Oracle® Database

Database Administrator’s Guide

21c
F31835-17
March 2025

Oracle Database Database Administrator’s Guide, 21c

F31835-17

Primary Authors: Mark Doran, Padmaja Potineni, Rajesh Bhatiya

Contributing Authors: Donna Keesling

Contributors: A. Agrawal, L. Ashdown, P. Avril, D. Austin, T. Baby, H. Baer, S. Ball, S. Battula, M. Bauer, T. Bednar, E. Belden, J. Byun, L. Carpenter, A. Chaudhry, C. Chang, B. Cheng, H. Chien, T. Chien, G. Christman, C. C. Chui, L. Clarke, D. Colello, C. Colrain, K. Cook, J. Creighton, A. Dadhich, S. Datta, S. Davidson, M. Dilman, S. Doraiswamy, J. Draaijer, M. Fallen, M. Fuller, D. Gagne, A. Ganesh, M. Girkar, GP Gongloor, J. Gonzalez, V. Goorah, S. Gopalan, S. Gupta, B. Habeck, S. Hase, W. Hu, P. Huey, K. Inoue, M. Ito, C. Iyer, K. Itikarlapalli, P. Jaganath, S. Jain, C. Jones, S. Joshi, B. Khaladkar, F. Kobylanski, B. Krishnan, V. Krishnaswamy, A. Kruglikov, B. Kuchibhotla, V. Kuhr, R. Kumar, S. Kumar, V. Kumar, H. Lakshmanan, A. Lee, B. Lee, J. Lee, S. K. Lee, T. Lee, C. Lei, B. Leung, Y. Li, I. Listvinsky, B. Llewellyn, H. Lombera, B. Lundhild, S. Lynn, R. Mani, V. Marwah, C. McGregor, J. McDonnell, J. McHugh, B. McGuirk, J. Meeks, K. Mensah, M. Minhas, K. Mohan, H. Mohankumar, A. Munnolimath, G. Mulagund, P. Murguia, P. Murthy, A. Mylavarapu, V. Moore, N. Muthukrishnan, S. Muthulingam, L. Nim, S. Panchumarthy, R. Pang, V. Panteleenko, R. Pingte, K. Rajamani, A. Raghavan, M. Ramacher, R. Ramkissoon, S. Ravindhran, G. Ravipati, A. Ray, W. Ren, K. Rich, J. Rivera, C. A. L. Rueda, R. Rungta, S. Sahu, P. Shanthaveerappa, Y. Sarig, M. Savanur, S. Shankar, D. Sharma, A. Shen, B. Sinha, S. Sonawane, J. Spiller, J. Stamos, D. Steiner, J. Stern, M. Stewart, S. Stoian, R. Swonger, M. Subramaniam, N. Sundarappa, M. Susairaj, A. Tran, A. Tsukerman, C. Tuzla, T. Ueda, K. Umamageswaran, D. Utzig, E. Voss, N. Wagner, X. Wang, M. Wei, S. Wertheimer, P. Wheeler, D. Williams, A. Witkowski, S. Wolicki, D. M. Wong, Z. Yang, T. F. Yu, W. Zhang

Contents

10 Managing Archived Redo Log Files

11 Managing Tablespaces

13 Transporting Data

14 Managing Undo

20 Managing Indexes

33 Distributed Transactions Concepts

34 Managing Distributed Transactions

Preface

This document describes how to create, configure, and administer an Oracle database.

Audience

This document is intended for database administrators who perform the following tasks:

- Create and configure one or more Oracle databases
- Monitor and tune Oracle databases
- Oversee routine maintenance operations for Oracle databases
- Create and maintain schema objects, such as tables, indexes, and views
- Schedule system and user jobs
- Diagnose, repair, and report problems

To use this document, you should be familiar with relational database concepts. You should also be familiar with the operating system environment under which you are running Oracle Database.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Concepts*
- *Oracle Multitenant Administrator’s Guide*
- *Oracle Database SQL Language Reference*

- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Automatic Storage Management Administrator's Guide*
- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database Error Messages*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database SQL Tuning Guide*
- *Oracle Database Development Guide*
- *Oracle Database PL/SQL Language Reference*
- *SQL*Plus User's Guide and Reference*
- *Oracle Database In-Memory Guide*
- *Oracle Database Using Oracle Sharding*

Many of the examples in this book use the sample schemas. See *Oracle Database Sample Schemas* for information about these schemas.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I
Basic Database Administration

Database administrators have specific responsibilities and must understand how to complete database administration tasks.

- Getting Started with Database Administration
  To get started with database administration, you must understand basic database concepts, such as the types of database users, database security, and privileges. You must also be able to complete basic tasks, such as submitting commands and SQL to the database and creating a password file.
- Configuring Automatic Restart of an Oracle Database
  Configure your Oracle database with the Oracle Restart feature to automatically restart the database, the listener, and other Oracle components after a hardware or software failure or whenever your database host computer restarts.
- Managing Processes
  Oracle Databases uses several processes so that multiple users and applications can connect to a single database instance simultaneously.
- Managing Memory
  Memory management involves maintaining optimal sizes for the Oracle Database instance memory structures as demands on the database change.
- Managing Users and Securing the Database
  Establish a security policy for every database.
- Monitoring the Database
  It is important that you monitor the operation of your database on a regular basis. Doing so not only informs you of errors that have not yet come to your attention but also gives you a better understanding of the normal operation of your database. Being familiar with normal behavior in turn helps you recognize when something is wrong.
- Diagnosing and Resolving Problems
  Oracle Database includes an advanced fault diagnosability infrastructure for collecting and managing diagnostic data, so as to diagnose and resolve database problems. **Diagnostic data** includes the trace files, dumps, and core files that are also present in previous releases, plus new types of diagnostic data that enable customers and Oracle Support to identify, investigate, track, and resolve problems quickly and effectively.

1 Getting Started with Database Administration

To get started with database administration, you must understand basic database concepts, such as the types of database users, database security, and privileges. You must also be able to complete basic tasks, such as submitting commands and SQL to the database and creating a password file.

- Changes on Oracle Database Release 21c for Oracle Database Administrator's Guide
  The following are changes in *Oracle Database Administrator's Guide* for Oracle Database Release 21c.
- Types of Oracle Database Users
  The types of users and their roles and responsibilities depend on the database site. A small site can have one database administrator who administers the database for application developers and users. A very large site can find it necessary to divide the duties of a database administrator among several people and among several areas of specialization.
- Tasks of a Database Administrator
  You must complete several specific tasks to design, implement, and maintain an Oracle Database.
- SQL Statements
  The primary means of communicating with Oracle Database is by submitting SQL statements.
- Identifying Your Oracle Database Software Release
  As many as five numbers may be required to fully identify a release.
- About Database Administrator Security and Privileges
  To perform the administrative tasks of an Oracle Database DBA, you need specific privileges within the database and possibly in the operating system of the server on which the database runs. Ensure that access to a database administrator's account is tightly controlled.
- Database Administrator Authentication
  As a DBA, you often perform special operations such as shutting down or starting up a database. Because only a DBA should perform these operations, the database administrator user names require a secure authentication scheme.
- Creating and Maintaining a Database Password File
  You can create a database password file using the password file creation utility, `ORAPWD`. For some operating systems, you can create this file as part of your standard installation.
- Data Utilities
  Oracle utilities are available to help you maintain the data in your Oracle Database.

1.1 Changes on Oracle Database Release 21c for Oracle Database Administrator's Guide

The following are changes in *Oracle Database Administrator's Guide* for Oracle Database Release 21c.

- New Features
  The following features are new in this release.
- Deprecated Features
  The following features are deprecated in this release.
- Desupported Features
  The following features are desupported in this release.

1.1.1 New Features

The following features are new in this release.

- Blockchain tables are tamper-proof, insert-only, tables that enable you to implement centralized blockchain applications.

  See Managing Blockchain Tables.
- Database Resident Connection Pooling (DRCP) can be configured at the PDB level.

  See Enabling Database Resident Connection Pooling.
- The attention log provides quick access to information about critical events that need action. Trace file records are classified based on the level of sensitivity of the record.

  See Attention Log and Trace Files.
- For Oracle Autonomous Database, the size of the sequence cache is dynamically computed based on the rate of usage of sequence numbers.

  See About Automatic Sizing of the Sequence Cache.
- You can enable automatic indexing at the table level.

  See Configuring Automatic Indexing in an Oracle Database.
- Transportable tablespace jobs can be restarted.
- In Oracle Cloud environments, a PDB can be downsized by reducing the value of the `CPU_MIN_COUNT` parameter. The default value for `JOB_QUEUE_PROCESSES` across all containers is automatically derived from the number of sessions and CPUs configured in the system.
- Password files created using older releases can be forced to use case sensitivity by migrating the password file to the new format.
- SecureFiles segments can be shrunk thereby reclaiming unused space in SecureFiles segments and improving performance.

1.1.2 Deprecated Features

The following features are deprecated in this release.

- Traditional auditing

  Oracle recommends that you use unified auditing instead of traditional auditing. Unified auditing enables selective and more effective auditing inside the Oracle database.

1.1.3 Desupported Features

The following features are desupported in this release.

- `REMOTE_OS_AUTHENT` initialization parameter

1.2 Types of Oracle Database Users

The types of users and their roles and responsibilities depend on the database site. A small site can have one database administrator who administers the database for application developers and users. A very large site can find it necessary to divide the duties of a database administrator among several people and among several areas of specialization.

- Database Administrators
  Each database requires at least one database administrator (DBA). An Oracle Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility.
- Security Officers
  In some cases, a site assigns one or more security officers to a database. A security officer enrolls users, controls and monitors user access to the database, and maintains system security.
- Network Administrators
  Some sites have one or more network administrators. A network administrator, for example, administers Oracle networking products, such as Oracle Net Services.
- Application Developers
  Application developers design and implement database applications.
- Application Administrators
  An Oracle Database site can assign one or more application administrators to administer a particular application. Each application can have its own administrator.
- Database Users
  Database users interact with the database through applications or utilities.

1.2.1 Database Administrators

Each database requires at least one database administrator (DBA). An Oracle Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility.

A database administrator's responsibilities can include the following tasks:

- Installing and upgrading the Oracle Database server and application tools
- Allocating system storage and planning future storage requirements for the database system
- Creating primary database storage structures (tablespaces) after application developers have designed an application
- Creating primary objects (tables, views, indexes) once application developers have designed an application
- Modifying the database structure, as necessary, from information given by application developers
- Enrolling users and maintaining system security
- Ensuring compliance with Oracle license agreements
- Controlling and monitoring user access to the database
- Monitoring and optimizing the performance of the database
- Planning for backup and recovery of database information

- Maintaining archived data on tape
- Backing up and restoring the database
- Contacting Oracle for technical support

1.2.2 Security Officers

In some cases, a site assigns one or more security officers to a database. A security officer enrolls users, controls and monitors user access to the database, and maintains system security.

As a DBA, you might not be responsible for these duties if your site has a separate security officer.

See *Oracle Database Security Guide* for information about the duties of security officers.

1.2.3 Network Administrators

Some sites have one or more network administrators. A network administrator, for example, administers Oracle networking products, such as Oracle Net Services.

See *Oracle Database Net Services Administrator's Guide* for information about the duties of network administrators.

See Also:

Distributed Database Management, for information on network administration in a distributed environment

1.2.4 Application Developers

Application developers design and implement database applications.

Their responsibilities include the following tasks:

- Designing and developing the database application
- Designing the database structure for an application
- Estimating storage requirements for an application
- Specifying modifications of the database structure for an application
- Relaying this information to a database administrator
- Tuning the application during development
- Establishing security measures for an application during development

Application developers can perform some of these tasks in collaboration with DBAs. See *Oracle Database Development Guide* for information about application development tasks.

1.2.5 Application Administrators

An Oracle Database site can assign one or more application administrators to administer a particular application. Each application can have its own administrator.

1.2.6 Database Users

Database users interact with the database through applications or utilities.

A typical user's responsibilities include the following tasks:

- Entering, modifying, and deleting data, where permitted
- Generating reports from the data

1.3 Tasks of a Database Administrator

You must complete several specific tasks to design, implement, and maintain an Oracle Database.

> **Note:**
>
> When upgrading to a new release, back up your existing production environment, both software and database, before installation. For information on preserving your existing production database, see *Oracle Database Upgrade Guide*.

- Task 1: Evaluate the Database Server Hardware
  Evaluate how Oracle Database and its applications can best use the available computer resources.
- Task 2: Install the Oracle Database Software
  As the database administrator, you install the Oracle Database server software and any front-end tools and database applications that access the database.
- Task 3: Plan the Database
  As the database administrator, you must plan the logical storage structure of the database, the overall database design, and a backup strategy for the database.
- Task 4: Create and Open the Database
  After you complete the database design, you can create the database and open it for normal use.
- Task 5: Back Up the Database
  After you create the database structure, perform the backup strategy you planned for the database.
- Task 6: Enroll System Users
  After you back up the database structure, you can enroll the users of the database in accordance with your Oracle license agreement, and grant appropriate privileges and roles to these users.
- Task 7: Implement the Database Design
  After you create and start the database, and enroll the system users, you can implement the planned logical structure database by creating all necessary tablespaces. When you have finished creating tablespaces, you can create the database objects.
- Task 8: Back Up the Fully Functional Database
  When the database is fully implemented, again back up the database. In addition to regularly scheduled backups, you should always back up your database immediately after implementing changes to the database structure.

- Task 9: Tune Database Performance
  Optimizing the performance of the database is one of your ongoing responsibilities as a DBA. Oracle Database provides a database resource management feature that helps you to control the allocation of resources among various user groups.
- Task 10: Download and Install Release Updates and Release Update Revisions
  After the database installation, download and install Release Updates (Updates) and Release Update Revisions (Revisions) for your Oracle software on a regular basis.
- Task 11: Roll Out to Additional Hosts
  After you have an Oracle Database installation properly configured, tuned, patched, and tested, you may want to roll that exact installation out to other hosts.

1.3.1 Task 1: Evaluate the Database Server Hardware

Evaluate how Oracle Database and its applications can best use the available computer resources.

This evaluation should reveal the following information:

- How many disk drives are available to the Oracle products
- How many, if any, dedicated tape drives are available to Oracle products
- How much memory is available to the instances of Oracle Database you will run (see your system configuration documentation)

1.3.2 Task 2: Install the Oracle Database Software

As the database administrator, you install the Oracle Database server software and any front-end tools and database applications that access the database.

In some distributed processing installations, the database is controlled by a central computer (database server) and the database tools and applications are executed on remote computers (clients). In this case, you must also install the Oracle Net components necessary to connect the remote systems to the computer that executes Oracle Database.

For more information on what software to install, see "Identifying Your Oracle Database Software Release".

See Also:

For specific requirements and instructions for installation, see the following documentation:

- The Oracle documentation specific to your operating system
- The installation guides for your front-end tools and Oracle Net drivers

1.3.3 Task 3: Plan the Database

As the database administrator, you must plan the logical storage structure of the database, the overall database design, and a backup strategy for the database.

It is important to plan how the logical storage structure of the database will affect system performance and various database management operations. For example, before creating any tablespaces for your database, you should know how many data files will comprise the tablespace, what type of information will be stored in each tablespace, and on which disk drives the data files will be physically stored. When planning the overall logical storage of the database structure, take into account the effects that this structure will have when the database is actually created and running. Consider how the logical storage structure of the database will affect:

- The performance of the computer running Oracle Database
- The performance of the database during data access operations
- The efficiency of backup and recovery procedures for the database

Plan the relational design of the database objects and the storage characteristics for each of these objects. By planning the relationship between each object and its physical storage before creating it, you can directly affect the performance of the database as a unit. Be sure to plan for the growth of the database.

In distributed database environments, this planning stage is extremely important. The physical location of frequently accessed data dramatically affects application performance.

During the planning stage, develop a backup strategy for the database. You can alter the logical storage structure or design of the database to improve backup efficiency.

It is beyond the scope of this book to discuss relational and distributed database design. If you are not familiar with such design issues, see accepted industry-standard documentation.

Oracle Database Structure and Storage, and Schema Objects, provide specific information on creating logical storage structures, objects, and integrity constraints for your database.

1.3.4 Task 4: Create and Open the Database

After you complete the database design, you can create the database and open it for normal use.

You can create a database at installation time, using the Database Configuration Assistant, or you can supply your own scripts for creating a database.

See Also:

- *Oracle Multitenant Administrator's Guide* for information on creating a database
- *Oracle Database SQL Language Reference* for guidance in starting up the database

1.3.5 Task 5: Back Up the Database

After you create the database structure, perform the backup strategy you planned for the database.

Create any additional redo log files, take the first full database backup (online or offline), and schedule future database backups at regular intervals.

See Also:

Oracle Database Backup and Recovery User's Guide

1.3.6 Task 6: Enroll System Users

After you back up the database structure, you can enroll the users of the database in accordance with your Oracle license agreement, and grant appropriate privileges and roles to these users.

See Managing Users and Securing the Database for guidance in this task.

1.3.7 Task 7: Implement the Database Design

After you create and start the database, and enroll the system users, you can implement the planned logical structure database by creating all necessary tablespaces. When you have finished creating tablespaces, you can create the database objects.

Oracle Database Structure and Storage and Schema Objects provide information on creating logical storage structures and objects for your database.

1.3.8 Task 8: Back Up the Fully Functional Database

When the database is fully implemented, again back up the database. In addition to regularly scheduled backups, you should always back up your database immediately after implementing changes to the database structure.

1.3.9 Task 9: Tune Database Performance

Optimizing the performance of the database is one of your ongoing responsibilities as a DBA. Oracle Database provides a database resource management feature that helps you to control the allocation of resources among various user groups.

The database resource manager is described in Managing Resources with Oracle Database Resource Manager.

See Also:

Oracle Database Performance Tuning Guide for information about tuning your database and applications

1.3.10 Task 10: Download and Install Release Updates and Release Update Revisions

After the database installation, download and install Release Updates (Updates) and Release Update Revisions (Revisions) for your Oracle software on a regular basis.

Starting with Oracle Database 18c, Oracle provides quarterly updates in the form of Release Updates (Updates) and Release Update Revisions (Revisions). Oracle no longer releases patch sets. Check the My Oracle Support website for required updates for your installation.

See Also:

- *Oracle Database Installation Guide* for your platform for instructions on downloading and installing Release Updates (Updates) and Release Update Revisions (Revisions)
- My Oracle Support Note 2285040.1

1.3.11 Task 11: Roll Out to Additional Hosts

After you have an Oracle Database installation properly configured, tuned, patched, and tested, you may want to roll that exact installation out to other hosts.

Reasons to do this include the following:

- You have multiple production database systems.
- You want to create development and test systems that are identical to your production system.

Instead of installing, tuning, and patching on each additional host, you can clone your tested Oracle Database installation to other hosts, saving time and avoiding inconsistencies. There are two types of cloning available to you:

- Cloning an Oracle home—Just the configured and patched binaries from the Oracle home directory and subdirectories are copied to the destination host and *fixed* to match the new environment. You can then start an instance with this cloned home and create a database.

  You can use Oracle Enterprise Manager Cloud Control to clone an Oracle home to one or more destination hosts. You can manually clone an Oracle home using a set of provided scripts and Oracle Universal Installer.
- Cloning a database—The tuned database, including database files, initialization parameters, and so on, are cloned to an existing Oracle home (possibly a cloned home).

  You can use Cloud Control to clone an Oracle database instance to an existing Oracle home.

See Also:

- *Oracle Enterprise Manager Cloud Administration Guide*
- *Oracle Enterprise Manager Lifecycle Management Administrator's Guide*
- Cloud Control online help
- *Oracle Multitenant Administrator's Guide*

1.4 SQL Statements

The primary means of communicating with Oracle Database is by submitting SQL statements.

- Submitting Commands and SQL to the Database
  There are several ways to submit SQL statements and commands to Oracle Database.
- About SQL*Plus
  SQL*Plus is the primary command-line interface to your Oracle database. You use SQL*Plus to start up and shut down the database, set database initialization parameters, create and manage users, create and alter database objects (such as tables and indexes), insert and update data, run SQL queries, and more.
- Connecting to the Database with SQL*Plus
  Connect to the Oracle Database instance using SQL*Plus.

1.4.1 Submitting Commands and SQL to the Database

There are several ways to submit SQL statements and commands to Oracle Database.

- Directly, using the command-line interface of SQL*Plus
- Indirectly, using a graphical user interface, such as Oracle Enterprise Manager Database Express (EM Express) or Oracle Enterprise Manager Cloud Control (Cloud Control)

  With these tools, you use an intuitive graphical interface to administer the database, and the tool submits SQL statements and commands behind the scenes.

  See the online help for the tool for more information.
- Directly, using SQL Developer

  Developers use SQL Developer to create and test database schemas and applications, although you can also use it for database administration tasks.

  See *Oracle SQL Developer User's Guide* for more information.

Oracle Database also supports a superset of SQL, which includes commands for starting up and shutting down the database, modifying database configuration, and so on.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

1.4.2 About SQL*Plus

SQL*Plus is the primary command-line interface to your Oracle database. You use SQL*Plus to start up and shut down the database, set database initialization parameters, create and manage users, create and alter database objects (such as tables and indexes), insert and update data, run SQL queries, and more.

Before you can submit SQL statements and commands, you must connect to the database. With SQL*Plus, you can connect locally or remotely. **Connecting locally** means connecting to an Oracle database running on the same computer on which you are running SQL*Plus. **Connecting remotely** means connecting over a network to an Oracle database that is running on a remote computer. Such a database is referred to as a **remote database**. The SQL*Plus executable on the local computer is provided by a full Oracle Database installation, an Oracle Client installation, or an Instant Client installation.

See Also:

*SQL*Plus User's Guide and Reference*

1.4.3 Connecting to the Database with SQL*Plus

Connect to the Oracle Database instance using SQL*Plus.

- About Connecting to the Database with SQL*Plus
  Oracle Database includes the following components: the Oracle Database instance, which is a collection of processes and memory, and a set of disk files that contain user data and system data.
- Step 1: Open a Command Window
  Take the necessary action on your platform to open a window into which you can enter operating system commands.
- Step 2: Set Operating System Environment Variables
  Depending on your platform, you may have to set environment variables before starting SQL*Plus, or at least verify that they are set properly.
- Step 3: Start SQL*Plus
  To connect to Oracle Database, use one of these options to start SQL*Plus.
- Step 4: Submit the SQL*Plus CONNECT Command
  Submit the SQL*Plus `CONNECT` command to initially connect to the Oracle database instance or at any time to reconnect as a different user.

1.4.3.1 About Connecting to the Database with SQL*Plus

Oracle Database includes the following components: the Oracle Database instance, which is a collection of processes and memory, and a set of disk files that contain user data and system data.

Each instance has an instance ID, also known as a system ID (SID). Because there can be multiple Oracle instances on a host computer, each with its own set of data files, you must identify the instance to which you want to connect. For a local connection, you identify the instance by setting operating system environment variables. For a remote connection, you identify the instance by specifying a network address and a database service name. For both local and remote connections, you must set environment variables to help the operating system find the SQL*Plus executable and to provide the executable with a path to its support files and scripts.

To manage objects that are shared by the multitenant container database and its pluggable databases (PDBs), such as control files, redo log files, or archived redo log files, connect to the CDB root. Objects such as tablespaces, data files, or temp files can be created in the CDB root or a PDB. To manage such objects, connect to the container that owns the object.

In the remainder of this book, connecting to the database means connecting to the CDB root.

See Also:

Oracle Database Concepts for background information about the Oracle instance

1.4.3.2 Step 1: Open a Command Window

Take the necessary action on your platform to open a window into which you can enter operating system commands.

- Open a command window.

1.4.3.3 Step 2: Set Operating System Environment Variables

Depending on your platform, you may have to set environment variables before starting SQL*Plus, or at least verify that they are set properly.

For example, on most platforms, you must set the environment variables `ORACLE_SID` and `ORACLE_HOME`. In addition, you must configure the `PATH` environment variable to include the `ORACLE_HOME/bin` directory. Some platforms may require additional environment variables:

- On Unix and Linux, set environment variables by entering operating system commands as needed.
- On Microsoft Windows, the installer automatically assigns values to `ORACLE_HOME` and `ORACLE_SID` in the Windows registry. Modify the `PATH` environment variable as needed.

If you did not create a database upon installation, then the installer does not set `ORACLE_SID` in the registry; after you create your database at a later time, you must set the `ORACLE_SID` environment variable from a command window.

Unix and Linux installations come with two scripts, `oraenv` and `coraenv`, that you can use to easily set environment variables.

For all platforms, when switching between instances with different Oracle homes, you must change the `ORACLE_HOME` environment variable. If multiple instances share the same Oracle home, then you must change only `ORACLE_SID` when switching instances.

Example 1-1 Setting Environment Variables in Unix (C Shell)

```
setenv ORACLE_SID orcl
setenv ORACLE_HOME /u01/app/oracle/product/database_release_number/dbhome_1
setenv LD_LIBRARY_PATH $ORACLE_HOME/lib:/usr/lib:/usr/dt/lib:/usr/openwin/lib:/usr/ccs/lib
```

Example 1-2 Setting Environment Variables in Linux (Bash Shell)

```
export ORACLE_SID=orcl
export ORACLE_HOME=/u01/app/oracle/product/database_release_number/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:/usr/lib:/usr/dt/lib:/usr/openwin/lib:/usr/ccs/lib
```

1.4.3.4 Step 3: Start SQL*Plus

To connect to Oracle Database, use one of these options to start SQL*Plus.

1. Do one of the following:
   - Ensure that the `PATH` environment variable contains *`$ORACLE_HOME`*`/bin`.

- Change directory to *$ORACLE_HOME*/bin. Ensure that the PATH environment variable contains a dot (".").

2. Enter the following command (case-sensitive on Unix and Linux):

   ```
   sqlplus /nolog
   ```

   You can also run the sqlplus command by specifying its complete path:

   ```
   $ORACLE_HOME/bin/sqlplus /nolog
   ```

1.4.3.5 Step 4: Submit the SQL*Plus CONNECT Command

Submit the SQL*Plus CONNECT command to initially connect to the Oracle database instance or at any time to reconnect as a different user.

- In SQL*Plus, submit the CONNECT command.

  This command is used to connect to the CDB root or a particular PDB.

Example 1-3 Connecting to a Local Database User

This simple example connects to a local database as user SYSTEM. SQL*Plus prompts for the SYSTEM user password.

```
connect system
```

Example 1-4 Connecting to a Local Database User with SYSDBA Privilege

This example connects to a local database as user SYS with the SYSDBA privilege. SQL*Plus prompts for the SYS user password.

```
connect sys as sysdba
```

When connecting as user SYS, you must connect AS SYSDBA.

Example 1-5 Connecting to a Local Database User with SYSBACKUP Privilege

This example connects to a local database as user SYSBACKUP with the SYSBACKUP privilege. SQL*Plus prompts for the SYSBACKUP user password.

```
connect sysbackup as sysbackup
```

When connecting as user SYSBACKUP, you must connect AS SYSBACKUP.

Example 1-6 Connecting Locally with SYSDBA Privilege with Operating System Authentication

This example connects locally with the SYSDBA privilege with operating system authentication.

```
connect / as sysdba
```

Example 1-7 Connecting to a Pluggable Database with SYSDBA Privilege

This example connects locally to a pluggable database (PDB) named sales_pdb as user SYS with the SYSDBA privilege. SQL*Plus prompts for the SYS user password.

```
connect sys@sales_pdb as sysdba
```

When connecting as user `SYS`, you must connect `AS SYSDBA`.

Example 1-8   Connecting with Easy Connect Syntax

This example uses Easy Connect syntax to connect as user `salesadmin` to a remote database running on the host `dbhost.example.com`. The Oracle Net listener (the listener) is listening on the default port (1521). The database service is `sales.example.com`. SQL*Plus prompts for the `salesadmin` user password.

```
connect salesadmin@"dbhost.example.com/sales.example.com"
```

Example 1-9   Connecting with Easy Connect Syntax with the Service Handler Type Indicated

This example is identical to the preceding example of connecting with Easy Connect, except that the service handler type is indicated.

```
connect salesadmin@"dbhost.example.com/sales.example.com:dedicated"
```

Example 1-10   Connecting with Easy Connect Syntax with a Nondefault Listener Port

This example is identical to the preceding example of connecting with Easy Connect, except that the listener is listening on the nondefault port number 1522.

```
connect salesadmin@"dbhost.example.com:1522/sales.example.com"
```

Example 1-11   Connecting with Easy Connect Syntax with the Host IP Address

This example is identical to the preceding example of connecting with Easy Connect, except that the host IP address is substituted for the host name.

```
connect salesadmin@"192.0.2.5/sales.example.com"
```

Example 1-12   Connecting with an IPv6 Address

This example connects to the database using an Internet Protocol version 6 (IPv6) address. Note the enclosing square brackets.

```
connect salesadmin@"[2001:0DB8:0:0::200C:417A]/sales.example.com"
```

Example 1-13   Connecting by Specifying an Instance

This example specifies the instance to which to connect, and omits the database service name. Note that when you specify only the instance, you cannot specify the service handler type.

```
connect salesadmin@"dbhost.example.com/orcl"
```

Example 1-14   Connecting with a Net Service Name

This example connects remotely as user `salesadmin` to the database service designated by the net service name `sales1`. SQL*Plus prompts for the `salesadmin` user password.

```
connect salesadmin@sales1
```

Example 1-15   Connecting with External Authentication

This example connects remotely with external authentication to the database service designated by the net service name `sales1`.

```
connect /@sales1
```

Example 1-16 Connecting with SYSDBA Privilege and External Authentication

This example connects remotely with the `SYSDBA` privilege and with external authentication to the database service designated by the net service name `sales1`.

```
connect /@sales1 as sysdba
```

Example 1-17 Connecting as a User with a Service Name

This example connects remotely as user `salesadmin` to the database service designated by the net service name `sales1`. The database session starts in the `rev21` edition. SQL*Plus prompts for the `salesadmin` user password.

```
connect salesadmin@sales1 edition=rev21
```

Note:

If you come across any issues while connecting to the database as a user with the `SYSDBA` privileges, then refer to My Oracle Support Notes 69642.1, 233223.1, 18089.1, and 747456.1.

- Syntax of the SQL*Plus CONNECT Command
  Use the SQL*Plus `CONNECT` command to initially connect to the Oracle instance or to reconnect to the Oracle instance.

1.4.3.5.1 Syntax of the SQL*Plus CONNECT Command

Use the SQL*Plus `CONNECT` command to initially connect to the Oracle instance or to reconnect to the Oracle instance.

Syntax

```
CONN[ECT] [logon] [AS {SYSOPER | SYSDBA | SYSBACKUP | SYSDG | SYSKM | SYSRAC}]
```

The syntax of *`logon`* is as follows:

```
{username | /}[@connect_identifier] [edition={edition_name | DATABASE_DEFAULT}]
```

When you provide the *`username`*, SQL*Plus prompts for a password. The password is not echoed as you type it.

The following table describes the syntax components of the `CONNECT` command.

Syntax Component	Description
/	Calls for external authentication of the connection request. A database password is not used in this type of authentication. The most common form of external authentication is operating system authentication, where the database user is authenticated by having logged in to the host operating system with a certain host user account. External authentication can also be performed with an Oracle wallet or by a network service. See *Oracle Database Security Guide* for more information. See also "Using Operating System Authentication".

Syntax Component	Description
`AS {SYSOPER | SYSDBA | SYSBACKUP | SYSDG | SYSKM | SYSRAC}`	Indicates that the database user is connecting with an administrative privilege. Only certain predefined administrative users or users who have been added to the password file may connect with these privileges. See "Administrative Privileges" for more information.
`username`	A valid database user name. The database authenticates the connection request by matching *`username`* against the data dictionary and prompting for a user password.
`connect_identifier` (1)	An Oracle Net connect identifier, for a remote connection. The exact syntax depends on the Oracle Net configuration. If omitted, SQL*Plus attempts connection to a local instance. A common connect identifier is a *net service name*. This is an alias for an Oracle Net connect descriptor (network address and database service name). The alias is typically resolved in the `tnsnames.ora` file on the local computer, but can be resolved in other ways. See *Oracle Database Net Services Administrator's Guide* for more information on connect identifiers.

Syntax Component	Description

`connect_identifier` (2)	As an alternative, a connect identifier can use *easy connect* syntax. Easy connect provides out-of-the-box TCP/IP connectivity for remote databases without having to configure Oracle Net Services on the client (local) computer.
	Easy connect syntax for the connect identifier is as follows (the enclosing double-quotes must be included):
	`"host[:port][/service_name][:server][/instance_name]"`
	where:
	• *`host`* is the host name or IP address of the computer hosting the remote database.
	Both IP version 4 (IPv4) and IP version 6 (IPv6) addresses are supported. IPv6 addresses must be enclosed in square brackets. See *Oracle Database Net Services Administrator's Guide* for information about IPv6 addressing.
	• *`port`* is the TCP port on which the Oracle Net listener on *`host`* listens for database connections. If omitted, 1521 is assumed.
	• *`service_name`* is the database service name to which to connect. It can be omitted if the Net Services listener configuration on the remote host designates a default service. If no default service is configured, then *`service_name`* must be supplied. Each database typically offers a standard service with a name equal to the global database name, which is made up of the `DB_NAME` and `DB_DOMAIN` initialization parameters as follows:
	`DB_NAME.DB_DOMAIN`
	If `DB_DOMAIN` is null, then the standard service name is just the `DB_NAME`. For example, if `DB_NAME` is `orcl` and `DB_DOMAIN` is `us.example.com`, then the standard service name is `orcl.us.example.com`.
	See "*Oracle Database SQL Language Reference*" for more information.
	• *`server`* is the type of service handler. Acceptable values are `dedicated`, `shared`, and `pooled`. If omitted, then the default type of server is chosen by the listener: shared server if configured, otherwise dedicated server.
	• *`instance_name`* is the instance to which to connect. You can specify both service name and instance name, which you would typically do only for Oracle Real Application Clusters (Oracle RAC) environments. For Oracle RAC or single instance environments, if you specify only instance name, then you connect to the default database service. If there is no default service configured in the `listener.ora` file, then an error is generated. You can obtain the instance name from the `INSTANCE_NAME` initialization parameter.
	See *Oracle Database Net Services Administrator's Guide* for more information on easy connect.

`edition={`*`edition_name`*` | DATABASE_DEFAULT}`	Specifies the edition in which the new database session starts. If you specify an edition, then it must exist, and you must have the `USE` privilege on it. If this clause is not specified, then the database default edition is used for the session. See *Oracle Database Development Guide* for information on editions and edition-based redefinition.

See Also:

- "Using Operating System Authentication"
- *SQL*Plus User's Guide and Reference* for more information on the `CONNECT` command
- *Oracle Database Net Services Administrator's Guide* for more information on net service names
- *Oracle Database Net Services Reference* for information on how to define the default service in `listener.ora`

1.5 Identifying Your Oracle Database Software Release

As many as five numbers may be required to fully identify a release.

Because Oracle Database continues to evolve and can require maintenance, Oracle periodically produces new releases. Not all customers initially subscribe to a new release or require specific maintenance for their existing release. As a result, multiple releases of the product exist simultaneously.

- About Oracle Database Release Numbers
  Oracle Database releases are categorized by five numeric segments that indicate release information.
- Checking Your Current Release Number
  To identify the release of Oracle Database that is currently installed and to see the release levels of other database components you are using, query the data dictionary view `PRODUCT_COMPONENT_VERSION`.

1.5.1 About Oracle Database Release Numbers

Oracle Database releases are categorized by five numeric segments that indicate release information.

Note:

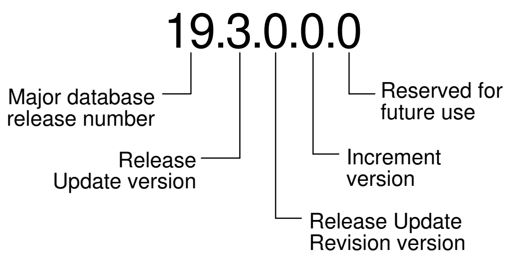
Starting with Oracle Database 18c, Oracle provides quarterly updates in the form of Release Updates (Updates) and Release Update Revisions (Revisions). Oracle no longer releases patch sets. For more information, see My Oracle Support Note 2285040.1.

Oracle Database releases are released in `version` and `version_full` releases.

The `version` release is designated in the form *`major release version`*`.0.0.0.0`. The major release version is based on the last two digits of the year in which an Oracle Database version is released for the first time. For example, the Oracle Database version released for the first time in the year 2018 has the major release version of 18, and thus its `version` release is `18.0.0.0.0`.

The `version_full` release is an update of a `version` release and is designated based on the major release version, the quarterly release update version (Update), and the quarterly release update revision version (Revision). The `version_full` releases are categorized by five numeric segments separated by periods as shown in the following example:

Figure 1-1 Example of an Oracle Database Release Number



- First numeral: This numeral indicates the major release version. It also denotes the last two digits of the year in which the Oracle Database version was released for the first time.
- Second numeral: This numeral indicates the release update version (Update).
- Third numeral: This numeral indicates the release update revision version (Revision).
- Fourth numeral: This numeral indicates the increment version. This nomenclature can apply to updates in future releases.
- Fifth numeral: This numeral is reserved for future use.

Note:

The first three numerals mainly identify an Oracle Database release.

Caution:

Oracle strongly recommends that you apply the most recent release update (Update) or bundle patch or patch set update to your source and target databases before starting an upgrade, and before starting a downgrade.

Related Topics

- My Oracle Support note 2285040.1

1.5.2 Checking Your Current Release Number

To identify the release of Oracle Database that is currently installed and to see the release levels of other database components you are using, query the data dictionary view `PRODUCT_COMPONENT_VERSION`.

A sample query follows. Other product release levels may increment independent of the database server.

```
COL PRODUCT FORMAT A38
COL VERSION FORMAT A10
COL VERSION_FULL FORMAT A12
COL STATUS FORMAT A12
SELECT * FROM PRODUCT_COMPONENT_VERSION;

PRODUCT                              VERSION    VERSION_FULL STATUS
------------------------------------ ---------- ------------ ------------
NLSRTL                               19.0.0.0.0 19.2.0.0.0   Production
Oracle Database 19c Enterprise Edition 19.0.0.0.0 19.2.0.0.0   Production
PL/SQL                               19.0.0.0.0 19.2.0.0.0   Production
...
```

It is important to convey to Oracle the results of this query when you report problems with the software.

Note:

You can also query the `V$VERSION` view to see component-level information about all the Oracle Database components that are currently installed.

1.6 About Database Administrator Security and Privileges

To perform the administrative tasks of an Oracle Database DBA, you need specific privileges within the database and possibly in the operating system of the server on which the database runs. Ensure that access to a database administrator's account is tightly controlled.

- The Database Administrator's Operating System Account
  To perform many of the administrative duties for a database, you must be able to execute operating system commands.
- Administrative User Accounts
  Oracle Database provides several administrative user accounts that are associated with administrative privileges.

1.6.1 The Database Administrator's Operating System Account

To perform many of the administrative duties for a database, you must be able to execute operating system commands.

Depending on the operating system on which Oracle Database is running, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require operating system privileges or access rights that other database users do not require (for example, to perform Oracle Database software installation). Although you do not need the Oracle Database files to be stored in your account, you should have access to them.

See Also:

Your operating system-specific Oracle documentation. The method of creating the account of the database administrator is specific to the operating system.

1.6.2 Administrative User Accounts

Oracle Database provides several administrative user accounts that are associated with administrative privileges.

- About Administrative User Accounts
  Administrative user accounts have special privileges required to administer areas of the database, such as the `CREATE ANY TABLE` or `ALTER SESSION` privilege, or `EXECUTE` privilege on packages owned by the `SYS` schema.
- SYS
  When you create an Oracle database, the user `SYS` is automatically created with all the privileges.
- SYSTEM
  When you create an Oracle database, the user `SYSTEM` is also automatically created and granted the `DBA` role.
- SYSBACKUP, SYSDG, SYSKM, and SYSRAC
  When you create an Oracle database, the following users are automatically created to facilitate separation of duties for database administrators: `SYSBACKUP`, `SYSDG`, `SYSKM`, and `SYSRAC`.
- The DBA Role
  A predefined `DBA` role is automatically created with every Oracle Database installation. This role contains most database system privileges. Therefore, the DBA role should be granted only to actual database administrators.

1.6.2.1 About Administrative User Accounts

Administrative user accounts have special privileges required to administer areas of the database, such as the `CREATE ANY TABLE` or `ALTER SESSION` privilege, or `EXECUTE` privilege on packages owned by the `SYS` schema.

The following administrative user accounts are automatically created when Oracle Database is installed:

- `SYS`
- `SYSTEM`
- `SYSBACKUP`
- `SYSDG`
- `SYSKM`
- `SYSRAC`

Oracle recommends that you create at least one additional administrative user and grant it appropriate privileges for performing daily administrative tasks. Do not use `SYS` and `SYSTEM` for these purposes.

Note:

Both Oracle Universal Installer (OUI) and Database Configuration Assistant (DBCA) prompt for `SYS` and `SYSTEM` passwords and do not accept default passwords.

See Also:

- *Oracle Multitenant Administrator's Guide* for information about passwords for the SYS and SYSTEM users
- *Oracle Database Security Guide* for the security checklist for configuring a database

1.6.2.2 SYS

When you create an Oracle database, the user SYS is automatically created with all the privileges.

All of the base tables and views for the database data dictionary are stored in the schema SYS. These base tables and views are critical for the operation of Oracle Database. To maintain the integrity of the data dictionary, tables in the SYS schema are manipulated only by the database. They should never be modified by any user or database administrator, and no one should create any tables in the schema of user SYS. (However, you can change the storage parameters of the data dictionary settings if necessary.)

Ensure that most database users are never able to connect to Oracle Database using the SYS account.

1.6.2.3 SYSTEM

When you create an Oracle database, the user SYSTEM is also automatically created and granted the DBA role.

The SYSTEM user name is used to create additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the SYSTEM schema to store tables of interest to non-administrative users.

1.6.2.4 SYSBACKUP, SYSDG, SYSKM, and SYSRAC

When you create an Oracle database, the following users are automatically created to facilitate separation of duties for database administrators: SYSBACKUP, SYSDG, SYSKM, and SYSRAC.

These users separate duties in the following ways:

- SYSBACKUP facilitates Oracle Recovery Manager (RMAN) backup and recovery operations either from RMAN or SQL*Plus.
- SYSDG facilitates Data Guard operations. The user can perform operations either with Data Guard Broker or with the DGMGRL command-line interface.
- SYSKM facilitates Transparent Data Encryption keystore operations.
- SYSRAC facilitates Oracle Real Application Clusters (Oracle RAC) operations by connecting to the database by the Clusterware agent on behalf of Oracle RAC utilities such as SRVCTL.

  The SYSRAC administrative privilege cannot be granted to database users and is not supported in a password file. The SYSRAC administrative privilege is used only by the

Oracle agent of Oracle Clusterware to connect to the database using operating system authentication.

Each of these accounts provides a designated user for the new administrative privilege with the same name. Specifically, the `SYSBACKUP` account provides a designated user for the `SYSBACKUP` administrative privilege. The `SYSDG` account provides a designated user for the `SYSDG` administrative privilege. The `SYSKM` account provides a designated user for the `SYSKM` administrative privilege.

Create a user and grant to that user an appropriate administrative privilege to use when performing daily administrative tasks. Doing so enables you to manage each user account separately, and each user account can have a distinct password. Do not use the `SYSBACKUP`, `SYSDG`, or `SYSKM` user account for these purposes.

To use one of these administrative privileges, a user must exercise the privilege when connecting to a database by specifying the privilege, for example `AS SYSBACKUP`, `AS SYSDG`, or `AS SYSKM`. If the authentication succeeds, then the user is connected to a database with a session in which the administrative privilege is enabled. In this case, the session user is the corresponding administrative user account. For example, if user `bradmin` connects with the `AS SYSBACKUP` administrative privilege, then the session user is `SYSBACKUP`.

Note:

- These user accounts cannot be dropped.
- These user accounts are *schema only* accounts, that is, they are created without passwords. You can assign passwords to these user accounts whenever you want them to be authenticated.

See Also:

- "Administrative Privileges"
- *Oracle Database Security Guide*

1.6.2.5 The DBA Role

A predefined `DBA` role is automatically created with every Oracle Database installation. This role contains most database system privileges. Therefore, the DBA role should be granted only to actual database administrators.

Note:

The DBA role does not include the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` system privileges. These are special administrative privileges that allow an administrator to perform basic database administration tasks, such as creating the database and instance startup and shutdown. These administrative privileges are discussed in "Administrative Privileges".

See Also:

- *Oracle Database Security Guide* for more information about administrative user accounts
- "Using Password File Authentication"

1.7 Database Administrator Authentication

As a DBA, you often perform special operations such as shutting down or starting up a database. Because only a DBA should perform these operations, the database administrator user names require a secure authentication scheme.

- Administrative Privileges
  Administrative privileges that are required for an administrator to perform basic database operations are granted through special system privileges.
- Operations Authorized by Administrative Privileges
  Each administrative privilege authorizes a specific set of operations.
- Authentication Methods for Database Administrators
  Database administrators can be authenticated with account passwords, operating system (OS) authentication, password files, or strong authentication with a directory-based authentication service, such as Oracle Internet Directory.
- Using Operating System Authentication
  Membership in special operating system groups enables a DBA to authenticate to the database through the operating system rather than with a database user name and password. This is known as operating system authentication.
- Using Password File Authentication
  You can use password file authentication for an Oracle database instance and for an Oracle Automatic Storage Management (Oracle ASM) instance. The password file for an Oracle database is called a database password file, and the password file for Oracle ASM is called an Oracle ASM password file.

1.7.1 Administrative Privileges

Administrative privileges that are required for an administrator to perform basic database operations are granted through special system privileges.

These privileges are:

- `SYSDBA`
- `SYSOPER`
- `SYSBACKUP`
- `SYSDG`
- `SYSKM`
- `SYSRAC`

Excluding the `SYSRAC` privilege, grant these privileges to users depending upon the level of authorization they require. The `SYSRAC` privilege cannot be granted to users because it is used only by the Oracle agent of Oracle Clusterware to connect to the database using operating system authentication.

Starting with Oracle Database 12*c* Release 1 (12.1), the `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges are available. Starting with Oracle Database 12*c* Release 2 (12.2), the `SYSRAC` administrative privilege is available. Each new administrative privilege grants the minimum required privileges to complete tasks in each area of administration. The new administrative privileges enable you to avoid granting `SYSDBA` administrative privilege for many common tasks.

Note:

These administrative privileges allow access to a database instance even when the database is not open. Control of these privileges is totally outside of the database itself. Methods for authenticating database administrators with these privileges include operating system (OS) authentication, password files, and strong authentication with a directory-based authentication service.

These privileges can also be thought of as types of connections that enable you to perform certain database operations for which privileges cannot be granted in any other fashion. For example, if you have the `SYSDBA` privilege, then you can connect to the database by specifying the `AS SYSDBA` clause in the `CONNECT` command and perform `STARTUP` and `SHUTDOWN` operations. See "Authentication Methods for Database Administrators".

1.7.2 Operations Authorized by Administrative Privileges

Each administrative privilege authorizes a specific set of operations.

The following table lists the operations that are authorized by each administrative privilege:

Administrative Privilege	Operations Authorized
`SYSDBA`	• Perform `STARTUP` and `SHUTDOWN` operations • `ALTER DATABASE`: open, mount, back up, or change character set • `CREATE DATABASE` • `DROP DATABASE` • `CREATE SPFILE` • `ALTER DATABASE ARCHIVELOG` • `ALTER DATABASE RECOVER` • Includes the `RESTRICTED SESSION` privilege This administrative privilege allows most operations, including the ability to view user data. It is the most powerful administrative privilege.

Administrative Privilege	Operations Authorized
SYSOPER	• Perform STARTUP and SHUTDOWN operations • CREATE SPFILE • ALTER DATABASE: open, mount, or back up • ALTER DATABASE ARCHIVELOG • ALTER DATABASE RECOVER (Complete recovery only. Any form of incomplete recovery, such as UNTIL TIME|CHANGE|CANCEL|CONTROLFILE requires connecting as SYSDBA.) • Includes the RESTRICTED SESSION privilege This privilege allows a user to perform basic operational tasks, but without the ability to view user data.
SYSBACKUP	This privilege allows a user to perform backup and recovery operations either from Oracle Recovery Manager (RMAN) or SQL*Plus. See *Oracle Database Security Guide* for the full list of operations allowed by this administrative privilege.
SYSDG	This privilege allows a user to perform Data Guard operations. You can use this privilege with either Data Guard Broker or the DGMGRL command-line interface. See *Oracle Database Security Guide* for the full list of operations allowed by this administrative privilege.
SYSKM	This privilege allows a user to perform Transparent Data Encryption keystore operations. See *Oracle Database Security Guide* for the full list of operations allowed by this administrative privilege.
SYSRAC	This privilege allows the Oracle agent of Oracle Clusterware to perform Oracle Real Application Clusters (Oracle RAC) operations. See *Oracle Database Security Guide* for the full list of operations allowed by this administrative privilege.

The manner in which you are authorized to use these privileges depends upon the method of authentication that you use.

When you connect with an administrative privilege, you connect with a current schema that is not generally associated with your username. For SYSDBA, the current schema is SYS. For SYSOPER, the current schema is PUBLIC. For SYSBACKUP, SYSDG, and SYSRAC, the current schema is SYS for name resolution purposes. However, the current schema for SYSKM is SYSKM.

Also, when you connect with an administrative privilege, you connect with a specific session user. When you connect as SYSDBA, the session user is SYS. For SYSOPER, the session user is PUBLIC. For SYSBACKUP, SYSDG, SYSKM, and SYSRAC, the session user is SYSBACKUP, SYSDG, SYSKM, and SYSRAC, respectively.

See Also:

- "Administrative User Accounts"
- "Using Operating System Authentication"
- "Using Password File Authentication"
- *Oracle Database SQL Language Reference* for more information about the current schema and the session user
- *Oracle Database Security Guide*

Example 1-18 Current Schema When Connecting AS SYSDBA

This example illustrates that a user is assigned another schema (SYS) when connecting with the SYSDBA administrative privilege. Assume that the sample user mydba has been granted the SYSDBA administrative privilege and has issued the following command and statement:

```
CONNECT mydba
CREATE TABLE admin_test(name VARCHAR2(20));
```

Later, user mydba issues this command and statement:

```
CONNECT mydba AS SYSDBA
SELECT * FROM admin_test;
```

User mydba now receives the following error:

```
ORA-00942: table or view does not exist
```

Having connected as SYSDBA, user mydba now references the SYS schema, but the table was created in the mydba schema.

Example 1-19 Current Schema and Session User When Connecting AS SYSBACKUP

This example illustrates that a user is assigned another schema (SYS) and another session user (SYSBACKUP) when connecting with the SYSBACKUP administrative privilege. Assume that the sample user mydba has been granted the SYSBACKUP administrative privilege and has issued the following command and statements:

```
CONNECT mydba AS SYSBACKUP

SELECT SYS_CONTEXT('USERENV', 'CURRENT_SCHEMA') FROM DUAL;

SYS_CONTEXT('USERENV','CURRENT_SCHEMA')
--------------------------------------------------------------------------------
SYS

SELECT SYS_CONTEXT('USERENV', 'SESSION_USER') FROM DUAL;

SYS_CONTEXT('USERENV','SESSION_USER')
--------------------------------------------------------------------------------
SYSBACKUP
```

1.7.3 Authentication Methods for Database Administrators

Database administrators can be authenticated with account passwords, operating system (OS) authentication, password files, or strong authentication with a directory-based authentication service, such as Oracle Internet Directory.

- About Authentication Methods for Database Administrators
  There are several ways to authenticate database administrators.
- Nonsecure Remote Connections
  To connect to Oracle Database as a privileged user over a nonsecure connection, you must be authenticated by a password file.
- Local Connections and Secure Remote Connections
  You can connect to Oracle Database as a privileged user over a local connection or a secure remote connection.

1.7.3.1 About Authentication Methods for Database Administrators

There are several ways to authenticate database administrators.

Oracle database can authenticate database administrators through the data dictionary, (using an account password) like other users. Keep in mind that database passwords are case-sensitive. See *Oracle Database Security Guide* for more information about case-sensitive database passwords.

In addition to normal data dictionary authentication, the following methods are available for authenticating database administrators with the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` privilege:

- Operating system (OS) authentication
- Password file including Kerberos and SSL authentication services
- Strong authentication with a directory-based authentication service, such as Oracle Internet Directory

Note:

The `SYSRAC` privilege only allows OS authentication by the Oracle agent of Oracle Clusterware. Password files and strong authentication cannot be used with the `SYSRAC` privilege.

These methods are required to authenticate a database administrator when the database is not started or otherwise unavailable. (They can also be used when the database is available.)

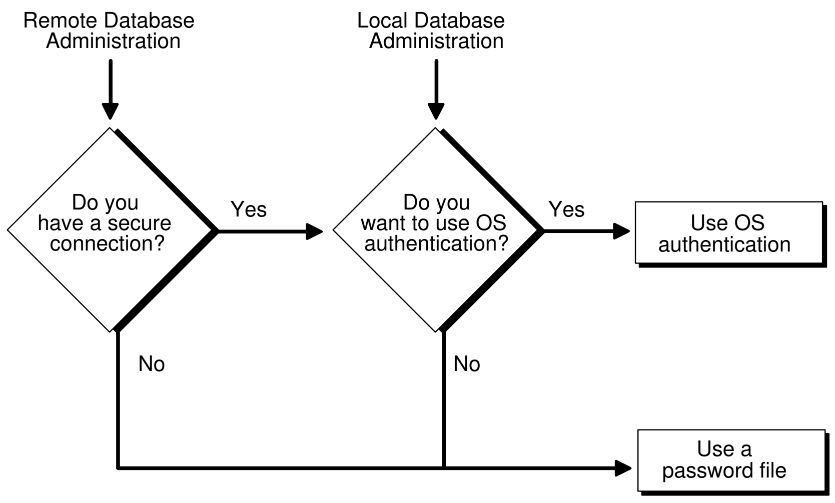
The remainder of this section focuses on operating system authentication and password file authentication. See *Oracle Database Security Guide* for information about authenticating database administrators with directory-based authentication services.

Note:

Operating system authentication takes precedence over password file authentication. If you meet the requirements for operating system authentication, then even if you use a password file, you will be authenticated by operating system authentication.

Your choice is influenced by whether you intend to administer your database locally on the same system where the database resides, or whether you intend to administer many different databases from a single remote client. The following figure illustrates the choices you have for database administrator authentication schemes.

Figure 1-2 Database Administrator Authentication Methods


Remote Database Administration
Local Database Administration
Do you have a secure connection?
Yes
Do you want to use OS authentication?
Yes
Use OS authentication
No
No
Use a password file

If you are performing remote database administration, then consult your Oracle Net documentation to determine whether you are using a secure connection. Most popular connection protocols, such as TCP/IP and DECnet, are not secure.

See Also:

- *Oracle Database Security Guide* for information about authenticating database administrators with directory-based authentication services.
- *Oracle Database Net Services Administrator's Guide*

1.7.3.2 Nonsecure Remote Connections

To connect to Oracle Database as a privileged user over a nonsecure connection, you must be authenticated by a password file.

When using password file authentication, the database uses a password file to keep track of database user names that have been granted the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege. This form of authentication is discussed in "Using Password File Authentication".

1.7.3.3 Local Connections and Secure Remote Connections

You can connect to Oracle Database as a privileged user over a local connection or a secure remote connection.

You can connect in two ways:

- If the database has a password file and you have been granted a system privilege, then you can connect and be authenticated by a password file.
- If the server is not using a password file, or if you have not been granted a system privilege and are therefore not in the password file, then you can use operating system authentication. On most operating systems, authentication for database administrators involves placing the operating system username of the database administrator in a special group.

  For example, users in the OSDBA group are granted the `SYSDBA` administrative privilege. Similarly, the OSOPER group is used to grant `SYSOPER` administrative privilege to users, the OSBACKUPDBA group is used to grant `SYSBACKUP` administrative privilege to users, the OSDGDBA group is used to grant `SYSDG` administrative privilege to users, the OSKMDBA group is used to grant `SYSKM` administrative privilege to users, and the OSRACDBA group is used to grant `SYSRAC` administrative privilege to users.

1.7.4 Using Operating System Authentication

Membership in special operating system groups enables a DBA to authenticate to the database through the operating system rather than with a database user name and password. This is known as operating system authentication.

- Operating System Groups
  Operating system groups are created and assigned specific names as part of the database installation process.
- Preparing to Use Operating System Authentication
  DBAs can authenticate to the database through the operating system rather than with a database user name and password.
- Connecting Using Operating System Authentication
  A user can connect to the database using operating system authentication.

1.7.4.1 Operating System Groups

Operating system groups are created and assigned specific names as part of the database installation process.

The default names of the operating system groups vary depending upon your operating system, and are listed in the following table:

Operating System Group	UNIX or Linux User Group	Windows User Group
OSDBA	`dba`	`ORA_DBA` (for all Oracle homes) `ORA_`*`HOMENAME`*`_DBA` (for each specific Oracle home)

Operating System Group	UNIX or Linux User Group	Windows User Group
OSOPER	`oper`	`ORA_OPER` (for all Oracle homes) `ORA_`*`HOMENAME`*`_OPER` (for each specific Oracle home)
OSBACKUPDBA	`backupdba`	`ORA_`*`HOMENAME`*`_SYSBACKUP`
OSDGDBA	`dgdba`	`ORA_`*`HOMENAME`*`_SYSDG`
OSKMDBA	`kmdba`	`ORA_`*`HOMENAME`*`_SYSKM`
OSRACDBA	`racdba`	`ORA_`*`HOMENAME`*`_SYSRAC`

For the Windows user group names, replace *HOMENAME* with the Oracle home name.

Oracle Universal Installer uses these default names, but, on UNIX or Linux, you can override them. On UNIX or Linux, one reason to override them is if you have multiple instances running on the same host computer in different Oracle homes. If each instance has a different person as the principal DBA, then you can improve the security of each instance by creating different groups for each instance.

For example, for two instances on the same UNIX or Linux host in different Oracle homes, the OSDBA group for the first instance might be named `dba1`, and OSDBA for the second instance might be named `dba2`. The first DBA would be a member of `dba1` only, and the second DBA would be a member of `dba2` only. Thus, when using operating system authentication, each DBA would be able to connect only to his assigned instance.

On Windows, default user group names cannot be changed. The *HOMENAME* placeholder enables you to have different user group names when you have multiple instances running on the same host Windows computer.

Membership in a group affects your connection to the database in the following ways:

- If you are a member of the OSDBA group, and you specify `AS SYSDBA` when you connect to the database, then you connect to the database with the `SYSDBA` administrative privilege.
- If you are a member of the OSOPER group, and you specify `AS SYSOPER` when you connect to the database, then you connect to the database with the `SYSOPER` administrative privilege.
- If you are a member of the OSBACKUPDBA group, and you specify `AS SYSBACKUP` when you connect to the database, then you connect to the database with the `SYSBACKUP` administrative privilege.
- If you are a member of the OSDGDBA group, and you specify `AS SYSDG` when you connect to the database, then you connect to the database with the `SYSDG` administrative privilege.
- If you are a member of the OSKMDBA group, and you specify `AS SYSKM` when you connect to the database, then you connect to the database with the `SYSKM` administrative privilege.
- If you are a member of the OSRACDBA group, and you specify `AS SYSRAC` when you connect to the database, then you connect to the database with the `SYSRAC` administrative privilege.
- If you are not a member of one of these operating system groups, and you attempt to connect as `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, `SYSKM`, or `SYSRAC`, then the `CONNECT` command fails.

See Also:

Your operating system specific Oracle documentation for information about creating the OSDBA and OSOPER groups

1.7.4.2 Preparing to Use Operating System Authentication

DBAs can authenticate to the database through the operating system rather than with a database user name and password.

To enable operating system authentication of an administrative user:

1. Create an operating system account for the user.
2. Add the account to the appropriate operating-system defined groups.

1.7.4.3 Connecting Using Operating System Authentication

A user can connect to the database using operating system authentication.

You can use operating system authentication by performing one of the following actions.

- A user can be authenticated, enabled as an administrative user, and connected to a local database by typing one of the following SQL*Plus commands:

  ```
  CONNECT / AS SYSDBA
  CONNECT / AS SYSOPER
  CONNECT / AS SYSBACKUP
  CONNECT / AS SYSDG
  CONNECT / AS SYSKM
  ```

- For the Windows platform only, remote operating system authentication over a secure connection is supported. You must specify the net service name for the remote database:

  ```
  CONNECT /@net_service_name AS SYSDBA
  CONNECT /@net_service_name AS SYSOPER
  CONNECT /@net_service_name AS SYSBACKUP
  CONNECT /@net_service_name AS SYSDG
  CONNECT /@net_service_name AS SYSKM
  ```

  Both the client computer and database host computer must be on a Windows domain.

Note:

The `SYSRAC` administrative privilege is used only by the Oracle agent of Oracle Clusterware to connect to the database using operating system authentication.

See Also:

- "Connecting to the Database with SQL*Plus"
- *SQL*Plus User's Guide and Reference* for the syntax of the `CONNECT` command

1.7.5 Using Password File Authentication

You can use password file authentication for an Oracle database instance and for an Oracle Automatic Storage Management (Oracle ASM) instance. The password file for an Oracle database is called a database password file, and the password file for Oracle ASM is called an Oracle ASM password file.

- Preparing to Use Password File Authentication
  To prepare for password file authentication, you must create the password file, set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter, and grant privileges.
- Connecting Using Password File Authentication
  Using password file authentication, administrative users can be connected and authenticated to a local or remote database by using the SQL*Plus `CONNECT` command. By default, passwords are case-sensitive.

See Also:

Oracle Automatic Storage Management Administrator's Guide for information about creating an Oracle ASM password file.

1.7.5.1 Preparing to Use Password File Authentication

To prepare for password file authentication, you must create the password file, set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter, and grant privileges.

To enable authentication of an administrative user using password file authentication, you must do the following:

1. If it is not already created, then create the password file using the `ORAPWD` utility:

   ```
   orapwd FILE=filename FORMAT=12.2
   ```

   See "Creating and Maintaining a Database Password File" for details.

Note:

- When you invoke the Database Configuration Assistant (DBCA) as part of the Oracle Database installation process, DBCA creates a password file.
- The administrative privileges `SYSBACKUP`, `SYSDG`, and `SYSKM` are not supported in the password file when the file is created with the `FORMAT=LEGACY` argument.
- `12.2` is the default for the `FORMAT` command-line argument.
- The administrative privilege `SYSRAC` is not supported in the password file.
- The administrative privileges can be granted to external users only when the file is created with the `FORMAT=12.2` argument. `FORMAT=12.2` also enables SSL and Kerberos authentication for administrative users.
- When you create a database password file that is stored in an Oracle ASM disk group, it can be shared among the multiple Oracle RAC database instances. The password file is not duplicated on each Oracle RAC database instance.

2. Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `exclusive`. (This is the default).

   Note:

   `REMOTE_LOGIN_PASSWORDFILE` is a static initialization parameter and therefore cannot be changed without restarting the database.

3. Connect to the database as user `SYS` (or as another user with the administrative privileges).
4. If the user does not already exist in the database, then create the user and assign a password.

   Keep in mind that database passwords are case-sensitive. See *Oracle Database Security Guide* for more information about case-sensitive database passwords.
5. Grant the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege to the user. For example:

   ```
   GRANT SYSDBA to mydba;
   ```

   This statement adds the user to the password file, thereby enabling connection `AS SYSDBA`, `AS SYSOPER`, `AS SYSBACKUP`, `AS SYSDG`, or `AS SYSKM`.

   See Also:

   "Creating and Maintaining a Database Password File" for instructions for creating and maintaining a password file

1.7.5.2 Connecting Using Password File Authentication

Using password file authentication, administrative users can be connected and authenticated to a local or remote database by using the SQL*Plus `CONNECT` command. By default, passwords are case-sensitive.

To connect using password file authentication:

- In SQL*Plus, execute the `CONNECT` command with a valid username and password and the `AS SYSDBA`, `AS SYSOPER`, `AS SYSBACKUP`, `AS SYSDG`, or `AS SYSKM` clause.

For example, if user `mydba` has been granted the `SYSDBA` privilege, then `mydba` can connect as follows:

```
CONNECT mydba AS SYSDBA
```

However, if user `mydba` has not been granted the `SYSOPER` privilege, then the following command fails:

```
CONNECT mydba AS SYSOPER
```

Note:

Operating system authentication takes precedence over password file authentication. Specifically, if you are a member of the appropriate operating system group, such as OSDBA or OSOPER, and you connect with the appropriate clause (for example, `AS SYSDBA`), then you will be connected with associated administrative privileges regardless of the *username/password* that you specify.

If you are not in the one of the operating system groups, and you are not in the password file, then attempting to connect with the clause fails.

See Also:

- "About Connecting to the Database with SQL*Plus"
- "Creating a Database Password File with ORAPWD"
- *SQL*Plus User's Guide and Reference* for syntax of the `CONNECT` command
- *Oracle Database Security Guide*

1.8 Creating and Maintaining a Database Password File

You can create a database password file using the password file creation utility, `ORAPWD`. For some operating systems, you can create this file as part of your standard installation.

- ORAPWD Syntax and Command Line Argument Descriptions
  The `ORAPWD` command creates and maintains a password file.

- Creating a Database Password File with ORAPWD
  You can create a database password file with ORAPWD.
- Sharing and Disabling the Database Password File
  You use the initialization parameter REMOTE_LOGIN_PASSWORDFILE to control whether a database password file is shared among multiple Oracle Database instances. You can also use this parameter to disable password file authentication.
- Keeping Administrator Passwords Synchronized with the Data Dictionary
  If you change the REMOTE_LOGIN_PASSWORDFILE initialization parameter from none to exclusive or shared, then you must ensure that the passwords stored in the data dictionary and the passwords stored in the password file for the non-SYS administrative users, such as SYSDBA, SYSOPER, SYSBACKUP, SYSDG, and SYSKM users are the same.
- Adding Users to a Database Password File
  When you grant SYSDBA, SYSOPER, SYSBACKUP, SYSDG, or SYSKM administrative privilege to a user, that user's name and privilege information are added to the database password file.
- Granting and Revoking Administrative Privileges
  Use the GRANT statement to grant administrative privileges. Use the REVOKE statement to revoke administrative privileges.
- Viewing Database Password File Members
  The V$PWFILE_USERS view contains information about users that have been granted administrative privileges.
- Removing a Database Password File
  You can remove a database password file if it is no longer needed.

See Also:

- "Using Password File Authentication"
- "Authentication Methods for Database Administrators"
- *Oracle Automatic Storage Management Administrator's Guide* for information about creating and maintaining an Oracle ASM password file

1.8.1 ORAPWD Syntax and Command Line Argument Descriptions

The ORAPWD command creates and maintains a password file.

The syntax of the ORAPWD command is as follows:

```
orapwd FILE=filename
[FORCE={y|n}]
[ASM={y|n}]
[DBUNIQUENAME=dbname]
[FORMAT={12.2|12}]
[SYS={y|n|password|external('sys-external-name')|global('sys-directory-DN')}]
[SYSBACKUP={y|n|password|external('sysbackup-external-name')|global('sysbackup-directory-
DN')}]
[SYSDG={y|n|password|external('sysdg-external-name')|global('sysdg-directory-DN')}]
[SYSKM={y|n|password|external('syskm-external-name')|global('syskm-directory-DN')}]
[DELETE={y|n}]
[INPUT_FILE=input-fname]

orapwd DESCRIBE FILE=filename
```

Command arguments are summarized in the following table.

Argument	Description
`FILE`	If the `DESCRIBE` argument is not included, then specify the name to assign to the new password file. You must supply a complete path. If you supply only a file name, the file is written to the current directory. If the `DESCRIBE` argument is included, then specify the name of an existing password file.
`FORCE`	(Optional) If `y`, permits overwriting an existing password file. It also clears CRS resources, if they already have the password file registered.
`ASM`	(Optional) If `y`, create an Oracle ASM password file in an Oracle ASM disk group. If `n`, the default, create a password file in the operating system file system. When the `DBUNIQUENAME` argument is specified, the password file is a database password file. When the `DBUNIQUENAME` argument is not specified, the password file can be a database password file or an Oracle ASM password file.
`DBUNIQUENAME`	Unique database name used to identify database password files residing in an ASM disk group only. This argument is required when the database password file is stored on an Oracle ASM disk group. This argument is ignored when an Oracle ASM password file is created by setting the `ASM` argument to `y`.
`FORMAT`	(Optional) Specify one of the following values: • `12.2`, the default, creates the password file in 12.2. format. This format supports granting administrative privileges to external users and enables SSL and Kerberos authentication for administrative users. • `12` creates the password file in Oracle Database 12*c* format. This format supports the `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges.
`SYS`	(Optional) This argument specifies if `SYS` user is password, externally, or globally authenticated. This argument can be set to `y`, `n`, *`password`*, `external('`*`sys-external-name`*`')`, or `global(`*`sys-directory-DN`*`)`. If `SYS=y` and `INPUT_FILE` is specified to migrate password file entries, then you will be prompted to enter the new password for the `SYS` administrative user. If *`password`*, then you will be prompted to enter the password for the `SYS` administrative user. If `external('`*`sys-external-name`*`')`, then replace *`sys-external-name`* with the external name for SSL or Kerberos authentication for the `SYS` administrative user. If `global(`*`sys-directory-DN`*`)`, then specify the directory service name for the global `SYS` user.
`SYSBACKUP`	(Optional) Creates `SYSBACKUP` entry. This argument specifies if `SYSBACKUP` user is password, externally, or globally authenticated. This argument can be set to `y`, `n`, *`password`*, `external('`*`sysbackup-external-name`*`')`, or `global(`*`sysbackup-directory-DN`*`)`. If *`password`*, then you will be prompted to enter the password for the `SYSBACKUP` administrative user. If `external('`*`sysbackup-external-name`*`')`, then replace *`sysbackup-external-name`* with the external name for SSL or Kerberos authentication for the `SYSBACKUP` administrative user. If `global(`*`sysbackup-directory-DN`*`)`, then specify the directory service name for the global `SYSBACKUP` user.

Argument	Description
`SYSDG`	(Optional) Creates `SYSDG` entry. This argument specifies if `SYSDG` user is password, externally, or globally authenticated. This argument can be set to `y`, `n`, *`password`*, `external('`*`sysdg-external-name`*`')`, or `global(`*`sysdg-directory-DN`*`)`. If *`password`*, then you will be prompted to enter the password for the `SYSDG` administrative user. If `external('`*`sysdg-external-name`*`')`, then replace *`sysdg-external-name`* with the external name for SSL or Kerberos authentication for the `SYSDG` administrative user. If `global(`*`sysdg-directory-DN`*`)`, then specify the directory service name for the global `SYSDG` user.
`SYSKM`	(Optional) Creates `SYSKM` entry. This argument specifies if `SYSKM` user is password, externally, or globally authenticated. (Optional) This argument can be set to `y`, `n`, *`password`*, `external('`*`syskm-external-name`*`')`, or `global(`*`syskm-directory-DN`*`)`. If *`password`*, then you will be prompted to enter the password for the `SYSKM` administrative user. If `external('`*`syskm-external-name`*`')`, then replace *`syskm-external-name`* with the external name for SSL or Kerberos authentication for the `SYSKM` administrative user. If `y`, creates a `SYSKM` entry in the password file. You are prompted for the password. The password is stored in the created password file. If `n`, no `SYSKM` entry is created in the password file. **Note:** The `y` and `n` values in the `SYSKM` argument are deprecated in Oracle Database 12*c* Release 2 (12.2) and may be desupported in a future release. If `global(`*`syskm-directory-DN`*`)`, then specify the directory service name for the global `SYSKM` user.
`DELETE`	(Optional) If `y`, delete the specified password file. If `n`, the default, create the specified password file.
`INPUT_FILE`	(Optional) Name of the input password file. `ORAPWD` migrates the entries in the input file to a new password file. This argument can be used to convert a password file from one format to another, for example from 12 format to 12.2 format. This argument also can be used to reset the password for the `SYS` administrative user. `ORAPWD` cannot migrate an input password that is stored in an Oracle ASM disk group.
`DESCRIBE`	Describes the properties of the specified password file, including the `FORMAT` value (`12.2` or `12`).

There are no spaces permitted around the equal-to (=) character.

Note:

Each external name must be unique.

The following sections provide more information about some of the `ORAPWD` command line arguments.

FILE

This argument sets the name of the password file being created. This argument is mandatory.
If you specify a location on an Oracle ASM disk group, then the database password file is shared automatically among the nodes in the cluster. When you use an Oracle ASM disk group to store the password file, and you are not using Oracle Managed Files, you must specify the name of the password file, including its full path. The full path is not required if you are using Oracle Managed Files.
If you do not specify a location on an Oracle ASM disk group, then the file name required for the password file is operating system specific. Some operating systems require the password file to adhere to a specific format and be located in a specific directory. Other operating systems allow the use of environment variables to specify the name and location of the password file.
The following table lists the required name and location for the password file on the UNIX, Linux, and Windows platforms. For other platforms, consult your platform-specific documentation.

Platform	Required Name	Required Location
UNIX and Linux	`orapw`*`ORACLE_SID`*	`ORACLE_BASE/dbs`
Windows	`PWD`*`ORACLE_SID`*`.ora`	`ORACLE_BASE\database`

For example, for a database instance with the SID `orcldw`, the password file must be named `orapworcldw` on Linux and `PWDorcldw.ora` on Windows.
In an Oracle Real Application Clusters (Oracle RAC) environment on a platform that requires an environment variable to be set to the path of the password file, the environment variable for each instance must point to the same password file.
For a policy-managed Oracle RAC database or an Oracle RAC One Node database with `ORACLE_SID` of the form *`db_unique_name_n`*, where *`n`* is a number, the password file is searched for first using `ORACLE_BASE/dbs/orapw`*`sid_prefix`* or `ORACLE_BASE\database\PWD`*`sid_prefix`*`.ora`. The *`sid_prefix`* (the first 8 characters of the database name) is used to locate the password file.

Note:

- It is critically important to the security of your system that you protect your password file and the environment variables that identify the location of the password file. Any user with access to these could potentially compromise the security of the connection.
- For Oracle Database 18c and later, if the password file is not found in its default directory, then the database checks for the password file in the directory that was the default directory in the earlier database releases. In the Oracle Database releases earlier to 18c, the default directory of the password file on UNIX and Linux platforms was `ORACLE_HOME/dbs` and on Windows was `ORACLE_HOME\database`.

See Also:

Using Oracle Managed Files

FORCE
This argument, if set to `y`, enables you to overwrite an existing password file. An error is returned if a password file of the same name already exists and this argument is omitted or set to `n`.

ASM
If this argument is set to `y`, then `ORAPWD` creates an Oracle ASM password file. The `FILE` argument must specify a location in the Oracle ASM disk group.
If this argument is set to `n`, the default, then `ORAPWD` creates a password file. The `FILE` argument can specify a location in the Oracle ASM disk group or in the operating system file system. When the `DBUNIQUENAME` argument is specified, the password file is a database password file. When the `DBUNIQUENAME` argument is not specified, the password file can be a database password file or an Oracle ASM password file.

See Also:

Oracle Automatic Storage Management Administrator's Guide for information about creating and maintaining an Oracle ASM password file

DBUNIQUENAME
This argument sets the unique database name for a database password file being created on an Oracle ASM disk group. It identifies which database resource to update with the database password file location.
This argument is not required when a database password file is created on an operating system file system.
This argument is ignored when an Oracle ASM password file is created by setting the `ASM` argument to `y`.

FORMAT
If this argument is set to 12.2, the default, then `ORAPWD` creates a database password file in 12.2 format. 12.2 format is required for the password file to support granting administrative privileges to external users and SSL and Kerberos authentication for administrative users. Password profiles assigned to the users are also enforced on the administrative users.
If this argument is set to `12`, then `ORAPWD` creates a database password file in Oracle Database 12*c* format. Oracle Database 12*c* format is required for the password file to support `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges.
If this argument is set to `legacy`, then `ORAPWD` creates a database password file that is in the format before Oracle Database 12*c*. The password file supports `SYSDBA` and `SYSOPER` administrative privileges, but it does not support `SYSBACKUP`, `SYSDG`, and `SYSKM` administrative privileges.

SYS
If `SYS=Y` and `INPUT_FILE` is specified to migrate password file entries, then you will be prompted to enter the new password for the `SYS` administrative user.
If *`password`*, then you will be prompted to enter the password for the `SYS` administrative user.
If `external('`*`sys-external-name`*`')`, then replace *`sys-external-name`* with the external name for SSL or Kerberos authentication for the `SYS` administrative user.

If `global(`*`sys-directory-DN`*`)`, then specify the directory service name for the global `SYS` user.

SYSBACKUP
If *`password`*, then you will be prompted to enter the password for the `SYSBACKUP` administrative user.
If `external('`*`sysbackup-external-name`*`')`, then replace *`sysbackup-external-name`* with the external name for SSL or Kerberos authentication for the `SYSDG` administrative user.
If `global(`*`sysbackup-directory-DN`*`)`, then specify the directory service name for the global `SYSBACKUP` user.

SYSDG
If *`password`*, then you will be prompted to enter the password for the `SYSDG` administrative user.
If `external('`*`sysdg-external-name`*`')`, then replace *`sysdg-external-name`* with the external name for SSL or Kerberos authentication for the `SYSDG` administrative user.
If `global(`*`sysdg-directory-DN`*`)`, then specify the directory service name for the global `SYSDG` user.

SYSKM
If *`password`*, then you will be prompted to enter the password for the `SYSKM` administrative user.
If `external('`*`syskm-external-name`*`')`, then replace *`syskm-external-name`* with the external name for SSL or Kerberos authentication for the `SYSKM` administrative user.
If `global(`*`syskm-directory-DN`*`)`, then specify the directory service name for the global `SYSKM` user.

DELETE
If this argument is set to `y`, then `ORAPWD` deletes the specified password file. When `y` is specified, `FILE`, `ASM`, or `DBUNIQUENAME` must be specified. When `FILE` is specified, the file must be located on an ASM disk group.
If this argument is set to `n`, the default, then `ORAPWD` creates the password file.

INPUT_FILE
This argument specifies the name of the input password file. `ORAPWD` migrates the entries in the input file to a new password file. This argument can convert a password file from one format to another, for example from 12 format to 12.2 format.
This argument also can be used to reset the password for the `SYS` administrative user.
When the `INPUT_FILE` argument is specified, `ORAPWD` does not create any new entries. Therefore, `ORAPWD` ignores the following arguments:

- `PASSWORD`
- `SYSBACKUP`
- `SYSDG`
- `SYSKM`

When an input file is specified and the new password file replaces the input file, `FORCE` must be set to `y`.

Note:

When the `FORMAT` argument is not specified, by default the new password file is created in 12.2 format from the input file.

See Also:

"Administrative Privileges" and "Adding Users to a Database Password File"

1.8.2 Creating a Database Password File with ORAPWD

You can create a database password file with `ORAPWD`.

Passwords are case-sensitive. However, password files created using an earlier Oracle Database release retain their case-insensitive passwords, if the `ignorecase` option was omitted during password file creation. Oracle recommends that you force case sensitivity in these older password files by migrating the password file from one format to another.

To create a database password file:

- Run the `ORAPWD` command.

Example 1-20 Creating a Database Password File Located in an Oracle ASM Disk Group

The following command creates a database password file in 12.2 format named `orapworcl` that is located in an Oracle ASM disk group. The `DBUNIQUENAME` argument is required because the database password file is located in an Oracle ASM disk group.

```
orapwd FILE='+DATA/orcl/orapworcl' DBUNIQUENAME='orcl' FORMAT=12.2
```

Example 1-21 Creating a Database Password File with a SYSBACKUP Entry

The following example is the similar to Example 1-20 except that it creates a `SYSBACKUP` entry in the database password file. The password file is in 12.2 format by default.

```
orapwd FILE='+DATA/orcl/orapworcl' DBUNIQUENAME='orcl' SYSBACKUP=password FORMAT=12.2
```

Example 1-22 Creating a Database Password File with External Authentication for SYS and SYSKM

The following example is the similar to Example 1-20 except that it specifies an external name for the `SYS` and `SYSKM` administrative users.

```
orapwd FILE='+DATA/orcl/orapworcl' DBUNIQUENAME='orcl' FORMAT=12.2
sys=external('KerberosUserSYS@example.com')
syskm=external('KerberosUserSYSKM@example.com')
```

Example 1-23 Creating a Database Password File Located in a File System

The following command creates a database password file in 12.2 format named `orapworcl` that is located in the default location in an operating system file system.

```
orapwd FILE='/u01/oracle/dbs/orapworcl' FORMAT=12.2
```

Example 1-24 Migrating a Database Password File to Oracle Database 12c Format

The following command migrates a database password file to the 12.2 format. The new password file is case-sensitive and will contain case-sensitive passwords. The password file is named `orapworcl`, and it is located in an operating system file system. The new database password file replaces the existing database password file. Therefore, `FORCE` must be set to `y`.

```
orapwd FILE='/u01/oracle/dbs/orapworcl' FORMAT=12.2 INPUT_FILE='/u01/oracle/dbs/
orapworcl' FORCE=y
```

Example 1-25 Resetting the Password for the SYS Administrative User

The following command resets the password for the `SYS` administrative user. The new database password file replaces the existing database password file. Therefore, `FORCE` must be set to `y`.

```
orapwd FILE='/u01/oracle/dbs/orapworcl' SYS=Y INPUT_FILE='/u01/oracle/dbs/orapworcl'
FORCE=y
```

You are prompted to enter the new password for the `SYS` administrative user.

Example 1-26 Describing a Password File

The following command describes the `orapworcl` password file.

```
orapwd DESCRIBE FILE='orapworcl'
Password file Description : format=12.2
```

Note:

If the database password file name or location is changed, then run the following command for the changes to take effect:

```
SQL> ALTER SYSTEM FLUSH PASSWORDFILE_METADATA_CACHE;
```

This command flushes the metadata cache and the subsequent logins to the database use the new password file. In an Oracle RAC environment, this command clears cache in all the Oracle RAC databases, but there could be some databases that may still continue using the old password file till the change is propagated across all the Oracle RAC databases.

After running this command, you can verify the changes by querying the `V$PASSWORDFILE_INFO` view.

See Also:

Oracle Automatic Storage Management Administrator's Guide for information about managing a shared password file in an Oracle ASM disk group

1.8.3 Sharing and Disabling the Database Password File

You use the initialization parameter `REMOTE_LOGIN_PASSWORDFILE` to control whether a database password file is shared among multiple Oracle Database instances. You can also use this parameter to disable password file authentication.

To share a password file or disable password file authentication:

- Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter.

You can set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to one of the following values:

- `none`: Setting this parameter to `none` causes Oracle Database to behave as if the password file does not exist. That is, no privileged connections are allowed over nonsecure connections.
- `exclusive`: (The default) An `exclusive` password file can be used with only one database. Only an `exclusive` file can be modified. Using an `exclusive` password file enables you to add, modify, and delete users. It also enables you to change the password for `SYS`, `SYSBACKUP`, `SYSDG`, or `SYSKM` with the `ALTER USER` command.

  When an `exclusive` password file is stored on an Oracle ASM disk group, it can be used by a single-instance database or multiple instances of an Oracle Real Application Clusters (Oracle RAC) database.

  When an `exclusive` password file is stored on an operating system, it can be used with only one instance of one database.
- `shared`: A `shared` password file can be used by multiple databases running on the same server, or multiple instances of an Oracle RAC database, even when it is stored on an operating system. A `shared` password file is read-only and cannot be modified. Therefore, you cannot add users to a `shared` password file. Any attempt to do so or to change the password of `SYS` or other users with the administrative privileges generates an error. All users needing administrative privileges must be added to the password file when `REMOTE_LOGIN_PASSWORDFILE` is set to `exclusive`. After all users are added, you can change `REMOTE_LOGIN_PASSWORDFILE` to `shared`, and then share the file.

  This option is useful if you are administering multiple databases with a single password file.

  You cannot specify `shared` for an Oracle ASM password file.

If `REMOTE_LOGIN_PASSWORDFILE` is set to `exclusive` or `shared` and the password file is missing, this is equivalent to setting `REMOTE_LOGIN_PASSWORDFILE` to `none`.

1.8.4 Keeping Administrator Passwords Synchronized with the Data Dictionary

If you change the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter from `none` to `exclusive` or `shared`, then you must ensure that the passwords stored in the data dictionary and the passwords stored in the password file for the non-SYS administrative users, such as `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, and `SYSKM` users are the same.

> **Note:**
>
> Starting with Oracle Database 12*c* Release 2 (12.2), authentication for the `SYS` user happens using only the password file and not using the data dictionary.

To synchronize the passwords for non-`SYS` administrative users, such as `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, and `SYSKM` users, you must first revoke and then regrant the privileges to these users as follows:

1. Find all users who have been granted the `SYSDBA` privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSDBA='TRUE';
   ```
2. Revoke and then re-grant the `SYSDBA` privilege to these users.

```
REVOKE SYSDBA FROM non-SYS-user;
GRANT SYSDBA TO non-SYS-user;
```

3. Find all users who have been granted the `SYSOPER` privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSOPER='TRUE';
   ```

4. Revoke and regrant the `SYSOPER` privilege to these users.

   ```
   REVOKE SYSOPER FROM non-SYS-user;
   GRANT SYSOPER TO non-SYS-user;
   ```

5. Find all users who have been granted the `SYSBACKUP` privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSBACKUP ='TRUE';
   ```

6. Revoke and regrant the `SYSBACKUP` privilege to these users.

   ```
   REVOKE SYSBACKUP FROM non-SYS-user;
   GRANT SYSBACKUP TO non-SYS-user;
   ```

7. Find all users who have been granted the `SYSDG` privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSDG='TRUE';
   ```

8. Revoke and regrant the `SYSDG` privilege to these users.

   ```
   REVOKE SYSDG FROM non-SYS-user;
   GRANT SYSDG TO non-SYS-user;
   ```

9. Find all users who have been granted the `SYSKM` privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSKM='TRUE';
   ```

10. Revoke and regrant the `SYSKM` privilege to these users.

   ```
   REVOKE SYSKM FROM non-SYS-user;
   GRANT SYSKM TO non-SYS-user;
   ```

1.8.5 Adding Users to a Database Password File

When you grant `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege to a user, that user's name and privilege information are added to the database password file.

A user's name remains in the password file only as long as that user has at least one of these privileges. If you revoke all of these privileges, then Oracle Database removes the user from the password file.

Note:

The password file must be created with the `FORMAT=12.2` or `FORMAT=12` argument to support `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege.

Creating a Password File and Adding New Users to It

Use the following procedure to create a password file and add new users to it:

1. Follow the instructions for creating a password file as explained in "Creating a Database Password File with ORAPWD".
2. Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `exclusive`. (This is the default.)

Oracle Database issues an error if you attempt to grant these privileges and the initialization parameter `REMOTE_LOGIN_PASSWORDFILE` is not set correctly.

> **Note:**
>
> `REMOTE_LOGIN_PASSWORDFILE` is a static initialization parameter and therefore cannot be changed without restarting the database.

3. Connect with `SYSDBA` privileges as shown in the following example, and enter the `SYS` password when prompted:

   ```
   CONNECT SYS AS SYSDBA
   ```

4. Start up the instance and create the database if necessary, or mount and open an existing database.
5. Create users as necessary. Grant `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege to yourself and other users as appropriate. See "Granting and Revoking Administrative Privileges".

1.8.6 Granting and Revoking Administrative Privileges

Use the `GRANT` statement to grant administrative privileges. Use the `REVOKE` statement to revoke administrative privileges.

To grant the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` administrative privilege to a user:

- Run the `GRANT` statement.

For example:

```
GRANT SYSDBA TO mydba;
```

To revoke the administrative privilege from a user:

- Run the `REVOKE` statement.

For example:

```
REVOKE SYSDBA FROM mydba;
```

The `WITH ADMIN OPTION` is ignored if it is specified in the `GRANT` statement that grants an administrative privilege, and the following rules apply:

- A user currently connected as `SYSDBA` can grant any administrative privilege to another user and revoke any administrative privilege from another user.
- A user currently connected as `SYSOPER` *cannot* grant any administrative privilege to another user and *cannot* revoke any administrative privilege from another user.
- A user currently connected as `SYSBACKUP` can grant or revoke another user's `SYSBACKUP` administrative privilege.
- A user currently connected as `SYSDG` can grant or revoke another user's `SYSDG` administrative privilege.
- A user currently connected as `SYSKM` can grant or revoke another user's `SYSKM` administrative privilege.

Administrative privileges cannot be granted to roles, because roles are available only after database startup. Do not confuse the database administrative privileges with operating system roles.

See Also:

Oracle Database Security Guide for more information on administrative privileges

1.8.7 Viewing Database Password File Members

The `V$PWFILE_USERS` view contains information about users that have been granted administrative privileges.

To determine which users have been granted administrative privileges:

- Query the `V$PWFILE_USERS` view.

See Also:

Oracle Database Reference for information about the `V$PWFILE_USERS` view

1.8.8 Removing a Database Password File

You can remove a database password file if it is no longer needed.

If you determine that you no longer require a database password file to authenticate users, then to remove it:

- Delete the database password file, and optionally reset the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `none`.

After you remove this file, only those users who can be authenticated by the operating system can perform `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, or `SYSKM` database administration operations.

1.9 Data Utilities

Oracle utilities are available to help you maintain the data in your Oracle Database.

SQL*Loader

SQL*Loader is used both by database administrators and by other users of Oracle Database. It loads data from standard operating system files (such as, files in text or C data format) into database tables.

Export and Import Utilities

The Data Pump utility enables you to archive data and to move data between one Oracle Database and another. Also available are the original Import (IMP) and Export (EXP) utilities for importing and exporting data from and to earlier releases.

See Also:

- *Oracle Database Utilities* for detailed information about SQL*Loader
- *Oracle Database Utilities* for detailed information about Data Pump

2

Configuring Automatic Restart of an Oracle Database

Configure your Oracle database with the Oracle Restart feature to automatically restart the database, the listener, and other Oracle components after a hardware or software failure or whenever your database host computer restarts.

- About Oracle Restart
  Oracle Restart enhances the availability of Oracle databases in a single-instance environment.
- Configuring Oracle Restart
  To configure Oracle Restart, you can add components, remove components, or modify options for components.
- Starting and Stopping Components Managed by Oracle Restart
  When Oracle Restart is in use, Oracle strongly recommends that you use the SRVCTL utility to start and stop components.
- Stopping and Restarting Oracle Restart for Maintenance Operations
  When several components in an Oracle home are managed by Oracle Restart, you can stop Oracle Restart and the components managed by Oracle Restart in the Oracle home.
- SRVCTL Command Reference for Oracle Restart
  You can reference details about the syntax and options for SRVCTL commands specific to Oracle Restart.
- CRSCTL Command Reference
  You can reference details about the syntax for the CRSCTL commands that are relevant for Oracle Restart.

2.1 About Oracle Restart

Oracle Restart enhances the availability of Oracle databases in a single-instance environment.

- Oracle Restart Overview
  When you install Oracle Restart, various Oracle components can be automatically restarted after a hardware or software failure or whenever your database host computer restarts.
- About Startup Dependencies
  Oracle Restart ensures that Oracle components are started in the proper order, in accordance with component dependencies.
- About Starting and Stopping Components with Oracle Restart
  Oracle Restart automatically restarts various Oracle components when required, and automatically stops Oracle components in an orderly fashion when you manually shut down your system.
- About Starting and Stopping Oracle Restart
  The CRSCTL utility starts and stops Oracle Restart.

- Oracle Restart Configuration
  Oracle Restart maintains a list of all the Oracle components that it manages, and maintains configuration information for each component.
- Oracle Restart Integration with Oracle Data Guard
  Oracle Restart is integrated with Oracle Data Guard (Data Guard) and the Oracle Data Guard Broker (the broker).
- Fast Application Notification with Oracle Restart
  Oracle Restart uses Oracle Notification Services (ONS) and Oracle Advanced Queues to publish Fast Application Notification (FAN) high availability events. Integrated Oracle clients use FAN to provide fast notification to clients when the service or instance goes down. The client can automate the failover of database connections between a primary database and a standby database.

2.1.1 Oracle Restart Overview

When you install Oracle Restart, various Oracle components can be automatically restarted after a hardware or software failure or whenever your database host computer restarts.

Table 2-1 lists these components.

Table 2-1 Oracle Components Automatically Restarted by Oracle Restart

Component	Notes
Database instance	Oracle Restart can accommodate multiple databases on a single host computer.
Oracle Net listener	-
Database services	Does not include the default service created upon installation because it is automatically managed by Oracle Database. Also does not include any default services created during database creation or global services. For more information about global services, see the *Oracle Database Global Data Services Concepts and Administration Guide*.
Oracle Automatic Storage Management (Oracle ASM) instance	-
Oracle ASM disk groups	Restarting a disk group means mounting it.
Oracle Notification Services (ONS)	In an Oracle Grid Infrastructure for Standalone Servers (Oracle Restart) environment, ONS can be used in Oracle Data Guard installations for automating failover of connections between primary and standby database through Fast Application Notification (FAN). ONS is a service for sending FAN events to integrated clients upon failover.

Oracle Restart runs periodic check operations to monitor the health of these components. If a check operation fails for a component, the component is shut down and restarted.

Oracle Restart is used in standalone server (non-clustered) environments only. For Oracle Real Application Clusters (Oracle RAC) environments, the functionality to automatically restart components is provided by Oracle Clusterware.

Oracle Restart runs out of the Oracle Grid Infrastructure home, which you install separately from Oracle Database homes. See the *Oracle Grid Infrastructure Installation Guide* for your platform for information about installing the Oracle Grid Infrastructure home.

See Also:

- "Configuring Oracle Restart"
- *Oracle Automatic Storage Management Administrator's Guide* for information about Oracle Automatic Storage Management

2.1.2 About Startup Dependencies

Oracle Restart ensures that Oracle components are started in the proper order, in accordance with component dependencies.

For example, if database files are stored in Oracle ASM disk groups, then before starting the database instance, Oracle Restart ensures that the Oracle ASM instance is started and the required disk groups are mounted. Likewise, if a component must be shut down, Oracle Restart ensures that dependent components are cleanly shut down first.

Oracle Restart also manages the weak dependency between database instances and the Oracle Net listener (the listener): When a database instance is started, Oracle Restart attempts to start the listener. If the listener startup fails, then the database is still started. If the listener later fails, Oracle Restart does not shut down and restart any database instances.

2.1.3 About Starting and Stopping Components with Oracle Restart

Oracle Restart automatically restarts various Oracle components when required, and automatically stops Oracle components in an orderly fashion when you manually shut down your system.

There may be times, however, when you want to manually start or stop individual Oracle components. Oracle Restart includes the Server Control (SRVCTL) utility that you use to manually start and stop Oracle Restart–managed components. When Oracle Restart is in use, Oracle strongly recommends that you use SRVCTL to manually start and stop components.

After you stop a component with SRVCTL, Oracle Restart does not automatically restart that component if a failure occurs. If you then start the component with SRVCTL, that component is again available for automatic restart.

Oracle utilities such as SQL*Plus, the Listener Control utility (`LSNRCTL`), and `ASMCMD` are integrated with Oracle Restart. If you shut down the database with SQL*Plus, Oracle Restart does not interpret this as a database failure and does not attempt to restart the database. Similarly, if you shut down the Oracle ASM instance with SQL*Plus or `ASMCMD`, Oracle Restart does not attempt to restart it.

An important difference between starting a component with SRVCTL and starting it with SQL*Plus (or another utility) is the following:

- When you start a component with SRVCTL, any components on which this component depends are automatically started first, and in the proper order.
- When you start a component with SQL*Plus (or another utility), other components in the dependency chain are not automatically started; you must ensure that any components on which this component depends are started.

In addition, Oracle Restart enables you to start and stop all of the components managed by Oracle Restart in a specified Oracle home using a single command. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home. This capability is useful when you are installing a patch.

See Also:

"Starting and Stopping Components Managed by Oracle Restart"

2.1.4 About Starting and Stopping Oracle Restart

The CRSCTL utility starts and stops Oracle Restart.

You can also use the CRSCTL utility to enable or disable Oracle high availability services. Oracle Restart uses Oracle high availability services to start and stop automatically the components managed by Oracle Restart. For example, Oracle high availability services daemons automatically start databases, listeners, and Oracle ASM instances. When Oracle high availability services are disabled, none of the components managed by Oracle Restart are started when a node is rebooted.

Typically, you use the CRSCTL utility when you must stop all of the running Oracle software in an Oracle installation. For example, you might need to stop Oracle Restart when you are installing a patch or performing operating system maintenance. When the maintenance is complete, you use the CRSCTL utility to start Oracle Restart.

See Also:

"Stopping and Restarting Oracle Restart for Maintenance Operations" for information about using the CRSCTL utility

2.1.5 Oracle Restart Configuration

Oracle Restart maintains a list of all the Oracle components that it manages, and maintains configuration information for each component.

All of this information is collectively known as the **Oracle Restart configuration**. When Oracle Restart starts a component, it starts the component according to the configuration information for that component. For example, the Oracle Restart configuration includes the location of the server parameter file (SPFILE) for databases, and the TCP port to listen on for listeners.

If you install Oracle Restart and then create your database with Database Configuration Assistant (DBCA), DBCA automatically adds the database to the Oracle Restart configuration. When DBCA then starts the database, the required dependencies between the database and other components (for example disk groups in which the database stores data) are established, and Oracle Restart begins to manage the database.

You can manually add and remove components from the Oracle Restart configuration with SRVCTL commands. For example, if you install Oracle Restart onto a host on which a database is already running, you can use SRVCTL to add that database to the Oracle Restart configuration. When you manually add a component to the Oracle Restart configuration and then start it with SRVCTL, Oracle Restart begins to manage the component, restarting it when required.

Note:

Adding a component to the Oracle Restart configuration is also referred to as "registering a component with Oracle Restart."

Other SRVCTL commands enable you to view the status and configuration of Oracle Restart–managed components, temporarily disable and then reenable management for components, and more.

When Oracle Restart is installed, many operations that create Oracle components automatically add the components to the Oracle Restart configuration. Table 2-2 lists some create operations and whether the created component is automatically added.

Table 2-2 Create Operations and the Oracle Restart Configuration

Create Operation	Created Component Automatically Added to Oracle Restart Configuration?
Create a database with OUI or DBCA	Yes
Create a database with the `CREATE DATABASE` SQL statement	No
Create an Oracle ASM instance with OUI, DBCA, or ASMCA	Yes
Create a disk group (any method)	Yes
Add a listener with NETCA	Yes
Create a database service with SRVCTL	Yes
Create a database service by modifying the `SERVICE_NAMES` initialization parameter[1]	No
Create a database service with `DBMS_SERVICE.CREATE_SERVICE`	No
Create a standby database	No

1 Not recommended when Oracle Restart is in use

Table 2-3 lists some delete/drop/remove operations and whether the deleted component is also automatically removed from the Oracle Restart configuration.

Table 2-3 Delete/Drop/Remove Operations and the Oracle Restart Configuration

Operation	Deleted Component Automatically Removed from Oracle Restart Configuration?
Delete a database with DBCA	Yes
Delete a database by removing database files with operating system commands[1]	No
Delete a listener with NETCA	Yes
Drop an Oracle ASM disk group (any method)	Yes

Table 2-3 (Cont.) Delete/Drop/Remove Operations and the Oracle Restart Configuration

Operation	Deleted Component Automatically Removed from Oracle Restart Configuration?
Delete a database service with SRVCTL	Yes
Delete a database service by any other means	No

1 Not recommended

2.1.6 Oracle Restart Integration with Oracle Data Guard

Oracle Restart is integrated with Oracle Data Guard (Data Guard) and the Oracle Data Guard Broker (the broker).

When a database shutdown and restart is required in response to a role change request, Oracle Restart shuts down and restarts the database in an orderly fashion (taking dependencies into account), and according to the settings in the Oracle Restart configuration. Oracle Restart also ensures that, following a Data Guard role transition, all database services configured to run in the new database role are active and all services not configured to run in the new role are stopped.

In addition, the Oracle Restart configuration supports Data Guard–related configuration options for the following components:

- **Databases**—When you add a database to the Oracle Restart configuration, you can specify the current Data Guard role for the database: `PRIMARY`, `PHYSICAL_STANDBY`, `LOGICAL_STANDBY`, or `SNAPSHOT_STANDBY`. If the role is later changed using the broker, Oracle Restart automatically updates the database configuration with the new role. If you change the database role without using the broker, you must manually modify the database's role in the Oracle Restart configuration to reflect the new role.
- **Database Services**—When adding a database service to the Oracle Restart configuration, you can specify one or more Data Guard roles for the service. When this configuration option is present, upon database open Oracle Restart starts the service only if one of the service roles matches the current database role.

See Also:

- *Oracle Data Guard Concepts and Administration* for information about Oracle Data Guard
- "Fast Application Notification with Oracle Restart"
- "Automating the Failover of Connections Between Primary and Standby Databases"

2.1.7 Fast Application Notification with Oracle Restart

Oracle Restart uses Oracle Notification Services (ONS) and Oracle Advanced Queues to publish Fast Application Notification (FAN) high availability events. Integrated Oracle clients use FAN to provide fast notification to clients when the service or instance goes down. The client can automate the failover of database connections between a primary database and a standby database.

- Overview of Fast Application Notification
  FAN is a high availability notification mechanism that Oracle Restart can use to notify other processes about configuration changes that include service status changes, such as `UP` or `DOWN` events.
- Application High Availability with Services and FAN
  Oracle Database focuses on maintaining service availability. With Oracle Restart, Oracle services are designed to be continuously available. Oracle Restart monitors the database and its services and, when configured, sends event notifications using FAN.

See Also:

Oracle Database Advanced Queuing User's Guide

2.1.7.1 Overview of Fast Application Notification

FAN is a high availability notification mechanism that Oracle Restart can use to notify other processes about configuration changes that include service status changes, such as `UP` or `DOWN` events.

FAN provides the ability to immediately terminate inflight transaction when an instance or server fails. Integrated Oracle clients receive the events and respond. Applications can respond either by propagating the error to the user or by resubmitting the transactions and masking the error from the application user. When a `DOWN` event occurs, integrated clients immediately clean up connections to the terminated database. When an `UP` event occurs, the clients create new connections to the new primary database instance.

Oracle Restart publishes FAN events whenever a managed instance or service goes up or down. After a failover, the Oracle Data Guard Broker (broker) publishes FAN events. These FAN events can be used in the following ways:

- Applications can use FAN with Oracle Restart without programmatic changes if they use one of these Oracle integrated database clients: Oracle Database JDBC, Universal Connection Pool for Java, Oracle Call Interface, and Oracle Database ODP.NET. These clients can be configured for Fast Connection Failover (FCF) to automatically connect to a new primary database after a failover.
- FAN server-side callouts can be configured on the database tier.

For `DOWN` events, such as a failed primary database, FAN provides immediate notification to the clients so that they can failover as fast as possible to the new primary database. The clients do not wait for a timeout. The clients are notified immediately, and they must be configured to failover when they are notified.

For `UP` events, when services and instances are started, new connections can be created so that the application can immediately take advantage of the extra resources.

Through server-side callouts, you can also use FAN to:

- Log status information
- Page DBAs or open support tickets when resources fail to start

- Automatically start dependent external applications that must be co-located with a service

FAN events are published using ONS and Oracle Database Advanced Queuing queues. The queues are configured automatically when you configure a service. You must configure ONS manually using SRVCTL commands.

The Connection Manager (CMAN) and Oracle Net Services listeners are integrated with FAN events, enabling the CMAN and the listener to immediately de-register services provided by the failed instance and to avoid erroneously sending connection requests to a failed database.

See Also:

Oracle Data Guard Broker for information about FAN events in an Oracle Data Guard environment

2.1.7.2 Application High Availability with Services and FAN

Oracle Database focuses on maintaining service availability. With Oracle Restart, Oracle services are designed to be continuously available. Oracle Restart monitors the database and its services and, when configured, sends event notifications using FAN.

- Managing Unplanned Outages
  If Oracle Restart detects an outage, then it isolates the failed component and recovers the dependent components. If the failed component is the database instance, then after Oracle Data Guard fails over to the standby database, Oracle Restart on the new primary database starts any services defined with the current role.
- Managing Planned Outages
  For repairs, upgrades, and changes that require you to shut down the primary database, Oracle Restart provides interfaces that disable and enable services to minimize service disruption to application users.
- Fast Application Notification High Availability Events
  Understand FAN event record parameters and the event types.
- Using Fast Application Notification Callouts
  FAN callouts are server-side executables that Oracle Restart executes immediately when high availability events occur.
- Oracle Clients That Are Integrated with Fast Application Notification
  Oracle has integrated FAN with many of the common Oracle client drivers that are used to connect to Oracle Restart databases. Therefore, the easiest way to use FAN is to use an integrated Oracle Client.

2.1.7.2.1 Managing Unplanned Outages

If Oracle Restart detects an outage, then it isolates the failed component and recovers the dependent components. If the failed component is the database instance, then after Oracle Data Guard fails over to the standby database, Oracle Restart on the new primary database starts any services defined with the current role.

FAN events are published by Oracle Restart and the Oracle Data Guard Broker through ONS and Advanced Queuing. You can also perform notifications using FAN callouts.

Note:

Oracle Restart does not run callouts with guaranteed ordering. Callouts are run asynchronously, and they are subject to scheduling variability.

With Oracle Restart, restart and recovery are automatic, including the restarting of the subsystems, such as the listener and the Oracle Automatic Storage Management (Oracle ASM) processes, not just the database. You can use FAN callouts to report faults to your fault management system and to initiate repair jobs.

2.1.7.2.2 Managing Planned Outages

For repairs, upgrades, and changes that require you to shut down the primary database, Oracle Restart provides interfaces that disable and enable services to minimize service disruption to application users.

Using Oracle Data Guard Broker with Oracle Restart allows a coordinated failover of the database service from the primary to the standby for the duration of the planned outage. Once you complete the operation, you can return the service to normal operation.

The management policy for a service controls whether the service starts automatically when the database is restarted. If the management policy for a service is set to `AUTOMATIC`, then it restarts automatically. If the management policy for a service is set to `MANUAL`, then it must be started manually.

See Also:

"Modifying the Oracle Restart Configuration for a Component"

2.1.7.2.3 Fast Application Notification High Availability Events

Understand FAN event record parameters and the event types.

Table 2-4 describes the FAN event record parameters and the event types, followed by name-value pairs for the event properties. The event type is always the first entry and the timestamp is always the last entry. In the following example, the name in the name-value pair is shown in `Fan event type` (`service_member`), and the value in the name-value pair is shown in `Properties`:

```
FAN event type: service_member
Properties: version=1.0 service=ERP database=FINPROD instance=FINPROD host=node1
status=up
```

Table 2-4 Event Record Parameters and Descriptions

Parameter	Description
`VERSION`	Version of the event record. Used to identify release changes.
`EVENT TYPE`	`SERVICE`, `SERVICE_MEMBER`, `DATABASE`, `INSTANCE`, `NODE`, `ASM`, `SRV_PRECONNECT`. Note that database and Instance types provide the database service, such as `DB_UNIQUE_NAME.DB_DOMAIN`.

Table 2-4 (Cont.) Event Record Parameters and Descriptions

Parameter	Description
`DATABASE UNIQUE NAME`	The unique database supporting the service; matches the initialization parameter value for `DB_UNIQUE_NAME`, which defaults to the value of the initialization parameter `DB_NAME`.
`INSTANCE`	The name of the instance that supports the service; matches the `ORACLE_SID` value.
`NODE NAME`	The name of the node that supports the service or the node that has stopped; matches the node name known to Cluster Synchronization Services (CSS).
`SERVICE`	The service name; matches the service in `DBA_SERVICES`.
`STATUS`	Values are `UP`, `DOWN`, `NOT_RESTARTING`, `PRECONN_UP`, `PRECONN_DOWN`, and `UNKNOWN`.
`REASON`	`Data_Guard_Failover`, `Failure`, `Dependency`, `User`, `Autostart`, `Restart`.
`CARDINALITY`	The number of service members that are currently active; included in all `UP` events.
`TIMESTAMP`	The local time zone to use when ordering notification events.

A FAN record matches the database signature of each session as shown in Table 2-5.

Table 2-5 FAN Parameters and Matching Database Signatures

FAN Parameter	Matching Oracle Database Signature
`SERVICE`	`sys_context('userenv', 'service_name')`
`DATABASE UNIQUE NAME`	`sys_context('userenv', 'db_unique_name')`
`INSTANCE`	`sys_context('userenv', 'instance_name')`
`NODE NAME`	`sys_context('userenv', 'server_host')`

2.1.7.2.4 Using Fast Application Notification Callouts

FAN callouts are server-side executables that Oracle Restart executes immediately when high availability events occur.

You can use FAN callouts to automate the following activities when events occur, such as:

- Opening fault tracking tickets
- Sending messages to pagers
- Sending e-mail
- Starting and stopping server-side applications
- Maintaining an uptime log by logging each event as it occurs

To use FAN callouts:

- Place an executable in the directory grid_home/racg/usrco on both the primary and the standby database servers. If you are using scripts, then set the shell as the first line of the executable.

The following is an example file for the grid_home/racg/usrco/callout.sh callout:

```
#! /bin/ksh
FAN_LOGFILE= [your path name]/admin/log/`hostname`_uptime.log
echo $* "reported="`date` >> $FAN_LOGFILE &
```

The following output is from the previous example:

```
NODE VERSION=1.0 host=sun880-2 status=nodedown reason=
timestamp=08-Oct-2004 04:02:14 reported=Fri Oct 8 04:02:14 PDT 2004
```

A FAN record matches the database signature of each session, as shown in Table 2-5. Use this information to take actions on sessions that match the FAN event data.

See Also:

Table 2-4 for information about the callout and event details

2.1.7.2.5 Oracle Clients That Are Integrated with Fast Application Notification

Oracle has integrated FAN with many of the common Oracle client drivers that are used to connect to Oracle Restart databases. Therefore, the easiest way to use FAN is to use an integrated Oracle Client.

You can use the CMAN session pools, Oracle Call Interface, Universal Connection Pool for Java, JDBC simplefan API, and ODP.NET connection pools. The overall goal is to enable applications to consistently obtain connections to the available primary database at anytime.

See Also:

"Automating the Failover of Connections Between Primary and Standby Databases"

2.2 Configuring Oracle Restart

To configure Oracle Restart, you can add components, remove components, or modify options for components.

- About Configuring Oracle Restart
  If you install Oracle Restart by installing the Oracle Grid Infrastructure for Standalone Servers and then create your database, the database is automatically added to the Oracle Restart configuration, and is then automatically restarted when required. However, if you install Oracle Restart on a host computer on which a database already exists, you must manually add the database, the listener, the Oracle Automatic Storage Management (Oracle ASM) instance, and possibly other components to the Oracle Restart configuration.
- Preparing to Run SRVCTL
  Many Oracle Restart tasks require that you run the SRVCTL utility. You must ensure that you run SRVCTL from the correct Oracle home, and that you log in to the host computer with the correct user account.
- Obtaining Help for SRVCTL
  Online help is available for the SRVCTL utility.

- Adding Components to the Oracle Restart Configuration
  In most cases, creating an Oracle component on a host that is running Oracle Restart automatically adds the component to the Oracle Restart configuration. However, in some cases, you must add components manually.
- Removing Components from the Oracle Restart Configuration
  When you use an Oracle-recommended method to delete an Oracle component, the component is also automatically removed from the Oracle Restart configuration.
- Disabling and Enabling Oracle Restart Management for a Component
  You can temporarily disable Oracle Restart management for a component. One reason to do this is when you are performing maintenance on the component. For example, if a component must be repaired, then you might not want it to be automatically restarted if it fails or if the host computer is restarted. When maintenance is complete, you can reenable management for the component.
- Viewing Component Status
  You can use SRVCTL to view the running status (running or not running) for any component managed by Oracle Restart. For some components, additional information is also displayed.
- Viewing the Oracle Restart Configuration for a Component
  You can use SRVCTL to view the Oracle Restart configuration for any component. Oracle Restart maintains different configuration information for each component type. In one form of the SRVCTL command, you can obtain a list of components managed by Oracle Restart.
- Modifying the Oracle Restart Configuration for a Component
  You can use SRVCTL to modify the Oracle Restart configuration of a component. For example, you can modify the port number that a listener listens on when Oracle Restart starts it, or the server parameter file (SPFILE) that Oracle Restart points to when it starts a database.
- Managing Environment Variables in the Oracle Restart Configuration
  The Oracle Restart configuration can store name/value pairs for environment variables.
- Creating and Deleting Database Services with SRVCTL
  When managing a database with Oracle Restart, Oracle recommends that you use SRVCTL to create and delete database services. When you use SRVCTL to add a database service, the service is automatically added to the Oracle Restart configuration and a dependency between the service and the database is established. Thus, if you start the service, Oracle Restart first starts the database if it is not started.
- Enabling FAN Events in an Oracle Restart Environment
  To enable Oracle Restart to publish Fast Application Notification (FAN) events, you must create an Oracle Notification Services (ONS) network that includes the Oracle Restart servers and the integrated clients.
- Automating the Failover of Connections Between Primary and Standby Databases
  In a configuration that uses Oracle Restart and Oracle Data Guard primary and standby databases, the database services fail over automatically from the primary to the standby during either a switchover or failover.
- Enabling Clients for Fast Connection Failover
  Fast Connection Failover provides high availability to Fast Application Notification (FAN) integrated clients, such as clients that use JDBC, OCI, or ODP.NET. If you configure the client to use fast connection failover, then the client automatically subscribes to FAN events and can react to database UP and DOWN events. In response, Oracle Database gives the client a connection to an active instance that provides the requested database service.

2.2.1 About Configuring Oracle Restart

If you install Oracle Restart by installing the Oracle Grid Infrastructure for Standalone Servers and then create your database, the database is automatically added to the Oracle Restart configuration, and is then automatically restarted when required. However, if you install Oracle Restart on a host computer on which a database already exists, you must manually add the database, the listener, the Oracle Automatic Storage Management (Oracle ASM) instance, and possibly other components to the Oracle Restart configuration.

After configuring Oracle Restart to manage your database, you may want to:

- Add additional components to the Oracle Restart configuration.
- Remove components from the Oracle Restart configuration.
- Temporarily suspend Oracle Restart management for one or more components.
- Modify the Oracle Restart configuration options for an individual component.

See Also:

"About Oracle Restart"

2.2.2 Preparing to Run SRVCTL

Many Oracle Restart tasks require that you run the SRVCTL utility. You must ensure that you run SRVCTL from the correct Oracle home, and that you log in to the host computer with the correct user account.

Table 2-6 lists the components that you can configure with SRVCTL, and for each component, lists the Oracle home from which you must run SRVCTL.

Table 2-6 Determining the Oracle Home from which to Start SRVCTL

Component Being Configured	Oracle Home from which to Start SRVCTL
Database, database service	Database home
Oracle ASM instance, disk group, listener[1], ONS	Oracle Grid Infrastructure home

1 Assumes the listener was started from the Oracle Grid Infrastructure home. If you installed Oracle Restart for an existing database, the listener may have been started from the database home, in which case you start SRVCTL from the database home.

To prepare to run SRVCTL:

1. Use Table 2-6 to determine the Oracle home from which you must run SRVCTL.
2. If you intend to run a SRVCTL command that modifies the Oracle Restart configuration (`add`, `remove`, `enable`, `disable`, and so on), then do one of the following:
   - On UNIX and Linux, log in to the database host computer as the user who installed the Oracle home that you determined in Step 1.
   - On Windows, log in to the database host computer as an Administrator.

   Otherwise, log in to the host computer as any user.

3. Open the command window that you will use to enter the SRVCTL commands.

   To enter commands, you might need to ensure that the SRVCTL program is in your `PATH` environment variable. Otherwise, you can enter the absolute path to the program.

2.2.3 Obtaining Help for SRVCTL

Online help is available for the SRVCTL utility.

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl
   ```

For more detailed help, enter the following command:

```
srvctl -help
```

For detailed help on a particular command, enter:

```
srvctl command -help
```

For example, to obtain help for the `add` command and the different options for each component type, enter:

```
srvctl add -help
```

For detailed help on a particular command for a particular component type, enter:

```
srvctl command object -help
```

For example, to obtain help about adding a database service, enter the following command:

```
srvctl add service -help
```

See "SRVCTL Command Reference for Oracle Restart" for a list of SRVCTL commands and Table 2-7 for a list of components.

Starting with Oracle Database 12*c*, single-letter parameters are deprecated in favor of keyword parameters. To support backward compatibility, you can use a mix of single-letter parameters and new keyword parameters. The help shows the keyword parameters by default, but you can obtain the single-letter equivalents, where applicable, by adding the `-compatible` parameter after the `-help` parameter.

For example, to obtain help about adding a database service that includes the single-letter equivalents, enter the following command:

```
srvctl add service -help -compatible
```

The single-letter equivalents appear in parentheses next to the keyword parameters. Parameters that are new in Oracle Database 12*c* and later do not have single-letter equivalents.

2.2.4 Adding Components to the Oracle Restart Configuration

In most cases, creating an Oracle component on a host that is running Oracle Restart automatically adds the component to the Oracle Restart configuration. However, in some cases, you must add components manually.

(See Table 2-2.) The component is then automatically restarted when required.

The following are occasions when you must manually add components to the Oracle Restart configuration with SRVCTL:

- You install Oracle Restart after creating the database.
- You create an additional Oracle database on the same host computer using the `CREATE DATABASE` SQL statement.
- You create a database service with `DBMS_SERVICE.CREATE_SERVICE` package procedure. (The recommended way is to use SRVCTL.)

> **Note:**
>
> Adding a component to the Oracle Restart configuration is also referred to as "registering a component with Oracle Restart."

Adding a component to the Oracle Restart configuration does not start that component. You must use a `srvctl start` command to start it.

When you add a component to the Oracle Restart configuration with SRVCTL, you can specify optional configuration settings for the component.

To add a component to the Oracle Restart configuration with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl add object options
   ```

   where *`object`* is one of the components listed in Table 2-7. See the SRVCTL add command for available options for each component.

Example 2-1 Adding a Database

This example adds a database with a `DB_UNIQUE_NAME` of `dbcrm`. The mandatory `-oraclehome` option specifies the Oracle home location.

```
srvctl add database -db dbcrm -oraclehome /u01/app/oracle/product/
database_release_number/dbhome_1
```

Example 2-2 Adding a Database Service

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example both creates a new database service named `crmbatch` and adds it to the Oracle Restart configuration.

```
srvctl add service -db dbcrm -service crmbatch
```

See "Creating and Deleting Database Services with SRVCTL" for more examples.

Example 2-3 Adding the Default Listener

This example adds the default listener to the Oracle Restart configuration.

```
srvctl add listener
```

Note:

When you install a database or manually add a database to the Oracle Restart configuration, and you have a separate Oracle Grid Infrastructure installation owner user, then you must also add the grid user as a member of the `OSRACDBA` group of that database to enable Oracle Grid Infrastructure components to connect to the database. This is because the Oracle Grid Infrastructure components must be able to connect to the database as `SYSRAC` to start and stop the database.

For example, if the host user who installed the Oracle Grid Infrastructure home is named `grid` and the `OSRACDBA` group of the Oracle home is named `racdba`, then user `grid` must be a member of the `racdba` group.

See Also:

- "Starting and Stopping Components Managed by Oracle Restart"
- "Operating System Groups"
- "SRVCTL Command Reference for Oracle Restart"

2.2.5 Removing Components from the Oracle Restart Configuration

When you use an Oracle-recommended method to delete an Oracle component, the component is also automatically removed from the Oracle Restart configuration.

For example, if you use Database Configuration Assistant (DBCA) to delete a database, DBCA removes the database from the Oracle Restart configuration. Likewise, if you use Oracle Net Configuration Assistant (NETCA) to delete a listener, NETCA removes the listener from the Oracle Restart configuration. See Table 2-3 for more examples. If you use a non-recommended or manual method to delete an Oracle component, you must first use SRVCTL to remove the component from the Oracle Restart configuration. Failing to do so could result in an error.

To remove a component from the Oracle Restart configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl remove object [options]
   ```

   where *`object`* is one of the components listed in Table 2-7. See the SRVCTL remove command for available options for each component.

Example 2-4 Removing a Database

This example removes a database with a `DB_UNIQUE_NAME` of `dbcrm`.

```
srvctl remove database -db dbcrm
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.6 Disabling and Enabling Oracle Restart Management for a Component

You can temporarily disable Oracle Restart management for a component. One reason to do this is when you are performing maintenance on the component. For example, if a component must be repaired, then you might not want it to be automatically restarted if it fails or if the host computer is restarted. When maintenance is complete, you can reenable management for the component.

When you disable a component:

- It is no longer automatically restarted.
- It is no longer automatically started through a dependency.
- It cannot be started with SRVCTL.
- Any component dependent on this resource is no longer automatically started or restarted.

To disable or enable automatic restart for a component:

1. Prepare to run SRVCTL, as described in "Preparing to Run SRVCTL".
2. Do one of the following:
   - To disable a component, enter the following command:

     ```
     srvctl disable object [options]
     ```

   - To enable a component, enter the following command:

     ```
     srvctl enable object [options]
     ```

   Replace *object* with one of the components listed in Table 2-7. See the SRVCTL disable command and the enable command for available options for each component.

Example 2-5 Disabling Automatic Restart for a Database

This example disables automatic restart for a database with a `DB_UNIQUE_NAME` of `dbcrm`.

```
srvctl disable database -db dbcrm
```

Example 2-6 Disabling Automatic Restart for an Oracle ASM Disk Group

This example disables automatic restart for the Oracle ASM disk group named `recovery`.

```
srvctl disable diskgroup -diskgroup recovery
```

Example 2-7 Enabling Automatic Restart for an Oracle ASM Disk Group

This example reenables automatic restart for the disk group `recovery`.

```
srvctl enable diskgroup -diskgroup recovery
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.7 Viewing Component Status

You can use SRVCTL to view the running status (running or not running) for any component managed by Oracle Restart. For some components, additional information is also displayed.

To view component status:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl status object [options]
   ```

   where *object* is one of the components listed in Table 2-7. See the SRVCTL status command for available options for each component.

Example 2-8 Viewing Status of a Database

This example displays the status of the database with a `DB_UNIQUE_NAME` of `dbcrm`.

```
srvctl status database -db dbcrm

Database is running.
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.8 Viewing the Oracle Restart Configuration for a Component

You can use SRVCTL to view the Oracle Restart configuration for any component. Oracle Restart maintains different configuration information for each component type. In one form of the SRVCTL command, you can obtain a list of components managed by Oracle Restart.

To view component configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl config object options
   ```

   where *object* is one of the components listed in Table 2-7. See the SRVCTL config command for available options for each component.

Example 2-9 Viewing a List of All Databases Managed by Oracle Restart

```
srvctl config database

dbcrm
orcl
```

Example 2-10 Viewing the Configuration of a Particular Database

This example displays the configuration of the database with a `DB_UNIQUE_NAME` of `orcl`.

```
srvctl config database -db orcl

Database unique name: orcl
Database name: orcl
Oracle home: /u01/app/oracle/product/database_release_number/dbhome_1
Oracle user: oracle
Spfile: +DATA/orcl/spfileorcl.ora
Domain: us.example.com
Start options: open
Stop options: immediate
Database role:
Management policy: automatic
Disk Groups: DATA
Services: mfg,sales
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.9 Modifying the Oracle Restart Configuration for a Component

You can use SRVCTL to modify the Oracle Restart configuration of a component. For example, you can modify the port number that a listener listens on when Oracle Restart starts it, or the server parameter file (SPFILE) that Oracle Restart points to when it starts a database.

To modify the Oracle Restart configuration for a component:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl modify object options
   ```

   where *`object`* is one of the components listed in Table 2-7. See the SRVCTL modify command for available options for each component.

Example 2-11 Modifying the Oracle Restart Configuration for a Database

For the database with a `DB_UNIQUE_NAME` of `dbcrm`, the following command changes the management policy to `MANUAL` and the start option to `NOMOUNT`.

```
srvctl modify database -db dbcrm -policy MANUAL -startoption NOMOUNT
```

With a `MANUAL` management policy, the database is never automatically started when the database host computer is restarted. However, Oracle Restart continues to monitor the database and restarts it if a failure occurs.

See Also:

- "Viewing the Oracle Restart Configuration for a Component"
- "SRVCTL Command Reference for Oracle Restart"

2.2.10 Managing Environment Variables in the Oracle Restart Configuration

The Oracle Restart configuration can store name/value pairs for environment variables.

- About Environment Variables in the Oracle Restart Configuration
  You can set environment variable values in the Oracle Restart configuration.
- Setting and Unsetting Environment Variables
  You use SRVCTL to set and unset environment variable values in the Oracle Restart configuration for a component.
- Viewing Environment Variables
  You use SRVCTL to view the values of environment variables in the Oracle Restart configuration for a component.

2.2.10.1 About Environment Variables in the Oracle Restart Configuration

You can set environment variable values in the Oracle Restart configuration.

If you typically set environment variables (other than `ORACLE_HOME` and `ORACLE_SID`) before starting your Oracle database, then you can set these environment variable values in the Oracle Restart configuration. You can store any number environment variables in the individual configurations of the following components:

- Database instance
- Listener
- Oracle ASM instance

When Oracle Restart starts one of these components, it first sets environment variables for that component to the values stored in the component configuration. Although you can set environment variables that are used by Oracle components in this manner, this capability is primarily intended for operating system environment variables.

The following sections provide instructions for setting, unsetting, and viewing environment variables:

- Setting and Unsetting Environment Variables
- Viewing Environment Variables

Note:

Do not use this facility to set standard environment variables like `ORACLE_HOME` and `ORACLE_SID`; these are set automatically by Oracle Restart.

2.2.10.2 Setting and Unsetting Environment Variables

You use SRVCTL to set and unset environment variable values in the Oracle Restart configuration for a component.

To set or unset environment variables in the configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Do one of the following:
   - To set an environment variable in the configuration, enter the following command:

     ```
     srvctl setenv {asm|database|listener} options
     ```

   - To remove an environment variable from the configuration, enter the following command:

     ```
     srvctl unsetenv {asm|database|listener} options
     ```

   See the SRVCTL setenv command and the unsetenv command for available options for each component.

Example 2-12 Setting Database Environment Variables

This example sets the `NLS_LANG` and the AIX `AIXTHREAD_SCOPE` environment variables in the Oracle Restart configuration for the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl setenv database -db dbcrm -envs "NLS_LANG=AMERICAN_AMERICA.AL32UTF8,
   AIXTHREAD_SCOPE=S"
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.10.3 Viewing Environment Variables

You use SRVCTL to view the values of environment variables in the Oracle Restart configuration for a component.

To view environment variable values in the configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl getenv {database|listener|asm} options
   ```

   See the SRVCTL getenv command for available options for each component.

Example 2-13 Viewing All Environment Variables for a Database

This example gets and displays the environment variables in the Oracle Restart configuration for the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl getenv database -db dbcrm

dbcrm:
NLS_LANG=AMERICAN_AMERICA
```

```
AIXTHREAD_SCOPE=S
GCONF_LOCAL_LOCKS=1
```

Example 2-14   Viewing Specific Environment Variables for a Database

This example gets and displays the `NLS_LANG` and `AIXTHREAD_SCOPE` environment variables from the Oracle Restart configuration for the same database:

```
srvctl getenv database -db dbcrm -envs "NLS_LANG,AIXTHREAD_SCOPE"

dbcrm:
NLS_LANG=AMERICAN_AMERICA
AIXTHREAD_SCOPE=S
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.11 Creating and Deleting Database Services with SRVCTL

When managing a database with Oracle Restart, Oracle recommends that you use SRVCTL to create and delete database services. When you use SRVCTL to add a database service, the service is automatically added to the Oracle Restart configuration and a dependency between the service and the database is established. Thus, if you start the service, Oracle Restart first starts the database if it is not started.

When you use SRVCTL to delete a database service, the service is also removed from the Oracle Restart configuration.

To create a database service with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl add service -db db_unique_name -service service_name [options]
   ```

   The database service is created and added to the Oracle Restart configuration. See the srvctl add service command for available options.

To delete a database service with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Enter the following command:

   ```
   srvctl remove service -db db_unique_name -service service_name [-force]
   ```

   The database service is removed from the Oracle Restart configuration. If the `-force` flag is present, the service is removed even if it is still running. Without this flag, an error occurs if the service is running.

Example 2-15   Creating a Database Service

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example creates a new database service named `crmbatch`.

```
srvctl add service -db dbcrm -service crmbatch
```

Example 2-16 Creating a Role-Based Database Service

This example creates the `crmbatch` database service and assigns it the Data Guard role of `PHYSICAL_STANDBY`. The service is automatically started only if the current role of the `dbcrm` database is physical standby.

```
srvctl add service -db dbcrm -service crmbatch -role PHYSICAL_STANDBY
```

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.12 Enabling FAN Events in an Oracle Restart Environment

To enable Oracle Restart to publish Fast Application Notification (FAN) events, you must create an Oracle Notification Services (ONS) network that includes the Oracle Restart servers and the integrated clients.

These clients can include Oracle Connection Manager (CMAN), Java Database Connectivity (JDBC), and Universal Connection Pool (UCP) clients. If you are using Oracle Call Interface or ODP.NET clients, then you must enable Oracle Advanced Queuing (AQ) HA notifications for your services. In addition, ONS must be running on the server.

To enable FAN events in an Oracle Restart environment:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Add the database to the Oracle Restart Configuration if it is not already managed by Oracle Restart. See "Adding Components to the Oracle Restart Configuration".
3. Add ONS to the configuration:

   ```
   srvctl add ons
   ```

   ONS is disabled when it is added.
4. Enable ONS:

   ```
   srvctl enable ons
   ```
5. Start ONS:

   ```
   srvctl start ons
   ```
6. Add the service to the Oracle Restart Configuration.

   For Oracle Call Interface and ODP.NET clients, ensure that the `-notification` option is set to `TRUE` to enable the database queue.

   See "Creating and Deleting Database Services with SRVCTL".
7. Enable each client for fast connection failover. See "Enabling Clients for Fast Connection Failover".

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.13 Automating the Failover of Connections Between Primary and Standby Databases

In a configuration that uses Oracle Restart and Oracle Data Guard primary and standby databases, the database services fail over automatically from the primary to the standby during either a switchover or failover.

You can use Oracle Notification Services (ONS) to immediately notify clients of the failover of services between the primary and standby databases. The Oracle Data Guard Broker uses Fast Application Notification (FAN) to send notifications to clients when a failover occurs. Integrated Oracle clients automatically failover connections and applications can mask the failure from end-users.

To automate connection failover, you must create an ONS network that includes the Oracle Restart servers and the integrated clients (CMAN, listener, JDBC, and UCP). If you are using Oracle Call Interface or ODP.NET clients, you must enable the Oracle Advanced Queuing queue. The database and the services must be managed by Oracle Restart and the Oracle Data Guard Broker to automate the failover of services.

To automate the failover of services between primary and standby databases:

1. Configure the primary and standby database with the Oracle Data Guard Broker. See *Oracle Data Guard Broker*.
2. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
3. Add the primary database to the Oracle Restart configuration on the primary server if it has not been added. Ensure that you specify `PRIMARY` for the database role. See "Adding Components to the Oracle Restart Configuration".
4. Add the standby database to the Oracle Restart configuration on the standby server if it has not been added. Ensure that you specify the appropriate standby database role.
5. Enable FAN events on both the primary database server and the standby database server. "Enabling FAN Events in an Oracle Restart Environment".
6. Add the services that clients will use to connect to the databases to the Oracle Restart configuration on the primary database and the standby database. When you add a service, ensure that:
   - The `-role` option is set to the proper role for each service
   - The `-notification` option is set to `TRUE` if you are using ODP.NET or Oracle Call Interface

   See "Creating and Deleting Database Services with SRVCTL".
7. Enable each client for fast connection failover. See "Enabling Clients for Fast Connection Failover".

See Also:

"SRVCTL Command Reference for Oracle Restart"

2.2.14 Enabling Clients for Fast Connection Failover

Fast Connection Failover provides high availability to Fast Application Notification (FAN) integrated clients, such as clients that use JDBC, OCI, or ODP.NET. If you configure the client to use fast connection failover, then the client automatically subscribes to FAN events and can react to database UP and DOWN events. In response, Oracle Database gives the client a connection to an active instance that provides the requested database service.

- About Enabling Clients for Fast Connection Failover
In a configuration with a standby database, after you have added Oracle Notification Services (ONS) to your Oracle Restart configurations and enabled Oracle Advanced Queuing (AQ) HA notifications for your services, you can enable clients for fast connection failover.
- Enabling Fast Connection Failover for JDBC Clients
Enabling FAN for the Oracle Universal Connection Pool enables Fast Connection Failover (FCF) for the client. Your application can use either thick or thin JDBC clients to use FCF.
- Enabling Fast Connection Failover for Oracle Call Interface Clients
Oracle Call Interface clients can enable Fast Connection Failover (FCF) by registering to receive notifications about Oracle Restart high availability FAN events and respond when events occur.
- Enabling Fast Connection Failover for ODP.NET Clients
Oracle Data Provider for .NET (ODP.NET) connection pools can subscribe to notifications that indicate when services are down. After a `DOWN` event, Oracle Database cleans up sessions in the connection pool that go to the instance that stops, and ODP.NET proactively disposes connections that are no longer valid.

2.2.14.1 About Enabling Clients for Fast Connection Failover

In a configuration with a standby database, after you have added Oracle Notification Services (ONS) to your Oracle Restart configurations and enabled Oracle Advanced Queuing (AQ) HA notifications for your services, you can enable clients for fast connection failover.

The clients receive Fast Application Notification (FAN) events and can relocate connections to the current primary database after an Oracle Data Guard failover. See "Automating the Failover of Connections Between Primary and Standby Databases" for information about adding ONS.

For databases with no standby database configured, you can still configure the client FAN events. When there is a failure, you can configure the client to retry the connection to the database. Since Oracle Restart will restart the failed database, the client can reconnect when the database restarts. Ensure that you program the appropriate delay and retries on the connection string, as illustrated in the examples in this section.

2.2.14.2 Enabling Fast Connection Failover for JDBC Clients

Enabling FAN for the Oracle Universal Connection Pool enables Fast Connection Failover (FCF) for the client. Your application can use either thick or thin JDBC clients to use FCF.

To configure the JDBC client, set the `FastConnectionFailoverEnabled` property before making the first `getConnection()` request to a data source. When you enable Fast Connection Failover, the failover applies to every connection in the connection cache. If your application explicitly creates a connection cache using the Connection Cache Manager, then you must first set `FastConnectionFailoverEnabled`.

This section describes how to enable FCF for JDBC with the Universal Connection Pool. For thick JDBC clients, if you enable Fast Connection Failover, do not enable Transparent Application Failover (TAF), either on the client or for the service. Enabling FCF with thin or thick JDBC clients enables the connection pool to receive and react to all FAN events.

To enable Fast Connection Failover for JDBC clients:

1. On a cache enabled DataSource, set the DataSource property `FastConnectionFailoverEnabled` to `true` as in the following example to enable FAN for the Oracle JDBC Implicit Connection Cache:

   ```
   PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
   pds.setONSConfiguration("nodes=primaryhost:6200,standbyhost:6200");
   pds.setFastConnectionFailoverEnabled(true);
   pds.setURL("jdbc:oracle:thin:@(DESCRIPTION=
     (LOAD_BALANCE=on)
     (ADDRESS=(PROTOCOL=TCP)(HOST=primaryhost)(PORT=1521))
     (ADDRESS=(PROTOCOL=TCP)(HOST=standbyhost)(PORT=1521))
     (CONNECT_DATA=(service_name=service_name)))");

   ......
   ```

   In this example, `primaryhost` is the server for the primary database, and `standbyhost` is the server for the standby database.

   Applications must have both ucp.jar and ons.jar in their `CLASSPATH`.

   Note:

   Use the following system property to enable FAN without making data source changes: `-D oracle.jdbc.FastConnectionFailover=true`.

2. When you start the application, ensure that the ons.jar file is located on the application `CLASSPATH`. The ons.jar file is part of the Oracle client installation.

See Also:

- *Oracle Database JDBC Developer's Guide*
- *Oracle Universal Connection Pool Developer's Guide*

2.2.14.3 Enabling Fast Connection Failover for Oracle Call Interface Clients

Oracle Call Interface clients can enable Fast Connection Failover (FCF) by registering to receive notifications about Oracle Restart high availability FAN events and respond when events occur.

This improves the session failover response time in Oracle Call Interface and removes terminated connections from connection and session pools. This feature works on Oracle Call Interface applications, including those that use Transparent Application Failover (TAF), connection pools, or session pools.

First, you must enable a service for high availability events to automatically populate the Advanced Queuing `ALERT_QUEUE`. If your application is using TAF, then enable the TAF settings for the service. Configure client applications to connect to an Oracle Restart database. Clients can register callbacks that are used whenever an event occurs. This reduces the time that it takes to detect a connection failure.

During `DOWN` event processing, Oracle Call Interface:

- Terminates affected connections at the client and returns an error
- Removes connections from the Oracle Call Interface connection pool and the Oracle Call Interface session pool

  The session pool maps each session to a physical connection in the connection pool, and there can be multiple sessions for each connection.
- Fails over the connection if you have configured TAF

If TAF is not configured, then the client only receives an error.

Note:

Oracle Call Interface does not manage `UP` events.

To Enable Fast Connection Failover for an Oracle Call Interface client:

1. Ensure that the service that you are using has Advanced Queuing notifications enabled by setting the services' values using the SRVCTL `modify` command. For example:

   ```
   srvctl modify service -db proddb -service gl.us.example.com -notification
   true -role primary -failovertype select -failovermethod basic -failoverretry 5
   -failoverdelay 180 -clbgoal long
   ```
2. Enable `OCI_EVENTS` at environment creation time on the client as follows:

   ```
   ( OCIEnvCreate(...) )
   ```
3. Link client applications with the client thread or operating system library.
4. (Optional) Register a client `EVENT` callback.
5. Ensure that the client uses an Oracle Net connect descriptor that includes all primary and standby hosts in the `ADDRESS_LIST`. For example:

   ```
   gl =
   (DESCRIPTION =
    (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
      (ADDRESS_LIST =
   ```

```
          (ADDRESS = (PROTOCOL = TCP)(HOST = BOSTON1)(PORT = 1521))
          (ADDRESS = (PROTOCOL = TCP)(HOST = CHICAGO1)(PORT = 1521))
          (LOAD_BALANCE = yes)
     )
     (CONNECT_DATA=
       (SERVICE_NAME=gl.us.example.com)))
```

To see the alert information, query the views `DBA_OUTSTANDING_ALERTS` and `DBA_ALERT_HISTORY`.

See Also:

- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Net Services Administrator's Guide* for information about configuring TAF

2.2.14.4 Enabling Fast Connection Failover for ODP.NET Clients

Oracle Data Provider for .NET (ODP.NET) connection pools can subscribe to notifications that indicate when services are down. After a `DOWN` event, Oracle Database cleans up sessions in the connection pool that go to the instance that stops, and ODP.NET proactively disposes connections that are no longer valid.

All three ODP.NET providers (core, managed, and unmanaged) support FCF.

To enable Fast Connection Failover for ODP.NET clients:

1. Enable Fast Application Notification (FAN) by using SRVCTL `modify service` command, as in the following example:

   ```
   srvctl modify service -db dbname -service gl -notification true
   ```

2. Enable Fast Connection Failover for ODP.NET connection pools by subscribing to FAN high availability events. Set the `HA` Events connection string attribute to true at connection time. In newer ODP.NET versions, `HA` Events is set to true by default. The pooling attribute must be set to `true`, which is the default. The following example illustrates these settings, where *`user_name`* is the name of the user and *`password`* is the user password:

   ```
   // C#
   using System;
   using Oracle.ManagedDataAccess.Client;
   //using Oracle.DataAccess.Client;

   class HAEventEnablingSample
   {
     static void Main()
     {
       OracleConnection con = new OracleConnection();

       // Open a connection using ConnectionString attributes
       // Also, enable "load balancing"
       con.ConnectionString =
         "User Id=user_name;Password=password;Data Source=oracle;" +
         "Min Pool Size=10;Connection Lifetime=120;Connection Timeout=60;" +
         "HA Events=true;Incr Pool Size=5;Decr Pool Size=2";
   ```

```
      con.Open();

      // Create more connections and perform work against the database here.

      // Dispose OracleConnection object
      con.Dispose();
    }
  }
```

3. Ensure that the client uses an Oracle Net connect descriptor that includes all primary and standby hosts in the `ADDRESS_LIST`. For example:

```
gl =
(DESCRIPTION =
 (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
   (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = BOSTON1)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = CHICAGO1)(PORT = 1521))
     (LOAD_BALANCE = yes)
 )
 (CONNECT_DATA=
  (SERVICE_NAME=gl.us.example.com))
 )
)
```

See Also:

- *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows* for information about ODP.NET
- "SRVCTL Command Reference for Oracle Restart"

2.3 Starting and Stopping Components Managed by Oracle Restart

When Oracle Restart is in use, Oracle strongly recommends that you use the SRVCTL utility to start and stop components.

Use the SRVCTL utility to start and stop components for the following reasons:

- When starting a component with SRVCTL, Oracle Restart can first start any components on which this component depends. When stopping a component with SRVCTL, Oracle Restart can stop any dependent components first.
- SRVCTL always starts a component according to its Oracle Restart configuration. Starting a component by other means may not.

  For example, if you specified a server parameter file (SPFILE) location when you added a database to the Oracle Restart configuration, and that location is not the default location for SPFILEs, if you start the database with SQL*Plus, the SPFILE specified in the configuration may not be used.

  See the srvctl add database command for a table of configuration options for a database instance.

- When you start a component with SRVCTL, environment variables stored in the Oracle Restart configuration for the component are set.

  See "Managing Environment Variables in the Oracle Restart Configuration" for more information.

You can start and stop any component managed by Oracle Restart with SRVCTL.

To start or stop a component managed by Oracle Restart with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Do one of the following:
   - To start a component, enter the following command:

     ```
     srvctl start object [options]
     ```

   - To stop a component, enter the following command:

     ```
     srvctl stop object [options]
     ```

   where `object` is one of the components listed in Table 2-7. See the SRVCTL start command and the stop command for available options for each component.

Example 2-17 Starting a Database

This example starts the database with a `DB_UNIQUE_NAME` of dbcrm:

```
srvctl start database -db dbcrm
```

Example 2-18 Starting a Database NOMOUNT

This example starts the database instance without mounting the database:

```
srvctl start database -db dbcrm -startoption nomount
```

Example 2-19 Starting the Default Listener

This example starts the default listener:

```
srvctl start listener
```

Example 2-20 Starting a Specified Listener

This example starts the listener named `crmlistener`:

```
srvctl start listener -listener crmlistener
```

Example 2-21 Starting Database Services

This example starts the database services `bizdev` and `support` for the database with a `DB_UNIQUE_NAME` of `dbcrm`. If the database is not started, Oracle Restart first starts the database.

```
srvctl start service -db dbcrm -service "bizdev,support"
```

Example 2-22 Starting (Mounting) Oracle ASM Disk Groups

This example starts (mounts) the Oracle ASM disk groups `data` and `recovery`. The user running this command must be a member of the OSASM group.

```
srvctl start diskgroup -diskgroup "data,recovery"
```

Example 2-23 Shutting Down a Database

This example stops (shuts down) the database with a `DB_UNIQUE_NAME` of `dbcrm`. Because a stop option (`-stopoption`) is not provided, the database shuts down according to the stop option in its Oracle Restart configuration. The default stop option is `IMMEDIATE`.

```
srvctl stop database -db dbcrm
```

Example 2-24 Shutting Down a Database with the ABORT option

This example does a `SHUTDOWN ABORT` of the database with a `DB_UNIQUE_NAME` of `dbcrm`.

```
srvctl stop database -db dbcrm -stopoption abort
```

Note:

After relinking Oracle executables, use the SRVCTL utility to start and stop components when Oracle Restart is in use. Typically, relinking Oracle executables is required on a Linux or UNIX-based operating system after you apply an operating system patch or after an operating system upgrade. See *Oracle Database Administrator's Reference for Linux and UNIX-Based Operating Systems* for more information about relinking.

If you use SQL*Plus to start and stop components, then you must first run the setasmgidwrap script after relinking. See *Oracle Database Upgrade Guide* for information about running this script.

See Also:

The SRVCTL start command

2.4 Stopping and Restarting Oracle Restart for Maintenance Operations

When several components in an Oracle home are managed by Oracle Restart, you can stop Oracle Restart and the components managed by Oracle Restart in the Oracle home.

You can also disable Oracle Restart so that it is not restarted if the node reboots. You might need to do this when you are performing maintenance that includes the Oracle home, such as installing a patch. When the maintenance operation is complete, you can enable and restart Oracle Restart, and you can restart the components managed by Oracle Restart in the Oracle home.

Use both the SRVCTL utility and the CRSCTL utility for the stop and start operations:

- The `stop home` SRVCTL command stops all of the components that are managed by Oracle Restart in the specified Oracle home. The `start home` SRVCTL command starts these components. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

When you use the `home` object, a state file, specified in the `-statefile` option, tracks the state of each component. The `stop` and `status` commands create the state file. The `start` command uses the state file to identify the components to restart.

In addition, you can check the status of the components managed by Oracle Restart using the `status home` command.

- The `stop` CRSCTL command stops Oracle Restart, and the `disable` CRSCTL command ensures that the components managed by Oracle Restart do not restart automatically. The `enable` CRSCTL command enables automatic restart and the `start` CRSCTL command restarts Oracle Restart.

To stop and start the components in an Oracle home while installing a patch:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL".
2. Use the SRVCTL utility to stop the components managed by Oracle Restart in an Oracle home:

   ```
   srvctl stop home -oraclehome oracle_home -statefile state_file [-stopoption
   stop_options] [-force]
   ```

   where *oracle_home* is the complete path of the Oracle home and *state_file* is the complete path to the state file. State information for the Oracle home is recorded in the specified state file. Make a note of the state file location because it must be specified in Step 7.

   Before stopping the components in an Oracle Grid Infrastructure home, ensure that you first stop the components in a dependent Oracle Database home.
3. If you are patching an Oracle Grid Infrastructure home, then disable and stop Oracle Restart. Otherwise, go to Step 4.

   To disable and stop Oracle Restart, use the CRSCTL utility to run the following commands:

   ```
   crsctl disable has

   crsctl stop has
   ```
4. Perform the maintenance operation.
5. Use the CRSCTL utility to enable automatic restart of the components managed by Oracle Restart:

   ```
   crsctl enable has
   ```
6. Use the CRSCTL utility to start Oracle Restart:

   ```
   crsctl start has
   ```
7. Use the SRVCTL utility to start the components that were stopped in Step 2:

   ```
   srvctl start home -oraclehome oracle_home -statefile state_file
   ```

   The state file must match the state file specified in Step 2.
8. (Optional) Use the SRVCTL utility to check the status of the components managed by Oracle Restart in the Oracle home:

   ```
   srvctl status home -oraclehome oracle_home -statefile state_file
   ```

Example 2-25 Stopping Components Managed by Oracle Restart in an Oracle Home

```
srvctl stop home -oraclehome /u01/app/oracle/product/database_release_number/dbhome_1 -
statefile /usr1/or_state
```

Example 2-26 Starting Components Managed by Oracle Restart in an Oracle Home

```
srvctl start home -oraclehome /u01/app/oracle/product/database_release_number/dbhome_1 -
statefile /usr1/or_state
```

Example 2-27 Displaying the Status of Components Managed by Oracle Restart in an Oracle Home

```
srvctl status home -oraclehome /u01/app/oracle/product/database_release_number/dbhome_1 -
statefile /usr1/or_state
```

See Also:

- The srvctl stop home command
- The srvctl status home command
- The srvctl start home command
- "CRSCTL Command Reference"

2.5 SRVCTL Command Reference for Oracle Restart

You can reference details about the syntax and options for SRVCTL commands specific to Oracle Restart.

See *Oracle Real Application Clusters Administration and Deployment Guide* for the full list of SRVCTL commands.

SRVCTL Command Syntax and Options Overview

SRVCTL expects the following command syntax:

```
srvctl command object options
```

where:

- `command` is a verb such as `start`, `stop`, or `remove`.
- `object` is the component on which SRVCTL performs the command, such as database, listener, and so on. You can also use component abbreviations. See Table 2-7 for a complete list of components and their abbreviations.
- `options` extend the use of a preceding command combination to include additional parameters for the command. For example, the `-db` option indicates that a database unique name follows, and the `-service` option indicates that a comma-delimited list of database service names follows.

Note:

On the Windows platform, when specifying a comma-delimited list, you must enclose the list within double-quotes ("...,..."). You must also use double-quotes on the UNIX and Linux platforms if any list member contains shell metacharacters.

Case Sensitivity

SRVCTL commands and components are case insensitive. Options are case sensitive. Database and database service names are case insensitive and case preserving.

Command Parameters Input File

You can specify command parameters in a file rather than directly on the command line. Using a command parameters input file is useful in the following situations:

- You want to run a command with very long parameter values or a command with numerous parameters
- You want to bypass shell processing of certain special characters

To specify a command parameters input file, use the `-file` parameter with a value that is the location of the command parameters file. SRVCTL processes the command parameters from the command parameters file instead of from the command line.

SRVCTL Components Summary

Table 2-7 lists the keywords that can be used for the *`object`* portion of SRVCTL commands. You can use either the full name or the abbreviation for each component keyword.

Table 2-7 Component Keywords and Abbreviations

Component	Abbreviation	Description
`asm`	`asm`	Oracle ASM instance
`database`	`db`	Database instance
`diskgroup`	`dg`	Oracle ASM disk group
`home`	`home`	Oracle home or Oracle Clusterware home
`listener`	`lsnr`	Oracle Net listener
`service`	`serv`	Database service
`ons`	`ons`	Oracle Notification Services (ONS)

- add
  The `srvctl add` command adds the specified component to the Oracle Restart configuration, and optionally sets Oracle Restart configuration parameters for the component. After a component is added, Oracle Restart begins to manage it, restarting it when required.
- config
  The `srvctl config` command displays the Oracle Restart configuration of the specified component or set of components.
- disable
  Disables a component, which suspends management of that component by Oracle Restart.
- downgrade
  The `srvctl downgrade` command downgrades the database configuration after you manually downgrade the database.

- enable
  The `srvctl enable` command reenables the specified disabled component.
- getenv
  Gets and displays environment variables and their values from the Oracle Restart configuration for a database, listener, or Oracle ASM instance.
- modify
  Modifies the Oracle Restart configuration of a component. The change takes effect when the component is next restarted.
- remove
  Removes the specified component from the Oracle Restart configuration. Oracle Restart no longer manages the component. Any environment variable settings for the component are also removed.
- setenv
  The `setenv` command sets values of environment variables in the Oracle Restart configuration for a database, a listener, or the Oracle ASM instance.
- start
  Starts the specified component or components.
- status
  Displays the running status of the specified component or set of components.
- stop
  Stops the specified component or components.
- unsetenv
  The `unsetenv` command deletes one or more environment variables from the Oracle Restart configuration for a database, a listener, or an Oracle ASM instance.
- update
  The `srvctl update` command updates the running database to switch to the specified startup option.
- upgrade
  The `srvctl upgrade` command upgrades the resources types and resources from an older version to a newer version.

See Also:

Table 2-1

2.5.1 add

The `srvctl add` command adds the specified component to the Oracle Restart configuration, and optionally sets Oracle Restart configuration parameters for the component. After a component is added, Oracle Restart begins to manage it, restarting it when required.

To perform `srvctl add` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

Note:

There is no `srvctl add` command for Oracle ASM disk groups. Disk groups are automatically added to the Oracle Restart configuration when they are first mounted. If you remove a disk group from the Oracle Restart configuration and later want to add it back, connect to the Oracle ASM instance with SQL*Plus and use an `ALTER DISKGROUP` ... `MOUNT` command.

- srvctl add asm
  Adds an Oracle ASM instance to the Oracle Restart configuration.
- srvctl add database
  Adds a database to the Oracle Restart configuration.
- srvctl add listener
  Adds a listener to the Oracle Restart configuration.
- srvctl add ons
  Adds Oracle Notification Services (ONS) to an Oracle Restart configuration.
- srvctl add service
  Adds a database service to the Oracle Restart configuration.

2.5.1.1 srvctl add asm

Adds an Oracle ASM instance to the Oracle Restart configuration.

- Syntax and Options
- Example

2.5.1.1.1 Syntax and Options

Use the `srvctl add asm` command with the following syntax:

```
srvctl add asm [-listener listener_name] [-spfile spfile]
  [-pwfile password_file_path] [-diskstring asm_diskstring]
```

Table 2-8 srvctl add asm Options

Option	Description
`-listener` *`listener_name`*	Name of the listener with which Oracle ASM should register. A weak dependency is established with this listener. (Before starting the Oracle ASM instance, Oracle Restart attempts to start the listener. If the listener does not start, the Oracle ASM instance is still started. If the listener later fails, Oracle Restart does not restart Oracle ASM.) If omitted, defaults to the listener named `listener`.
`-spfile` *`spfile`*	The full path of the server parameter file for the database. If omitted, the default SPFILE is used.
`-pwfile` *`password_file_path`*	The full path of the Oracle ASM password file.

Table 2-8 (Cont.) srvctl add asm Options

Option	Description
`-diskstring` *`asm_diskstring`*	Oracle ASM disk group discovery string. An Oracle ASM discovery string is a comma-delimited list of strings that limits the set of disks that an Oracle ASM instance discovers. The discovery strings can include wildcard characters. Only disks that match one of the strings are discovered.

2.5.1.1.2 Example

An example of this command is:

```
srvctl add asm -listener crmlistener
```

See Also:

Oracle Automatic Storage Management Administrator's Guide for more information about Oracle ASM disk group discovery strings

2.5.1.2 srvctl add database

Adds a database to the Oracle Restart configuration.

After adding a database to the Oracle Restart configuration, if the database then accesses data in an Oracle ASM disk group, a dependency between the database that disk group is created. Oracle Restart then ensures that the disk group is mounted before attempting to start the database.

However, if the database and Oracle ASM instance are not running when you add the database to the Oracle Restart configuration, you must manually establish the dependency between the database and its disk groups by specifying the `-diskgroup` option in the SRVCTL command. See the example later in this section.

Note:

When you manually add a database to the Oracle Restart configuration, you must also add the Oracle grid infrastructure software owner as a member of the OSDBA group of that database. This is because the grid infrastructure components must be able to connect to the database as `SYSDBA` to start and stop the database.

For example, if the host user who installed the grid infrastructure home is named `grid` and the OSDBA group of the new database is named `dba`, then user `grid` must be a member of the `dba` group.

- Syntax and Options
- Examples

2.5.1.2.1 Syntax and Options

Use the `srvctl add database` command with the following syntax:

```
srvctl add database -db db_unique_name -oraclehome oracle_home
  [-domain domain_name] [-dbname db_name] [-instance instance_name]
  [-spfile spfile][-pwfile password_file_path] [-startoption start_options]
  [-stopoption stop_options]
  [-role {PRIMARY | PHYSICAL_STANDBY | LOGICAL_STANDBY |
         SNAPSHOT_STANDBY | FAR_SYNC}]
  [-policy {AUTOMATIC | MANUAL | NORESTART}] [-diskgroup disk_group_list]
  [-verbose]
```

Table 2-9 srvctl add database Options

Syntax	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-oraclehome` *`oracle_home`*	The full path of Oracle home for the database
`-domain` *`domain_name`*	The domain for the database. Must match the `DB_DOMAIN` initialization parameter.
`-dbname` *`db_name`*	If provided, must match the `DB_NAME` initialization parameter setting. You must include this option if `DB_NAME` is different from the unique name given by the `-db` option
`-instance` *`instance_name`*	The instance name. You must include this option if the instance name is different from the unique name given by the `-db` option. For example, if the unique name includes an underscore, and the instance name omits the underscore, then use this parameter to specify the instance name.
`-spfile` *`spfile`*	The full path of the server parameter file for the database. If omitted, the default SPFILE is used.
`-pwfile` *`password_file_path`*	The full path of the database password file.
`-startoption` *`start_options`*	Startup options for the database (`OPEN`, `MOUNT`, or `NOMOUNT`). If omitted, defaults to `OPEN`. **See Also:** *SQL*Plus User's Guide and Reference* for more information about startup options
`-stopoption` *`stop_options`*	Shutdown options for the database (`NORMAL`, `IMMEDIATE`, `TRANSACTIONAL`, or `ABORT`). If omitted, defaults to `IMMEDIATE`. **See Also:** *SQL*Plus User's Guide and Reference* for more information about shutdown options
`-role {PRIMARY | PHYSICAL_STANDBY | LOGICAL_STANDBY | SNAPSHOT_STANDBY | FAR_SYNC}`	The current role of the database (`PRIMARY`, `PHYSICAL_STANDBY`, `LOGICAL_STANDBY`, `SNAPSHOT_STANDBY`, or `FAR_SYNC`). The default is `PRIMARY`. Applicable in Oracle Data Guard environments only. **See Also:** *Oracle Data Guard Concepts and Administration* for more information about database roles

Table 2-9 (Cont.) srvctl add database Options

Syntax	Description
`-policy {AUTOMATIC | MANUAL | NORESTART}`	Management policy for the database. • `AUTOMATIC` (default): The database is automatically restored to its previous running condition (started or stopped) upon restart of the database host computer. • `MANUAL`: The database is never automatically restarted upon restart of the database host computer. A `MANUAL` setting does not prevent Oracle Restart from monitoring the database while it is running and restarting it if a failure occurs. • `NORESTART`: Similar to the `MANUAL` setting, the database is never automatically restarted upon restart of the database host computer. A `NORESTART` setting, however, never restarts the database even if a failure occurs.
`-diskgroup` *`disk_group_list`*	Comma separated list of disk groups upon which the database is dependent. When starting the database, Oracle Restart first ensures that these disk groups are mounted. This option is required only if the database instance and the Oracle ASM instance are not started when adding the database. Otherwise, the dependency is recorded automatically between the database and its disk groups.
`-verbose`	Verbose output

2.5.1.2.2 Examples

This example adds the database with the `DB_UNIQUE_NAME` `dbcrm`:

```
srvctl add database -db dbcrm -oraclehome /u01/app/oracle/product/
database_release_number/dbhome_1
```

This example adds the same database and also establishes a dependency between the database and the disk groups `DATA` and `RECOVERY`.

```
srvctl add database -db dbcrm -oraclehome /u01/app/oracle/product/
database_release_number/dbhome_1
  -diskgroup "DATA,RECOVERY"
```

See Also:

- "Oracle Restart Integration with Oracle Data Guard"
- *Oracle Data Guard Concepts and Administration*

2.5.1.3 srvctl add listener

Adds a listener to the Oracle Restart configuration.

- Syntax and Options
- Example

2.5.1.3.1 Syntax and Options

Use the `srvctl add listener` command with the following syntax:

```
srvctl add listener [-listener listener_name] [-endpoints endpoints] [-skip]
  [-oraclehome oracle_home]
```

Table 2-10 srvctl add listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, defaults to `LISTENER`
`-endpoints` *`endpoints`*	Comma separated TCP ports or listener endpoints. If omitted, defaults to TCP:1521. *endpoints* syntax is: `"[TCP:]port[, ...] [/IPC:key] [/NMP:pipe_name] [/TCPS:s_port] [/SDP:port]"`
`-skip`	Skip checking for port conflicts with the supplied endpoints
`-oraclehome` *`oracle_home`*	Oracle home for the listener. If omitted, the Oracle Grid Infrastructure home is assumed.

2.5.1.3.2 Example

The following command adds a listener (named `LISTENER`) running out of the database Oracle home and listening on TCP port 1522:

```
srvctl add listener -endpoints TCP:1522
  -oraclehome /u01/app/oracle/product/database_release_number/dbhome_1
```

2.5.1.4 srvctl add ons

Adds Oracle Notification Services (ONS) to an Oracle Restart configuration.

ONS must be added to an Oracle Restart configuration to enable the sending of Fast Application Notification (FAN) events after an Oracle Data Guard failover.

When ONS is added to an Oracle Restart configuration, it is initially disabled. You can enable it with the `srvctl enable ons` command.

- Syntax and Options

See Also:

"srvctl enable ons"

2.5.1.4.1 Syntax and Options

Use the `srvctl add ons` command with the following syntax:

```
srvctl add ons [-emport em_port] [-onslocalport ons_local_port]
  [-onsremoteport ons_remote_port] [-remoteservers host[:port],[host[:port]...]]
  [-verbose]
```

Table 2-11 srvctl add ons Options

Option	Description
`-emport` *`em_port`*	ONS listening port for Oracle Enterprise Manager Cloud Control (Cloud Control). The default is 2016.
`-onslocalport` *`ons_local_port`*	ONS listening port for local client connections. The default is 6100.
`-onsremoteport` *`ons_remote_port`*	ONS listening port for connections from remote hosts. The default is 6200.
`-remoteservers host[:port], [host[:port],...`	A list of *`host:port`* pairs of remote hosts that are part of the ONS network **Note:** If *`port`* is not specified for a remote host, then `ons_remote_port` is used.
`-verbose`	Verbose output

2.5.1.5 srvctl add service

Adds a database service to the Oracle Restart configuration.

Creates the database service if it does not exist. This method of creating a service is preferred over using the `DBMS_SERVICE` PL/SQL package.

- Syntax and Options
- Example

2.5.1.5.1 Syntax and Options

Use the `srvctl add service` command with the following syntax:

```
srvctl add service -db db_unique_name -service service_name
  [-role [PRIMARY][,PHYSICAL_STANDBY][,LOGICAL_STANDBY][,SNAPSHOT_STANDBY]]
  [-policy {AUTOMATIC | MANUAL}]
  [-failovertype {NONE | SESSION | SELECT | TRANSACTION}]
  [-failovermethod {NONE | BASIC}] [-failoverdelay integer]
  [-failoverretry integer] [-clbgoal {SHORT | LONG}]
  [-rlbgoal {SERVICE_TIME | THROUGHPUT | NONE}] [-notification {TRUE | FALSE}]
  [-edition edition_name] [-pdb pluggable_database]
  [-sql_translation_profile sql_translation_profile]
  [-commit_outcome {TRUE | FALSE}] [-retention retention]
  [-replay_init_time replay_init_time] [-drain_timeout timeout]
  [-stopoption stop_option] [-session_state {STATIC | DYNAMIC}]
  [-global {TRUE | FALSE}] [-maxlag max_lag_time] [-force] [-verbose]
```

Table 2-12 srvctl add service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database The name must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name`*	The database service name

Table 2-12 (Cont.) srvctl add service Options

Option	Description
`-role [PRIMARY][,PHYSICAL_STANDBY][,LOGICAL_STANDBY][,SNAPSHOT_STANDBY]`	A list of service roles This option is applicable in Oracle Data Guard environments only. When this option is present, upon database open, the service is started only when one of its service roles matches the current database role. **See Also:** *Oracle Data Guard Concepts and Administration* for more information about database roles
`-policy {AUTOMATIC | MANUAL}`	Management policy for the service If `AUTOMATIC` (the default), the service is automatically started upon restart of the database, either by a planned restart (with SRVCTL) or after a failure. Automatic restart is also subject to the service role, however (the `-role` option). If `MANUAL`, the service is never automatically restarted upon planned restart of the database (with SRVCTL). A `MANUAL` setting does not prevent Oracle Restart from monitoring the service when it is running and restarting it if a failure occurs.
`-failovertype {NONE |SESSION | SELECT | TRANSACTION}`	To enable Application Continuity for OCI and Java, use `TRANSACTION`. If the failover type is `TRANSACTION`, then OCI and Java attempt to recover the in progress transaction upon receipt of a recoverable error. When failover type is `TRANSACTION`, the `-commit_outcome` option must be set to `TRUE`. To enable Transparent Application Failover (TAF) for OCI, use `SELECT` or `SESSION`.
`-failovermethod {NONE | BASIC}`	TAF failover method for backward compatibility only If the failover type (`-failovertype`) is set to a value other than `NONE`, then use `BASIC` for this option.
`-failoverdelay` *`integer`*	For Application Continuity and TAF, the time delay, in seconds, between reconnect attempts for each incident at failover
`-failoverretry` *`integer`*	For Application Continuity and TAF, the number of attempts to connect after an incident
`-clbgoal {SHORT | LONG}`	Connection load balancing goal Use `SHORT` for run-time load balancing. Use `LONG` for long running connections, such as batch jobs.
`-rlbgoal {SERVICE_TIME | THROUGHPUT | NONE}`	Run-time load balancing goal Use `SERVICE_TIME` to balance connections by response time. Use `THROUGHPUT` to balance connections by throughput.
`-notification {TRUE | FALSE}`	Enable Fast Application Notification (FAN) for OCI connections
`-edition` *`edition_name`*	The initial session edition of the service When an edition is specified for a service, all subsequent connections that specify the service use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the initial session edition. SRVCTL does not validate the specified edition name. During connection, the connect user must have `USE` privilege on the specified edition. If the edition does not exist or if the connect user does not have `USE` privilege on the specified edition, then an error is raised.

Table 2-12 (Cont.) srvctl add service Options

Option	Description
`-pdb` *`pluggable_database`*	In a multitenant container database (CDB), the name of the pluggable database (PDB) to associate with the service If this option is set to an empty string, then the service is associated with root.
`-sql_translation_profile` *`sql_translation_profile`*	A SQL translation profile for a service that you are adding after you have migrated applications from a non-Oracle database to an Oracle database This parameter corresponds to the SQL translation profile parameter in the `DBMS_SERVICE` service attribute. **Notes:** • Before using the SQL translation framework, you must migrate all server-side application objects and data to the Oracle database. • Use the `srvctl config service` command to display the SQL translation profile. **See Also:** *Oracle Database SQL Translation and Migration Guide* for more information about using a SQL translation profile
`-commit_outcome {TRUE | FALSE}`	For Transaction Guard, when `TRUE` a transaction's commit outcome is accessible after the transaction's session fails due to a recoverable outage. If `FALSE`, the default, then a transaction's commit outcome is not retained. When this option is set to `TRUE`, the outcome of a transaction's commit is durable, and an applications can determine the commit status of a transaction after an outage. You can set `commit_outcome` to `TRUE` for a user-defined service. The `commit_outcome` setting has no effect on Oracle Active Data Guard and read-only databases. **See Also:** See *Oracle Database Development Guide* for more information.
`-retention` *`retention`*	If `commit_outcome` is set to `TRUE`, then this option determines the amount of time, in seconds, that the commit outcome is retained. The default is 24 hours (86400). If `commit_outcome` is set to `FALSE`, then this option cannot be set.
`-replay_init_time` *`replay_init_time`*	For Application Continuity, this option specifies the difference between the time, in seconds, of original execution of the first operation of a request and the time that the replay is ready to start after a successful reconnect. Application Continuity will not replay after the specified amount of time has passed. This option is intended to avoid the unintentional execution of a transaction when a system is recovered after a long period of time. The default is 5 minutes (300). The maximum value is 24 hours (86400). If `failovertype` is not set to `TRANSACTION`, then this option is not used.
`-drain_timeout` *`timeout`*	This option specifies the time allowed for resource draining to be completed in seconds. Permitted values are `NULL`, `0`, or any positive integer. The draining period is intended for planned maintenance operations. During the draining period, all current client requests are processed, but new requests are not accepted. How draining works depends on the setting of the `-stopoption` option. The default value is `NULL`, which means that this option is not set. If the option is not set, and `-drain_timeout` has been set on the service, then this value is used. If it is set to `0`, then draining does not occur.

Table 2-12 (Cont.) srvctl add service Options

Option	Description
`-stopoption` *`stop_option`*	This option specifies the mode in which the service is stopped. The following values are permitted: • `IMMEDIATE` specifies that sessions are permitted to drain before the service is stopped. • `TRANSACTIONAL` specifies that sessions are permitted to drain for the amount of time specified in the `–drain_timeout` option. The service is stopped when the time limit is reached, and any remaining sessions are terminated. • `NONE` is the default.
`-session_state {STATIC | DYNAMIC}`	For Application Continuity, this parameter specifies whether the session state that is not transactional is changed by the application. Oracle recommends a setting of `DYNAMIC` for most applications. **Note:** This parameter is considered only if `-failovertype` is set to `TRANSACTION` for Application Continuity. It describes how non-transactional is changed during a request. Examples of session state are NLS settings, optimizer preferences, event settings, PL/SQL global variables, temporary tables, advanced queues, LOBs, and result cache. If non-transactional values change after the request starts, then use the default, `DYNAMIC`. Most applications should use `DYNAMIC` mode. If you are unsure, then use `DYNAMIC` mode.
`-global {TRUE | FALSE}`	If `TRUE`, then the service is a Global Data Services (GDS) service and is managed by the Global Services Manager (GSM). If `FALSE`, the default, then the service is not a GDS service. The global attribute of a service cannot be changed after the service is added. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.
`-maxlag` *`maximum_lag_time`*	Maximum replication lag time in seconds. Must be a non-negative integer. The default value is `ANY`.
`-force`	Force the add operation even though a listener is not configured for a network.
`-verbose`	Verbose output

2.5.1.5.2 Example

This example adds the `sales` service for the database with `DB_UNIQUE_NAME` `dbcrm`. The service is started only when `dbcrm` is in `PRIMARY` mode.

```
srvctl add service -db dbcrm -service sales -role PRIMARY
```

See Also:

- The section in *Oracle Database PL/SQL Packages and Types Reference* on the `DBMS_SERVICE` package for more information about the options for this command
- "Oracle Restart Integration with Oracle Data Guard"
- *Oracle Data Guard Concepts and Administration*
- *Oracle Multitenant Administrator's Guide* for information about creating, modifying, or removing a service for a pluggable database (PDB)

2.5.2 config

The `srvctl config` command displays the Oracle Restart configuration of the specified component or set of components.

- srvctl config asm
  Displays the Oracle Restart configuration information for the Oracle ASM instance.
- srvctl config database
  Displays the Oracle Restart configuration information for the specified database, or lists all databases managed by Oracle Restart.
- srvctl config listener
  Displays the Oracle Restart configuration information for all Oracle Restart–managed listeners or for the specified listener.
- srvctl config ons
  Displays the current configuration information for Oracle Notification Services (ONS).
- srvctl config service
  For the specified database, displays the Oracle Restart configuration information for the specified database service or for all Oracle Restart–managed database services.

2.5.2.1 srvctl config asm

Displays the Oracle Restart configuration information for the Oracle ASM instance.

- Syntax and Options
- Example

2.5.2.1.1 Syntax and Options

Use the `srvctl config asm` command with the following syntax:

```
srvctl config asm [-all]
```

Table 2-13 srvctl config asm Options

Option	Description
`-all`	Display enabled/disabled status also

2.5.2.1.2 Example

An example of this command is:

```
srvctl config asm -all

asm home: /u01/app/oracle/product/database_release_number/grid
ASM is enabled.
```

2.5.2.2 srvctl config database

Displays the Oracle Restart configuration information for the specified database, or lists all databases managed by Oracle Restart.

- Syntax and Options
- Example

2.5.2.2.1 Syntax and Options

Use the `srvctl config database` command with the following syntax:

```
srvctl config database [-db db_unique_name [-all]] [-verbose]
```

Table 2-14 srvctl config database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-all`	Display detailed configuration information
`-verbose`	Verbose output

2.5.2.2.2 Example

An example of this command to list all Oracle Restart–managed databases is:

```
srvctl config database

dbcrm
orcl
```

An example of this command to display configuration and enabled/disabled status for the database with the `DB_UNIQUE_ID` `orcl` is:

```
srvctl config database -db orcl -all

Database unique name: orcl
Database name: orcl
Oracle home: /u01/app/oracle/product/database_release_number/dbhome_1
Oracle user: oracle
Spfile: +DATA/orcl/spfileorcl.ora
Domain: us.example.com
```

```
Start options: open
Stop options: immediate
Database role:
Management policy: automatic
Disk Groups: DATA
Services: mfg,sales
Database is enabled
```

2.5.2.3 srvctl config listener

Displays the Oracle Restart configuration information for all Oracle Restart–managed listeners or for the specified listener.

- Syntax and Options
- Example

2.5.2.3.1 Syntax and Options

Use the `srvctl config listener` command with the following syntax:

```
srvctl config listener [-listener listener_name]
```

Table 2-15 srvctl config listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, configuration information for all Oracle Restart–managed listeners is displayed.

2.5.2.3.2 Example

This example displays the configuration information and enabled/disabled status for the default listener:

```
srvctl config listener

Name: LISTENER
Home: /u01/app/oracle/product/database_release_number/dbhome_1
End points: TCP:1521
Listener is enabled.
```

2.5.2.4 srvctl config ons

Displays the current configuration information for Oracle Notification Services (ONS).

- Syntax and Options

2.5.2.4.1 Syntax and Options

Use the `srvctl config ons` command with the following syntax:

```
srvctl config ons
```

2.5.2.5 srvctl config service

For the specified database, displays the Oracle Restart configuration information for the specified database service or for all Oracle Restart–managed database services.

- Syntax and Options
- Example

2.5.2.5.1 Syntax and Options

Use the `srvctl config service` command with the following syntax:

```
srvctl config service -db db_unique_name [-service service_name] [-verbose]
```

Table 2-16 srvctl config service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name`*	Database service name. If omitted, SRVCTL displays configuration information for all Oracle Restart–managed services for the database.
`-verbose`	Verbose output

2.5.2.5.2 Example

An example of this command is:

```
srvctl config service -db dbcrm -service sales

Service name: sales
Service is enabled
Cardinality: SINGLETON
Disconnect: true
Service role: PRIMARY
Management policy: automatic
DTP transaction: false
AQ HA notifications: false
Failover type: NONE
Failover method: NONE
TAF failover retries: 0
TAF failover delay: 0
Connection Load Balancing Goal: NONE
Runtime Load Balancing Goal: NONE
TAF policy specification: NONE
Edition: e2
```

2.5.3 disable

Disables a component, which suspends management of that component by Oracle Restart.

The `srvctl disable` command is intended to be used when a component must be repaired or shut down for maintenance, and should not be restarted automatically. When you disable a component:

- It is no longer automatically restarted.
- It is no longer automatically started through a dependency.
- It cannot be started with SRVCTL.

To perform `srvctl disable` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

- srvctl disable asm
  Disables the Oracle ASM instance.
- srvctl disable database
  Disables the specified database.
- srvctl disable diskgroup
  Disables an Oracle ASM disk group.
- srvctl disable listener
  Disables the specified listener or all listeners.
- srvctl disable ons
  Disables Oracle Notification Services (ONS).
- srvctl disable service
  Disables one or more database services.

See Also:

The enable command

2.5.3.1 srvctl disable asm

Disables the Oracle ASM instance.

- Syntax and Options

2.5.3.1.1 Syntax and Options

Use the `srvctl disable asm` command with the following syntax:

```
srvctl disable asm
```

2.5.3.2 srvctl disable database

Disables the specified database.

- Syntax and Options
- Example

2.5.3.2.1 Syntax and Options

Use the `srvctl disable database` command with the following syntax:

```
srvctl disable database -db db_unique_name
```

Table 2-17   srvctl disable database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

2.5.3.2.2 Example

An example of this command is:

```
srvctl disable database -db dbcrm
```

2.5.3.3 srvctl disable diskgroup

Disables an Oracle ASM disk group.

- Syntax and Options
- Example

2.5.3.3.1 Syntax and Options

Use the `srvctl disable diskgroup` command with the following syntax:

```
srvctl disable diskgroup -diskgroup diskgroup_name
```

Table 2-18   srvctl disable diskgroup Options

Option	Description
`-diskgroup` *`diskgroup_name`*	Disk group name

2.5.3.3.2 Example

An example of this command is:

```
srvctl disable diskgroup -diskgroup DATA
```

2.5.3.4 srvctl disable listener

Disables the specified listener or all listeners.

- Syntax and Options
- Example

2.5.3.4.1 Syntax and Options

Use the `srvctl disable listener` command with the following syntax:

```
srvctl disable listener [-listener listener_name]
```

Table 2-19 srvctl disable listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, all listeners are disabled.

2.5.3.4.2 Example

An example of this command is:

```
srvctl disable listener -listener crmlistener
```

2.5.3.5 srvctl disable ons

Disables Oracle Notification Services (ONS).

- Syntax and Options

2.5.3.5.1 Syntax and Options

Use the `srvctl disable ons` command with the following syntax:

```
srvctl disable ons [-verbose]
```

Table 2-20 srvctl disable ons Options

Option	Description
`-verbose`	Verbose output

2.5.3.6 srvctl disable service

Disables one or more database services.

- Syntax and Options
- Example

2.5.3.6.1 Syntax and Options

Use the `srvctl disable service` command with the following syntax:

```
srvctl disable service -db db_unique_name -service service_name_list
  [-global_override
```

Table 2-21 srvctl disable service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

Table 2-21 (Cont.) srvctl disable service Options

Option	Description
`-service` *`service_name_list`*	Comma-delimited list of database service names
`-global_override`	If the service is a Global Data Services (GDS) service, then this option must be specified to disable the service. An error is returned if you attempt to disable a GDS service and –`global_override` is not included. This option is ignored if the service is not a GDS service. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.

2.5.3.6.2 Example

The following example disables the database service `sales` and `mfg`:

```
srvctl disable service -db dbcrm -service sales,mfg
```

2.5.4 downgrade

The `srvctl downgrade` command downgrades the database configuration after you manually downgrade the database.

- srvctl downgrade database
  The `srvctl downgrade database` command downgrades the configuration of a database and its services from its current version to the specified lower version.

2.5.4.1 srvctl downgrade database

The `srvctl downgrade database` command downgrades the configuration of a database and its services from its current version to the specified lower version.

- Syntax and Options

2.5.4.1.1 Syntax and Options

Use the `srvctl downgrade database` command with the following syntax:

```
srvctl downgrade database -db db_unique_name -oraclehome oracle_home
       -targetversion to_version
```

Table 2-22 srvctl downgrade database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-oraclehome` *`oracle_home`*	The full path of Oracle home for the database

Table 2-22 (Cont.) srvctl downgrade database Options

Option	Description
`-targetversion` *`to_version`*	The version to which to downgrade

2.5.5 enable

The `srvctl enable` command reenables the specified disabled component.

When you enable a component:

- Oracle Restart can automatically restart it.
- It can be automatically started through a dependency.
- You can start it manually with SRVCTL.

If the component is already enabled, then the command is ignored.

When you add a component to the Oracle Restart configuration, it is enabled by default.

To perform `srvctl enable` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

- srvctl enable asm
  Enables an Oracle ASM instance.
- srvctl enable database
  Enables the specified database.
- srvctl enable diskgroup
  Enables an Oracle ASM disk group.
- srvctl enable listener
  Enables the specified listener or all listeners.
- srvctl enable ons
  Enables Oracle Notification Services (ONS).
- srvctl enable service
  Enables one or more database services for the specified database.

See Also:

The disable command

2.5.5.1 srvctl enable asm

Enables an Oracle ASM instance.

- Syntax and Options

2.5.5.1.1 Syntax and Options

Use the `srvctl enable asm` command with the following syntax:

```
srvctl enable asm
```

2.5.5.2 srvctl enable database

Enables the specified database.

- Syntax and Options
- Example

2.5.5.2.1 Syntax and Options

Use the `srvctl enable database` command with the following syntax:

```
srvctl enable database -db db_unique_name
```

Table 2-23 srvctl enable database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

2.5.5.2.2 Example

An example of this command is:

```
srvctl enable database -db dbcrm
```

2.5.5.3 srvctl enable diskgroup

Enables an Oracle ASM disk group.

- Syntax and Options
- Example

2.5.5.3.1 Syntax and Options

Use the `srvctl enable diskgroup` command with the following syntax:

```
srvctl enable diskgroup -diskgroup diskgroup_name
```

Table 2-24 srvctl enable diskgroup Options

Option	Description
`-diskgroup` *`diskgroup_name`*	Disk group name

2.5.5.3.2 Example

An example of this command is:

```
srvctl enable diskgroup -diskgroup DATA
```

2.5.5.4 srvctl enable listener

Enables the specified listener or all listeners.

- Syntax and Options
- Example

2.5.5.4.1 Syntax and Options

Use the `srvctl enable listener` command with the following syntax:

```
srvctl enable listener [-listener listener_name]
```

Table 2-25 srvctl enable listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, all listeners are enabled.

2.5.5.4.2 Example

An example of this command is:

```
srvctl enable listener -listener crmlistener
```

2.5.5.5 srvctl enable ons

Enables Oracle Notification Services (ONS).

- Syntax and Options

2.5.5.5.1 Syntax and Options

Use the `srvctl enable ons` command with the following syntax:

```
srvctl enable ons [-verbose]
```

Table 2-26 srvctl enable ons Options

Option	Description
`-verbose`	Verbose output

2.5.5.6 srvctl enable service

Enables one or more database services for the specified database.

- Syntax and Options
- Example

2.5.5.6.1 Syntax and Options

Use the `srvctl enable service` command with the following syntax:

```
srvctl enable service -db db_unique_name -service service_name_list
  [-global_override]
```

Table 2-27 srvctl enable service Options

Option	Description
`-db db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service service_name_list`	Comma-delimited list of database service names
`-global_override`	If the service is a Global Data Services (GDS) service, then this option must be specified to enable the service. An error is returned if you attempt to enable a GDS service and `-global_override` is not included. This option is ignored if the service is not a GDS service. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.

2.5.5.6.2 Example

The following example enables the database services `sales` and `mfg` in the database with `DB_UNIQUE_NAME dbcrm`:

```
srvctl enable service -db dbcrm -service "sales,mfg"
```

2.5.6 getenv

Gets and displays environment variables and their values from the Oracle Restart configuration for a database, listener, or Oracle ASM instance.

- srvctl getenv asm
  Displays the configured environment variables for the Oracle ASM instance.
- srvctl getenv database
  Displays the configured environment variables for the specified database.
- srvctl getenv listener
  Displays the configured environment variables for the specified listener.

See Also:

- setenv command
- unsetenv command
- "Managing Environment Variables in the Oracle Restart Configuration"

2.5.6.1 srvctl getenv asm

Displays the configured environment variables for the Oracle ASM instance.

- Syntax and Options
- Example

2.5.6.1.1 Syntax and Options

Use the `srvctl getenv asm` command with the following syntax:

```
srvctl getenv asm [-envs name_list]
```

Table 2-28 srvctl getenv asm Options

Options	Description
`-envs name_list`	Comma-delimited list of names of environment variables to display. If omitted, SRVCTL displays all configured environment variables for Oracle ASM.

2.5.6.1.2 Example

The following example displays all configured environment variables for the Oracle ASM instance:

```
srvctl getenv asm
```

2.5.6.2 srvctl getenv database

Displays the configured environment variables for the specified database.

- Syntax and Options
- Example

2.5.6.2.1 Syntax and Options

Use the `srvctl getenv database` command with the following syntax:

```
srvctl getenv database -db db_unique_name [-envs name_list]
```

Table 2-29 srvctl getenv database Options

Options	Description
`-db db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-envs name_list`	Comma-delimited list of names of environment variables to display. If omitted, SRVCTL displays all configured environment variables.

2.5.6.2.2 Example

The following example displays all configured environment variables for the database with `DB_UNIQUE_NAME` `dbcrm`:

```
srvctl getenv database -db dbcrm
```

2.5.6.3 srvctl getenv listener

Displays the configured environment variables for the specified listener.

- Syntax and Options
- Example

2.5.6.3.1 Syntax and Options

Use the `srvctl getenv listener` command with the following syntax:

```
srvctl getenv listener [-listener listener_name] [-envs name_list]
```

Table 2-30 srvctl getenv listener Options

Options	Description
`-listener` *`listener_name`*	Listener name. If omitted, SRVCTL lists environment variables for all listeners.
`-envs` *`name_list`*	Comma-delimited list of names of environment variables to display. If omitted, SRVCTL displays all configured environment variables.

2.5.6.3.2 Example

The following example displays all configured environment variables for the listener named `crmlistener`:

```
srvctl getenv listener -listener crmlistener
```

2.5.7 modify

Modifies the Oracle Restart configuration of a component. The change takes effect when the component is next restarted.

To perform `srvctl modify` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

- srvctl modify asm
  Modifies the Oracle Restart configuration for the Oracle ASM instance.
- srvctl modify database
  Modifies the Oracle Restart configuration for a database.
- srvctl modify listener
  Modifies the Oracle Restart configuration for the specified listener or all listeners.
- srvctl modify ons
  Modifies Oracle Notification Services (ONS).

- srvctl modify service
  Modifies the Oracle Restart configuration of a database service.

2.5.7.1 srvctl modify asm

Modifies the Oracle Restart configuration for the Oracle ASM instance.

- Syntax and Options
- Example

2.5.7.1.1 Syntax and Options

Use the `srvctl modify asm` command with the following syntax:

```
srvctl modify asm [-listener listener_name] [-spfile spfile]
  [-pwfile password_file_path] [-diskstring asm_diskstring]
```

Table 2-31 srvctl modify asm Options

Option	Description
`-listener` *`listener_name`*	Name of the listener with which Oracle ASM must register. A weak dependency is established with this listener. (Before Oracle ASM is started, Oracle Restart ensures that this listener is started.)
`-spfile` *`spfile`*	The full path of the server parameter file for the database. If omitted, the default SPFILE is used.
`-pwfile` *`password_file_path`*	The full path of the Oracle ASM password file.
`-diskstring` *`asm_diskstring`*	Oracle ASM disk group discovery string. An Oracle ASM discovery string is a comma-delimited list of strings that limits the set of disks that an Oracle ASM instance discovers. The discovery strings can include wildcard characters. Only disks that match one of the strings are discovered.

2.5.7.1.2 Example

An example of this command is:

```
srvctl modify asm -listener crmlistener
```

See Also:

Oracle Automatic Storage Management Administrator's Guide for more information about Oracle ASM disk group discovery strings

2.5.7.2 srvctl modify database

Modifies the Oracle Restart configuration for a database.

- Syntax and Options
- Example

2.5.7.2.1 Syntax and Options

Use the `srvctl modify database` command with the following syntax:

```
srvctl modify database -db db_unique_name [-oraclehome oracle_home]
  [-user oracle_user] [-domain domain_name] [-dbname db_name]
  [-instance instance_name] [-instance instance_name] [-spfile spfile]
  [-pwfile password_file_path] [-startoption start_options]
  [-stopoption stop_options]
  [-role {PRIMARY | PHYSICAL_STANDBY | LOGICAL_STANDBY | SNAPSHOT_STANDBY}]
  [-policy {AUTOMATIC | MANUAL | NORESTART}]
  [{-diskgroup "diskgroup_list" | -nodiskgroup}] [-force]
```

Table 2-32 srvctl modify database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-user` *`oracle_user`*	Name of the Oracle user who owns the Oracle home directory
`-diskgroup` *`disk_group_list`*	Comma separated list of disk groups upon which the database is dependent. When starting the database, Oracle Restart first ensures that these disk groups are mounted. This option is required only if the database instance and the Oracle ASM instance are not started when adding the database. Otherwise, the dependency is recorded automatically between the database and its disk groups.
`-nodiskgroup`	Remove the database's dependency on Oracle ASM disk groups
`-force`	Force the operation even though the some resources might be stopped.
`(Other options)`	See Table 2-9

2.5.7.2.2 Example

The following example changes the role of the database with `DB_UNIQUE_NAME dbcrm` to `LOGICAL_STANDBY`:

```
srvctl modify database -db dbcrm -role logical_standby
```

See Also:

- "Oracle Restart Integration with Oracle Data Guard"
- *Oracle Data Guard Concepts and Administration*

2.5.7.3 srvctl modify listener

Modifies the Oracle Restart configuration for the specified listener or all listeners.

- Syntax and Options
- Example

2.5.7.3.1 Syntax and Options

Use the `srvctl modify listener` command with the following syntax:

```
srvctl modify listener [-listener listener_name] [-endpoints endpoints]
  [-oraclehome oracle_home]
```

Table 2-33 srvctl modify listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, all listener configurations are modified.
`-endpoints` *`endpoints`*	Comma separated TCP ports or listener endpoints. *endpoints* syntax is: `"[TCP:]port[, ...] [/IPC:key] [/NMP:pipe_name] [/TCPS:s_port] [/SDP:port]"`
`-oraclehome` *`oracle_home`*	New Oracle home for the listener

2.5.7.3.2 Example

This example modifies the TCP port on which the listener named `crmlistener` listens:

```
srvctl modify listener -listener crmlistener -endpoints TCP:1522
```

2.5.7.4 srvctl modify ons

Modifies Oracle Notification Services (ONS).

- Syntax and Options

2.5.7.4.1 Syntax and Options

Use the `srvctl modify ons` command with the following syntax:

```
srvctl modify ons [-emport em_port] [-onslocalport ons_local_port]
  [-onsremoteport ons_remote_port] [-remoteservers host[:port],[host[:port]...]]
  [-verbose]
```

Table 2-34 srvctl modify ons Options

Option	Description
`-emport` *`em_port`*	ONS listening port for Cloud Control. The default is 2016.
`-onslocalport` *`ons_local_port`*	ONS listening port for local client connections
`-onsremoteport` *`ons_remote_port`*	ONS listening port for connections from remote hosts
`-remoteservers host[:port],[host[:port],...`	A list of *`host:port`* pairs of remote hosts that are part of the ONS network **Note:** If *`port`* is not specified for a remote host, then `ons_remote_port` is used.
`-verbose`	Verbose output

2.5.7.5 srvctl modify service

Modifies the Oracle Restart configuration of a database service.

> **Note:**
>
> Oracle recommends that you limit configuration changes to the minimum requirement and that you not perform other service operations while the online service modification is in progress.

- Syntax and Options
- Example

2.5.7.5.1 Syntax and Options

Use the `srvctl modify service` command with the following syntax:

```
srvctl modify service -db db_unique_name -service service_name
  [-role [PRIMARY][,PHYSICAL_STANDBY][,LOGICAL_STANDBY][,SNAPSHOT_STANDBY]]
  [-policy {AUTOMATIC | MANUAL}]
  [-failovertype {NONE | SESSION | SELECT | TRANSACTION}]
  [-failovermethod {NONE | BASIC}] [-failoverdelay integer]
  [-failoverretry integer] [-clbgoal {SHORT | LONG}]
  [-rlbgoal {SERVICE_TIME | THROUGHPUT | NONE}] [-notification {TRUE | FALSE}]
  [-edition edition_name] [-pdb pluggable_database]
  [-sql_translation_profile sql_translation_profile]
  [-commit_outcome {TRUE | FALSE}] [-retention retention]
  [-replay_init_time replay_init_time] [-drain_timeout timeout]
  [-stopoption stop_option] [-session_state {STATIC | DYNAMIC}]
  [-global_override] [-verbose]
```

Table 2-35 srvctl modify service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database The name must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name`*	Service name
`-role [PRIMARY] [,PHYSICAL_STANDBY] [,LOGICAL_STANDBY] [,SNAPSHOT_STANDBY]`	A list of service roles This option is applicable in Oracle Data Guard environments only. When this option is present, upon database startup, the service is started only when one of its service roles matches the current database role. **See Also:** *Oracle Data Guard Concepts and Administration* for more information about database roles

Table 2-35 (Cont.) srvctl modify service Options

Option	Description
`-policy {AUTOMATIC | MANUAL}`	Management policy for the service If `AUTOMATIC` (the default), the service is automatically started upon restart of the database, either by a planned restart (with SRVCTL) or after a failure. Automatic restart is also subject to the service role, however (the `-role` option). If `MANUAL`, the service is never automatically restarted upon planned restart of the database (with SRVCTL). A `MANUAL` setting does not prevent Oracle Restart from monitoring the service when it is running and restarting it if a failure occurs.
`-failovertype {NONE |SESSION | SELECT | TRANSACTION}`	To enable Application Continuity for OCI and Java, use `TRANSACTION`. If the failover type is `TRANSACTION`, then OCI and Java attempt to recover the inflight transaction upon receipt of a recoverable error. When failover type is `TRANSACTION`, the `commit_outcome` option must be set to `TRUE`. To enable Transparent Application Failover (TAF) for OCI, use `SELECT` or `SESSION`.
`-failovermethod {NONE | BASIC}`	TAF failover method for backward compatibility only If the failover type (`-failovertype`) is set to a value other than `NONE`, then use `BASIC` for this option.
`-failoverdelay` *`integer`*	For Application Continuity and TAF, the time delay, in seconds, between reconnect attempts for each incident at failover
`-failoverretry` *`integer`*	For Application Continuity and TAF, the number of attempts to connect after an incident
`-clbgoal {SHORT | LONG}`	Connection load balancing goal Use `SHORT` for run-time load balancing. Use `LONG` for long running connections, such as batch jobs.
`-rlbgoal {SERVICE_TIME | THROUGHPUT | NONE}`	Run-time load balancing goal Use `SERVICE_TIME` to balance connections by response time. Use `THROUGHPUT` to balance connections by throughput.
`-notification {TRUE | FALSE}`	Enable Fast Application Notification (FAN) for OCI connections
`-edition` *`edition_name`*	The initial session edition of the service If this option is not specified, then the edition is not modified for the service. If this option is specified but *`edition_name`* is empty, then the edition is set to `NULL`. A `NULL` edition has no effect. When an edition is specified for a service, all subsequent connections that specify the service use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the initial session edition. SRVCTL does not validate the specified edition name. During connection, the connect user must have `USE` privilege on the specified edition. If the edition does not exist or if the connect user does not have `USE` privilege on the specified edition, then an error is raised.
`-pdb` *`pluggable_database`*	In a CDB, the name of the PDB to associate with the service If this option is set to an empty string, then the service is associated with root.

Table 2-35 (Cont.) srvctl modify service Options

Option	Description
`-sql_translation_profile` *`sql_translation_profile`*	A SQL translation profile for a service that you are adding after you have migrated applications from a non-Oracle database to an Oracle database **Note:** Before using the SQL translation framework, you must migrate all server-side application objects and data to the Oracle database. **See Also:** *Oracle Database SQL Translation and Migration Guide* for more information about using a SQL translation profile
`-commit_outcome {TRUE | FALSE}`	For Transaction Guard, when `TRUE` a transaction's commit outcome is accessible after the transaction's session fails due to a recoverable outage. If `FALSE`, the default, then a transaction's commit outcome is not retained. When this option is set to `TRUE`, the outcome of a transaction's commit is durable, and an applications can determine the commit status of a transaction after an outage. You can set `commit_outcome` to `TRUE` for a user-defined service. The `commit_outcome` setting has no effect on Oracle Active Data Guard and read-only databases. **See Also:** See *Oracle Database Development Guide* for more information.
`-retention` *`retention`*	If `commit_outcome` is set to `TRUE`, then this option determines the amount of time, in seconds, that the commit outcome is retained. The default is 24 hours (86400). If `commit_outcome` is set to `FALSE`, then this option cannot be set.
`-replay_init_time` *`replay_init_time`*	For Application Continuity, this option specifies the difference between the time, in seconds, of original execution of the first operation of a request and the time that the replay is ready to start after a successful reconnect. Application Continuity will not replay after the specified amount of time has passed. This option is intended to avoid the unintentional execution of a transaction when a system is recovered after a long period of time. The default is 5 minutes (300). The maximum value is 24 hours (86400). If `failovertype` is not set to `TRANSACTION`, then this option is not used.
`-drain_timeout` *`timeout`*	This option specifies the time allowed for resource draining to be completed in seconds. Permitted values are `NULL`, `0`, or any positive integer. The draining period is intended for planned maintenance operations. During the draining period, all current client requests are processed, but new requests are not accepted. How draining works depends on the setting of the `-stopoption` option. The default value is `NULL`, which means that this option is not set. If the option is not set, and `-drain_timeout` has been set on the service, then this value is used. If it is set to `0`, then draining does not occur.
`-stopoption` *`stop_option`*	This option specifies the mode in which the service is stopped. The following values are permitted: • `IMMEDIATE` specifies that sessions are permitted to drain before the service is stopped. • `TRANSACTIONAL` specifies that sessions are permitted to drain for the amount of time specified in the `–drain_timeout` option. The service is stopped when the time limit is reached, and any remaining sessions are terminated. • `NONE` is the default.

Table 2-35 (Cont.) srvctl modify service Options

Option	Description
`-session_state {STATIC | DYNAMIC}`	For Application Continuity, this parameter specifies whether the session state that is not transactional is changed by the application. Oracle recommends a setting of `DYNAMIC` for most applications. **Note:** This parameter is considered only if `-failovertype` is set to `TRANSACTION` for Application Continuity. It describes how non-transactional is changed during a request. Examples of session state are NLS settings, optimizer preferences, event settings, PL/SQL global variables, temporary tables, advanced queues, LOBs, and result cache. If non-transactional values change after the request starts, then use the default, `DYNAMIC`. Most applications should use `DYNAMIC` mode. If you are unsure, then use `DYNAMIC` mode.
`-global_override`	If the service is a Global Data Services (GDS) service, then this option must be specified to modify any of the following service attributes: • `-role` • `-policy` • `-failovertype` • `-failovermethod` • `-failoverdelay` • `-failoverretry` • `-edition` • `-clbgoal` • `-rlbgoal` • `-notification` An error is returned if you attempt to modify one of these options for a GDS service and `-global_override` is not included. This option is ignored if the service is not a GDS service. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.
`-verbose`	Verbose output

2.5.7.5.2 Example

For the database with a `DB_UNIQUE_NAME` of `dbcrm`, the following command changes the Oracle Data Guard role of the database service named `support` to `standby`:

```
srvctl modify service -db dbcrm -service support -role standby
```

See Also:

Oracle Multitenant Administrator's Guide for information about managing services associated with PDBs

2.5.8 remove

Removes the specified component from the Oracle Restart configuration. Oracle Restart no longer manages the component. Any environment variable settings for the component are also removed.

Before you remove a component from the Oracle Restart configuration, you must use SRVCTL to stop it. Oracle recommends that you disable the component before removing it, but this is not required.

To perform `srvctl remove` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

- srvctl remove asm
  Removes an Oracle ASM instance.
- srvctl remove database
  Removes a database. Prompts for confirmation first.
- srvctl remove diskgroup
  Removes an Oracle ASM disk group.
- srvctl remove listener
  Removes the specified listener or all listeners.
- srvctl remove ons
  Removes Oracle Notification Services (ONS).
- srvctl remove service
  Removes the specified database service.

See Also:

- stop command
- disable command

2.5.8.1 srvctl remove asm

Removes an Oracle ASM instance.

- Syntax and Options
- Example

2.5.8.1.1 Syntax and Options

Use the `srvctl remove asm` command with the following syntax:

```
srvctl remove asm [-force]
```

Table 2-36 srvctl remove asm Options

Options	Description
`-force`	Force remove, even when disk groups and databases that use Oracle ASM exist or when the Oracle ASM instance is running.

2.5.8.1.2 Example

An example of this command is:

```
srvctl remove asm
```

2.5.8.2 srvctl remove database

Removes a database. Prompts for confirmation first.

- Syntax and Options
- Example

2.5.8.2.1 Syntax and Options

Use the `srvctl remove database` command with the following syntax:

> **Note:**
>
> After running this command, ensure that the password file is in the default location if you want to connect to the database as the `SYS` user with the `SYS` user's password.

```
srvctl remove database -db db_unique_name [-force] [-noprompt] [-verbose]
```

Table 2-37 srvctl remove database Options

Options	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-force`	Force. Removes the database even if it is running.
`-noprompt`	Suppresses the confirmation prompt and removes immediately
`-verbose`	Verbose output. A success or failure message is displayed.

2.5.8.2.2 Example

An example of this command is:

```
srvctl remove database -db dbcrm
```

2.5.8.3 srvctl remove diskgroup

Removes an Oracle ASM disk group.

- Syntax and Options
- Example

2.5.8.3.1 Syntax and Options

Use the `srvctl remove diskgroup` command with the following syntax:

```
srvctl remove diskgroup -diskgroup diskgroup_name [-force]
```

Table 2-38 srvctl remove diskgroup Options

Option	Description
`-diskgroup` *`diskgroup_name`*	Disk group name
`-force`	Force. Removes the disk group even if files are open on it.

2.5.8.3.2 Example

This example removes the disk group named `DATA`. An error is returned if files are open on this disk group.

```
srvctl remove diskgroup -diskgroup DATA
```

2.5.8.4 srvctl remove listener

Removes the specified listener or all listeners.

- Syntax and Options
- Example

2.5.8.4.1 Syntax and Options

Use the `srvctl remove listener` command with the following syntax:

```
srvctl remove listener [-listener listener_name | -all] [-force]
```

Table 2-39 srvctl remove listener Options

Options	Description
`-listener` *`listener_name`*	Name of the listener that you want to remove. If omitted, then the default is `LISTENER`.
`-all`	Remove all listeners
`-force`	Force. Removes the listener even if databases are using it.

2.5.8.4.2 Example

The following command removes the listener `lsnr01`:

```
srvctl remove listener -listener lsnr01
```

2.5.8.5 srvctl remove ons

Removes Oracle Notification Services (ONS).

- Syntax and Options

2.5.8.5.1 Syntax and Options

Use the `srvctl remove ons` command as follows:

```
srvctl remove ons [-force] [-verbose]
```

Table 2-40 srvctl remove ons Options

Options	Description
`-force`	Force. Removes ONS even if it is enabled.
`-verbose`	Verbose output

2.5.8.6 srvctl remove service

Removes the specified database service.

- Syntax and Options
- Example

2.5.8.6.1 Syntax and Options

Use the `srvctl remove service` command as follows:

```
srvctl remove service -db db_unique_name -service service_name [-global_override]
```

Table 2-41 srvctl remove service Options

Options	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name`*	Service name

Table 2-41 (Cont.) srvctl remove service Options

Options	Description
`-global_override`	If the service is a Global Data Services (GDS) service, then this option must be specified to remove the service. An error is returned if you attempt to remove a GDS service and `-global_override` is not included. This option is ignored if the service is not a GDS service. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.

2.5.8.6.2 Example

An example of this command is:

```
srvctl remove service -db dbcrm -service sales
```

2.5.9 setenv

The `setenv` command sets values of environment variables in the Oracle Restart configuration for a database, a listener, or the Oracle ASM instance.

To perform `srvctl setenv` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

- srvctl setenv asm
  Sets the values of environment variables in the Oracle Restart configuration for the Oracle ASM instance. Before starting the instance, Oracle Restart sets environment variables to the values stored in the configuration.
- srvctl setenv database
  Sets the values of environment variables in the Oracle Restart configuration for a database instance. Before starting the instance, Oracle Restart sets environment variables to the values stored in the configuration.
- srvctl setenv listener
  Sets the values of environment variables in the Oracle Restart configuration for a listener. Before starting the listener, Oracle Restart sets environment variables to the values stored in the configuration.

See Also:

- getenv command
- unsetenv command
- "Managing Environment Variables in the Oracle Restart Configuration"

2.5.9.1 srvctl setenv asm

Sets the values of environment variables in the Oracle Restart configuration for the Oracle ASM instance. Before starting the instance, Oracle Restart sets environment variables to the values stored in the configuration.

- Syntax and Options
- Example

2.5.9.1.1 Syntax and Options

Use the `srvctl setenv asm` command with the following syntax:

```
srvctl setenv asm {-envs name=val[,name=val,...] | -env name=val}
```

Table 2-42 srvctl setenv database Options

Options	Description
`-envs name=val[,name=val,...]`	Comma-delimited list of name/value pairs of environment variables
`-env name=val`	Enables single environment variable to be set to a value that contains commas or other special characters

2.5.9.1.2 Example

The following example sets the AIX operating system environment variable `AIXTHREAD_SCOPE` in the Oracle ASM instance configuration:

```
srvctl setenv asm -envs AIXTHREAD_SCOPE=S
```

2.5.9.2 srvctl setenv database

Sets the values of environment variables in the Oracle Restart configuration for a database instance. Before starting the instance, Oracle Restart sets environment variables to the values stored in the configuration.

- Syntax and Options
- Example

2.5.9.2.1 Syntax and Options

Use the `srvctl setenv database` command with the following syntax:

```
srvctl setenv database -db db_unique_name
  {-envs name=val[,name=val,...] | -env name=val}
```

Table 2-43 srvctl setenv database Options

Options	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-envs` *`name=val`*`[,`*`name=val`*`,...]`	Comma-delimited list of name/value pairs of environment variables
`-env` *`name=val`*	Enables single environment variable to be set to a value that contains commas or other special characters

2.5.9.2.2 Example

The following example sets the `LANG` environment variable in the configuration of the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl setenv database -db dbcrm -envs LANG=en
```

2.5.9.3 srvctl setenv listener

Sets the values of environment variables in the Oracle Restart configuration for a listener. Before starting the listener, Oracle Restart sets environment variables to the values stored in the configuration.

- Syntax and Options
- Example

2.5.9.3.1 Syntax and Options

Use the `srvctl setenv listener` command with the following syntax:

```
srvctl setenv listener [-listener listener_name]
  {-envs name=val[,name=val,...] | -env name=val}
```

Table 2-44 srvctl setenv listener Options

Options	Description
`-listener` *`listener_name`*	Listener name. If omitted, sets the specified environment variables in all listener configurations.
`-envs` *`name=val`*`[,`*`name=val`*`,...]`	Comma-delimited list of name/value pairs of environment variables
`-env` *`name=val`*	Enables single environment variable to be set to a value that contains commas or other special characters

2.5.9.3.2 Example

The following example sets the AIX operating system environment variable `AIXTHREAD_SCOPE` in the configuration of the listener named `crmlistener`:

```
srvctl setenv listener -listener crmlistener -envs AIXTHREAD_SCOPE=S
```

2.5.10 start

Starts the specified component or components.

- srvctl start asm
  Starts the Oracle ASM instance.
- srvctl start database
  Starts the specified database instance.
- srvctl start diskgroup
  Starts (mounts) an Oracle ASM disk group.
- srvctl start home
  Starts all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.
- srvctl start listener
  Starts the specified listener or all listeners.
- srvctl start ons
  Starts Oracle Notification Services (ONS).
- srvctl start service
  Starts the specified database service or services.

See Also:

"Starting and Stopping Components Managed by Oracle Restart"

2.5.10.1 srvctl start asm

Starts the Oracle ASM instance.

For this command, SRVCTL connects "/ as sysasm" to perform the operation. To run such operations, the owner of the executables in the Oracle Grid Infrastructure home must be a member of the OSASM group, and users running the commands must also be in the OSASM group.

- Syntax and Options
- Example

2.5.10.1.1 Syntax and Options

Use the `srvctl start asm` command with the following syntax:

```
srvctl start asm [-startoption start_options]
```

Table 2-45 srvctl start asm Option

Option	Description
`-startoption` *`start_options`*	Comma-delimited list of options for the startup command (`OPEN`, `MOUNT`, `NOMOUNT`, or `FORCE`). If omitted, defaults to normal startup (`OPEN`). **See Also:** *SQL*Plus User's Guide and Reference* for more information about startup options

2.5.10.1.2 Example

This example starts the Oracle ASM instance, which then mounts any disk groups named in the `ASM_DISKGROUPS` initialization parameter:

```
srvctl start asm
```

This example starts the Oracle ASM instance without mounting any disk groups:

```
srvctl start asm -startoption nomount
```

2.5.10.2 srvctl start database

Starts the specified database instance.

For this command, SRVCTL connects "/ as sysdba" to perform the operation. To run such operations, the owner of the Oracle executables in the database Oracle home must be a member of the OSDBA group (for example, the `dba` group on UNIX and Linux), and users running the commands must also be in the OSDBA group.

- Syntax and Options
- Example

2.5.10.2.1 Syntax and Options

Use the `srvctl start database` command with the following syntax:

```
srvctl start database -db db_unique_name [-startoption start_options] [-verbose]
```

Table 2-46 srvctl start database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

Table 2-46 (Cont.) srvctl start database Options

Option	Description
`-startoption` *`start_options`*	Comma-delimited list of options for the startup command (for example: `OPEN`, `MOUNT`, `NOMOUNT`, `RESTRICT`, and so on) **Notes:** • This command parameter does not support the `PFILE` option or the `QUIET` option, but it supports all other database startup options. • For multi-word startup options, such as `read only` and `read write`, separate the words with a space and enclose in single quotation marks (`''`). For example, `'read only'`. **See Also:** *SQL*Plus User's Guide and Reference* for more information about startup options
`-verbose`	Verbose output

2.5.10.2.2 Example

An example of this command is:

```
srvctl start database -db dbcrm -startoption nomount
```

2.5.10.3 srvctl start diskgroup

Starts (mounts) an Oracle ASM disk group.

- Syntax and Options
- Example

2.5.10.3.1 Syntax and Options

Use the `srvctl start diskgroup` command with the following syntax:

```
srvctl start diskgroup -diskgroup diskgroup_name
```

Table 2-47 srvctl start diskgroup Options

Option	Description
`-diskgroup` *`diskgroup_name`*	Disk group name

2.5.10.3.2 Example

An example of this command is:

```
srvctl start diskgroup -diskgroup DATA
```

2.5.10.4 srvctl start home

Starts all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

This command starts the components that were stopped by a `srvctl stop home`. This command uses the information in the specified state file to identify the components to start.

Note:

Use this command to restart components after you install a patch in an Oracle home.

- Syntax and Options

2.5.10.4.1 Syntax and Options

Use the `srvctl start home` command with the following syntax:

```
srvctl start home -oraclehome oracle_home -statefile state_file
```

Table 2-48 srvctl start home Options

Option	Description
`-oraclehome` *`oracle_home`*	Complete path of the Oracle home
`-statefile` *`state_file`*	Complete path of the state file. The state file contains the current state information for the components in the Oracle home and is created when the `srvctl stop home` command or the `srvctl status home` command is run.

2.5.10.5 srvctl start listener

Starts the specified listener or all listeners.

- Syntax and Options
- Example

2.5.10.5.1 Syntax and Options

Use the `srvctl start listener` command with the following syntax:

```
srvctl start listener [-listener listener_name]
```

Table 2-49 srvctl start listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, all Oracle Restart–managed listeners are started.

2.5.10.5.2 Example

An example of this command is:

```
srvctl start listener -listener listener
```

2.5.10.6 srvctl start ons

Starts Oracle Notification Services (ONS).

- Syntax and Options

2.5.10.6.1 Syntax and Options

Use the `srvctl start ons` command with the following syntax:

```
srvctl start ons [-verbose]
```

Table 2-50 srvctl start ons Options

Option	Description
`-verbose`	Verbose output

2.5.10.7 srvctl start service

Starts the specified database service or services.

- Syntax and Options
- Example

2.5.10.7.1 Syntax and Options

Use the `srvctl start service` command with the following syntax:

```
srvctl start service -db db_unique_name [-service service_name_list |
  -pdb pluggable_database] [-startoption start_options] [-global_override] [-verbose]
```

Table 2-51 srvctl start service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name_list`*	Comma-delimited list of service names. The service name list is optional and, if not provided, SRVCTL starts all of the database's services.
`-pdb` *`pluggable_database`*	In a CDB, the name of the PDB associated with the service If this option is set to an empty string, then the service is associated with root.
`-startoption` *`start_options`*	Options for database startup (for example: `OPEN`, `MOUNT`, `NOMOUNT` and so on) if the database must be started first **See Also:** *SQL*Plus User's Guide and Reference* for more information about startup options
`-global_override`	If the service is a Global Data Services (GDS) service, then this option must be specified to start the service. An error is returned if you attempt to start a GDS service and `–global_override` is not included. This option is ignored if the service is not a GDS service. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.
`-verbose`	Verbose output

2.5.10.7.2 Example

For the database with a `DB_UNIQUE_NAME` of `dbcrm`, the following example starts the `sales` database service:

```
srvctl start service -db dbcrm -service sales
```

2.5.11 status

Displays the running status of the specified component or set of components.

- srvctl status asm
  Displays the running status of the Oracle ASM instance.
- srvctl status database
  Displays the running status of the specified database.
- srvctl status diskgroup
  Displays the running status of an Oracle ASM disk group.
- srvctl status home
  Displays the running status of all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.
- srvctl status listener
  Displays the running status of the specified listener or of all Oracle Restart–managed listeners.
- srvctl status ons
  Displays the running status of Oracle Notification Services (ONS).
- srvctl status service
  Displays the running status of one or more database services.

2.5.11.1 srvctl status asm

Displays the running status of the Oracle ASM instance.

- Syntax and Options
- Example

2.5.11.1.1 Syntax and Options

Use the `srvctl status asm` command with the following syntax:

```
srvctl status asm [-all] [-verbose]
```

Table 2-52 srvctl status asm Options

Option	Description
`-all`	Display enabled/disabled status also
`-verbose`	Verbose output

2.5.11.1.2 Example

An example of this command is:

```
srvctl status asm

ASM is running on dbhost
```

2.5.11.2 srvctl status database

Displays the running status of the specified database.

- Syntax and Options
- Example

2.5.11.2.1 Syntax and Options

Use the `srvctl status database` command with the following syntax:

```
srvctl status database -db db_unique_name [-force] [-verbose]
```

Table 2-53 srvctl status database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-force`	Display a message if the database is disabled
`-verbose`	Verbose output. Lists the database services that are running.

2.5.11.2.2 Example

An example of this command is:

```
srvctl status database -db dbcrm -verbose

Database dbcrm is running with online services mfg,sales
```

2.5.11.3 srvctl status diskgroup

Displays the running status of an Oracle ASM disk group.

- Syntax and Options
- Example

2.5.11.3.1 Syntax and Options

Use the `srvctl status diskgroup` command with the following syntax:

```
srvctl status diskgroup -diskgroup diskgroup_name [-all] [-verbose]
```

Table 2-54 srvctl status diskgroup Options

Option	Description
`-diskgroup` *`diskgroup_name`*	Disk group name
`-all`	Display enabled/disabled status also
`-verbose`	Verbose output. Lists the database services that are running.

2.5.11.3.2 Example

An example of this command is:

```
srvctl status diskgroup -diskgroup DATA

Disk Group DATA is running on dbhost
```

2.5.11.4 srvctl status home

Displays the running status of all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

This command writes the current status of the components to the specified state file.

- Syntax and Options

2.5.11.4.1 Syntax and Options

Use the `srvctl status home` command with the following syntax:

```
srvctl status home -oraclehome oracle_home -statefile state_file
```

Table 2-55 srvctl status home Options

Option	Description
`-oraclehome` *`oracle_home`*	Complete path of the Oracle home
`-statefile` *`state_file`*	Complete path of the state file

2.5.11.5 srvctl status listener

Displays the running status of the specified listener or of all Oracle Restart–managed listeners.

- Syntax and Options
- Example

2.5.11.5.1 Syntax and Options

Use the `srvctl status listener` command with the following syntax:

```
srvctl status listener [-listener listener_name] [-verbose]
```

Table 2-56 srvctl status listener Options

Option	Description
`-listener listener_name`	Listener name. If omitted, the status of all listeners is displayed.
`-verbose`	Verbose output. Lists the database services that are running.

2.5.11.5.2 Example

An example of this command is:

```
srvctl status listener -listener crmlistener

Listener CRMLISTENER is running on dbhost
```

2.5.11.6 srvctl status ons

Displays the running status of Oracle Notification Services (ONS).

- Syntax and Options

2.5.11.6.1 Syntax and Options

Use the `srvctl status ons` command with the following syntax:

```
srvctl status ons [-verbose]
```

Table 2-57 srvctl status ons Options

Option	Description
`-verbose`	Verbose output. Lists the database services that are running.

2.5.11.7 srvctl status service

Displays the running status of one or more database services.

- Syntax and Options
- Example

2.5.11.7.1 Syntax and Options

Use the `srvctl status service` command with the following syntax:

```
srvctl status service -db db_unique_name
  [-service service_name_list | -pdb pluggable_database]
  [-force] [-verbose]
```

Table 2-58 srvctl status service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name_list`*	Comma-delimited list of service names. If omitted, status is listed for all database services for the designated database.
`-pdb` *`pluggable_database`*	In a multitenant container database (CDB), the name of the pluggable database (PDB) associated with the service If this option is set to an empty string, then the service is associated with root.
`-force`	Display a message if a service is disabled
`-verbose`	Verbose output

2.5.11.7.2 Example

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, the following example displays the running status of the service `sales`:

```
srvctl status service -db dbcrm -service sales

Service sales is running on dbhost
```

2.5.12 stop

Stops the specified component or components.

If you want a component to remain stopped after you issue a `srvctl stop` command, disable the component. See the disable command.

Note:

If a component is stopped and is not disabled, it could restart as a result of another planned operation. That is, although a stopped component will not restart as a result of a failure, it might be started if a dependent component is started with a `srvctl start` command.

- srvctl stop asm
  Stops the Oracle ASM instance.
- srvctl stop database
  Stops a database and its services.
- srvctl stop diskgroup
  Stops (dismounts) an Oracle ASM disk group.

- srvctl stop home
  Stops all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.
- srvctl stop listener
  Stops the designated listener or all Oracle Restart–managed listeners. Stopping a listener does not cause databases that are registered with the listener to be stopped.
- srvctl stop ons
  Stops Oracle Notification Services (ONS).
- srvctl stop service
  Stops one or more database services.

See Also:

"Starting and Stopping Components Managed by Oracle Restart"

2.5.12.1 srvctl stop asm

Stops the Oracle ASM instance.

- Syntax and Options
- Example

2.5.12.1.1 Syntax and Options

Use the `srvctl stop asm` command with the following syntax:

```
srvctl stop asm [-stopoption stop_options] [-force]
```

Table 2-59 srvctl stop asm Option

Option	Description
`-stopoption` *`stop_options`*	Options for the shutdown operation, for example, `NORMAL`, `TRANSACTIONAL`, `IMMEDIATE`, or `ABORT` **See Also:** *SQL*Plus User's Guide and Reference* for more information about shutdown options
`-force`	Force. Must be present if disk groups are currently started (mounted). This option enables SRVCTL to stop the disk groups before stopping Oracle ASM. Each dependent database instance is also stopped according to its stop options, or with the `ABORT` option if the configured stop options fail.

2.5.12.1.2 Example

An example of this command is:

```
srvctl stop asm -stopoption abort -force
```

2.5.12.2 srvctl stop database

Stops a database and its services.

- Syntax and Options
- Example

2.5.12.2.1 Syntax and Options

Use the `srvctl stop database` command with the following syntax:

```
srvctl stop database -db db_unique_name [-stopoption stop_options]
  [-drain_timeout timeout] [-force] [-verbose]
```

Table 2-60 srvctl stop database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-stopoption` *`stop_options`*	`SHUTDOWN` command options (for example: `NORMAL`, `TRANSACTIONAL`, `IMMEDIATE`, or `ABORT`). Default is `IMMEDIATE`.
`-drain_timeout` *`timeout`*	This option specifies the time allowed for resource draining to be completed in seconds. Permitted values are `NULL`, `0`, or any positive integer. The draining period is intended for planned maintenance operations. During the draining period, all current client requests are processed, but new requests are not accepted. How draining works depends on the setting of the `-stopoption` option. The default value is `NULL`, which means that this option is not set. If the option is not set, and `-drain_timeout` has been set on the service, then this value is used. If it is set to `0`, then draining does not occur.
`-force`	Stops the database, its services, and any resources that depend on the services
`-verbose`	Verbose output

2.5.12.2.2 Example

An example of this command is:

```
srvctl stop database -db dbcrm
```

2.5.12.3 srvctl stop diskgroup

Stops (dismounts) an Oracle ASM disk group.

- Syntax and Options
- Example

2.5.12.3.1 Syntax and Options

Use the `srvctl stop diskgroup` command with the following syntax:

```
srvctl stop diskgroup -diskgroup diskgroup_name [-force]
```

Table 2-61 srvctl stop diskgroup Options

Option	Description
`-diskgroup` *`diskgroup_name`*	Disk group name
`-force`	Force. Dismount the disk group even if some files in the disk group are open.

2.5.12.3.2 Example

This example stops the disk group named `DATA`. An error is returned if files are open on this disk group.

```
srvctl stop diskgroup -diskgroup DATA
```

2.5.12.4 srvctl stop home

Stops all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

This command identifies the components that it stopped in the specified state file.

Note:

- Before stopping the components in an Oracle Grid Infrastructure home, stop the components in a dependent Oracle Database home.
- Use this command to stop components before you install a patch in an Oracle home.

- Syntax and Options

2.5.12.4.1 Syntax and Options

Use the `srvctl stop home` command with the following syntax:

```
srvctl stop home -oraclehome oracle_home -statefile state_file
  [-stopoption stop_options] [-force]
```

Table 2-62 srvctl stop home Options

Option	Description
`-oraclehome` *`oracle_home`*	Complete path of the Oracle home
`-statefile` *`state_file`*	Complete path to where you want the state file to be written
`-stopoption` *`stop_options`*	`SHUTDOWN` command options for the database (for example: `NORMAL`, `TRANSACTIONAL`, `IMMEDIATE`, or `ABORT`). Default is `IMMEDIATE`. **See Also:** *SQL*Plus User's Guide and Reference* for more information about shutdown options

Table 2-62 (Cont.) srvctl stop home Options

Option	Description
`-force`	Force stop each component

2.5.12.5 srvctl stop listener

Stops the designated listener or all Oracle Restart–managed listeners. Stopping a listener does not cause databases that are registered with the listener to be stopped.

- Syntax and Options
- Example

2.5.12.5.1 Syntax and Options

Use the `srvctl stop listener` command with the following syntax:

```
srvctl stop listener [-listener listener_name] [-force]
```

Table 2-63 srvctl stop listener Options

Option	Description
`-listener` *`listener_name`*	Listener name. If omitted, all Oracle Restart–managed listeners are stopped.
`-force`	Force. Passes the stop command with the `-f` option to Oracle Clusterware. See *Oracle Clusterware Administration and Deployment Guide* for more information about the Oracle Clusterware `-f` option.

2.5.12.5.2 Example

An example of this command is:

```
srvctl stop listener -listener crmlistener
```

2.5.12.6 srvctl stop ons

Stops Oracle Notification Services (ONS).

- Syntax and Options

2.5.12.6.1 Syntax and Options

Use the `srvctl stop ons` command with the following syntax:

```
srvctl stop ons [-verbose]
```

Table 2-64 srvctl stop ons Options

Option	Description
`-verbose`	Verbose output

2.5.12.7 srvctl stop service

Stops one or more database services.

- Syntax and Options
- Example

2.5.12.7.1 Syntax and Options

Use the `srvctl stop service` command with the following syntax:

```
srvctl stop service -db db_unique_name [-service service_name_list |
  -pdb pluggable_database] [-drain_timeout timeout] [-stopoption stop_option]
  [-global_override] [-wait wait_option] [-force] [-verbose]
```

Table 2-65 srvctl stop service Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-service` *`service_name_list`*	Comma-delimited list of database service names. If you do not provide a service name list, then SRVCTL stops all services on the database
`-pdb` *`pluggable_database`*	In a CDB, the name of the PDB associated with the service If this option is set to an empty string, then the service is associated with root.
`-drain_timeout` *`timeout`*	This option specifies the time allowed for resource draining to be completed in seconds. Permitted values are `NULL`, `0`, or any positive integer. The draining period is intended for planned maintenance operations. During the draining period, all current client requests are processed, but new requests are not accepted. How draining works depends on the setting of the `-stopoption` option. The default value is `NULL`, which means that this option is not set. If the option is not set, and `-drain_timeout` has been set on the service, then this value is used. If it is set to `0`, then draining does not occur.
`-stopoption` *`stop_option`*	This option specifies the mode in which the service is stopped. The following values are permitted: • `IMMEDIATE` specifies that sessions are permitted to drain before the service is stopped. • `TRANSACTIONAL` specifies that sessions are permitted to drain for the amount of time specified in the `–drain_timeout` option. The service is stopped when the time limit is reached, and any remaining sessions are terminated. • `NONE` is the default.

Table 2-65 (Cont.) srvctl stop service Options

Option	Description
`-global_override`	If the service is a Global Data Services (GDS) service, then this option must be specified to stop the service. An error is returned if you attempt to stop a GDS service and `-global_override` is not included. This option is ignored if the service is not a GDS service. See *Oracle Database Global Data Services Concepts and Administration Guide* for more information.
`-wait` *`wait_option`*	This option specifies whether to wait until service draining is completed before stopping the service. Specify `YES` to wait or `NO` to stop the service without waiting.
`-force`	Force. This option disconnects all of the stopped services' sessions immediately. Uncommitted transactions are rolled back. If this option is omitted, active sessions remain connected to the services, but no further connections to the services can be made.
`-verbose`	Verbose output

2.5.12.7.2 Example

The following example stops the `sales` database service on the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl stop service -db dbcrm -service sales
```

2.5.13 unsetenv

The `unsetenv` command deletes one or more environment variables from the Oracle Restart configuration for a database, a listener, or an Oracle ASM instance.

To perform `srvctl unsetenv` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" for more information.

- srvctl unsetenv asm
  Removes the specified environment variables from the Oracle Restart configuration for the Oracle ASM instance.
- srvctl unsetenv database
  Removes the specified environment variables from the Oracle Restart configuration for the specified database.
- srvctl unsetenv listener
  Removes the specified environment variables from the Oracle Restart configuration for the specified listener or all listeners.

See Also:

- setenv command
- getenv command
- "Managing Environment Variables in the Oracle Restart Configuration"

2.5.13.1 srvctl unsetenv asm

Removes the specified environment variables from the Oracle Restart configuration for the Oracle ASM instance.

- Syntax and Options
- Example

2.5.13.1.1 Syntax and Options

Use the `srvctl unsetenv asm` command with the following syntax:

```
srvctl unsetenv asm -envs name_list
```

Table 2-66 srvctl unsetenv asm Options

Options	Description
`-envs name_list`	Comma-delimited list of environment variables to remove

2.5.13.1.2 Example

The following example removes the AIX operating system environment variable `AIXTHREAD_SCOPE` from the Oracle ASM instance configuration:

```
srvctl unsetenv asm -envs AIXTHREAD_SCOPE
```

2.5.13.2 srvctl unsetenv database

Removes the specified environment variables from the Oracle Restart configuration for the specified database.

- Syntax and Options
- Example

2.5.13.2.1 Syntax and Options

Use the `srvctl unsetenv database` command as follows:

```
srvctl unsetenv database -db db_unique_name -envs name_list
```

Table 2-67 srvctl unsetenv database Options

Options	Description
`-db db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-envs name_list`	Comma-delimited list of environment variables to remove

2.5.13.2.2 Example

The following example deletes the `AIXTHREAD_SCOPE` environment variable from the Oracle Restart configuration for the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl unsetenv database -db dbcrm -envs AIXTHREAD_SCOPE
```

2.5.13.3 srvctl unsetenv listener

Removes the specified environment variables from the Oracle Restart configuration for the specified listener or all listeners.

- Syntax and Options
- Example

2.5.13.3.1 Syntax and Options

Use the `srvctl unsetenv listener` command with the following syntax:

```
srvctl unsetenv listener [-listener listener_name] -envs name_list
```

Table 2-68 srvctl unsetenv listener Options

Options	Description
`-listener listener_name`	Listener name. If omitted, the specified environment variables are removed from the configurations of all listeners.
`-envs name_list`	Comma-delimited list of environment variables to remove

2.5.13.3.2 Example

The following example removes the AIX operating system environment variable `AIXTHREAD_SCOPE` from the listener configuration for the listener named `crmlistener`:

```
srvctl unsetenv listener -listener crmlistener -envs AIXTHREAD_SCOPE
```

2.5.14 update

The `srvctl update` command updates the running database to switch to the specified startup option.

- srvctl update database
  The `srvctl update database` command changes the open mode of the database.

2.5.14.1 srvctl update database

The `srvctl update database` command changes the open mode of the database.

- Syntax and Options

2.5.14.1.1 Syntax and Options

Use the `srvctl update database` command as follows:

```
srvctl update database -db db_unique_name --startoption start_options
```

Table 2-69 srvctl upgrade database Options

Option	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-startoption` *`start_options`*	Startup options for the database. Examples of startup options are `OPEN`, `MOUNT`, or `"READ ONLY"`.

2.5.15 upgrade

The `srvctl upgrade` command upgrades the resources types and resources from an older version to a newer version.

- srvctl upgrade database
  The `srvctl upgrade database` command upgrades the configuration of a database and all of its services to the version of the database home from where this command is run.

2.5.15.1 srvctl upgrade database

The `srvctl upgrade database` command upgrades the configuration of a database and all of its services to the version of the database home from where this command is run.

- Syntax and Options

2.5.15.1.1 Syntax and Options

Use the `srvctl upgrade database` command as follows:

```
srvctl upgrade database -db db_unique_name -oraclehome oracle_home
```

Table 2-70 srvctl upgrade database Options

Parameter	Description
`-db` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

Table 2-70 (Cont.) srvctl upgrade database Options

Parameter	Description
`-oraclehome` *`oracle_home`*	The full path of Oracle home for the database

2.6 CRSCTL Command Reference

You can reference details about the syntax for the CRSCTL commands that are relevant for Oracle Restart.

Note:

You must be the root user or Oracle grid infrastructure software owner to run these CRSCTL commands.

CRSCTL Command Syntax Overview

CRSCTL expects the following command syntax:

```
crsctl command has
```

where *`command`* is a verb such as `start`, `stop`, or `enable`. The `has` object indicates Oracle high availability services.

Case Sensitivity

CRSCTL commands and components are case insensitive.

- check
  Displays the Oracle Restart status.
- config
  Displays the Oracle Restart configuration.
- disable
  Disables automatic restart of Oracle Restart.
- enable
  Enables automatic restart of Oracle Restart.
- start
  Starts Oracle Restart.
- stop
  Stops Oracle Restart.

2.6.1 check

Displays the Oracle Restart status.

Syntax and Options

```
crsctl check has
```

2.6.2 config

Displays the Oracle Restart configuration.

Syntax and Options

```
crsctl config has
```

2.6.3 disable

Disables automatic restart of Oracle Restart.

Syntax and Options

```
crsctl disable has
```

2.6.4 enable

Enables automatic restart of Oracle Restart.

Syntax and Options

```
crsctl enable has
```

2.6.5 start

Starts Oracle Restart.

Syntax and Options

```
crsctl start has
```

2.6.6 stop

Stops Oracle Restart.

Syntax and Options

```
crsctl stop has [-f]
```

Table 2-71 crsctl stop has Options

Options	Description
`-f`	Force. If any resources that are managed by Oracle Restart are still running, then try to stop these resources gracefully. If a resource cannot be stopped gracefully, then try to force the resource to stop. For example, if an Oracle ASM instance is running, then `SHUTDOWN IMMEDIATE` attempts to stop the Oracle ASM instance gracefully, while `SHUTDOWN ABORT` attempts to force the Oracle ASM instance to stop. When the `-f` option *is not* specified, this command tries to stop resources managed by Oracle Restart gracefully but does not try to force them to stop. **Note:** For a database resource, this command always uses `SHUTDOWN ABORT`, regardless of whether the `-f` option is specified.

3 Managing Processes

Oracle Databases uses several processes so that multiple users and applications can connect to a single database instance simultaneously.

- About Dedicated and Shared Server Processes
  Oracle Database creates server processes to handle the requests of user processes connected to an instance.
- About Database Resident Connection Pooling
  Database Resident Connection Pooling (DRCP) provides a connection pool in the database server for typical Web application usage scenarios where the application acquires a database connection, works on it for a relatively short duration, and then releases it. DRCP pools "dedicated" servers. A **pooled server** is the equivalent of a server foreground process and a database session combined.
- About Proxy Resident Connection Pooling
  Proxy resident connection pooling uses Proxy Resident Connection Pool that can be configured using Oracle Connection Manager in Traffic Director Mode. Proxy resident connection pooling provides high availability, security, and performance for database clients.
- Configuring Oracle Database for Shared Server
  You can enable shared server and set or alter shared server initialization parameters.
- Configuring Database Resident Connection Pooling
  The database server is preconfigured to allow database resident connection pooling. However, you must explicitly enable this feature by starting the connection pool.
- About Oracle Database Background Processes
  To maximize performance and accommodate many users, a multiprocess Oracle Database system uses **background processes**. Background processes consolidate functions that would otherwise be handled by multiple database programs running for each user process. Background processes asynchronously perform I/O and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.
- Managing Prespawned Processes
  Oracle Database can prespawn processes for better client connection performance.
- Managing Processes for Parallel SQL Execution
  You can manage parallel processing of SQL statements. In this configuration, Oracle Database can divide the work of processing an SQL statement among multiple parallel processes.
- Managing Processes for External Procedures
  An external procedure is a procedure or function written in a programming language and stored in a shared library. An Oracle server can call external procedures or functions using PL/SQL routines.
- Terminating Sessions
  Sometimes it is necessary to terminate current user sessions. For example, you might want to perform an administrative operation and need to terminate all non-administrative sessions.
- Process and Session Data Dictionary Views
  You can query data dictionary views for information about processes and sessions.

3.1 About Dedicated and Shared Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to an instance.

A server process can be either of the following:

- A **dedicated server process**, which services only one user process
- A **shared server process**, which can service multiple user processes

Your database is always enabled to allow dedicated server processes, but you must specifically configure and enable shared server by setting one or more initialization parameters.

- Dedicated Server Processes
  A dedicated server process services only one user process.
- Shared Server Processes
  A shared server process can service multiple user processes.

3.1.1 Dedicated Server Processes

A dedicated server process services only one user process.

Figure 3-1 illustrates how dedicated server processes work. In this diagram two user processes are connected to the database through dedicated server processes.

In general, it is better to be connected through a **dispatcher** and use a shared server process. This is illustrated in Figure 3-2. A shared server process can be more efficient because it keeps the number of processes required for the running instance low.

In the following situations, however, users and administrators should explicitly connect to an instance using a dedicated server process:

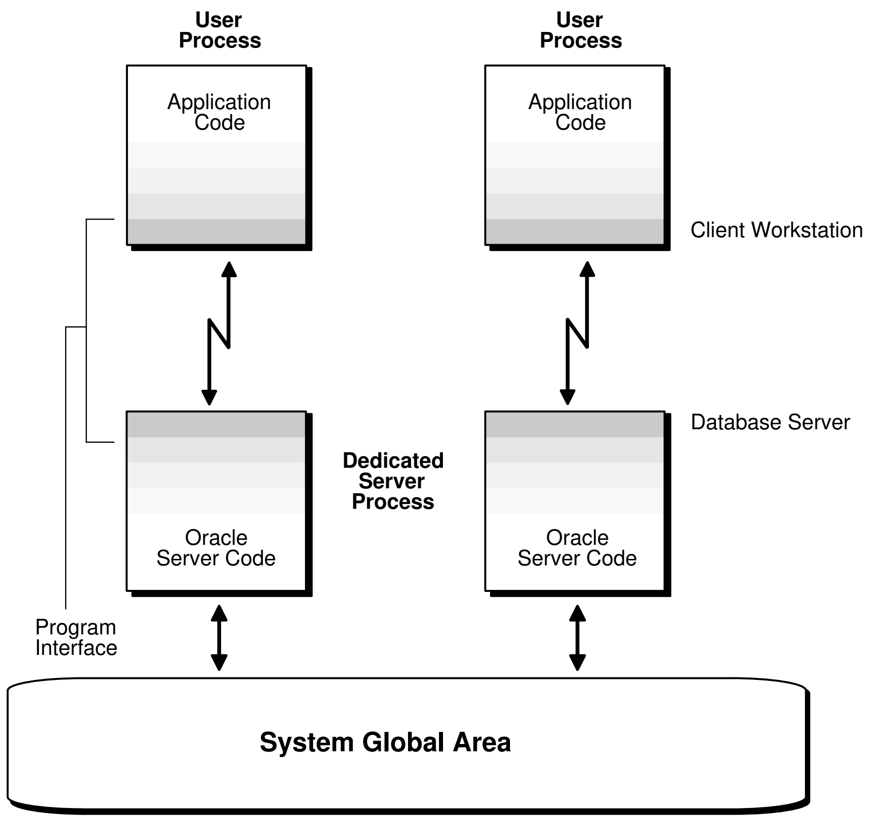
- To submit a batch job (for example, when a job can allow little or no idle time for the server process)
- To use Recovery Manager (RMAN) to back up, restore, or recover a database

To request a dedicated server connection when Oracle Database is configured for shared server, users must connect using a net service name that is configured to use a dedicated server. Specifically, the net service name value should include the `SERVER=DEDICATED` clause in the connect descriptor.

See Also:

Oracle Database Net Services Administrator's Guide for more information about requesting a dedicated server connection

Figure 3-1 Oracle Database Dedicated Server Processes



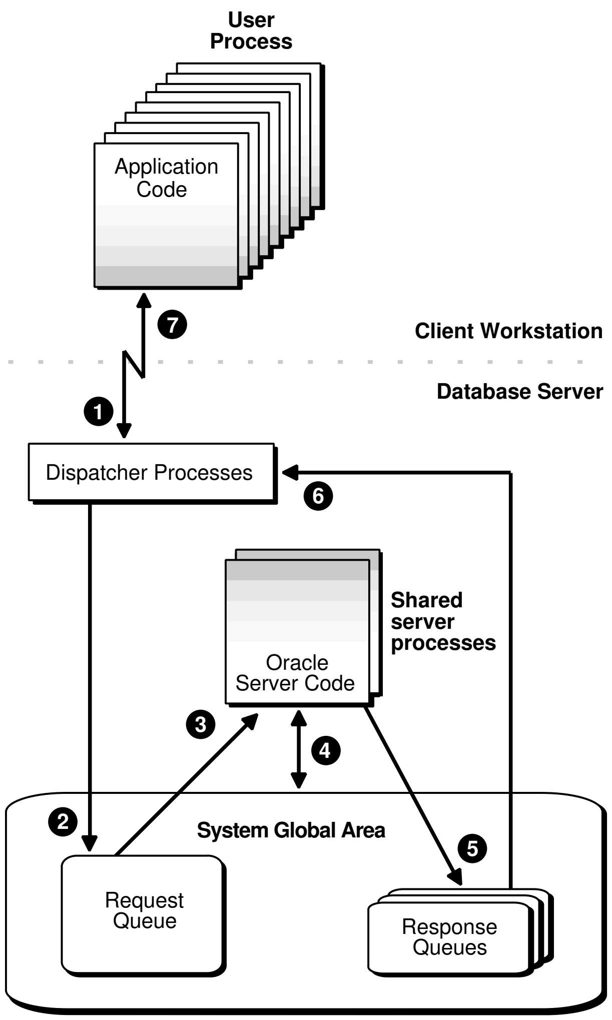
3.1.2 Shared Server Processes

A shared server process can service multiple user processes.

Consider an order entry system with dedicated server processes. A customer phones the order desk and places an order, and the clerk taking the call enters the order into the database. For most of the transaction, the clerk is on the telephone talking to the customer. A server process is not needed during this time, so the server process dedicated to the clerk's user process remains idle. The system is slower for other clerks entering orders, because the idle server process is holding system resources.

Shared server architecture eliminates the need for a dedicated server process for each connection (see Figure 3-2).

Figure 3-2 Oracle Database Shared Server Processes



In a shared server configuration, client user processes connect to a dispatcher. The dispatcher can support multiple client connections concurrently. Each client connection is bound to a **virtual circuit**, which is a piece of shared memory used by the dispatcher for client database connection requests and replies. The dispatcher places a virtual circuit on a common queue when a request arrives.

An idle shared server process picks up the virtual circuit from the common queue, services the request, and relinquishes the virtual circuit before attempting to retrieve another virtual circuit from the common queue. This approach enables a small pool of server processes to serve a large number of clients. A significant advantage of shared server architecture over the dedicated server model is the reduction of system resources, enabling the support of an increased number of users.

For even better resource management, shared server can be configured for **session multiplexing**, which combines multiple sessions for transmission over a single network connection in order to conserve the operating system's resources.

Shared server architecture requires Oracle Net Services. User processes targeting the shared server must connect through Oracle Net Services, even if they are on the same system as the Oracle Database instance.

See Also:

Oracle Database Net Services Administrator's Guide for more detailed information about shared server, including features such as session multiplexing

3.2 About Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) provides a connection pool in the database server for typical Web application usage scenarios where the application acquires a database connection, works on it for a relatively short duration, and then releases it. DRCP pools "dedicated" servers. A **pooled server** is the equivalent of a server foreground process and a database session combined.

DRCP complements middle-tier connection pools that share connections between threads in a middle-tier process. In addition, DRCP enables sharing of database connections across middle-tier processes on the same middle-tier host and even across middle-tier hosts. This results in significant reduction in key database resources needed to support a large number of client connections, thereby reducing the database tier memory footprint and boosting the scalability of both the middle-tier and the database tier. Having a pool of readily available servers also has the additional benefit of reducing the cost of creating and tearing down client connections.

DRCP is especially relevant for architectures with multi-process single threaded application servers (such as PHP/Apache) that cannot perform middle-tier connection pooling. The database can still scale to tens of thousands of simultaneous connections with DRCP.

Starting with Oracle Database Release 21c, DRCP can be configured based on the requirements of specific pluggable databases (PDBs). PDB administrators can independently configure, manage, and monitor a connection pool for individual PDBs. Note that the broker processes are owned and configured by the root and shared among the PDB pools.

Note:

- Starting with Oracle Database 12*c* Release 2 (12.2), proxy sessions that belong to the same user can be shared.
- On Windows platforms, setting the `SQLNET.AUTHENTICATION_SERVICES` parameter value to `nts` is not supported with DRCP.

See Also:

- *Oracle Database Concepts* for more details on DRCP
- *Oracle Database Development Guide* for more information about DRCP, including restrictions on using DRCP
- *Oracle Call Interface Programmer's Guide* for information about options that are available when obtaining a DRCP session
- *Oracle Database Development Guide* for information about sharing proxy sessions

When To Use Database Resident Connection Pooling

Database resident connection pooling is useful when multiple clients access the database and when any of the following apply:

- A large number of client connections need to be supported with minimum memory usage.
- The client applications are similar and can share or reuse sessions.

  Applications are similar if they connect with the same database credentials and use the same schema.
- The client applications acquire a database connection, work on it for a relatively short duration, and then release it.
- Session affinity is not required across client requests.
- There are multiple processes and multiple hosts on the client side.

Advantages of Database Resident Connection Pooling

Using database resident connection pooling provides the following advantages:

- Enables resource sharing among multiple middle-tier client applications.
- Improves scalability of databases and applications by reducing resource usage.

Database Resident Connection Pooling and LOGON/LOGOFF Triggers

`LOGON` triggers fire for every authentication and every time a new session is created in DRCP.

`LOGOFF` triggers fire on every log off and when the sessions are destroyed in DRCP. Therefore, a `LOGOFF` trigger fires when a session is terminated due to an idle time limit.

- Comparing DRCP to Dedicated Server and Shared Server
  Understand the differences between dedicated server, shared server, and database resident connection pooling.

See Also:

- *Oracle Database PL/SQL Language Reference*
- *Oracle Database Security Guide*

3.2.1 Comparing DRCP to Dedicated Server and Shared Server

Understand the differences between dedicated server, shared server, and database resident connection pooling.

Table 3-1 lists the differences between dedicated server, shared server, and database resident connection pooling.

Table 3-1 Dedicated Servers, Shared Servers, and Database Resident Connection Pooling

Dedicated Server	Shared Server	Database Resident Connection Pooling
When a client request is received, a new server process and a session are created for the client.	When the first request is received from a client, the Dispatcher process places this request on a common queue. The request is picked up by an available shared server process. The Dispatcher process then manages the communication between the client and the shared server process.	When the first request is received from a client, the Connection Broker picks an available pooled server and hands off the client connection to the pooled server. If no pooled servers are available, the Connection Broker creates one. If the pool has reached its maximum size, the client request is placed on the wait queue until a pooled server is available.
Releasing database resources involves terminating the session and server process.	Releasing database resources involves terminating the session.	Releasing database resources involves releasing the pooled server to the pool.
Memory requirement is proportional to the number of server processes and sessions. There is one server and one session for each client.	Memory requirement is proportional to the sum of the shared servers and sessions. There is one session for each client.	Memory requirement is proportional to the number of pooled servers and their sessions. There is one session for each pooled server.
Session memory is allocated from the PGA.	Session memory is allocated from the SGA.	Session memory is allocated from the PGA.

Example of Memory Usage for Dedicated Server, Shared Server, and Database Resident Connection Pooling

Consider an application in which the memory required for each session is 400 KB and the memory required for each server process is 4 MB. The pool size is 100 and the number of shared servers used is 100.

If there are 5000 client connections, the memory used by each configuration is as follows:

- Dedicated Server

  Memory used = 5000 X (400 KB + 4 MB) = 22 GB
- Shared Server

  Memory used = 5000 X 400 KB + 100 X 4 MB = 2.5 GB

  Out of the 2.5 GB, 2 GB is allocated from the SGA.
- Database Resident Connection Pooling

  Memory used = 100 X (400 KB + 4 MB) + (5000 X 35KB)= 615 MB

  The cost of each connection to the broker is approximately 35 KB.

3.3 About Proxy Resident Connection Pooling

Proxy resident connection pooling uses Proxy Resident Connection Pool that can be configured using Oracle Connection Manager in Traffic Director Mode. Proxy resident connection pooling provides high availability, security, and performance for database clients.

A database client that is based on any of the following technologies can use proxy resident connection pooling to connect to a database instance – Oracle Call Interface (OCI), Java Database Connectivity (JDBC), Oracle Data Provider for .NET (ODP.Net), Open Database Connectivity (ODBC), Pro*C, Pro*COBOL, PHP OCI8 extension, Node.js node-oracledb driver, Python cx_Oracle, ROracle, Ruby-oci8, Perl DBD::Oracle, or Oracle C++ Call Interface (OCCI).

Note:

Proxy resident connection pooling is available starting with Oracle Database 18c.

When to Use Proxy Resident Connection Pooling

Proxy resident connection pooling is useful when multiple clients access a database and when any of the following apply:

- A large number of client connections need to be supported using fewer number of connections to a database.
- A database connection needs to be shared across middle tier connection pools.
- More than 64K sessions need to be supported (when shared servers cannot be used as they have a limit of 64K sessions).
- High availability needs to be supported for older clients that do not support Transparent Application Failover (TAF) and Oracle RAC, or the clients that do not use Oracle Database Resident Connection Pooling (DRCP) or Fast Application Notification (FAN) or Application Continuity (AC).

Advantages of Proxy Resident Connection Pooling

Using proxy resident connection pooling provides the following major advantages:

- Improved high availability (planned and unplanned)
- Improved database security
- Database connection multiplexing

See Also:

The following sections in *Oracle Database Net Services Administrator's Guide* for more information about enabling proxy resident connection pooling using Oracle Connection Manager in Traffic Director Mode.

- "About Using Oracle Connection Manager in Traffic Director Mode"
- "Configuring Oracle Connection Manager in Traffic Director Mode"

3.4 Configuring Oracle Database for Shared Server

You can enable shared server and set or alter shared server initialization parameters.

- Initialization Parameters for Shared Server
  A set of initialization parameters control shared server operation.
- Memory Management for Shared Server
  Shared server requires some user global area (UGA) in either the shared pool or large pool. For installations with a small number of simultaneous sessions, the default sizes for these system global area (SGA) components are generally sufficient. However, if you expect a large number of sessions for your installation, you may have to tune memory to support shared server.
- Enabling Shared Server
  Shared server is enabled by setting the `SHARED_SERVERS` initialization parameter to a value greater than 0. The other shared server initialization parameters need not be set.
- Configuring Dispatchers
  The `DISPATCHERS` initialization parameter configures dispatcher processes in the shared server architecture. At least one dispatcher process is required for shared server to work. If you do not specify a dispatcher, but you enable shared server by setting `SHARED_SERVER` to a nonzero value, then by default Oracle Database creates one dispatcher for the TCP protocol.
- Disabling Shared Server
  You disable shared server by setting the `SHARED_SERVERS` initialization parameter to 0. You can do this dynamically with the `ALTER SYSTEM` statement.
- Shared Server Data Dictionary Views
  You can query data dictionary views for information about your shared server configuration and to monitor performance.

See Also:

- "About Dedicated and Shared Server Processes"
- *Oracle Database SQL Language Reference* for further information about the `ALTER SYSTEM` statement

3.4.1 Initialization Parameters for Shared Server

A set of initialization parameters control shared server operation.

The following initialization parameters control shared server operation:

- `SHARED_SERVERS`: Specifies the initial number of shared servers to start and the minimum number of shared servers to keep. This is the only required parameter for using shared servers.
- `MAX_SHARED_SERVERS`: Specifies the maximum number of shared servers that can run simultaneously.

- `SHARED_SERVER_SESSIONS`: Specifies the total number of shared server user sessions that can run simultaneously. Setting this parameter enables you to reserve user sessions for dedicated servers.
- `DISPATCHERS`: Configures dispatcher processes in the shared server architecture.
- `MAX_DISPATCHERS`: Specifies the maximum number of dispatcher processes that can run simultaneously. This parameter can be ignored for now. It will only be useful in a future release when the number of dispatchers is auto-tuned according to the number of concurrent connections.
- `CIRCUITS`: Specifies the total number of virtual circuits that are available for inbound and outbound network sessions.

See Also:

Oracle Database Reference for more information about these initialization parameters

3.4.2 Memory Management for Shared Server

Shared server requires some user global area (UGA) in either the shared pool or large pool. For installations with a small number of simultaneous sessions, the default sizes for these system global area (SGA) components are generally sufficient. However, if you expect a large number of sessions for your installation, you may have to tune memory to support shared server.

See the "Configuring and Using Memory" section of *Oracle Database Performance Tuning Guide* for guidelines.

3.4.3 Enabling Shared Server

Shared server is enabled by setting the `SHARED_SERVERS` initialization parameter to a value greater than 0. The other shared server initialization parameters need not be set.

- Set shared server dynamically by setting the `SHARED_SERVERS` initialization parameter to a nonzero value with the `ALTER SYSTEM` statement.
- Set the `SHARED_SERVERS` initialization parameter to a nonzero value at database startup by including it in the initialization parameter file.

Because shared server requires at least one dispatcher in order to work, a dispatcher is brought up even if no dispatcher has been configured. Dispatchers are discussed in "Configuring Dispatchers".

Note:

If SHARED_SERVERS is not included in the initialization parameter file at database startup, but DISPATCHERS is included and it specifies at least one dispatcher, shared server is enabled. In this case, the default for SHARED_SERVERS is 1.

If neither SHARED_SERVERS nor DISPATCHERS is included in the initialization file, you cannot start shared server after the instance is brought up by just altering the DISPATCHERS parameter. You must specifically alter SHARED_SERVERS to a nonzero value to start shared server.

Note:

If you create your Oracle database with Database Configuration Assistant (DBCA), DBCA configures a dispatcher for Oracle XML DB (XDB). This is because XDB protocols like HTTP and FTP require shared server. This results in a SHARED_SERVER value of 1. Although shared server is enabled, this configuration permits only sessions that connect to the XDB service to use shared server. To enable shared server for regular database sessions (for submitting SQL statements), you must add an additional dispatcher configuration, or replace the existing configuration with one that is not specific to XDB. See "Configuring Dispatchers" for instructions.

- About Determining a Value for SHARED_SERVERS
  The SHARED_SERVERS initialization parameter specifies the minimum number of shared servers that you want created when the instance is started. After instance startup, Oracle Database can dynamically adjust the number of shared servers based on how busy existing shared servers are and the length of the request queue.
- Decreasing the Number of Shared Server Processes
  You can decrease the minimum number of shared servers that must be kept active by dynamically setting the SHARED_SERVERS parameter to a lower value. Thereafter, until the number of shared servers is decreased to the value of the SHARED_SERVERS parameter, any shared servers that become inactive are marked by PMON for termination.
- Limiting the Number of Shared Server Processes
  The MAX_SHARED_SERVERS initialization parameter specifies the maximum number of shared servers that can be automatically created by PMON. It has no default value.
- Limiting the Number of Shared Server Sessions
  The SHARED_SERVER_SESSIONS initialization parameter specifies the maximum number of concurrent shared server user sessions.
- Protecting Shared Memory
  The CIRCUITS initialization parameter sets a maximum limit on the number of virtual circuits that can be created in shared memory. This parameter has no default. If it is not specified, then the system can create circuits as needed, limited by the DISPATCHERS initialization parameter and system resources.

3.4.3.1 About Determining a Value for SHARED_SERVERS

The SHARED_SERVERS initialization parameter specifies the minimum number of shared servers that you want created when the instance is started. After instance startup, Oracle Database can dynamically adjust the number of shared servers based on how busy existing shared servers are and the length of the request queue.

In typical systems, the number of shared servers stabilizes at a ratio of one shared server for every ten connections. For OLTP applications, when the rate of requests is low, or when the ratio of server usage to request is low, the connections-to-servers ratio could be higher. In contrast, in applications where the rate of requests is high or the server usage-to-request ratio is high, the connections-to-server ratio could be lower.

The PMON (process monitor) background process cannot terminate shared servers below the value specified by `SHARED_SERVERS`. Therefore, you can use this parameter to stabilize the load and minimize strain on the system by preventing PMON from terminating and then restarting shared servers because of coincidental fluctuations in load.

If you know the average load on your system, you can set `SHARED_SERVERS` to an optimal value. The following example shows how you can use this parameter:

Assume a database is being used by a telemarketing center staffed by 1000 agents. On average, each agent spends 90% of the time talking to customers and only 10% of the time looking up and updating records. To keep the shared servers from being terminated as agents talk to customers and then spawned again as agents access the database, a DBA specifies that the optimal number of shared servers is 100.

However, not all work shifts are staffed at the same level. On the night shift, only 200 agents are needed. Since `SHARED_SERVERS` is a dynamic parameter, a DBA reduces the number of shared servers to 20 at night, thus allowing resources to be freed up for other tasks such as batch jobs.

3.4.3.2 Decreasing the Number of Shared Server Processes

You can decrease the minimum number of shared servers that must be kept active by dynamically setting the `SHARED_SERVERS` parameter to a lower value. Thereafter, until the number of shared servers is decreased to the value of the `SHARED_SERVERS` parameter, any shared servers that become inactive are marked by PMON for termination.

- Set shared server dynamically by setting the `SHARED_SERVERS` initialization parameter to a nonzero value with the `ALTER SYSTEM` statement.

For example, the following statement reduces the number of shared servers:

```
ALTER SYSTEM SET SHARED_SERVERS = 5;
```

Setting `SHARED_SERVERS` to 0 disables shared server. For more information, see "Disabling Shared Server".

3.4.3.3 Limiting the Number of Shared Server Processes

The `MAX_SHARED_SERVERS` initialization parameter specifies the maximum number of shared servers that can be automatically created by PMON. It has no default value.

If no value is specified, then PMON starts as many shared servers as is required by the load, subject to these limitations:

- The process limit (set by the `PROCESSES` initialization parameter)
- A minimum number of free process slots (at least one-eighth of the total process slots, or two slots if `PROCESSES` is set to less than 24)
- System resources

To limit the number of shared server processes:

- Set the `MAX_SHARED_SERVERS` initialization parameter.

The value of `SHARED_SERVERS` overrides the value of `MAX_SHARED_SERVERS`. Therefore, you can force PMON to start more shared servers than the `MAX_SHARED_SERVERS` value by setting `SHARED_SERVERS` to a value higher than `MAX_SHARED_SERVERS`. You can subsequently place a new upper limit on the number of shared servers by dynamically altering the `MAX_SHARED_SERVERS` to a value higher than `SHARED_SERVERS`.

The primary reason to limit the number of shared servers is to reserve resources, such as memory and CPU time, for other processes. For example, consider the case of the telemarketing center discussed previously:

The DBA wants to reserve two thirds of the resources for batch jobs at night. He sets `MAX_SHARED_SERVERS` to less than one third of the maximum number of processes (`PROCESSES`). By doing so, the DBA ensures that even if all agents happen to access the database at the same time, batch jobs can connect to dedicated servers without having to wait for the shared servers to be brought down after processing agents' requests.

Another reason to limit the number of shared servers is to prevent the concurrent run of too many server processes from slowing down the system due to heavy swapping, although `PROCESSES` can serve as the upper bound for this rather than `MAX_SHARED_SERVERS`.

Still other reasons to limit the number of shared servers are testing, debugging, performance analysis, and tuning. For example, to see how many shared servers are needed to efficiently support a certain user community, you can vary `MAX_SHARED_SERVERS` from a very small number upward until no delay in response time is noticed by the users.

3.4.3.4 Limiting the Number of Shared Server Sessions

The `SHARED_SERVER_SESSIONS` initialization parameter specifies the maximum number of concurrent shared server user sessions.

Setting this parameter, which is a dynamic parameter, lets you reserve database sessions for dedicated servers. This in turn ensures that administrative tasks that require dedicated servers, such as backing up or recovering the database, are not preempted by shared server sessions.

To limit the number of shared server sessions:

- Set the `SHARED_SERVER_SESSIONS` initialization parameter.

This parameter has no default value. If it is not specified, the system can create shared server sessions as needed, limited by the `SESSIONS` initialization parameter.

3.4.3.5 Protecting Shared Memory

The `CIRCUITS` initialization parameter sets a maximum limit on the number of virtual circuits that can be created in shared memory. This parameter has no default. If it is not specified, then the system can create circuits as needed, limited by the `DISPATCHERS` initialization parameter and system resources.

To protect shared memory by limiting the number of virtual circuits that can be created in shared memory:

- Set the `CIRCUITS` initialization parameter.

3.4.4 Configuring Dispatchers

The `DISPATCHERS` initialization parameter configures dispatcher processes in the shared server architecture. At least one dispatcher process is required for shared server to work. If you do not specify a dispatcher, but you enable shared server by setting `SHARED_SERVER` to a nonzero value, then by default Oracle Database creates one dispatcher for the TCP protocol.

The equivalent `DISPATCHERS` explicit setting of the initialization parameter for this configuration is:

```
dispatchers="(PROTOCOL=tcp)"
```

You can configure more dispatchers, using the `DISPATCHERS` initialization parameter, if either of the following conditions apply:

- You must configure a protocol other than TCP/IP. You configure a protocol address with one of the following attributes of the DISPATCHERS parameter:
  - ADDRESS
  - DESCRIPTION
  - PROTOCOL
- You want to configure one or more of the optional dispatcher attributes:
  - DISPATCHERS
  - CONNECTIONS
  - SESSIONS
  - LISTENER
  - MULTIPLEX
  - SERVICE

> **Note:**
>
> Database Configuration Assistant helps you configure this parameter.

To configure a protocol other than TCP/IP or to configure additional dispatchers:

- Set the `DISPATCHERS` initialization parameter and specify the appropriate attributes.

- DISPATCHERS Initialization Parameter Attributes
  You can set several attributes for the `DISPATCHERS` initialization parameter.
- Determining the Number of Dispatchers
  Once you know the number of possible connections for each process for the operating system, calculate the initial number of dispatchers to create during instance startup, for each network protocol.
- Setting the Initial Number of Dispatchers
  You can specify multiple dispatcher configurations by setting `DISPATCHERS` to a comma separated list of strings, or by specifying multiple `DISPATCHERS` initialization parameters in the initialization parameter file.

- Altering the Number of Dispatchers
  You can control the number of dispatcher processes in the instance. Unlike the number of shared servers, the number of dispatchers does not change automatically. You change the number of dispatchers explicitly with the `ALTER SYSTEM` statement. You can increase the number of dispatchers to more than the limit specified by the `MAX_DISPATCHERS` parameter.
- Shutting Down Specific Dispatcher Processes
  With the `ALTER SYSTEM SET DISPATCHERS` statement, you leave it up to the database to determine which dispatchers to shut down to reduce the number of dispatchers. Alternatively, it is possible to shut down specific dispatcher processes.

3.4.4.1 DISPATCHERS Initialization Parameter Attributes

You can set several attributes for the `DISPATCHERS` initialization parameter.

A protocol address is required and is specified using one or more of the following attributes:

Attribute	Description
`ADDRESS`	Specify the network protocol address of the endpoint on which the dispatchers listen.
`DESCRIPTION`	Specify the network description of the endpoint on which the dispatchers listen, including the network protocol address. The syntax is as follows: `(DESCRIPTION=(ADDRESS=...))`
`PROTOCOL`	Specify the network protocol for which the dispatcher generates a listening endpoint. For example: `(PROTOCOL=tcp)` See the *Oracle Database Net Services Reference* for further information about protocol address syntax.

The following attribute specifies how many dispatchers this configuration should have. It is optional and defaults to 1.

Attribute	Description
`DISPATCHERS`	Specify the initial number of dispatchers to start.

The following attributes tell the instance about the network attributes of each dispatcher of this configuration. They are all optional.

Attribute	Description
`CONNECTIONS`	Specify the maximum number of network connections to allow for each dispatcher.
`SESSIONS`	Specify the maximum number of network sessions to allow for each dispatcher.
`LISTENER`	Specify an alias name for the listeners with which the LREG process registers dispatcher information. Set the alias to a name that is resolved through a naming method.
`MULTIPLEX`	Used to enable the Oracle Connection Manager session multiplexing feature.
`SERVICE`	Specify the service names the dispatchers register with the listeners.

You can specify either an entire attribute name a substring consisting of at least the first three characters. For example, you can specify `SESSIONS=3`, `SES=3`, `SESS=3`, or `SESSI=3`, and so forth.

See Also:

Oracle Database Reference for more detailed descriptions of the attributes of the `DISPATCHERS` initialization parameter

3.4.4.2 Determining the Number of Dispatchers

Once you know the number of possible connections for each process for the operating system, calculate the initial number of dispatchers to create during instance startup, for each network protocol.

To calculate the initial number of dispatchers to create during instance startup, use the following formula:

```
Number of dispatchers =
   CEIL ( max. concurrent sessions / connections for each dispatcher )
```

`CEIL` returns the result roundest up to the next whole integer.

For example, assume a system that can support 970 connections for each process, and that has:

- A maximum of 4000 sessions concurrently connected through TCP/IP and
- A maximum of 2,500 sessions concurrently connected through TCP/IP with SSL

The `DISPATCHERS` attribute for TCP/IP should be set to a minimum of five dispatchers (4000 / 970), and for TCP/IP with SSL three dispatchers (2500 / 970:

```
DISPATCHERS='(PROT=tcp)(DISP=5)', '(PROT=tcps)(DISP=3)'
```

Depending on performance, you may need to adjust the number of dispatchers.

3.4.4.3 Setting the Initial Number of Dispatchers

You can specify multiple dispatcher configurations by setting `DISPATCHERS` to a comma separated list of strings, or by specifying multiple `DISPATCHERS` initialization parameters in the initialization parameter file.

- Set the `DISPATCHERS` initialization parameter.

If you specify `DISPATCHERS` multiple times, then the lines must be adjacent to each other in the initialization parameter file. Internally, Oracle Database assigns an `INDEX` value (beginning with zero) to each `DISPATCHERS` parameter. You can later refer to that `DISPATCHERS` parameter in an `ALTER SYSTEM` statement by its index number.

Some examples of setting the `DISPATCHERS` initialization parameter follow.

Example: Typical

This is a typical example of setting the `DISPATCHERS` initialization parameter.

```
DISPATCHERS="(PROTOCOL=TCP)(DISPATCHERS=2)"
```

Example: Forcing the IP Address Used for Dispatchers

The following hypothetical example will create two dispatchers that will listen on the specified IP address. The address must be a valid IP address for the host that the instance is on. (The host may be configured with multiple IP addresses.)

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.16.201))(DISPATCHERS=2)"
```

Example: Forcing the Port Used by Dispatchers

To force the dispatchers to use a specific port as the listening endpoint, add the `PORT` attribute as follows:

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5000))"
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5001))"
```

3.4.4.4 Altering the Number of Dispatchers

You can control the number of dispatcher processes in the instance. Unlike the number of shared servers, the number of dispatchers does not change automatically. You change the number of dispatchers explicitly with the `ALTER SYSTEM` statement. You can increase the number of dispatchers to more than the limit specified by the `MAX_DISPATCHERS` parameter.

1. Monitor the following views to determine the load on the dispatcher processes:
   - `V$QUEUE`
   - `V$DISPATCHER`
   - `V$DISPATCHER_RATE`

   If these views indicate that the load on the dispatcher processes is consistently high, then performance may be improved by starting additional dispatcher processes to route user requests. In contrast, if the load on dispatchers is consistently low, reducing the number of dispatchers may improve performance.

   See Also:

   Oracle Database Performance Tuning Guide for information about monitoring these views to determine dispatcher load and performance

2. To dynamically alter the number of dispatchers when the instance is running, use the `ALTER SYSTEM` statement to modify the `DISPATCHERS` attribute setting for an existing dispatcher configuration. You can also add new dispatcher configurations to start dispatchers with different network attributes.

When you reduce the number of dispatchers for a particular dispatcher configuration, the dispatchers are not immediately removed. Rather, as users disconnect, Oracle Database terminates dispatchers down to the limit you specify in `DISPATCHERS`,

For example, suppose the instance was started with this `DISPATCHERS` setting in the initialization parameter file:

```
DISPATCHERS='(PROT=tcp)(DISP=2)', '(PROT=tcps)(DISP=2)'
```

To increase the number of dispatchers for the TCP/IP protocol from 2 to 3, and decrease the number of dispatchers for the TCP/IP with SSL protocol from 2 to 1, you can issue the following statement:

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=0)(DISP=3)', '(INDEX=1)(DISP=1)';
```

or

```
ALTER SYSTEM SET DISPATCHERS = '(PROT=tcp)(DISP=3)', '(PROT=tcps)(DISP=1)';
```

Note:

You need not specify (`DISP=1`). It is optional because 1 is the default value for the `DISPATCHERS` parameter.

If fewer than three dispatcher processes currently exist for TCP/IP, the database creates new ones. If multiple dispatcher processes currently exist for TCP/IP with SSL, then the database terminates the extra ones as the connected users disconnect.

- Notes on Altering Dispatchers
  Understand details about altering dispatchers.

3.4.4.4.1 Notes on Altering Dispatchers

Understand details about altering dispatchers.

- The `INDEX` keyword can be used to identify which dispatcher configuration to modify. If you do not specify `INDEX`, then the first dispatcher configuration matching the `DESCRIPTION`, `ADDRESS`, or `PROTOCOL` specified will be modified. If no match is found among the existing dispatcher configurations, then a new dispatcher will be added.
- The `INDEX` value can range from 0 to *n*-1, where *n* is the current number of dispatcher configurations. If your `ALTER SYSTEM` statement specifies an `INDEX` value equal to *n*, where *n* is the current number of dispatcher configurations, a new dispatcher configuration will be added.
- To see the values of the current dispatcher configurations--that is, the number of dispatchers and so forth--query the `V$DISPATCHER_CONFIG` dynamic performance view. To see which dispatcher configuration a dispatcher is associated with, query the `CONF_INDX` column of the `V$DISPATCHER` view.
- When you change the `DESCRIPTION`, `ADDRESS`, `PROTOCOL`, `CONNECTIONS`, and `MULTIPLEX` attributes of a dispatcher configuration, the change does not take effect for existing dispatchers but only for new dispatchers. Therefore, in order for the change to be effective for all dispatchers associated with a configuration, you must forcibly terminate existing dispatchers after altering the `DISPATCHERS` parameter, and let the database start new ones in their place with the newly specified properties.

  The attributes `LISTENER` and `SERVICES` are not subject to the same constraint. They apply to existing dispatchers associated with the modified configuration. Attribute `SESSIONS` applies to existing dispatchers only if its value is reduced. However, if its value is increased, it is applied only to newly started dispatchers.

3.4.4.5 Shutting Down Specific Dispatcher Processes

With the `ALTER SYSTEM SET DISPATCHERS` statement, you leave it up to the database to determine which dispatchers to shut down to reduce the number of dispatchers. Alternatively, it is possible to shut down specific dispatcher processes.

1. To identify the name of the specific dispatcher process to shut down, use the `V$DISPATCHER` dynamic performance view.

   ```
   SELECT NAME, NETWORK FROM V$DISPATCHER;
   ```

   Each dispatcher is uniquely identified by a name of the form D*nnn*.
2. Run an ALTER SYSTEM SHUTDOWN IMMEDIATE statement and specify the dispatcher name.

   For example, to shut down dispatcher `D002`, issue the following statement:

   ```
   ALTER SYSTEM SHUTDOWN IMMEDIATE 'D002';
   ```

   The `IMMEDIATE` keyword stops the dispatcher from accepting new connections, and the database immediately terminates all existing connections through that dispatcher. After all sessions are cleaned up, the dispatcher process shuts down. If `IMMEDIATE` were not specified, then the dispatcher would wait until all of its users disconnected and all of its connections terminated before shutting down.

3.4.5 Disabling Shared Server

You disable shared server by setting the `SHARED_SERVERS` initialization parameter to 0. You can do this dynamically with the `ALTER SYSTEM` statement.

- Set the `SHARED_SERVERS` initialization parameter to 0.

When you disable shared server, no new clients can connect in shared mode. However, Oracle Database retains some shared servers until all shared server connections are closed. The number of shared servers retained is either the number specified by the preceding setting of `SHARED_SERVERS` or the value of the `MAX_SHARED_SERVERS` parameter, whichever is smaller. If both `SHARED_SERVERS` and `MAX_SHARED_SERVERS` are set to 0, then all shared servers will terminate and requests from remaining shared server clients will be queued until the value of `SHARED_SERVERS` or `MAX_SHARED_SERVERS` is raised again.

To terminate dispatchers once all shared server clients disconnect, enter this statement:

```
ALTER SYSTEM SET DISPATCHERS = '';
```

3.4.6 Shared Server Data Dictionary Views

You can query data dictionary views for information about your shared server configuration and to monitor performance.

View	Description
`V$DISPATCHER`	Provides information on the dispatcher processes, including name, network address, status, various usage statistics, and index number.
`V$DISPATCHER_CONFIG`	Provides configuration information about the dispatchers.
`V$DISPATCHER_RATE`	Provides rate statistics for the dispatcher processes.

View	Description
V$QUEUE	Contains information on the shared server message queues.
V$SHARED_SERVER	Contains information on the shared servers.
V$CIRCUIT	Contains information about virtual circuits, which are user connections to the database through dispatchers and servers.
V$SHARED_SERVER_MONITOR	Contains information for tuning shared server.
V$SGA	Contains size information about various system global area (SGA) groups. May be useful when tuning shared server.
V$SGASTAT	Contains detailed statistical information about the SGA, useful for tuning.
V$SHARED_POOL_RESERVED	Lists statistics to help tune the reserved pool and space within the shared pool.

See Also:

Oracle Database Performance Tuning Guide for specific information about monitoring and tuning shared server

3.5 Configuring Database Resident Connection Pooling

The database server is preconfigured to allow database resident connection pooling. However, you must explicitly enable this feature by starting the connection pool.

- Database Resident Connection Pooling Initialization Parameters
  You can set initialization parameters to configure database resident connection pooling.
- Enabling Database Resident Connection Pooling
  Oracle Database includes a default connection pool called `SYS_DEFAULT_CONNECTION_POOL`. This connection pool must be started to enable database resident connection pooling.
- Configuring the Connection Pool for Database Resident Connection Pooling
  The connection pool is configured using default parameter values. You can use the procedures in the `DBMS_CONNECTION_POOL` package to configure the connection pool according to your usage. In an Oracle Real Application Clusters (Oracle RAC) environment, the configuration parameters are applicable to each Oracle RAC instance.
- Data Dictionary Views for Database Resident Connection Pooling
  You can query data dictionary views to obtain information about your connection pool and to monitor the performance of database resident connection pooling.
- Determining the States of Connections in the Connection Pool
  You can query the `V$CPOOL_CONN_INFO` view to determine the current state of each connection in the connection pool.

See Also:

"About Database Resident Connection Pooling"

3.5.1 Database Resident Connection Pooling Initialization Parameters

You can set initialization parameters to configure database resident connection pooling.

Use the `DRCP_DEDICATED_OPT` initialization parameter to configure the use of dedicated optimization with Database Resident Connection Pooling (DRCP). You enable dedicated optimization by setting `DRCP_DEDICATED_OPT` to `Yes`. Dedicated optimization makes DRCP behave like a dedicated server when the number of connections to the DRCP broker is less than the DRCP maximum size.

The following initialization parameters are used to configure the authentication pool:

- `MAX_AUTH_SERVERS`

  Specifies the maximum number of authentication servers in the authentication pool. The authentication pool, which is separate from the connection pool, authenticates user connections when client applications connect to DRCP. Set this parameter to a positive integer that is greater than the value specified by the `MIN_AUTH_SERVERS` initialization parameter.

- `MIN_AUTH_SERVERS`

  Specifies the minimum number of authentication servers in the authentication pool. Set this parameter to a positive integer that is lesser than the value specified by the `MAX_AUTH_SERVERS` initialization parameter.

- `ENABLE_PER_PDB_DRCP`

  Set this parameter to `TRUE` to enable the creation of connection pools at the pluggable database (PDB) level. DRCP is started for both authentication pools and connection pools when a PDB is opened. The `MIN_AUTH_SERVERS` and `MAX_AUTH_SERVERS` parameters can be set at the PDB level. Only the PDB administrator can configure DRCP parameters for a PDB. The SYS user must grant the following to the PDB administrator: `CREATE SESSION` privilege, `CREATE SYNOMYM` privilege, `EXECUTE` privilege on the `DBMS_CONNECTION_POOL` package, and `SELECT` privilege on the data dictionary views containing DRCP information.

  When this parameter is set to `FALSE`, only the CDB root can manage the connection pool. You can configure DRCP either for the entire database or at the PDB level. When DRCP is enabled at the PDB level, database level DRCP is automatically disabled.

- `DRCP_CONNECTION_LIMIT`

  Specifies the maximum number of Database Resident Connection Pooling (DRCP) connections for a PDB. In `CDB$ROOT`, the default value is `0` (unlimited). In a PDB, if a persistent value for `SESSIONS` was explicitly set for the PDB (`ALTER SYSTEM SET SESSIONS = n SCOPE={SPFILE|BOTH}`), and the PDB was subsequently restarted, then the default value is (10 * `SESSIONS`). Otherwise, the PDB inherits the value for this parameter from `CDB$ROOT`. `DRCP_CONNECTION_LIMIT` is a PDB-inherited parameter. The value of this parameter in `CDB$ROOT` is not a CDB-wide limit. It is the default value for each PDB.

  In an Oracle RAC environment, different instances can use different values.

Related Topics

- MAX_AUTH_SERVERS
- MIN_AUTH_SERVERS
- ENABLE_PER_PDB_DRCP
- DRCP_CONNECTION_LIMIT

3.5.2 Enabling Database Resident Connection Pooling

Oracle Database includes a default connection pool called `SYS_DEFAULT_CONNECTION_POOL`. This connection pool must be started to enable database resident connection pooling.

> **Note:**
>
> If DRCP is configured at the PDB level, the connection pool is started when the PDB is opened. When a PDB is closed, its connection pool is stopped. The PDB administrator can stop, start, or modify the connection pool by using the `DBMS_CONNECTION_POOL` package.

To enable database resident connection pooling:

1. Start the database resident connection pool, as described in "Starting the Database Resident Connection Pool".
2. Route the client connection requests to the connection pool, as described in "Routing Client Connection Requests to the Connection Pool".

Starting the Database Resident Connection Pool

To start the connection pool:

1. Start SQL*Plus and connect to the database as the `SYS` user.
2. Issue the following command:

   ```
   SQL> EXECUTE DBMS_CONNECTION_POOL.START_POOL();
   ```

Once started, the connection pool remains in this state until it is explicitly stopped. The connection pool is automatically restarted when the database instance is restarted if the pool was active at the time of instance shutdown.

In an Oracle Real Application Clusters (Oracle RAC) environment, you can use any instance to manage the connection pool. Any changes you make to the pool configuration are applicable on all Oracle RAC instances.

Routing Client Connection Requests to the Connection Pool

In the client application, the connect string must specify the connect type as `POOLED`.

The following example shows an easy connect string that enables clients to connect to a database resident connection pool:

```
examplehost.company.com:1521/books.company.com:POOLED
```

The following example shows a TNS connect descriptor that enables clients to connect to a database resident connection pool:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=myhost)
         (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=sales)
         (SERVER=POOLED)))
```

Note:

Only the TCP protocol is supported for client connections to a database resident connection pool.

Disabling Database Resident Connection Pooling

To disable database resident connection pooling, you must explicitly stop the connection pool. Use the following steps:

1. Start SQL*Plus and connect to the database as the `SYS` user.
2. Issue the following command:

   ```
   SQL> EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
   ```

   See Also:

   Oracle Database PL/SQL Packages and Types Reference for more information on the `DBMS_CONNECTION_POOL` package.

Note:

The operation of disabling the database resident connection pool can be completed only when all client requests that have been handed off to a server are completed.

3.5.3 Configuring the Connection Pool for Database Resident Connection Pooling

The connection pool is configured using default parameter values. You can use the procedures in the `DBMS_CONNECTION_POOL` package to configure the connection pool according to your usage. In an Oracle Real Application Clusters (Oracle RAC) environment, the configuration parameters are applicable to each Oracle RAC instance.

Using the CONFIGURE_POOL Procedure

The `CONFIGURE_POOL` procedure of the `DBMS_CONNECTION_POOL` package enables you to configure the connection pool with advanced options. This procedure is usually used when you must modify all the parameters of the connection pool.

Using the ALTER_PARAM Procedure

The `ALTER_PARAM` procedure of the `DBMS_CONNECTION_POOL` package enables you to alter a specific configuration parameter without affecting other parameters.For example, the following command changes the minimum number of pooled servers used:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','MINSIZE','10');
```

The following example, changes the maximum number of connections that each connection broker can handle to 50000.

```
SQL> EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','MAXCONN_CBROK','50000');
```

Before you run this command, ensure that the maximum number of connections allowed by the platform on which your database is installed is not less than the value you set for `MAXCONN_CBROK`. Note that you cannot use this command when PDB-level connection pooling is configured.

For example, in Linux, the following entry in the `/etc/security/limits.conf` file indicates that the maximum number of connections allowed for the user `test_user` is 30000.

```
test_user HARD NOFILE 30000
```

To set the maximum number of connections that each connection broker can allow to 50000, first change the value in the `limits.conf` file to a value not less than 50000.

Restoring the Connection Pool Default Settings

If you have made changes to the connection pool parameters, but you want to revert to the default pool settings, use the `RESTORE_DEFAULT` procedure of the `DBMS_CONNECTION_POOL` package. The command to restore the connection pool to its default settings is:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.RESTORE_DEFAULTS();
```

- Configuration Parameters for Database Resident Connection Pooling
  You can specify parameters for subprograms in the `DBMS_CONNECTION_POOL` package to configure database resident connection pooling.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information on the `DBMS_CONNECTION_POOL` package.

3.5.3.1 Configuration Parameters for Database Resident Connection Pooling

You can specify parameters for subprograms in the `DBMS_CONNECTION_POOL` package to configure database resident connection pooling.

When DRCP is enabled at the database level, you must connect to the CDB root and then modify configuration parameters. When DRCP is configured at the PDB level, you must connect to the PDB as the PDB administrator and then modify configuration parameters.

Note:

When DRCP is enabled at the PDB level, the following parameters cannot be altered or set to the maximum value of 2147483647: `MINSIZE`, `NUM_CBROK`, and `MAXCONN_CBROK`.

The following table lists the parameters that you can configure for the connection pool.

Table 3-2 Configuration Parameters for Database Resident Connection Pooling

Parameter Name	Description
`MINSIZE`	The minimum number of pooled servers in the pool. The default value is 4 when configuring DRCP at the database level. If DRCP is confired at the PDB level, the default value is 0.
`MAXSIZE`	The maximum number of pooled servers in the pool. The default value is 40.
`INCRSIZE`	The number of pooled servers by which the pool is incremented if servers are unavailable when a client application request is received. The default value is 2.
`SESSION_CACHED_CURSORS`	The number of session cursors to cache in each pooled server session. The default value is 20.
`INACTIVITY_TIMEOUT`	The maximum time, in seconds, the pooled server can stay idle in the pool. After this time, the server is terminated. The default value is 300. This parameter does not apply if the pool is at `MINSIZE`.
`MAX_THINK_TIME`	The maximum time of inactivity, in seconds, for a client after it obtains a pooled server from the pool. After obtaining a pooled server from the pool, if the client application does not issue a database call for the time specified by `MAX_THINK_TIME`, then the pooled server is freed and the client connection is terminated. As a result, if a round trip call is attempted on such a connection, the application may encounter an `ORA-3113` or `ORA-3115` error.
`MAX_TXN_THINK_TIME`	The maximum time of inactivity, in seconds, for a client after it obtains a pooled server from the pool with an open transaction. After obtaining the pooled server from the pool, if the client application does not issue a database call for the time specified by `MAX_TXN_THINK_TIME`, then the pooled server is freed, and the client connection is terminated. The default value of this parameter is the value of the `MAX_THINK_TIME` parameter. Applications can set the value of the `MAX_TXN_THINK_TIME` parameter to a value higher than the `MAX_THINK_TIME` value to allow more time for the connections with open transactions.
`MAX_USE_SESSION`	The number of times a pooled server can be taken and released to the pool. The default value is 500000.
`MAX_LIFETIME_SESSION`	The time, in seconds, to live for a pooled server in the pool. The default value is 86400.

`NUM_CBROK`	The number of Connection Brokers that are created to handle client requests. The default value is 1.
	Creating multiple Connection Broker processes helps distribute the load of client connection requests if there are a large number of client applications.
	When using PDB-level connection pooling, the PDB administrator cannot modify the value of this parameter. It can only be modified by using the `CONNECTION_BROKERS` initialization parameter. For example:
	`ALTER SYSTEM SET`
	`CONNECTION_BROKERS='((TYPE=POOLED)(BROKERS=2)`
	`(CONNECTIONS=45000))'`

Table 3-2 (Cont.) Configuration Parameters for Database Resident Connection Pooling

Parameter Name	Description
`MAXCONN_CBROK`	The maximum number of connections that each Connection Broker can handle. The default value is 40000. But if the maximum connections allowed by the platform on which the database is installed is lesser than the default value, this value overrides the value set using `MAXCONN_CBROK`. Set the per-process file descriptor limit of the operating system sufficiently high so that it supports the number of connections specified by `MAXCONN_CBROK`. When using PDB-level connection pooling, you can modify the value of this parameter only by setting `CONNECTION_BROKERS` initialization parameter.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information on the `DBMS_CONNECTION_POOL` package.

3.5.4 Data Dictionary Views for Database Resident Connection Pooling

You can query data dictionary views to obtain information about your connection pool and to monitor the performance of database resident connection pooling.

To view information about all connection pools in the database, connect to the root as the CDB administrator. To view information about connection pools for a pluggable database (PDB), connect to the PDB as the PDB administrator and query the views. You can view statistics for your PDB only. The PDB administrator must be granted permissions to view the connection pooling data dictionary views.

Table 3-3 Data Dictionary Views for Database Resident Connection Pooling

View	Description
`DBA_CPOOL_INFO`	Contains information about the connection pool such as the pool status, the maximum and minimum number of connections, and timeout for idle sessions.
`V$CPOOL_CONN_INFO`	Contains information about each connection to the connection broker.
`V$CPOOL_STATS`	Contains pool statistics such as the number of session requests, number of times a session that matches the request was found in the pool, and total wait time for a session request.
`V$CPOOL_CC_INFO`	Contains information about the pool-to-connection class mapping for the pool.
`V$CPOOL_CC_STATS`	Contains connection class level statistics for the pool.

3.5.5 Determining the States of Connections in the Connection Pool

You can query the `V$CPOOL_CONN_INFO` view to determine the current state of each connection in the connection pool.

You can query this view for detailed information about the state of each connection. For example, you can determine which connections are busy or idle. To determine this information:

- Query the `V$CPOOL_CONN_INFO` view.

Example 3-1 Determining How Long Connections Have Been Waiting

The following query shows the wait time for connections in the `WAITING` state:

```
SELECT USERNAME, SERVICE, LAST_WAIT_TIME
   FROM V$CPOOL_CONN_INFO
   WHERE CONNECTION_STATUS = 'WAITING';
```

Example 3-2 Determining How Long Connections Have Been Active

The following query shows the amount of time each connection has been active for connections in the `ACTIVE` state:

```
SELECT USERNAME, SERVICE, LAST_ACTIVE_TIME
   FROM V$CPOOL_CONN_INFO
   WHERE CONNECTION_STATUS = 'ACTIVE';
```

Example 3-3 Listing the Longest Running Active Connections

The following query shows lists the connections that have been in the `ACTIVE` state the longest amount of time:

```
SELECT USERNAME, SERVICE, ACTIVE_TIME
   FROM V$CPOOL_CONN_INFO
   WHERE CONNECTION_STATUS = 'ACTIVE'
   ORDER BY ACTIVE_TIME DESC;
```

Example 3-4 Determining the Wait Time of the Oldest Connection in the Wait Queue

The following query shows the wait time for sessions in the `WAITING` state:

```
SELECT USERNAME, SERVICE, LAST_WAIT_TIME
   FROM V$CPOOL_CONN_INFO
   WHERE LAST_WAIT_TIME = (
      SELECT max(LAST_WAIT_TIME)
         FROM V$CPOOL_CONN_INFO
         WHERE CONNECTION_STATUS = 'WAITING');
```

3.6 About Oracle Database Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses **background processes**. Background processes consolidate functions that would otherwise be handled by multiple database programs running for each user process.

Background processes asynchronously perform I/O and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.

Table 3-4 describes the fundamental background processes, many of which are discussed in more detail elsewhere in this book. The use of additional database features or options can cause more background processes to be present. For example:

- When you use Oracle Database Advanced Queuing, the queue monitor (QMN*n*) background process is present.
- When you set the `FILE_MAPPING` initialization parameter to `true` for mapping data files to physical devices on a storage subsystem, the FMON process is present.
- If you use Oracle Automatic Storage Management (Oracle ASM), then additional Oracle ASM–specific background processes are present.

Table 3-4 Oracle Database Background Processes

Process Name	Description
Database writer (DBW*n* or BW*nn*)	The database writer writes modified blocks from the database buffer cache to the data files. Oracle Database allows a maximum of 100 database writer processes. The names of the first 36 database writer processes are DBW0-DBW9 and DBWa-DBWz. The names of the 37th through 100th database writer processes are BW36-BW99. The `DB_WRITER_PROCESSES` initialization parameter specifies the number of database writer processes. The database selects an appropriate default setting for this initialization parameter or adjusts a user-specified setting based on the number of CPUs and the number of processor groups. For more information about setting the `DB_WRITER_PROCESSES` initialization parameter, see the "*Oracle Database Performance Tuning Guide*".
Log writer (LGWR)	The log writer process writes redo log entries to disk. Redo log entries are generated in the redo log buffer of the system global area (SGA). LGWR writes the redo log entries sequentially into a redo log file. If the database has a multiplexed redo log, then LGWR writes the redo log entries to a group of redo log files. See " Managing the Redo Log" for information about the log writer process.
Checkpoint (CKPT)	At specific times, all modified database buffers in the system global area are written to the data files by DBW*n*. This event is called a checkpoint. The checkpoint process is responsible for signalling DBW*n* at checkpoints and updating all the data files and control files of the database to indicate the most recent checkpoint.
System monitor (SMON)	The system monitor performs recovery when a failed instance starts up again. In an Oracle Real Application Clusters database, the SMON process of one instance can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers terminated transactions skipped during system failure and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online.
Process monitor (PMON)	The process monitor performs process recovery when a user process fails. PMON is responsible for detecting processes that have failed. PMON is then responsible for coordinating cleanup performed by the CLMN process and the CL*`nn`* slaves. The cleanup frees resources that the process was using.
Archiver (ARC*n*)	One or more archiver processes copy the redo log files to archival storage when they are full or a log switch occurs. Archiver processes are the subject of " Managing Archived Redo Log Files".

Table 3-4 (Cont.) Oracle Database Background Processes

Process Name	Description
Recoverer (RECO)	The recoverer process is used to resolve distributed transactions that are pending because of a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions. For information about this process and how to start it, see " Managing Distributed Transactions".
Dispatcher (D*nnn*)	Dispatchers are optional background processes, present only when the shared server configuration is used. Shared server was discussed previously in "Configuring Oracle Database for Shared Server".

See Also:

Oracle Database Reference for a complete list of Oracle Database background processes

3.7 Managing Prespawned Processes

Oracle Database can prespawn processes for better client connection performance.

- About Managing Prespawned Processes
  Oracle Database can prespawn foreground and background processes in process pools.
- Managing Pools for Prespawned Processes
  You can use the `DBMS_PROCESS` package to configure and modify the number of prespawned processes in the foreground process pool.

3.7.1 About Managing Prespawned Processes

Oracle Database can prespawn foreground and background processes in process pools.

Oracle Database prespawns foreground processes when a dedicated broker is enabled or threaded execution mode is enabled. When a foreground process is required, it uses the prespawned processes internally to reduce the creation time. A database runs in threaded execution mode when the `THREADED_EXECUTION` initialization parameter is set to `TRUE`. When this parameter is set to `FALSE`, the default, the database runs in process mode, and Oracle Database does not prespawn foreground and background processes in process pools.

Client connection time can be more efficient when processes are prespawned. If threaded execution mode is enabled, then Oracle Database prespawns processes by default in various request pools. Each request pool is for a different kind of process. The `V$PROCESS_POOL` view shows information about these pools, and you can manage these pools using the `DBMS_PROCESS` package.

3.7.2 Managing Pools for Prespawned Processes

You can use the `DBMS_PROCESS` package to configure and modify the number of prespawned processes in the foreground process pool.

Oracle Database can create process pools to improve the efficiency of client connections. You can use the `DBMS_PROCESS` package to manage these pools. You can view the current process pools by querying the `V$PROCESS_POOL` view.

Process pools are created only if the database is running in the multithreaded Oracle Database model.

1. Connect to the database as a user with the required privileges.

   The user must have `SYSDBA` administrative privilege, and you must exercise this privilege using `AS SYSDBA` at connect time.

2. Run a subprogram in the `DBMS_PROCESS` package to manage a process pool.

Example 3-5 Stopping a Process Pool

```
SYS_DEFAULT_FOREGROUND_POOL

exec DBMS_PROCESS.STOP_POOL('SYS_DEFAULT_FOREGROUND_POOL');
```

The `ENABLED` column in the `V$PROCESS_POOL` view is `FALSE` for the process pool when it is stopped.

Example 3-6 Starting a Process Pool

```
SYS_DEFAULT_FOREGROUND_POOL

exec DBMS_PROCESS.START_POOL('SYS_DEFAULT_FOREGROUND_POOL');
```

The `ENABLED` column in the `V$PROCESS_POOL` view is `TRUE` for the process pool when it is enabled.

Example 3-7 Configuring a Process Pool

You can check the current configuration of a process pool by querying the `V$PROCESS_POOL` view. For example, the following query shows the current configuration of the process pools:

```
COLUMN POOL_NAME FORMAT A30
COLUMN ENABLED FORMAT A7
COLUMN MIN_COUNT FORMAT 9999999
COLUMN BATCH_COUNT FORMAT 9999999
COLUMN INIT_COUNT FORMAT 9999999

SELECT POOL_NAME, ENABLED, MIN_COUNT, BATCH_COUNT, INIT_COUNT
   FROM V$PROCESS_POOL;
```

Assume the results are the following:

```
POOL_NAME                      ENABLED MIN_COUNT BATCH_COUNT INIT_COUNT
------------------------------ ------- --------- ----------- ----------
SYS_DEFAULT_FOREGROUND_POOL    TRUE           10          20         29
```

For this process pool, to change the minimum number of prespawned process to 20, the number of prespawned processes created in a batch to 30, and the initial number of prespawned processes to 40, run the following procedure:

```
BEGIN
   DBMS_PROCESS.CONFIGURE_POOL(
      POOL_NAME   => 'SYS_DEFAULT_FOREGROUND_POOL',
      MIN_COUNT   => 20,
      BATCH_COUNT => 30,
      INIT_COUNT  => 40);
END;
/
```

You can confirm your changes by running the query again.

See Also:

- *Oracle Database Reference* for more information about the THREADED_EXECUTION initialization parameter
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_PROCESS package

3.8 Managing Processes for Parallel SQL Execution

You can manage parallel processing of SQL statements. In this configuration, Oracle Database can divide the work of processing an SQL statement among multiple parallel processes.

Note:

The parallel execution feature described in this section is available with the Oracle Database Enterprise Edition.

- About Parallel Execution Servers
  The execution of many SQL statements can be parallelized. The **degree of parallelism** is the number of parallel execution servers that can be associated with a single operation.
- Altering Parallel Execution for a Session
  You control parallel SQL execution for a session using the ALTER SESSION statement.

3.8.1 About Parallel Execution Servers

The execution of many SQL statements can be parallelized. The **degree of parallelism** is the number of parallel execution servers that can be associated with a single operation.

The degree of parallelism is determined by any of the following:

- A PARALLEL clause in a statement

- For objects referred to in a query, the `PARALLEL` clause that was used when the object was created or altered
- A parallel hint inserted into the statement
- A default determined by the database

An example of using parallel SQL execution is contained in "Parallelizing Table Creation".

When an instance starts up, Oracle Database creates a pool of parallel execution servers which are available for any parallel operation. A process called the **parallel execution coordinator** dispatches the execution of a pool of **parallel execution servers** and coordinates the sending of results from all of these parallel execution servers back to the user.

The parallel execution servers are enabled by default, because the `PARALLEL_MAX_SERVERS` initialization parameter value is set to greater than 0 by default. The processes are available for use by the various Oracle Database features that are capable of exploiting parallelism. Related initialization parameters are tuned by the database for the majority of users, but you can alter them as needed to suit your environment. For ease of tuning, some parameters can be altered dynamically.

Parallelism can be used by several features, including transaction recovery, replication, and SQL execution. In the case of parallel SQL execution, the topic discussed in this book, parallel execution server processes remain associated with a statement throughout its execution phase. When the statement is completely processed, these processes become available to process other statements.

Note:

To disable parallel SQL execution in a database, set the `PARALLEL_MAX_SERVERS` initialization parameter value to 0.

See Also:

- *Oracle Database SQL Tuning Guide* for information about using parallel hints
- *Oracle Database VLDB and Partitioning Guide* for more information about using parallel execution

3.8.2 Altering Parallel Execution for a Session

You control parallel SQL execution for a session using the `ALTER SESSION` statement.

- Disabling Parallel SQL Execution
  You disable parallel SQL execution with an `ALTER SESSION DISABLE PARALLEL DML|DDL|QUERY` statement. All subsequent DML (`INSERT`, `UPDATE`, `DELETE`), DDL (`CREATE`, `ALTER`), or query (`SELECT`) operations are executed serially after such a statement is issued. They will be executed serially regardless of any parallel attribute associated with the table or indexes involved. However, statements with a `PARALLEL` hint override the session settings.

- Enabling Parallel SQL Execution
  You enable parallel SQL execution with an `ALTER SESSION ENABLE PARALLEL DML|DDL|QUERY` statement. Subsequently, when a `PARALLEL` clause or parallel hint is associated with a statement, those DML, DDL, or query statements will execute in parallel. By default, parallel execution is enabled for DDL and query statements.
- Forcing Parallel SQL Execution
  You can force parallel execution of all subsequent DML, DDL, or query statements for which parallelization is possible with the `ALTER SESSION FORCE PARALLEL DML|DDL|QUERY` statement.

3.8.2.1 Disabling Parallel SQL Execution

You disable parallel SQL execution with an `ALTER SESSION DISABLE PARALLEL DML|DDL|QUERY` statement. All subsequent DML (`INSERT`, `UPDATE`, `DELETE`), DDL (`CREATE`, `ALTER`), or query (`SELECT`) operations are executed serially after such a statement is issued. They will be executed serially regardless of any parallel attribute associated with the table or indexes involved. However, statements with a `PARALLEL` hint override the session settings.

- Run the appropriate `ALTER SESSION DISABLE PARALLEL` statement to disable DML, DDL, or query operations.

For example, to disable parallel DDL operations, run the following statement:

```
ALTER SESSION DISABLE PARALLEL DDL;
```

3.8.2.2 Enabling Parallel SQL Execution

You enable parallel SQL execution with an `ALTER SESSION ENABLE PARALLEL DML|DDL|QUERY` statement. Subsequently, when a `PARALLEL` clause or parallel hint is associated with a statement, those DML, DDL, or query statements will execute in parallel. By default, parallel execution is enabled for DDL and query statements.

- Run the appropriate A`LTER SESSION DISABLE PARALLEL` statement to enable DML, DDL, or query operations.

For example, a DML statement can be parallelized only if you specifically issue an `ALTER SESSION` statement to enable parallel DML:

```
ALTER SESSION ENABLE PARALLEL DML;
```

3.8.2.3 Forcing Parallel SQL Execution

You can force parallel execution of all subsequent DML, DDL, or query statements for which parallelization is possible with the `ALTER SESSION FORCE PARALLEL DML|DDL|QUERY` statement.

You can force a specific degree of parallelism to be in effect, overriding any `PARALLEL` clause associated with subsequent statements. If you do not specify a degree of parallelism in the `ALTER SESSION` statement, the default degree of parallelism is used. Statement level parallel hints override the forced degree of parallelism. With table level parallel hints, the behavior depends on whether hints are provided for all tables. If all tables contain table-level parallel hints, the maximum value among these hints is used. If at least one table does not contain a table-level parallel hint, the degree of parallelism used is the highest value among the parallel hints and the degree of parallelism specified in the `ALTER SESSION` command .

To force parallel execution:

- Run an `ALTER SESSION FORCE PARALLEL` statement.

For example, the following statement forces parallel execution of subsequent statements and sets the overriding degree of parallelism to 5:

```
ALTER SESSION FORCE PARALLEL DDL PARALLEL 5;
```

3.9 Managing Processes for External Procedures

An external procedure is a procedure or function written in a programming language and stored in a shared library. An Oracle server can call external procedures or functions using PL/SQL routines.

- About External Procedures
  External procedures are procedures that are written in a programming language such as C, C++, or Java, compiled, and stored outside of the database, and then called by user sessions. For example, a PL/SQL program unit can call one or more C routines that are required to perform special-purpose processing.
- DBA Tasks to Enable External Procedure Calls
  To enable external procedure calls, you must modify the listener and manage libraries.

3.9.1 About External Procedures

External procedures are procedures that are written in a programming language such as C, C++, or Java, compiled, and stored outside of the database, and then called by user sessions. For example, a PL/SQL program unit can call one or more C routines that are required to perform special-purpose processing.

These callable routines are stored in a dynamic link library (DLL), or a libunit in the case of a Java class method, and are registered with the base language. Oracle Database provides a special-purpose interface, the **call specification** (call spec), that enables users to call external procedures.

When a user session calls an external procedure, the database starts an external procedure agent on the database host computer. The default name of the agent is `extproc`. Each session has its own dedicated agent. Optionally, you can create a credential so that the agent runs as a particular operating system user. When a session terminates, the database terminates its agent.

User applications pass to the external procedure agent the name of the DLL or libunit, the name of the external procedure, and any relevant parameters. The external procedure agent then loads the DLL or libunit, runs the external procedure, and passes back to the application any values returned by the external procedure.

See Also:

Oracle Database Development Guide for information about external procedures

3.9.2 DBA Tasks to Enable External Procedure Calls

To enable external procedure calls, you must modify the listener and manage libraries.

Enabling external procedure calls may involve the following DBA tasks:

- Configuring the listener to start the `extproc` agent

By default, the database starts the `extproc` process. Under the following circumstances, you must change this default configuration so that the listener starts the `extproc` process:

  – You want to use a multithreaded `extproc` agent
  – The database is running in shared server mode on Windows
  – An `AGENT` clause in the `LIBRARY` specification or an `AGENT IN` clause in the `PROCEDURE` or `FUNCTION` specification redirects external procedures to a different `extproc` agent

  Instructions for changing the default configuration are in *Oracle Database Development Guide*.

- Managing libraries or granting privileges related to managing libraries

  The database requires DLL statements to be accessed through a schema object called a library. For security purposes, by default, only users with the `DBA` role can create and manage libraries. Therefore, you may be asked to:

  – Create a directory object using the `CREATE DIRECTORY` statement for the location of the library. After the directory object is created, a `CREATE LIBRARY` statement can specify the directory object for the location of the library.
  – Create a credential using the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` PL/SQL procedure. After the credential is created, a `CREATE LIBRARY` statement can associate the credential with a library to run the `extproc` agent as a particular operating system user.
  – Use the `CREATE LIBRARY` statement to create the library objects that the developers need.
  – Grant the following privileges to developers: `CREATE LIBRARY`, `CREATE ANY LIBRARY`, `ALTER ANY LIBRARY`, `EXECUTE ANY LIBRARY`, `EXECUTE ON` *`library_name`*, and `EXECUTE ON` *`directory_object`*.

    Only make an explicit grant of these privileges to trusted users, and never to the `PUBLIC` role. If you plan to create PL/SQL interfaces to libraries, then only grant the `EXECUTE` privilege to the PL/SQL interface. Do not grant `EXECUTE` on the underlying library. You must have the `EXECUTE` object privilege on the library to create the PL/SQL interface. However, users have this privilege automatically in their own schemas. Explicit grants of `EXECUTE` object privilege on a library are rarely required.

See Also:

- *Oracle Database PL/SQL Language Reference* for information about the `CREATE LIBRARY` statement
- *Oracle Database Security Guide* for information about creating a credential using the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_CREDENTIAL` package
- "Specifying Scheduler Job Credentials" for information about using credentials with Oracle Scheduler jobs

3.10 Terminating Sessions

Sometimes it is necessary to terminate current user sessions. For example, you might want to perform an administrative operation and need to terminate all non-administrative sessions.

- About Terminating Sessions
  When a session is terminated, any active transactions of the session are rolled back, and resources held by the session (such as locks and memory areas) are immediately released and available to other sessions.
- Identifying Which Session to Terminate
  To identify which session to terminate, specify the session index number and serial number.
- Terminating an Active Session
  Terminating an active session ends the session.
- Terminating an Inactive Session
  If the session is not making a SQL call to Oracle Database (is `INACTIVE`) when it is terminated, the `ORA-00028` message is not returned immediately. The message is not returned until the user subsequently attempts to use the terminated session.
- Cancelling a SQL Statement in a Session
  You can cancel a SQL statement in a session using the `ALTER SYSTEM CANCEL SQL` statement.

3.10.1 About Terminating Sessions

When a session is terminated, any active transactions of the session are rolled back, and resources held by the session (such as locks and memory areas) are immediately released and available to other sessions.

You terminate a current session using the SQL statement `ALTER SYSTEM KILL SESSION`. The following statement terminates the session whose system identifier is 7 and serial number is 15:

```
ALTER SYSTEM KILL SESSION '7,15';
```

You can also use the `DBMS_SERVICE.DISCONNECT_SESSION` procedure to terminate sessions with a named service at the current instance.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DISCONNECT_SESSION` procedure

3.10.2 Identifying Which Session to Terminate

To identify which session to terminate, specify the session index number and serial number.

To identify the system identifier (SID) and serial number of a session:

- Query the `V$SESSION` dynamic performance view.

For example, the following query identifies all sessions for the user `jward`:

```
SELECT SID, SERIAL#, STATUS
  FROM V$SESSION
  WHERE USERNAME = 'JWARD';

SID    SERIAL#    STATUS
-----  ---------  --------
    7         15  ACTIVE
   12         63  INACTIVE
```

A session is `ACTIVE` when it is making a SQL call to Oracle Database. A session is `INACTIVE` if it is not making a SQL call to the database.

See Also:

Oracle Database Reference for a description of the status values for a session

3.10.3 Terminating an Active Session

Terminating an active session ends the session.

If a user session is processing a transaction (`ACTIVE` status) when you terminate the session, then the transaction is rolled back and the user immediately receives the following message:

```
ORA-00028: your session has been killed
```

If, after receiving the `ORA-00028` message, a user submits additional statements before reconnecting to the database, then Oracle Database returns the following message:

```
ORA-01012: not logged on
```

An active session cannot be interrupted when it is performing network I/O or rolling back a transaction. Such a session cannot be terminated until the operation completes. In this case, the session holds all resources until it is terminated. Additionally, the session that issues the `ALTER SYSTEM` statement to terminate a session waits up to 60 seconds for the session to be terminated. If the operation that cannot be interrupted continues past one minute, the issuer of the `ALTER SYSTEM` statement receives a message indicating that the session has been marked to be terminated. A session marked to be terminated is indicated in `V$SESSION` with a status of `KILLED` and a server that is something other than `PSEUDO`.

If you are using Application Continuity, then an active session's activity is recovered when the session terminates. If you do not want to recover a session after you terminate it, then you can include the `NOREPLAY` keyword in the `ALTER SYSTEM` statement. For example, the following statement specifies that the session will not be recovered:

```
ALTER SYSTEM KILL SESSION '7,15' NOREPLAY;
```

If you use the `DBMS_SERVICE.DISCONNECT_SESSION` procedure to terminate one or more sessions, then you can specify `DBMS_SERVICE.NOREPLAY` for the `disconnect_option` parameter to indicate that the sessions should not be recovered by Application Continuity. For example, to disconnect all sessions with the service `sales.example.com` and specify that the sessions should not be recovered, run the following procedure:

```
BEGIN
  DBMS_SERVICE.DISCONNECT_SESSION(
    service_name      => 'sales.example.com',
    disconnect_option => DBMS_SERVICE.NOREPLAY);
END;
/
```

See Also:

- "*Oracle Database SQL Language Reference*"
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DISCONNECT_SESSION` procedure

3.10.4 Terminating an Inactive Session

If the session is not making a SQL call to Oracle Database (is `INACTIVE`) when it is terminated, the `ORA-00028` message is not returned immediately. The message is not returned until the user subsequently attempts to use the terminated session.

When an inactive session has been terminated, the `STATUS` of the session in the `V$SESSION` view is `KILLED`. The row for the terminated session is removed from `V$SESSION` after the user attempts to use the session again and receives the `ORA-00028` message.

In the following example, an inactive session is terminated. First, `V$SESSION` is queried to identify the `SID` and `SERIAL#` of the session, and then the session is terminated.

```
SELECT SID,SERIAL#,STATUS,SERVER
   FROM V$SESSION
   WHERE USERNAME = 'JWARD';

SID    SERIAL#   STATUS     SERVER
-----  --------  ---------  ---------
    7        15  INACTIVE   DEDICATED
   12        63  INACTIVE   DEDICATED
2 rows selected.

ALTER SYSTEM KILL SESSION '7,15';
Statement processed.

SELECT SID, SERIAL#, STATUS, SERVER
   FROM V$SESSION
   WHERE USERNAME = 'JWARD';

SID    SERIAL#   STATUS     SERVER
-----  --------  ---------  ---------
    7        15  KILLED     PSEUDO
   12        63  INACTIVE   DEDICATED
2 rows selected.
```

3.10.5 Cancelling a SQL Statement in a Session

You can cancel a SQL statement in a session using the `ALTER SYSTEM CANCEL SQL` statement.

Instead of terminating a session, you can cancel a high-load SQL statement in a session. When you cancel a DML statement, the statement is rolled back.

The following clauses are required in an `ALTER SYSTEM CANCEL SQL` statement:

- `SID` – Session ID
- `SERIAL` – Session serial number

The following clauses are optional in an `ALTER SYSTEM CANCEL SQL` statement:

- `INST_ID` – Instance ID
- `SQL_ID` – SQL ID of the SQL statement

You can view this information for a session by querying the `GV$SESSION` view.

The following is the syntax for cancelling a SQL statement:

```
ALTER SYSTEM CANCEL SQL 'SID, SERIAL, @INST_ID, SQL_ID';
```

The following example cancels a SQL statement having the session identifier of `20`, session serial number of `51142`, and SQL ID of `8vu7s907prbgr`:

```
ALTER SYSTEM CANCEL SQL '20, 51142, 8vu7s907prbgr';
```

Note:

- If `@INST_ID` is not specified, the instance ID of the current session is used.
- If `SQL_ID` is not specified, the currently running SQL statement in the specified session is terminated.

See Also:

- *Oracle Database 2 Day + Performance Tuning Guide* for information about identifying high-load SQL statements
- *Oracle Database Reference* for information about the `GV$SESSION` view

3.11 Process and Session Data Dictionary Views

You can query data dictionary views for information about processes and sessions.

View	Description
`V$PROCESS`	Contains information about the currently active processes
`V$SESSION`	Lists session information for each current session
`V$SESS_IO`	Contains I/O statistics for each user session
`V$SESSION_LONGOPS`	Displays the status of various operations that run for longer than 6 seconds (in absolute time). These operations currently include many backup and recovery functions, statistics gathering, and query execution. More operations are added for every Oracle Database release.

View	Description
`V$SESSION_WAIT`	Displays the current or last wait for each session
`V$SESSION_WAIT_HISTORY`	Lists the last ten wait events for each active session
`V$WAIT_CHAINS`	Displays information about blocked sessions
`V$SESSTAT`	Contains session statistics
`V$RESOURCE_LIMIT`	Provides information about current and maximum global resource utilization for some system resources
`V$SQLAREA`	Contains statistics about shared SQL areas. Contains one row for each SQL string. Provides statistics about SQL statements that are in memory, parsed, and ready for execution

4 Managing Memory

Memory management involves maintaining optimal sizes for the Oracle Database instance memory structures as demands on the database change.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- About Memory Management
  The memory structures that must be managed are the system global area (SGA) and the instance program global area (instance PGA). Oracle Database supports various memory management methods, which are chosen by initialization parameter settings.
- Memory Architecture Overview
  Understand basic memory structures associated with Oracle Database.
- Using Automatic Memory Management
  You can allow the Oracle Database instance to automatically manage and tune memory for you.
- Configuring Memory Manually
  If you prefer to exercise more direct control over the sizes of individual memory components, you can disable automatic memory management and configure the database for manual memory management.
- Using Force Full Database Caching Mode
  An Oracle Database instance can cache the full database in the buffer cache.
- Configuring Database Smart Flash Cache
  The Database Smart Flash Cache feature is a transparent extension of the database buffer cache using solid state device (SSD) technology. Database Smart Flash Cache can greatly improve the performance of Oracle databases by reducing the amount of disk I/O at a much lower cost than adding an equivalent amount of RAM.
- Improving Query Response Time with the Server Result Cache
  The server result cache improves the performance of repetitive queries.
- Improving Query Performance with Oracle Database In-Memory
  Oracle Database In-Memory (Database In-Memory) is a suite of features, first introduced in Oracle Database 12*c* Release 1 (12.1.0.2), that greatly improves performance for real-time analytics and mixed workloads.
- Enabling High Performance Data Streaming with the Memoptimized Rowstore
  The Memoptimized Rowstore enables high performance data streaming for applications, such as Internet of Things (IoT) applications that typically stream small amounts of data in single-row inserts from a large number of clients simultaneously and also query data for clients at a very high frequency.

- Memory Management Reference
  Automatic memory management is supported only on some platforms. Also, you can query a set of data dictionary views for information on memory management.

4.1 About Memory Management

The memory structures that must be managed are the system global area (SGA) and the instance program global area (instance PGA). Oracle Database supports various memory management methods, which are chosen by initialization parameter settings.

Automatic Memory Management

Oracle Database can manage the SGA memory and instance PGA memory completely automatically. You designate only the total memory size to be used by the instance, and Oracle Database dynamically exchanges memory between the SGA and the instance PGA as needed to meet processing demands. This capability is referred to as *automatic memory management*. With this memory management method, the database also dynamically tunes the sizes of the individual SGA components and the sizes of the individual PGAs. Oracle recommends automatic memory management for databases where the total size of the SGA and PGA memory is less than or equal to four gigabytes.

Manual Memory Management

If you prefer to exercise more direct control over the sizes of individual memory components, you can disable automatic memory management and configure the database for manual memory management. There are a few different methods available for manual memory management. Some of these methods retain some degree of automation. The methods therefore vary in the amount of effort and knowledge required by the DBA. These methods are:

- Automatic shared memory management - for the SGA
- Manual shared memory management - for the SGA
- Automatic PGA memory management - for the instance PGA
- Manual PGA memory management - for the instance PGA

These memory management methods are described later in this chapter.

If you create your database with Database Configuration Assistant (DBCA) and choose the basic installation option, automatic memory management is enabled when system memory is less than or equal to 4 gigabytes. When system memory is greater than 4 gigabytes, automatic memory management is disabled, and automatic shared memory management is enabled. If you choose advanced installation, then DBCA enables you to select automatic memory management or automatic shared memory management.

Oracle recommends automatic shared memory management when the total size of the SGA and PGA memory is four gigabytes or larger.

Note:

The easiest way to manage memory is to use the graphical user interface of Oracle Enterprise Manager Database Express (EM Express) or Oracle Enterprise Manager Cloud Control (Cloud Control).

For information about managing memory with Cloud Control, see the Cloud Control online help.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

See Also:

Oracle Database Concepts for an introduction to the various automatic and manual methods of managing memory.

4.2 Memory Architecture Overview

Understand basic memory structures associated with Oracle Database.

The basic memory structures associated with Oracle Database include:

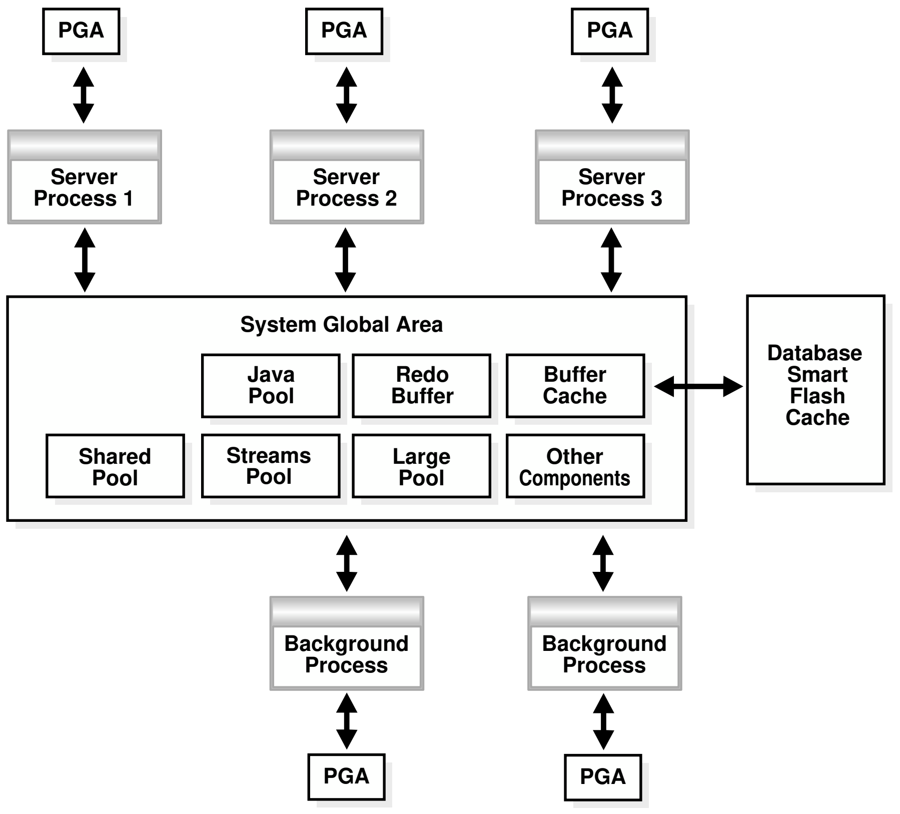
- System Global Area (SGA)

  The SGA is a group of shared memory structures, known as *SGA components*, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- Program Global Area (PGA)

  A PGA is a memory region that contains data and control information for a server process. It is nonshared memory created by Oracle Database when a server process is started. Access to the PGA is exclusive to the server process. There is one PGA for each server process. Background processes also allocate their own PGAs. The total PGA memory allocated for all background and server processes attached to an Oracle Database instance is referred to as the **total instance PGA memory**, and the collection of all individual PGAs is referred to as the **total instance PGA**, or just **instance PGA**.

Figure 4-1 illustrates the relationships among these memory structures.

Figure 4-1   Oracle Database Memory Structures



If your database is running on Solaris or Oracle Linux, you can optionally add another memory component: Database Smart Flash Cache. Database Smart Flash Cache is an extension of the SGA-resident buffer cache, providing a level 2 cache for database blocks. It can improve response time and overall throughput for both read-intensive online transaction processing (OLTP) workloads and ad hoc queries and bulk data modifications in a data warehouse environment. Database Smart Flash Cache resides on one or more flash disk devices, which are solid state storage devices that use flash memory. Database Smart Flash Cache is typically more economical than additional main memory, and is an order of magnitude faster than disk drives.

Starting with Oracle Database 12*c* Release 1 (12.1.0.2), the big table cache enables serial queries and parallel queries to use the buffer cache. The big table cache facilitates efficient caching for large tables in data warehousing environments, even if these tables do not fully fit in the buffer cache. Table scans can use the big table cache in the following scenarios:

- Parallel queries

  In single-instance and Oracle Real Application Clusters (Oracle RAC) databases, parallel queries can use the big table cache when the `DB_BIG_TABLE_CACHE_PERCENT_TARGET` initialization parameter is set to a non-zero value, and `PARALLEL_DEGREE_POLICY` is set to `AUTO` or `ADAPTIVE`.

- Serial queries

  In a single-instance configuration only, serial queries can use the big table cache when the `DB_BIG_TABLE_CACHE_PERCENT_TARGET` initialization parameter is set to a non-zero value.

See Also:

- "Configuring Database Smart Flash Cache"
- *Oracle Database Concepts* for more information on memory architecture in an Oracle Database instance
- *Oracle Database Reference* for more information about the `DB_BIG_TABLE_CACHE_PERCENT_TARGET` initialization parameter
- *Oracle Database Reference* for more information about the `PARALLEL_DEGREE_POLICY` initialization parameter
- *Oracle Database VLDB and Partitioning Guide* for more information about the big table cache

4.3 Using Automatic Memory Management

You can allow the Oracle Database instance to automatically manage and tune memory for you.

- About Automatic Memory Management
  The simplest way to manage instance memory is to allow the Oracle Database instance to automatically manage and tune it for you. To do so (on most platforms), you set only a *target* memory size initialization parameter (`MEMORY_TARGET`) and optionally a *maximum* memory size initialization parameter (`MEMORY_MAX_TARGET`).
- Enabling Automatic Memory Management
  If you did not enable automatic memory management upon database creation (either by selecting the proper options in DBCA or by setting the appropriate initialization parameters for the `CREATE DATABASE` SQL statement), then you can enable it at a later time. Enabling automatic memory management involves a shutdown and restart of the database.
- Monitoring and Tuning Automatic Memory Management
  The dynamic performance view `V$MEMORY_DYNAMIC_COMPONENTS` shows the current sizes of all dynamically tuned memory components, including the total sizes of the SGA and instance PGA.

4.3.1 About Automatic Memory Management

The simplest way to manage instance memory is to allow the Oracle Database instance to automatically manage and tune it for you. To do so (on most platforms), you set only a *target* memory size initialization parameter (`MEMORY_TARGET`) and optionally a *maximum* memory size initialization parameter (`MEMORY_MAX_TARGET`).

The total memory that the instance uses remains relatively constant, based on the value of `MEMORY_TARGET`, and the instance automatically distributes memory between the system global area (SGA) and the instance program global area (instance PGA). As memory requirements change, the instance dynamically redistributes memory between the SGA and instance PGA.

When automatic memory management is not enabled, you must size both the SGA and instance PGA manually.

Because the `MEMORY_TARGET` initialization parameter is dynamic, you can change `MEMORY_TARGET` at any time without restarting the database. `MEMORY_MAX_TARGET`, which is not dynamic, serves as an upper limit so that you cannot accidentally set MEMORY_TARGET too high, and so that enough memory is set aside for the database instance in case you do want to increase total instance memory in the future. Because certain SGA components either cannot easily shrink or must remain at a minimum size, the instance also prevents you from setting MEMORY_TARGET too low.

Note:

- If the total physical memory of a database instance is greater than 4 GB, then you cannot specify the Automatic Memory Management option during the database installation and creation. Oracle recommends that you use Automatic Shared Memory Management in such environments.
- You cannot enable automatic memory management if the LOCK_SGA initialization parameter is TRUE. See *Oracle Database Reference* for information about this parameter.

See Also:

"Platforms That Support Automatic Memory Management"

4.3.2 Enabling Automatic Memory Management

If you did not enable automatic memory management upon database creation (either by selecting the proper options in DBCA or by setting the appropriate initialization parameters for the CREATE DATABASE SQL statement), then you can enable it at a later time. Enabling automatic memory management involves a shutdown and restart of the database.

To enable automatic memory management:

1. Start SQL*Plus and connect to the Oracle Database instance with the SYSDBA administrative privilege.

   See "Connecting to the Database with SQL*Plus" and "Database Administrator Authentication" for instructions.

2. Calculate the minimum value for MEMORY_TARGET as follows:

   a. Determine the current sizes of SGA_TARGET and PGA_AGGREGATE_TARGET in megabytes by entering the following SQL*Plus commands:

```
SHOW PARAMETER SGA_TARGET

NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------------------
sga_target                           big integer 272M

SHOW PARAMETER PGA_AGGREGATE_TARGET

NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------------------
pga_aggregate_target                 big integer 90M
```

See "Enabling Automatic Shared Memory Management" for information about setting the `SGA_TARGET` parameter if it is not set.

b. Run the following query to determine the maximum instance PGA allocated in megabytes since the database was started:

```
SELECT VALUE/1048576 FROM V$PGASTAT WHERE NAME='maximum PGA allocated';
```

c. Compute the maximum value between the query result from step 2b and `PGA_AGGREGATE_TARGET`. Add `SGA_TARGET` to this value.

```
MEMORY_TARGET = SGA_TARGET + MAX(PGA_AGGREGATE_TARGET, MAXIMUM PGA ALLOCATED)
```

For example, if `SGA_TARGET` is 272M and `PGA_AGGREGATE_TARGET` is 90M as shown above, and if the maximum PGA allocated is determined to be 120M, then `MEMORY_TARGET` should be at least 392M (272M + 120M).

3. Choose the value for `MEMORY_TARGET` that you want to use.

   This can be the minimum value that you computed in step 2, or you can choose to use a larger value if you have enough physical memory available.

4. For the `MEMORY_MAX_TARGET` initialization parameter, decide on a maximum amount of memory that you would want to allocate to the database for the foreseeable future. That is, determine the maximum value for the sum of the SGA and instance PGA sizes. This number can be larger than or the same as the `MEMORY_TARGET` value that you chose in the previous step.

5. Do one of the following:

   - If you started your Oracle Database instance with a server parameter file, which is the default if you created the database with the Database Configuration Assistant (DBCA), enter the following command:

     ```
     ALTER SYSTEM SET MEMORY_MAX_TARGET = nM SCOPE = SPFILE;
     ```

     where *n* is the value that you computed in step 4.

     The `SCOPE = SPFILE` clause sets the value only in the server parameter file, and not for the running instance. You must include this `SCOPE` clause because `MEMORY_MAX_TARGET` is not a dynamic initialization parameter.

   - If you started your instance with a text initialization parameter file, manually edit the file so that it contains the following statements:

     ```
     memory_max_target = nM
     memory_target = mM
     ```

     where *n* is the value that you determined in step 4, and *m* is the value that you determined in step 3.

Note:

In a text initialization parameter file, if you omit the line for `MEMORY_MAX_TARGET` and include a value for `MEMORY_TARGET`, then the database automatically sets `MEMORY_MAX_TARGET` to the value of `MEMORY_TARGET`. If you omit the line for `MEMORY_TARGET` and include a value for `MEMORY_MAX_TARGET`, then the `MEMORY_TARGET` parameter defaults to zero. After startup, you can then dynamically change `MEMORY_TARGET` to a nonzero value, provided that it does not exceed the value of `MEMORY_MAX_TARGET`.

6. Shut down and restart the database.

   See *Oracle Database SQL Language Reference* for instructions.

7. If you started your Oracle Database instance with a server parameter file, enter the following commands:

   ```
   ALTER SYSTEM SET MEMORY_TARGET = nM;
   ALTER SYSTEM SET SGA_TARGET = 0;
   ALTER SYSTEM SET PGA_AGGREGATE_TARGET = 0;
   ```

   where *n* is the value that you determined in step 3.

Note:

With `MEMORY_TARGET` set, the `SGA_TARGET` setting becomes the minimum size of the SGA and the `PGA_AGGREGATE_TARGET` setting becomes the minimum size of the instance PGA. By setting both of these to zero as shown, there are no minimums, and the SGA and instance PGA can grow as needed as long as their sum is less than or equal to the `MEMORY_TARGET` setting. The sizing of SQL work areas remains automatic.

You can omit the statements that set the `SGA_TARGET` and `PGA_AGGREGATE_TARGET` parameter values to zero and leave either or both of the values as positive numbers. In this case, the values act as minimum values for the sizes of the SGA or instance PGA.

In addition, you can use the `PGA_AGGREGATE_LIMIT` initialization parameter to set an instance-wide hard limit for PGA memory. You can set `PGA_AGGREGATE_LIMIT` whether or not you use automatic memory management. See "Using Automatic PGA Memory Management".

See Also:

- "About Automatic Memory Management"
- "Memory Architecture Overview"
- *Oracle Database SQL Language Reference* for information on the `ALTER SYSTEM` SQL statement

4.3.3 Monitoring and Tuning Automatic Memory Management

The dynamic performance view `V$MEMORY_DYNAMIC_COMPONENTS` shows the current sizes of all dynamically tuned memory components, including the total sizes of the SGA and instance PGA.

- Query the `V$MEMORY_TARGET_ADVICE` view for tuning advice for the `MEMORY_TARGET` initialization parameter.

For example, run the following query:

```
SQL>  select * from v$memory_target_advice order by memory_size;

MEMORY_SIZE MEMORY_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR    VERSION
----------- ------------------ ------------ ------------------- ----------
        180                 .5          458               1.344          0
        270                .75          367              1.0761          0
        360                  1          341                   1          0
        450               1.25          335               .9817          0
        540                1.5          335               .9817          0
        630               1.75          335               .9817          0
        720                  2          335               .9817          0
```

The row with the `MEMORY_SIZE_FACTOR` of 1 shows the current size of memory, as set by the `MEMORY_TARGET` initialization parameter, and the amount of DB time required to complete the current workload. In previous and subsequent rows, the results show several alternative `MEMORY_TARGET` sizes. For each alternative size, the database shows the size factor (the multiple of the current size), and the estimated DB time to complete the current workload if the `MEMORY_TARGET` parameter were changed to the alternative size. Notice that for a total memory size smaller than the current `MEMORY_TARGET` size, estimated DB time increases. Notice also that in this example, there is nothing to be gained by increasing total memory size beyond 450MB. However, this situation might change if a complete workload has not yet been run.

EM Express provides an easy-to-use graphical memory advisor to help you select an optimal size for `MEMORY_TARGET`.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

See Also:

- *Oracle Database Reference* for more information about the `V$MEMORY_DYNAMIC_COMPONENTS` dynamic performance view
- *Oracle Database Reference* for more information about the `V$MEMORY_TARGET_ADVICE` dynamic performance view
- *Oracle Database Performance Tuning Guide* for a definition of DB time.

4.4 Configuring Memory Manually

If you prefer to exercise more direct control over the sizes of individual memory components, you can disable automatic memory management and configure the database for manual memory management.

- About Manual Memory Management
  There are two different manual memory management methods for the SGA, and two for the instance PGA.
- Using Automatic Shared Memory Management
  Automatic Shared Memory Management simplifies SGA memory management.

- Using Manual Shared Memory Management
  To manage shared memory manually, you first ensure that both automatic memory management and automatic shared memory management are disabled. You then manually configure, monitor, and tune memory components..
- Using Automatic PGA Memory Management
  By default, Oracle Database automatically and globally manages the total amount of memory dedicated to the instance PGA. You can control this amount by setting the initialization parameter `PGA_AGGREGATE_TARGET`.
- Using Manual PGA Memory Management
  Oracle Database supports manual PGA memory management, in which you manually tune SQL work areas.

4.4.1 About Manual Memory Management

There are two different manual memory management methods for the SGA, and two for the instance PGA.

The two manual memory management methods for the SGA vary in the amount of effort and knowledge required by the DBA. With *automatic shared memory management*, you set target and maximum sizes for the SGA. The database then sets the total size of the SGA to your designated target, and dynamically tunes the sizes of many SGA components. With *manual shared memory management*, you set the sizes of several individual SGA components, thereby determining the overall SGA size. You then manually tune these individual SGA components on an ongoing basis.

For the instance PGA, there is *automatic PGA memory management*, in which you set a target size for the instance PGA. The database then sets the size of the instance PGA to your target, and dynamically tunes the sizes of individual PGAs. There is also *manual PGA memory management*, in which you set maximum work area size for each type of SQL operator (such as sort or hash-join). This memory management method, although supported, is not recommended.

See Also:

Oracle Database Concepts for an overview of Oracle Database memory management methods.

4.4.2 Using Automatic Shared Memory Management

Automatic Shared Memory Management simplifies SGA memory management.

- About Automatic Shared Memory Management
  With automatic shared memory management, you specify the total amount of SGA memory available to an instance using the `SGA_TARGET` initialization parameter and Oracle Database automatically distributes this memory among the various SGA components to ensure the most effective memory utilization.
- Components and Granules in the SGA
  The SGA comprises several memory **components**, which are pools of memory used to satisfy a particular class of memory allocation requests.

- Setting Maximum SGA Size
  The `SGA_MAX_SIZE` initialization parameter specifies the maximum size of the System Global Area for the lifetime of the instance.
- Setting SGA Target Size
  You enable the automatic shared memory management feature by setting the `SGA_TARGET` initialization parameter to a nonzero value. This parameter sets the total size of the SGA. It replaces the parameters that control the memory allocated for a specific set of individual components, which are now automatically and dynamically resized (tuned) as needed.
- Enabling Automatic Shared Memory Management
  The procedure for enabling automatic shared memory management (ASMM) differs depending on whether you are changing to ASMM from manual shared memory management or from automatic memory management.
- Setting Minimums for Automatically Sized SGA Components
  You can exercise some control over the size of the automatically sized SGA components by specifying minimum values for the parameters corresponding to these components. Doing so can be useful if you know that an application cannot function properly without a minimum amount of memory in specific components.
- Dynamic Modification of SGA_TARGET
  The `SGA_TARGET` parameter can be dynamically increased up to the value specified for the `SGA_MAX_SIZE` parameter, and it can also be reduced.
- Modifying Parameters for Automatically Sized Components
  When automatic shared memory management is enabled, the manually specified sizes of automatically sized components serve as a lower bound for the size of the components. You can modify this limit dynamically by changing the values of the corresponding parameters.
- Modifying Parameters for Manually Sized Components
  Parameters for manually sized components can be dynamically altered as well. However, rather than setting a minimum size, the value of the parameter specifies the precise size of the corresponding component.

See Also:

- *Oracle Database Performance Tuning Guide* for information about tuning the components of the SGA

4.4.2.1 About Automatic Shared Memory Management

With automatic shared memory management, you specify the total amount of SGA memory available to an instance using the `SGA_TARGET` initialization parameter and Oracle Database automatically distributes this memory among the various SGA components to ensure the most effective memory utilization.

When automatic shared memory management is enabled, the sizes of the different SGA components are flexible and can adapt to the needs of a workload without requiring any additional configuration. The database automatically distributes the available memory among the various components as required, allowing the system to maximize the use of all available SGA memory.

If you are using a server parameter file (`SPFILE`), the database remembers the sizes of the automatically tuned SGA components across instance shutdowns. As a result, the database instance does not need to learn the characteristics of the workload again each time the instance is started. The instance can begin with information from the previous instance and continue evaluating workload where it left off at the last shutdown.

4.4.2.2 Components and Granules in the SGA

The SGA comprises several memory **components**, which are pools of memory used to satisfy a particular class of memory allocation requests.

Examples of memory components include the shared pool (used to allocate memory for SQL and PL/SQL execution), the java pool (used for java objects and other java execution memory), and the buffer cache (used for caching disk blocks). All SGA components allocate and deallocate space in units of **granules**. Oracle Database tracks SGA memory use in internal numbers of granules for each SGA component.

The memory for dynamic components in the SGA is allocated in the unit of granules. The granule size is determined by the amount of SGA memory requested when the instance starts. Specifically, the granule size is based on the value of the `SGA_MAX_SIZE` initialization parameter. Table 4-1 shows the granule size for different amounts of SGA memory.

Table 4-1 Granule Size

SGA Memory Amount	Granule Size
Less than or equal to 1 GB	4 MB
Greater than 1 GB and less than or equal to 8 GB	16 MB
Greater than 8 GB and less than or equal to 16 GB	32 MB
Greater than 16 GB and less than or equal to 32 GB	64 MB
Greater than 32 GB and less than or equal to 64 GB	128 MB
Greater than 64 GB and less than or equal to 128 GB	256 MB
Greater than 128 GB	512 MB

Some platform dependencies may arise. Consult your operating system specific documentation for more details.

You can query the `V$SGAINFO` view to see the granule size that is being used by an instance. The same granule size is used for all components in the SGA.

If you specify a size for a component that is not a multiple of granule size, Oracle Database rounds the specified size up to the nearest multiple. For example, if the granule size is 4 MB and you specify `DB_CACHE_SIZE` as 10 MB, the database actually allocates 12 MB.

4.4.2.3 Setting Maximum SGA Size

The `SGA_MAX_SIZE` initialization parameter specifies the maximum size of the System Global Area for the lifetime of the instance.

To set the maximum size of the System Global Area:

- Set the `SGA_MAX_SIZE` initialization parameter.

You can dynamically alter the initialization parameters affecting the size of the buffer caches, shared pool, large pool, Java pool, and streams pool but only to the extent that the sum of these sizes and the sizes of the other components of the SGA (fixed SGA, variable SGA, and redo log buffers) does not exceed the value specified by `SGA_MAX_SIZE`.

If you do not specify `SGA_MAX_SIZE`, then Oracle Database selects a default value that is the sum of all components specified or defaulted at initialization time. If you do specify `SGA_MAX_SIZE`, and at the time the database is initialized the value is less than the sum of the memory allocated for all components, either explicitly in the parameter file or by default, then the database ignores the setting for `SGA_MAX_SIZE` and chooses a correct value for this parameter.

4.4.2.4 Setting SGA Target Size

You enable the automatic shared memory management feature by setting the `SGA_TARGET` initialization parameter to a nonzero value. This parameter sets the total size of the SGA. It replaces the parameters that control the memory allocated for a specific set of individual components, which are now automatically and dynamically resized (tuned) as needed.

To enable the automatic shared memory management feature:

- Set the `SGA_TARGET` initialization parameter to a nonzero value.

Note:

- The `STATISTICS_LEVEL` initialization parameter must be set to `TYPICAL` (the default) or `ALL` for automatic shared memory management to function.
- An easier way to enable automatic shared memory management is to use EM Express. When you enable automatic shared memory management and set the Total SGA Size, EM Express automatically generates the `ALTER SYSTEM` statements to set `SGA_TARGET` to the specified size and to set all automatically sized SGA components to zero.

If you use SQL*Plus to set `SGA_TARGET`, then you must then set the automatically sized SGA components to zero or to a minimum value.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

- The SGA Target and Automatically Sized SGA Components
  Some SGA components are automatically sized when `SGA_TARGET` is set.
- SGA and Virtual Memory
  For optimal performance in most systems, the entire SGA should fit in real memory. If it does not, and if virtual memory is used to store parts of it, then overall database system performance can decrease dramatically. The reason for this is that portions of the SGA are paged (written to and read from disk) by the operating system.
- Monitoring and Tuning SGA Target Size
  The `V$SGAINFO` view provides information on the current tuned sizes of various SGA components. The `V$SGA_TARGET_ADVICE` view provides information that helps you decide on a value for `SGA_TARGET`.

4.4.2.4.1 The SGA Target and Automatically Sized SGA Components

Some SGA components are automatically sized when `SGA_TARGET` is set.

The following table lists the SGA components that are automatically sized when `SGA_TARGET` is set. For each SGA component, its corresponding initialization parameter is listed.

Table 4-2 Automatically Sized SGA Components and Corresponding Parameters

SGA Component	Initialization Parameter
Fixed SGA and other internal allocations needed by the Oracle Database instance	N/A
The shared pool	`SHARED_POOL_SIZE`
The large pool	`LARGE_POOL_SIZE`
The Java pool	`JAVA_POOL_SIZE`
The buffer cache	`DB_CACHE_SIZE`
The Streams pool	`STREAMS_POOL_SIZE`

The manually sized parameters listed in Table 4-3, if they are set, take their memory from `SGA_TARGET`, leaving what is available for the components listed in Table 4-2.

Table 4-3 Manually Sized SGA Components that Use SGA_TARGET Space

SGA Component	Initialization Parameter
The log buffer	`LOG_BUFFER`
The keep and recycle buffer caches	`DB_KEEP_CACHE_SIZE` `DB_RECYCLE_CACHE_SIZE`
Nonstandard block size buffer caches	`DB_`*`n`*`K_CACHE_SIZE`

In addition to setting `SGA_TARGET` to a nonzero value, you must set to zero all initialization parameters listed in Table 4-2 to enable full automatic tuning of the automatically sized SGA components.

Alternatively, you can set one or more of the automatically sized SGA components to a nonzero value, which is then used as the minimum setting for that component during SGA tuning. This is discussed in detail later in this section.

4.4.2.4.2 SGA and Virtual Memory

For optimal performance in most systems, the entire SGA should fit in real memory. If it does not, and if virtual memory is used to store parts of it, then overall database system performance can decrease dramatically. The reason for this is that portions of the SGA are paged (written to and read from disk) by the operating system.

See your operating system documentation for instructions for monitoring paging activity. You can also view paging activity using Cloud Control. See *Oracle Database 2 Day + Performance Tuning Guide* for more information.

4.4.2.4.3 Monitoring and Tuning SGA Target Size

The `V$SGAINFO` view provides information on the current tuned sizes of various SGA components. The `V$SGA_TARGET_ADVICE` view provides information that helps you decide on a value for `SGA_TARGET`.

To monitor and tune the SGA target size:

- Query the `V$SGAINFO` and `V$SGA_TARGET_ADVICE` views.

For example, run the following query:

```
SQL> select * from v$sga_target_advice order by sga_size;

  SGA_SIZE SGA_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR ESTD_PHYSICAL_READS
---------- --------------- ------------ ------------------- -------------------
       290              .5       448176              1.6578             1636103
       435             .75       339336              1.2552             1636103
       580               1       270344                   1             1201780
       725            1.25       239038               .8842              907584
       870             1.5       211517               .7824              513881
      1015            1.75       201866               .7467              513881
      1160               2       200703               .7424              513881
```

The information in this view is similar to that provided in the `V$MEMORY_TARGET_ADVICE` view for automatic memory management. See "Monitoring and Tuning Automatic Memory Management" for an explanation of that view.

EM Express provides an easy-to-use graphical memory advisor to help you select an optimal size for `SGA_TARGET`.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

See Also:

- *Oracle Database Reference* for more information about the `V$SGAINFO` view
- *Oracle Database Reference* for more information about the `V$SGA_TARGET_ADVICE` view

4.4.2.5 Enabling Automatic Shared Memory Management

The procedure for enabling automatic shared memory management (ASMM) differs depending on whether you are changing to ASMM from manual shared memory management or from automatic memory management.

To change to ASMM from manual shared memory management:

1. Run the following query to obtain a value for `SGA_TARGET`:

```
SELECT (
   (SELECT SUM(value) FROM V$SGA) -
   (SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY)
   ) "SGA_TARGET"
FROM DUAL;
```

2. Set the value of `SGA_TARGET`, either by editing the text initialization parameter file and restarting the database, or by issuing the following statement:

   ```
   ALTER SYSTEM SET SGA_TARGET=value [SCOPE={SPFILE|MEMORY|BOTH}]
   ```

   where *value* is the value computed in step 1 or is some value between the sum of all SGA component sizes and `SGA_MAX_SIZE`. For more information on the `ALTER SYSTEM` statement and its `SCOPE` clause, see *Oracle Database SQL Language Reference*.

3. Do one of the following:
   - For more complete automatic tuning, set the values of the automatically sized SGA components listed in Table 4-2 to zero. Do this by editing the text initialization parameter file or by issuing `ALTER SYSTEM` statements.
   - To control the minimum size of one or more automatically sized SGA components, set those component sizes to the desired value. (See the next section for details.) Set the values of the other automatically sized SGA components to zero. Do this by editing the text initialization parameter file or by issuing `ALTER SYSTEM` statements.

To change to ASMM from automatic memory management:

1. Set the `MEMORY_TARGET` initialization parameter to 0.

   ```
   ALTER SYSTEM SET MEMORY_TARGET = 0;
   ```

   The database sets `SGA_TARGET` based on current SGA memory allocation.

2. Do one of the following:
   - For more complete automatic tuning, set the sizes of the automatically sized SGA components listed in Table 4-2 to zero. Do this by editing the text initialization parameter file or by issuing `ALTER SYSTEM` statements.
   - To control the minimum size of one or more automatically sized SGA components, set those component sizes to the desired value. (See the next section for details.) Set the sizes of the other automatically sized SGA components to zero. Do this by editing the text initialization parameter file or by issuing `ALTER SYSTEM` statements.

Example 4-1 Using ASMM

For example, suppose you currently have the following configuration of parameters for an instance configured for manual shared memory management and with `SGA_MAX_SIZE` set to 1200M:

- `SHARED_POOL_SIZE` = 200M
- `DB_CACHE_SIZE` = 500M
- `LARGE_POOL_SIZE`=200M

Also assume the following query results:

Query	Result
`SELECT SUM(value) FROM V$SGA`	1200M

Query	Result
`SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY`	208M

You can take advantage of automatic shared memory management by issuing the following statements:

```
ALTER SYSTEM SET SGA_TARGET = 992M;
ALTER SYSTEM SET SHARED_POOL_SIZE = 0;
ALTER SYSTEM SET LARGE_POOL_SIZE = 0;
ALTER SYSTEM SET JAVA_POOL_SIZE = 0;
ALTER SYSTEM SET DB_CACHE_SIZE = 0;
ALTER SYSTEM SET STREAMS_POOL_SIZE = 0;
```

where 992M = 1200M minus 208M.

4.4.2.6 Setting Minimums for Automatically Sized SGA Components

You can exercise some control over the size of the automatically sized SGA components by specifying minimum values for the parameters corresponding to these components. Doing so can be useful if you know that an application cannot function properly without a minimum amount of memory in specific components.

To specify the minimum amount of SGA space for a component:

- Set a value for its corresponding initialization parameter.

Manually limiting the minimum size of one or more automatically sized components reduces the total amount of memory available for dynamic adjustment. This reduction in turn limits the ability of the system to adapt to workload changes. Therefore, this practice is not recommended except in exceptional cases. The default automatic management behavior maximizes both system performance and the use of available resources.

Related Topics

- The SGA Target and Automatically Sized SGA Components
  Some SGA components are automatically sized when `SGA_TARGET` is set.

4.4.2.7 Dynamic Modification of SGA_TARGET

The `SGA_TARGET` parameter can be dynamically increased up to the value specified for the `SGA_MAX_SIZE` parameter, and it can also be reduced.

If you reduce the value of `SGA_TARGET`, the system identifies one or more automatically tuned components for which to release memory. You can reduce `SGA_TARGET` until one or more automatically tuned components reach their minimum size. Oracle Database determines the minimum allowable value for `SGA_TARGET` taking into account several factors, including values set for the automatically sized components, manually sized components that use `SGA_TARGET` space, and number of CPUs.

The change in the amount of physical memory consumed when `SGA_TARGET` is modified depends on the operating system. On some UNIX platforms that do not support dynamic shared memory, the physical memory in use by the SGA is equal to the value of the `SGA_MAX_SIZE` parameter. On such platforms, there is no real benefit in setting `SGA_TARGET` to a value smaller than `SGA_MAX_SIZE`. Therefore, setting `SGA_MAX_SIZE` on those platforms is not recommended.

On other platforms, such as Solaris and Windows, the physical memory consumed by the SGA is equal to the value of `SGA_TARGET`.

For example, suppose you have an environment with the following configuration:

- `SGA_MAX_SIZE` = 1024M
- `SGA_TARGET` = 512M
- `DB_8K_CACHE_SIZE` = 128M

In this example, the value of `SGA_TARGET` can be resized up to 1024M and can also be reduced until one or more of the automatically sized components reaches its minimum size. The exact value depends on environmental factors such as the number of CPUs on the system. However, the value of `DB_8K_CACHE_SIZE` remains fixed at all times at 128M

> **Note:**
>
> When enabling automatic shared memory management, it is best to set `SGA_TARGET` to the desired nonzero value before starting the database. Dynamically modifying `SGA_TARGET` from zero to a nonzero value may not achieve the desired results because the shared pool may not be able to shrink. After startup, you can dynamically tune `SGA_TARGET` up or down as required.

4.4.2.8 Modifying Parameters for Automatically Sized Components

When automatic shared memory management is enabled, the manually specified sizes of automatically sized components serve as a lower bound for the size of the components. You can modify this limit dynamically by changing the values of the corresponding parameters.

If the specified lower limit for the size of a given SGA component is less than its current size, then there is no immediate change in the size of that component. The new setting only limits the automatic tuning algorithm to that reduced minimum size in the future.

To set the lower bound for the size of a component:

- Set the initialization parameter for the component to the minimum.

For example, consider the following configuration:

- `SGA_TARGET` = 512M
- `LARGE_POOL_SIZE` = 256M
- Current actual large pool size = 284M

In this example, if you increase the value of `LARGE_POOL_SIZE` to a value greater than the actual current size of the component, the system expands the component to accommodate the increased minimum size. For example, if you increase the value of `LARGE_POOL_SIZE` to 300M, then the system increases the large pool incrementally until it reaches 300M. This resizing occurs at the expense of one or more automatically tuned components. If you decrease the value of `LARGE_POOL_SIZE` to 200, there is no immediate change in the size of that component. The new setting only limits the reduction of the large pool size to 200 M in the future.

Note:

When `SGA_TARGET` is not set, the automatic shared memory management feature is not enabled. Therefore, the rules governing the resizing of all component parameters are the same as in earlier releases.

4.4.2.9 Modifying Parameters for Manually Sized Components

Parameters for manually sized components can be dynamically altered as well. However, rather than setting a minimum size, the value of the parameter specifies the precise size of the corresponding component.

When you increase the size of a manually sized component, extra memory is taken away from one or more automatically sized components. When you decrease the size of a manually sized component, the memory that is released is given to the automatically sized components.

To modify the precise size of a component:

- Set the initialization parameter for the component.

For example, consider this configuration:

- `SGA_TARGET` = 512M
- `DB_8K_CACHE_SIZE` = 128M

In this example, increasing `DB_8K_CACHE_SIZE` by 16M to 144M means that the 16M is taken away from the automatically sized components. Likewise, reducing `DB_8K_CACHE_SIZE` by 16M to 112M means that the 16M is given to the automatically sized components.

4.4.3 Using Manual Shared Memory Management

To manage shared memory manually, you first ensure that both automatic memory management and automatic shared memory management are disabled. You then manually configure, monitor, and tune memory components..

- About Manual Shared Memory Management
  If you decide not to use automatic memory management or automatic shared memory management, you must manually configure several SGA component sizes, and then monitor and tune these sizes on an ongoing basis as the database workload changes. You can follow guidelines on setting the parameters that control the sizes of these SGA components.
- Enabling Manual Shared Memory Management
  There is no initialization parameter that in itself enables manual shared memory management. You effectively enable manual shared memory management by disabling both automatic memory management and automatic shared memory management.
- Setting the Buffer Cache Initialization Parameters
  The buffer cache initialization parameters determine the size of the buffer cache component of the SGA.
- Specifying the Shared Pool Size
  The `SHARED_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the shared pool component of the SGA. Oracle Database selects an appropriate default value.

- Specifying the Large Pool Size
  The `LARGE_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the large pool component of the SGA.
- Specifying the Java Pool Size
  The `JAVA_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Java pool component of the SGA.
- Specifying the Streams Pool Size
  The `STREAMS_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Streams Pool component of the SGA.
- Specifying Miscellaneous SGA Initialization Parameters
  You can set a few additional initialization parameters to control how the SGA uses memory.

4.4.3.1 About Manual Shared Memory Management

If you decide not to use automatic memory management or automatic shared memory management, you must manually configure several SGA component sizes, and then monitor and tune these sizes on an ongoing basis as the database workload changes. You can follow guidelines on setting the parameters that control the sizes of these SGA components.

If you create your database with DBCA and choose manual shared memory management, DBCA provides fields where you must enter sizes for the buffer cache, shared pool, large pool, and Java pool. It then sets the corresponding initialization parameters in the server parameter file (`SPFILE`) that it creates. If you instead create the database with the `CREATE DATABASE` SQL statement and a text initialization parameter file, you can do one of the following:

- Provide values for the initialization parameters that set SGA component sizes.
- Omit SGA component size parameters from the text initialization file. Oracle Database chooses reasonable defaults for any component whose size you do not set.

4.4.3.2 Enabling Manual Shared Memory Management

There is no initialization parameter that in itself enables manual shared memory management. You effectively enable manual shared memory management by disabling both automatic memory management and automatic shared memory management.

To enable manual shared memory management:

1. Set the `MEMORY_TARGET` initialization parameter to 0.
2. Set the `SGA_TARGET` initialization parameter to 0.

You must then set values for the various SGA components, as described in the following sections.

4.4.3.3 Setting the Buffer Cache Initialization Parameters

The buffer cache initialization parameters determine the size of the buffer cache component of the SGA.

You use them to specify the sizes of caches for the various block sizes used by the database. These initialization parameters are all dynamic.

The size of a buffer cache affects performance. Larger cache sizes generally reduce the number of disk reads and writes. However, a large cache may take up too much memory and induce memory paging or swapping.

Oracle Database supports multiple block sizes in a database. If you create tablespaces with non-standard block sizes, you must configure non-standard block size buffers to accommodate these tablespaces. The standard block size is used for the `SYSTEM` tablespace. You specify the standard block size by setting the initialization parameter `DB_BLOCK_SIZE`. Legitimate values are from 2K to 32K.

If you intend to use multiple block sizes in your database, you must have the `DB_CACHE_SIZE` and at least one `DB_nK_CACHE_SIZE` parameter set. Oracle Database assigns an appropriate default value to the `DB_CACHE_SIZE` parameter, but the `DB_nK_CACHE_SIZE` parameters default to 0, and no additional block size caches are configured.

The sizes and numbers of non-standard block size buffers are specified by the following parameters:

```
DB_2K_CACHE_SIZE
DB_4K_CACHE_SIZE
DB_8K_CACHE_SIZE
DB_16K_CACHE_SIZE
DB_32K_CACHE_SIZE
```

Each parameter specifies the size of the cache for the corresponding block size.

Note:

- Platform-specific restrictions regarding the maximum block size apply, so some of these sizes might not be allowed on some platforms.
- A 32K block size is valid only on 64-bit platforms.

- Example of Setting Block and Cache Sizes
  An example illustrates setting block and cache sizes.
- Multiple Buffer Pools
  You can configure the database buffer cache with separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks.

See Also:

"Specifying Nonstandard Block Sizes for Tablespaces"

4.4.3.3.1 Example of Setting Block and Cache Sizes

An example illustrates setting block and cache sizes.

```
DB_BLOCK_SIZE=4096
DB_CACHE_SIZE=1024M
DB_2K_CACHE_SIZE=256M
DB_8K_CACHE_SIZE=512M
```

In the preceding example, the parameter `DB_BLOCK_SIZE` sets the standard block size of the database to 4K. The size of the cache of standard block size buffers is 1024MB. Additionally, 2K and 8K caches are also configured, with sizes of 256MB and 512MB, respectively.

Note:

The `DB_nK_CACHE_SIZE` parameters cannot be used to size the cache for the standard block size. If the value of `DB_BLOCK_SIZE` is *n*K, it is invalid to set `DB_nK_CACHE_SIZE`. The size of the cache for the standard block size is always determined from the value of `DB_CACHE_SIZE`.

The cache has a limited size, so not all the data on disk can fit in the cache. When the cache is full, subsequent cache misses cause Oracle Database to write dirty data already in the cache to disk to make room for the new data. (If a buffer is not dirty, it does not need to be written to disk before a new block can be read into the buffer.) Subsequent access to any data that was written to disk and then overwritten results in additional cache misses.

The size of the cache affects the likelihood that a request for data results in a cache hit. If the cache is large, it is more likely to contain the data that is requested. Increasing the size of a cache increases the percentage of data requests that result in cache hits.

You can change the size of the buffer cache while the instance is running, without having to shut down the database. Do this with the `ALTER SYSTEM` statement.

Use the fixed view `V$BUFFER_POOL` to track the sizes of the different cache components and any pending resize operations.

4.4.3.3.2 Multiple Buffer Pools

You can configure the database buffer cache with separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks.

Particular schema objects (tables, clusters, indexes, and partitions) can then be assigned to the appropriate buffer pool to control the way their data blocks age out of the cache.

- The `KEEP` buffer pool retains the schema object's data blocks in memory.
- The `RECYCLE` buffer pool eliminates data blocks from memory as soon as they are no longer needed.
- The `DEFAULT` buffer pool contains data blocks from schema objects that are not assigned to any buffer pool, as well as schema objects that are explicitly assigned to the `DEFAULT` pool.

The initialization parameters that configure the `KEEP` and `RECYCLE` buffer pools are `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE`.

Note:

Multiple buffer pools are only available for the standard block size. Non-standard block size caches have a single `DEFAULT` pool.

See Also:

Oracle Database Performance Tuning Guide for information about tuning the buffer cache and for more information about multiple buffer pools

4.4.3.4 Specifying the Shared Pool Size

The `SHARED_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the shared pool component of the SGA. Oracle Database selects an appropriate default value.

In releases before Oracle Database 10*g*, the amount of shared pool memory that was allocated was equal to the value of the `SHARED_POOL_SIZE` initialization parameter plus the amount of internal SGA overhead computed during instance startup. The internal SGA overhead refers to memory that is allocated by Oracle Database during startup, based on the values of several other initialization parameters. This memory is used to maintain state for different server components in the SGA. For example, if the `SHARED_POOL_SIZE` parameter is set to 64 MB and the internal SGA overhead is computed to be 12 MB, the real size of the shared pool is 64 + 12 = 76 MB, although the value of the `SHARED_POOL_SIZE` parameter is still displayed as 64 MB.

Starting with Oracle Database 10*g*, the size of the internal SGA overhead is included in the user-specified value of `SHARED_POOL_SIZE`. If you are not using automatic memory management or automatic shared memory management, the amount of shared pool memory that is allocated at startup is equal to the value of the `SHARED_POOL_SIZE` initialization parameter, rounded up to a multiple of the granule size. You must therefore set this parameter so that it includes the internal SGA overhead in addition to the desired value for shared pool size. In the previous example, if the `SHARED_POOL_SIZE` parameter is set to 64 MB at startup, then the available shared pool after startup is 64 - 12 = 52 MB, assuming the value of internal SGA overhead remains unchanged. In order to maintain an effective value of 64 MB for shared pool memory after startup, you must set the `SHARED_POOL_SIZE` parameter to 64 + 12 = 76 MB.

When migrating from a release earlier than Oracle Database 10*g*, the migration utilities recommend a new value for this parameter based on the value of internal SGA overhead in the pre-upgrade environment and based on the old value of this parameter. Beginning with Oracle Database 10*g*, the exact value of internal SGA overhead, also known as startup overhead in the shared pool, can be queried from the `V$SGAINFO` view. Also, in manual shared memory management mode, if the user-specified value of `SHARED_POOL_SIZE` is too small to accommodate even the requirements of internal SGA overhead, then Oracle Database generates an `ORA-00371` error during startup, with a suggested value to use for the `SHARED_POOL_SIZE` parameter. When you use automatic shared memory management, the shared pool is automatically tuned, and an `ORA-00371` error would not be generated.

- The Result Cache and Shared Pool Size

4.4.3.4.1 The Result Cache and Shared Pool Size

The result cache takes its memory from the shared pool. Therefore, if you expect to increase the maximum size of the result cache, take this into consideration when sizing the shared pool.

See Also:

"Specifying the Result Cache Maximum Size"

4.4.3.5 Specifying the Large Pool Size

The `LARGE_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the large pool component of the SGA.

The large pool is an optional component of the SGA. You must specifically set the `LARGE_POOL_SIZE` parameter to create a large pool. Configuring the large pool is discussed in *Oracle Database Performance Tuning Guide*.

4.4.3.6 Specifying the Java Pool Size

The `JAVA_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Java pool component of the SGA.

Oracle Database selects an appropriate default value. Configuration of the Java pool is discussed in *Oracle Database Java Developer's Guide*.

4.4.3.7 Specifying the Streams Pool Size

The `STREAMS_POOL_SIZE` initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Streams Pool component of the SGA.

If `STREAMS_POOL_SIZE` is set to 0, then the Oracle Streams product transfers memory from the buffer cache to the Streams Pool when it is needed. .

4.4.3.8 Specifying Miscellaneous SGA Initialization Parameters

You can set a few additional initialization parameters to control how the SGA uses memory.

- Physical Memory
  The `LOCK_SGA` parameter, when set to `TRUE`, locks the entire SGA into physical memory.
- SGA Starting Address
  The `SHARED_MEMORY_ADDRESS` and `HI_SHARED_MEMORY_ADDRESS` parameters specify the SGA's starting address at run time.

4.4.3.8.1 Physical Memory

The `LOCK_SGA` parameter, when set to `TRUE`, locks the entire SGA into physical memory.

This parameter cannot be used with automatic memory management.

See Also:

- *Oracle Database Reference* for more information on these initialization parameters
- "Using Automatic Memory Management"
- "Using Automatic Shared Memory Management"

4.4.3.8.2 SGA Starting Address

The `SHARED_MEMORY_ADDRESS` and `HI_SHARED_MEMORY_ADDRESS` parameters specify the SGA's starting address at run time.

These parameters are rarely used. For 64-bit platforms, `HI_SHARED_MEMORY_ADDRESS` specifies the high order 32 bits of the 64-bit address.

See Also:

- *Oracle Database Reference* for more information on the `SHARED_MEMORY_ADDRESS` initialization parameter
- *Oracle Database Reference* for more information on the `HI_SHARED_MEMORY_ADDRESS` initialization parameter
- "Using Automatic Memory Management"
- "Using Automatic Shared Memory Management"

4.4.4 Using Automatic PGA Memory Management

By default, Oracle Database automatically and globally manages the total amount of memory dedicated to the instance PGA. You can control this amount by setting the initialization parameter `PGA_AGGREGATE_TARGET`.

Oracle Database then tries to ensure that the total amount of PGA memory allocated across all database server processes and background processes never exceeds this target.

If you create your database with DBCA, you can specify a value for the total instance PGA. DBCA then sets the `PGA_AGGREGATE_TARGET` initialization parameters in the server parameter file (`SPFILE`) that it creates. If you do not specify the total instance PGA, DBCA chooses a reasonable default.

If you create the database with the `CREATE DATABASE` SQL statement and a text initialization parameter file, you can provide a value for `PGA_AGGREGATE_TARGET`. If you omit this parameter, the database chooses a default value for it.

With automatic PGA memory management, sizing of SQL work areas is automatic and all `*_AREA_SIZE` initialization parameters are ignored. At any given time, the total amount of PGA memory available to active work areas on the instance is automatically derived from the parameter `PGA_AGGREGATE_TARGET`. This amount is set to the value of `PGA_AGGREGATE_TARGET` minus the PGA memory allocated for other purposes (for example, session memory). The resulting PGA memory is then allotted to individual active work areas based on their specific memory requirements.

There are dynamic performance views that provide PGA memory use statistics. Most of these statistics are enabled when `PGA_AGGREGATE_TARGET` is set.

- Statistics on allocation and use of work area memory can be viewed in the following dynamic performance views:

  `V$SYSSTAT`

  `V$SESSTAT`

  `V$PGASTAT`

  `V$SQL_WORKAREA`

  `V$SQL_WORKAREA_ACTIVE`

- The following three columns in the `V$PROCESS` view report the PGA memory allocated and used by an Oracle Database process:

  `PGA_USED_MEM`

  `PGA_ALLOC_MEM`

  `PGA_MAX_MEM`

The `PGA_AGGREGATE_TARGET` setting is a target. Therefore, Oracle Database tries to limit PGA memory usage to the target, but usage can exceed the setting at times. To specify a hard limit on PGA memory usage, use the `PGA_AGGREGATE_LIMIT` initialization parameter. Oracle Database ensures that the PGA size does not exceed this limit. If the database exceeds the limit, then the database terminates calls from sessions that have the highest untunable PGA memory allocations. You can set `PGA_AGGREGATE_LIMIT` whether or not you use automatic memory management. If `PGA_AGGREGATE_LIMIT` is not set, then Oracle Database determines an appropriate default limit. See *Oracle Database Reference* for more information about this parameter.

Note:

The automatic PGA memory management method applies to work areas allocated by both dedicated and shared server process. See *Oracle Database Concepts* for information about PGA memory allocation in dedicated and shared server modes.

See Also:

- *Oracle Database Reference* for information about the initialization parameters and views described in this section
- *Oracle Database Performance Tuning Guide* for information about using the views described in this section

4.4.5 Using Manual PGA Memory Management

Oracle Database supports manual PGA memory management, in which you manually tune SQL work areas.

In releases earlier than Oracle Database 10*g*, the database administrator controlled the maximum size of SQL work areas by setting the following parameters: `SORT_AREA_SIZE`, `HASH_AREA_SIZE`, `BITMAP_MERGE_AREA_SIZE` and `CREATE_BITMAP_AREA_SIZE`. Setting these parameters is difficult, because the maximum work area size is ideally selected from the data input size and the total number of work areas active in the system. These two factors vary greatly from one work area to another and from one time to another. Thus, the various `*_AREA_SIZE` parameters are difficult to tune under the best of circumstances.

For this reason, Oracle strongly recommends that you leave automatic PGA memory management enabled.

If you decide to tune SQL work areas manually, you must set the `WORKAREA_SIZE_POLICY` initialization parameter to `MANUAL`.

Note:

The initialization parameter `WORKAREA_SIZE_POLICY` is a session- and system-level parameter that can take only two values: `MANUAL` or `AUTO`. The default is `AUTO`. You can set `PGA_AGGREGATE_TARGET`, and then switch back and forth from auto to manual memory management mode. When `WORKAREA_SIZE_POLICY` is set to `AUTO`, your settings for `*_AREA_SIZE` parameters are ignored.

4.5 Using Force Full Database Caching Mode

An Oracle Database instance can cache the full database in the buffer cache.

Note:

This feature is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

- About Force Full Database Caching Mode
  In default caching mode, Oracle Database does not always cache the underlying data when a user queries a large table because doing so might remove more useful data from the buffer cache. Starting with Oracle Database 12*c* Release 1 (12.1.0.2), if the Oracle Database instance determines that there is enough space to cache the full database in the buffer cache and that it would be beneficial to do so, then the instance automatically caches the full database in the buffer cache.
- Before Enabling Force Full Database Caching Mode
  The database must be at 12.0.0 or higher compatibility level to enable force full database caching mode for the database instance. In addition, ensure that the buffer cache is large enough to cache the entire database.
- Enabling Force Full Database Caching Mode
  You can enable force full database caching mode for a database.

- Disabling Force Full Database Caching Mode
  You can disable force full database caching mode for a database.

4.5.1 About Force Full Database Caching Mode

In default caching mode, Oracle Database does not always cache the underlying data when a user queries a large table because doing so might remove more useful data from the buffer cache. Starting with Oracle Database 12*c* Release 1 (12.1.0.2), if the Oracle Database instance determines that there is enough space to cache the full database in the buffer cache and that it would be beneficial to do so, then the instance automatically caches the full database in the buffer cache.

Caching the full database in the buffer cache might result in performance improvements. You can force an instance to cache the database in the buffer cache using an `ALTER DATABASE FORCE FULL DATABASE CACHING` statement. This statement puts the instance in force full database caching mode. In this mode, Oracle Database assumes that the buffer cache is large enough to cache the full database and tries to cache all blocks that are accessed subsequently.

When an Oracle Database instance is in force full database caching mode, the following query returns `YES`:

```
SELECT FORCE_FULL_DB_CACHING FROM V$DATABASE;
```

When an instance is in default caching mode, `NOCACHE` LOBs are not cached in the buffer cache. However, when an instance is in force full database caching mode, `NOCACHE` LOBs can be cached in the buffer cache. Also, both LOBs that use SecureFiles LOB storage and LOBs that use BasicFiles LOB storage can be cached in the buffer cache in force full database caching mode only.

Note:

- When an instance is put in force full database caching mode, database objects are not loaded into the buffer cache immediately. Instead, they are cached in the buffer cache when they are accessed.
- In a multitenant environment, force full database caching mode applies to the entire multitenant container database (CDB), including all of its pluggable databases (PDBs).
- Information about force full database caching mode is stored in the control file. If the control file is replaced or recreated, then the information about the force full database caching mode is lost. A restored control file might or might not include this information, depending on when the control file was backed up.

See Also:

- *Oracle Multitenant Administrator's Guide*
- "Managing Control Files"
- *Oracle Database Performance Tuning Guide* for information about when to use force full database caching mode

4.5.2 Before Enabling Force Full Database Caching Mode

The database must be at 12.0.0 or higher compatibility level to enable force full database caching mode for the database instance. In addition, ensure that the buffer cache is large enough to cache the entire database.

When a database is configured to use the `SGA_TARGET` or `MEMORY_TARGET` initialization parameter for automatic memory management, the size of the buffer cache might change depending on the workload. Run the following query to estimate the buffer cache size when the instance is under normal workload:

```
SELECT NAME, BYTES FROM V$SGAINFO WHERE NAME='Buffer Cache Size';
```

This query returns the buffer cache size for all possible block sizes. If your database uses multiple block sizes, then it is best to ensure that the buffer cache size for each possible block size is bigger than the total database size for that block size.

You can determine the buffer cache size for non-default block sizes with the `DB_`*`n`*`K_CACHE_SIZE` initialization parameter. With `SGA_TARGET` or `MEMORY_TARGET`, the buffer cache size for the default block size in the default pool might change depending on the workload. The following query returns the current buffer cache size for the default block size in the default pool:

```
SELECT COMPONENT, CURRENT_SIZE FROM V$SGA_DYNAMIC_COMPONENTS
   WHERE COMPONENT LIKE 'DEFAULT buffer cache';
```

If you are estimating memory requirements for running a database fully in the buffer cache, then you can estimate the size of the buffer cache as one of the following:

- If you plan to use `SGA_TARGET`, then you can estimate the buffer cache size as 60% of `SGA_TARGET`.
- If you plan to use `MEMORY_TARGET`, then you can estimate the SGA size as 60% of `MEMORY_TARGET`, and buffer cache size as 60% of SGA size. That is, you can estimate the buffer cache size as 36% of `MEMORY_TARGET`.

See Also:

"Using Automatic Memory Management"

4.5.3 Enabling Force Full Database Caching Mode

You can enable force full database caching mode for a database.

1. Connect to the instance as a user with `ALTER DATABASE` system privilege.
2. Ensure that the database is mounted but not open.

   See "*Oracle Database SQL Language Reference*".
3. Issue the following SQL statement:

   ```
   ALTER DATABASE FORCE FULL DATABASE CACHING;
   ```
4. (Optional) Open the database:

   ```
   ALTER DATABASE OPEN;
   ```

4.5.4 Disabling Force Full Database Caching Mode

You can disable force full database caching mode for a database.

1. Connect to the instance as a user with `ALTER DATABASE` system privilege.
2. Ensure that the database is mounted but not open.

   See "*Oracle Database SQL Language Reference*".
3. Issue the following SQL statement:

   ```
   ALTER DATABASE NO FORCE FULL DATABASE CACHING;
   ```
4. (Optional) Open the database:

   ```
   ALTER DATABASE OPEN;
   ```

4.6 Configuring Database Smart Flash Cache

The Database Smart Flash Cache feature is a transparent extension of the database buffer cache using solid state device (SSD) technology. Database Smart Flash Cache can greatly improve the performance of Oracle databases by reducing the amount of disk I/O at a much lower cost than adding an equivalent amount of RAM.

- When to Configure Database Smart Flash Cache
  You should consider configuring Database Smart Flash Cache when certain conditions are met.
- Sizing Database Smart Flash Cache
  As a general rule, size Database Smart Flash Cache to be between 2 times and 10 times the size of the buffer cache.
- Tuning Memory for Database Smart Flash Cache
  For each database block moved from the buffer cache to Database Smart Flash Cache, a small amount of metadata about the block is kept in the buffer cache.
- Database Smart Flash Cache Initialization Parameters
  You can use a set of initialization parameters to configure Database Smart Flash Cache.
- Database Smart Flash Cache in an Oracle Real Applications Clusters Environment
  Oracle recommends that you configure a Database Smart Flash Cache on either all or none of the instances in an Oracle Real Application Clusters environment. Also, the total flash cache size configured on each instance should be approximately the same.

See Also:

"Memory Architecture Overview" for a description of Database Smart Flash Cache

4.6.1 When to Configure Database Smart Flash Cache

You should consider configuring Database Smart Flash Cache when certain conditions are met.

Consider adding Database Smart Flash Cache when all of the following conditions are true:

- Your database is running on the Solaris or Oracle Linux operating systems. Database Smart Flash Cache is supported on these operating systems only.
- The Buffer Pool Advisory section of your Automatic Workload Repository (AWR) report or STATSPACK report indicates that doubling the size of the buffer cache would be beneficial.
- `db file sequential read` is a top wait event.
- You have spare CPU.

Note:

You cannot share one flash file among multiple instances. However, you can share a single flash device among multiple instances if you use a logical volume manager or similar tool to statically partition the flash device.

4.6.2 Sizing Database Smart Flash Cache

As a general rule, size Database Smart Flash Cache to be between 2 times and 10 times the size of the buffer cache.

Any multiplier less than two would not provide any benefit. If you are using automatic shared memory management, make Database Smart Flash Cache between 2 times and 10 times the size of `SGA_TARGET`. Using 80% of the size of `SGA_TARGET` instead of the full size would also suffice for this calculation.

4.6.3 Tuning Memory for Database Smart Flash Cache

For each database block moved from the buffer cache to Database Smart Flash Cache, a small amount of metadata about the block is kept in the buffer cache.

For a single instance database, the metadata consumes approximately 100 bytes. For an Oracle Real Application Clusters (Oracle RAC) database, it is closer to 200 bytes. You must therefore take this extra memory requirement into account when adding Database Smart Flash Cache.

To tune memory for the Database Smart Flash Cache, complete one of the following actions:

- If you are managing memory manually, then increase the size of the buffer cache by an amount approximately equal to the number of database blocks that fit into the Database Smart Flash Cache as configured, multiplied by 100 (or 200 for Oracle RAC).

- If you are using automatic memory management, then increase the size of the `MEMORY_TARGET` initialization parameter using the algorithm described above. You may first have to increase the size of the `MEMORY_MAX_TARGET` initialization parameter .
- If you are using automatic shared memory management, then increase the size of the `SGA_TARGET` initialization parameter .

Also, for an Oracle RAC database that uses the flash cache, additional memory must be allocated to the shared pool for Global Cache Service (GCS) resources. Each GCS resource requires approximately 208 bytes in the shared pool.

Note:

- You can choose to not increase the buffer cache size to account for Database Smart Flash Cache. In this case, the effective size of the buffer cache is reduced. In some cases, you can offset this loss by using a larger Database Smart Flash Cache.
- You can flush the Database Smart Flash Cache by issuing an `ALTER SYSTEM FLUSH FLASH_CACHE` statement. Flushing the Database Smart Flash Cache can be useful if you need to measure the performance of rewritten queries or a suite of queries from identical starting points, or if there might be corruption in the cache.

See Also:

"About Memory Management"

4.6.4 Database Smart Flash Cache Initialization Parameters

You can use a set of initialization parameters to configure Database Smart Flash Cache.

Table 4-4 Database Smart Flash Cache Initialization Parameters

Parameter	Description
`DB_FLASH_CACHE_FILE`	Specifies a list of paths and file names for the files to contain Database Smart Flash Cache, in either the operating system file system or an Oracle Automatic Storage Management disk group. If a specified file does not exist, then the database creates it during startup. Each file must reside on a flash device. If you configure Database Smart Flash Cache on a disk drive (spindle), then performance may suffer. A maximum of 16 files is supported.

Table 4-4 (Cont.) Database Smart Flash Cache Initialization Parameters

Parameter	Description
`DB_FLASH_CACHE_SIZE`	Specifies the size of each file in your Database Smart Flash Cache. Each size corresponds with a file specified in `DB_FLASH_CACHE_FILE`. The files and sizes correspond in the order that they are specified. An error is raised if the number of specified sizes does not match the number of specified files. Each size specification must be less than or equal to the physical memory size of its flash device. The size is expressed as *n*G, indicating the number of gigabytes (GB). For example, to specify a 16 GB Database Smart Flash Cache, set `DB_FLASH_CACHE_SIZE` value to `16G`.

For example, assume that your Database Smart Flash Cache uses following flash devices:

File	Size
/dev/sda	32G
/dev/sdb	32G
/dev/sdc	64G

You can set the initialization parameters to the following values:

```
DB_FLASH_CACHE_FILE = /dev/sda, /dev/sdb, /dev/sdc

DB_FLASH_CACHE_SIZE = 32G, 32G, 64G
```

You can query the `V$FLASHFILESTAT` view to determine the cumulative latency and read counts of each file and compute the average latency.

You can use `ALTER SYSTEM` to set `DB_FLASH_CACHE_SIZE` to zero for each flash device you wish to disable. You can also use `ALTER SYSTEM` to set the size for any disabled flash device back to its original size to reenable it. However, dynamically changing the size of Database Smart Flash Cache is not supported.

See Also:

Oracle Database Reference for more information about the initialization parameters described in this section and for more information about the `V$FLASHFILESTAT` view

4.6.5 Database Smart Flash Cache in an Oracle Real Applications Clusters Environment

Oracle recommends that you configure a Database Smart Flash Cache on either all or none of the instances in an Oracle Real Application Clusters environment. Also, the total flash cache size configured on each instance should be approximately the same.

4.7 Improving Query Response Time with the Server Result Cache

The server result cache improves the performance of repetitive queries.

- About the Server Result Cache
  The server result cache is a subcomponent of the shared pool.
- Using the Server Result Cache
  You control the use of server result cache by setting the `RESULT_CACHE_MODE` initialization parameter and using the `RESULT_CACHE` hint.
- Specifying the Result Cache Maximum Size
  The `RESULT_CACHE_MAX_SIZE` initialization parameter is a dynamic parameter that enables you to specify the maximum size of the result cache component of the SGA.
- Specifying the Use of Temporary Segments for Query Results
  You can specify a per-query limit on memory usage by setting the `RESULT_CACHE_MAX_SIZE` and `RESULT_CACHE_MAX_RESULT` initialization parameters. If a query exceeds the limit, the database server can store part of the results as a temporary segment in the `SYS` user's default temporary tablespace.

4.7.1 About the Server Result Cache

The server result cache is a subcomponent of the shared pool.

The server result cache is a memory pool within the shared pool that contains the SQL query result cache and PL/SQL function result cache. The SQL query result cache stores the results of queries and query fragments. Frequently executed queries will see performance improvements when using the SQL query result cache. The PL/SQL function result cache stores function result sets. Frequently invoked functions that depend on relatively static data are good candidates for result caching.

4.7.2 Using the Server Result Cache

You control the use of server result cache by setting the `RESULT_CACHE_MODE` initialization parameter and using the `RESULT_CACHE` hint.

The `RESULT_CACHE_MODE` initialization parameter determines whether the SQL query result cache is used for all queries (when possible) or only for annotated queries. Users can annotate a query or query fragment with a `RESULT_CACHE` hint to indicate that results should be stored in the SQL query result cache.

See Also:

- *Oracle Database Reference* to learn more about the `RESULT_CACHE_MODE` initialization parameter
- *Oracle Database SQL Language Reference* to learn about the `RESULT_CACHE` hint

4.7.3 Specifying the Result Cache Maximum Size

The `RESULT_CACHE_MAX_SIZE` initialization parameter is a dynamic parameter that enables you to specify the maximum size of the result cache component of the SGA.

Typically, there is no need to specify this parameter, because the default maximum size is chosen by the database based on total memory available to the SGA and on the memory management method currently in use. You can view the current default maximum size by displaying the value of the `RESULT_CACHE_MAX_SIZE` parameter. To change this maximum size, you can set `RESULT_CACHE_MAX_SIZE` with an `ALTER SYSTEM` statement, or you can specify this parameter in the text initialization parameter file. The value may be rounded up due to internal memory granularity.

If `RESULT_CACHE_MAX_SIZE` is 0 upon instance startup, the result cache is disabled. To reenable it you must set `RESULT_CACHE_MAX_SIZE` to a nonzero value (or remove this parameter from the text initialization parameter file to get the default maximum size) and then restart the database.

Note that after starting the database with the result cache disabled, if you use an `ALTER SYSTEM` statement to set `RESULT_CACHE_MAX_SIZE` to a nonzero value but do not restart the database, querying the value of the `RESULT_CACHE_MAX_SIZE` parameter returns a nonzero value even though the result cache is still disabled. The value of `RESULT_CACHE_MAX_SIZE` is therefore not the most reliable way to determine if the result cache is enabled. You can use the following query instead:

```
SELECT dbms_result_cache.status() FROM dual;

DBMS_RESULT_CACHE.STATUS()
---------------------------------------------
ENABLED
```

The result cache takes its memory from the shared pool, so if you increase the maximum result cache size, consider also increasing the shared pool size.

The view `V$RESULT_CACHE_STATISTICS` and the PL/SQL package procedure `DBMS_RESULT_CACHE.MEMORY_REPORT` display information to help you determine the amount of memory currently allocated to the result cache.

The PL/SQL package function `DBMS_RESULT_CACHE.FLUSH` clears the result cache and releases all the memory back to the shared pool.

See Also:

- *Oracle Database Performance Tuning Guide* for more information about the result cache
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_RESULT_CACHE` package procedures and functions
- *Oracle Database Reference* for more information about the `V$RESULT_CACHE_STATISTICS` view
- *Oracle Real Application Clusters Administration and Deployment Guide* for information on setting `RESULT_CACHE_MAX_SIZE` for a cluster database

4.7.4 Specifying the Use of Temporary Segments for Query Results

You can specify a per-query limit on memory usage by setting the `RESULT_CACHE_MAX_SIZE` and `RESULT_CACHE_MAX_RESULT` initialization parameters. If a query exceeds the limit, the database server can store part of the results as a temporary segment in the `SYS` user's default temporary tablespace.

You can query `V$RESULT_CACHE_OBJECTS` to determine whether temporary segments have been used. A value of `Temp` in the `TYPE` column indicates the use of temporary segments.

You can use the following initialization parameters, alterable at the PDB level, to control the use of space by temporary segments:

- `RESULT_CACHE_MODE`: Set to `MANUAL_TEMP` or `FORCE_TEMP`. In either mode, all query results will be allowed to spill to temporary segments unless prohibited by a hint. The default is `MANUAL`, which means that query results will only be cached when queries explicitly use a result cache hint.
- `RESULT_CACHE_MAX_TEMP_SIZE`: Set to a value to limit the amount of space in that the result cache will consume in a database's temporary tablespace. The parameter value defaults to 10 times the default or initialized value of `RESULT_CACHE_MAX_SIZE`. This parameter can only be modified at the system level, not the session. In addition, any value below 5% of the System Global Area (SGA) size will be sanitized to that 5%. A value of 0, however, will disable the feature. It also cannot exceed 10% of the currently estimated total free temporary tablespace in the `SYS` schema, sanitizing the value to that max.
- `RESULT_CACHE_MAX_TEMP_RESULT`: Set to a value to limit the maximum amount of space in the temporary tablespace that one cached query can consume. The value defaults to 5% of the value of `RESULT_CACHE_MAX_TEMP_SIZE`. This parameter can only be modified at the system level, not the session.

See Also:

Oracle Database Performance Tuning Guide for more information about the result cache

4.8 Improving Query Performance with Oracle Database In-Memory

Oracle Database In-Memory (Database In-Memory) is a suite of features, first introduced in Oracle Database 12*c* Release 1 (12.1.0.2), that greatly improves performance for real-time analytics and mixed workloads.

The Database In-Memory features can drastically improve the performance of queries that do the following:

- Scan a large number of rows and apply filters that use operators such as `<`, `>`, `=`, and `IN`
- Select a small number of columns from a table or a materialized view having large number of columns, such as a query that accesses 5 out of 100 columns
- Select LOB columns using SQL operators

- Join small dimension tables with large fact tables
- Aggregate data

The Database In-Memory feature set includes the In-Memory Column Store (IM column store), advanced query optimizations, and availability solutions.

- **IM Column Store**

  The IM column store is the key feature of Database In-Memory. The IM column store maintains copies of tables, partitions, and individual columns in a special compressed columnar format that is optimized for rapid scans. The IM column store resides in the In-Memory Area, which is an optional portion of the system global area (SGA).

  The IM column store does not replace row-based storage or the database buffer cache, but supplements it. The database enables data to be in memory in both a row-based and columnar format, providing the best of both worlds. The IM column store provides an additional transaction-consistent copy of table data that is independent of the disk format.

- **Advanced Query Optimizations**

  Database In-Memory includes several performance optimizations for analytic queries:

  – In-Memory Expression (IM expression): Enables to identify and populate *hot* expressions in the IM column store.

  – Join Group: Enables to eliminate the performance overhead of decompressing and hashing column values.

  – In-Memory Aggregation (IM aggregation): Enhances performance of aggregation queries that join small dimension tables with large fact tables.

  – Repopulation: Enhances performance of queries by automatically repopulating the IM column store with the modified objects.

  – In-Memory Dynamic Scans (IM dynamic scans): Enhances performance of queries by automatically using lightweight threads to parallelize table scans when the CPU resources are idle.

- **High Availability Support**

  Database In-Memory includes the following availability features:

  – Reduces the time to populate data into the IM column store when a database instance restarts. This functionality is achieved using the In-Memory FastStart (IM FastStart) feature.

  – Provides the IM column store on each node in an Oracle Real Application Clusters (Oracle RAC) environment.

  – Provides the IM column store on standby databases in an Active Data Guard environment.

Note:

By default, Oracle Database In-Memory is disabled in an Oracle database. It can be enabled by setting the `INMEMORY_SIZE` initialization parameter to a value greater than 0. When Oracle Database In-Memory is enabled, Oracle Database Resource Manager (the Resource Manager) also gets enabled automatically.

See Also:

- *Oracle Database In-Memory Guide*
- Oracle Video: Managing Oracle Database In-Memory

4.9 Enabling High Performance Data Streaming with the Memoptimized Rowstore

The Memoptimized Rowstore enables high performance data streaming for applications, such as Internet of Things (IoT) applications that typically stream small amounts of data in single-row inserts from a large number of clients simultaneously and also query data for clients at a very high frequency.

The Memoptimized Rowstore provides the following functionality:

- Fast ingest

  Fast ingest optimizes the processing of high-frequency, single-row data inserts into a database. Fast ingest uses the large pool for buffering the inserts before writing them to disk, so as to improve data insert performance.
- Fast lookup

  Fast lookup enables fast retrieval of data from a database for high-frequency queries. Fast lookup uses a separate memory area in the SGA called the *memoptimize pool* for buffering the data queried from tables, so as to improve query performance.

  Note:

  For using fast lookup, you must allocate appropriate memory size to the memoptimize pool using the `MEMOPTIMIZE_POOL_SIZE` initialization parameter.

See Also:

- *Oracle Database Performance Tuning Guide* for information about configuring and using the Memoptimized Rowstore
- *Oracle Database Concepts* for information about the memoptimize pool memory architecture
- *Oracle Database Reference* for information about the `MEMOPTIMIZE_POOL_SIZE` initialization parameter

4.10 Memory Management Reference

Automatic memory management is supported only on some platforms. Also, you can query a set of data dictionary views for information on memory management.

- Platforms That Support Automatic Memory Management
  Some platforms support automatic memory management.
- Memory Management Data Dictionary Views
  A set of dynamic performance views provide information on memory management.

4.10.1 Platforms That Support Automatic Memory Management

Some platforms support automatic memory management.

The following platforms support automatic memory management—the Oracle Database ability to automatically tune the sizes of the SGA and PGA, redistributing memory from one to the other on demand to optimize performance:

- Linux
- Solaris
- Windows
- HP-UX
- AIX

4.10.2 Memory Management Data Dictionary Views

A set of dynamic performance views provide information on memory management.

View	Description
`V$SGA`	Displays summary information about the system global area (SGA).
`V$SGAINFO`	Displays size information about the SGA, including the sizes of different SGA components, the granule size, and free memory.
`V$SGASTAT`	Displays detailed information about how memory is allocated within the shared pool, large pool, Java pool, and Streams pool.
`V$PGASTAT`	Displays PGA memory usage statistics as well as statistics about the automatic PGA memory manager when it is enabled (that is, when `PGA_AGGREGATE_TARGET` is set). Cumulative values in `V$PGASTAT` are accumulated since instance startup.
`V$MEMORY_DYNAMIC_COMPONENTS`	Displays information on the current size of all automatically tuned and static memory components, with the last operation (for example, grow or shrink) that occurred on each.
`V$SGA_DYNAMIC_COMPONENTS`	Displays the current sizes of all SGA components, and the last operation for each component.
`V$SGA_DYNAMIC_FREE_MEMORY`	Displays information about the amount of SGA memory available for future dynamic SGA resize operations.
`V$MEMORY_CURRENT_RESIZE_OPS`	Displays information about resize operations that are currently in progress. A resize operation is an enlargement or reduction of the SGA, the instance PGA, or a dynamic SGA component.
`V$SGA_CURRENT_RESIZE_OPS`	Displays information about dynamic SGA component resize operations that are currently in progress.

View	Description
V$MEMORY_RESIZE_OPS	Displays information about the last 800 completed memory component resize operations, including automatic grow and shrink operations for SGA_TARGET and PGA_AGGREGATE_TARGET.
V$SGA_RESIZE_OPS	Displays information about the last 800 completed SGA component resize operations.
V$MEMORY_TARGET_ADVICE	Displays information that helps you tune MEMORY_TARGET if you enabled automatic memory management.
V$SGA_TARGET_ADVICE	Displays information that helps you tune SGA_TARGET.
V$PGA_TARGET_ADVICE	Displays information that helps you tune PGA_AGGREGATE_TARGET.
V$IM_SEGMENTS	Displays information about the storage allocated for all segments in the IM column store. **Note:** This view is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

5

Managing Users and Securing the Database

Establish a security policy for every database.

- The Importance of Establishing a Security Policy for Your Database
  It is important to develop a security policy for every database. The security policy establishes methods for protecting your database from accidental or malicious destruction of data or damage to the database infrastructure.
- Managing Users and Resources
  To connect to the database, each user must specify a valid user name that has been previously defined to the database. An account must have been established for the user, with information about the user being stored in the data dictionary.
- User Privileges and Roles
  Privileges and roles are used to control user access to data and the types of SQL statements that can be executed.
- Auditing Database Activity
  You can monitor and record selected user database actions, including those performed by administrators. You can monitor system-wide actions as well as actions performed on individual database objects. This type of monitoring is called database auditing.
- Predefined User Accounts
  Oracle Database includes several predefined user accounts.

5.1 The Importance of Establishing a Security Policy for Your Database

It is important to develop a security policy for every database. The security policy establishes methods for protecting your database from accidental or malicious destruction of data or damage to the database infrastructure.

Each database can have an administrator, referred to as the security administrator, who is responsible for implementing and maintaining the database security policy If the database system is small, the database administrator can have the responsibilities of the security administrator. However, if the database system is large, a designated person or group of people may have sole responsibility as security administrator.

For information about establishing security policies for your database, see *Oracle Database Security Guide*.

5.2 Managing Users and Resources

To connect to the database, each user must specify a valid user name that has been previously defined to the database. An account must have been established for the user, with information about the user being stored in the data dictionary.

When you create a database user (account), you specify the following attributes of the user:

- User name

- Authentication method
- Default tablespace
- Temporary tablespace
- Other tablespaces and quotas
- User profile

To learn how to create and manage users, see *Oracle Database Security Guide*.

5.3 User Privileges and Roles

Privileges and roles are used to control user access to data and the types of SQL statements that can be executed.

The table that follows describes the three types of privileges and roles:

Type	Description
System privilege	A system-defined privilege usually granted only by administrators. These privileges allow users to perform specific database operations.
Object privilege	A system-defined privilege that controls access to a specific object.
Role	A collection of privileges and other roles. Some system-defined roles exist, but most are created by administrators. Roles group together privileges and other roles, which facilitates the granting of multiple privileges and roles to users.

Privileges and roles can be granted to other users by users who have been granted the privilege to do so. The granting of roles and privileges starts at the administrator level. At database creation, the administrative user `SYS` is created and granted all system privileges and predefined Oracle Database roles. User `SYS` can then grant privileges and roles to other users, and also grant those users the right to grant specific privileges to others.

To learn how to administer privileges and roles for users, see *Oracle Database Security Guide*.

5.4 Auditing Database Activity

You can monitor and record selected user database actions, including those performed by administrators. You can monitor system-wide actions as well as actions performed on individual database objects. This type of monitoring is called database auditing.

You can create unified audit policies and manage these audit policies using SQL statements. Oracle Database provides default unified audit policies that contain the standard audit settings, and you can create custom unified audit policies. You can also create fine-grained audit policies using the `DBMS_FGA` PL/SQL package.

See Also:

Oracle Database Security Guide for more information about database auditing

Note:

Starting with Oracle Database Release 21c, traditional auditing is deprecated. Oracle recommends that you use unified auditing, which enables selective and more effective auditing inside Oracle Database.

5.5 Predefined User Accounts

Oracle Database includes several predefined user accounts.

The three types of predefined accounts are:

- Administrative accounts (`SYS`, `SYSTEM`, `SYSBACKUP`, `SYSDG`, `SYSKM`, `SYSRAC`, `SYSMAN`, and `DBSNMP`)

  `SYS`, `SYSTEM`, `SYSBACKUP`, `SYSDG`, `SYSKM`, and `SYSRAC` are described in "About Database Administrator Security and Privileges". `SYSMAN` is used to perform Oracle Enterprise Manager Cloud Control (Cloud Control) administration tasks. The management agent of Cloud Control uses the `DBSNMP` account to monitor and manage the database. You must not delete these accounts.
- Sample schema accounts

  These optional accounts are used for examples in Oracle Database documentation and instructional materials. The sample schema accounts are – `HR`, `SH`, and `OE`.
- Internal accounts

  These accounts are created so that individual Oracle Database features or components can have their own schemas. You must not delete internal accounts, and you must not attempt to log in with them.

Note:

Starting with Oracle Database 19c, most of the Oracle Database supplied user accounts, except `SYS` and sample schemas are *schema only* accounts, that is, these accounts are created without passwords. This prevents malicious users from logging into these accounts. You can assign passwords to these accounts whenever you want them to be authenticated, but Oracle recommends that for better security, you should change these accounts back to schema only accounts, when you do not need to authenticate them anymore.

See Also:

- *Oracle Database Security Guide* for information about all the predefined accounts provided by Oracle Database
- *Oracle Database Security Guide* for information about schema only accounts
- *Oracle Database Sample Schemas* for information about all the sample schemas provided by Oracle Database

6

Monitoring the Database

It is important that you monitor the operation of your database on a regular basis. Doing so not only informs you of errors that have not yet come to your attention but also gives you a better understanding of the normal operation of your database. Being familiar with normal behavior in turn helps you recognize when something is wrong.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- Monitoring Errors and Alerts
  You can monitor database errors and alerts to prevent, detect, and solve problems.
- Monitoring Performance
  Monitoring performance includes monitoring locks and wait events and querying a set of data dictionary views.
- Monitoring Quarantined Objects
  Object quarantine enables an Oracle database to function even when there are corrupted, unrecoverable objects. The `V$QUARANTINE` view contains information about quarantined objects.
- Automatically Monitoring Schema Objects
  Oracle Database can automatically track the activities and usage of certain schema objects, such as tables and materialized views.

6.1 Monitoring Errors and Alerts

You can monitor database errors and alerts to prevent, detect, and solve problems.

> **Note:**
>
> The easiest and best way to monitor the database for errors and alerts is with the Database Home page in Oracle Enterprise Manager Cloud Control (Cloud Control). See the Cloud Control online help for more information. This section provides alternate methods for monitoring, using data dictionary views, PL/SQL packages, and other command-line facilities.

- Monitoring Errors with Trace Files and the Alert Log
  A trace file is a file that contains diagnostic data used to investigate problems. An alert log is a file that provides a chronological log of database messages and errors.

- Monitoring a Database with Server-Generated Alerts
  A server-generated alert is a notification from the Oracle Database server of an impending problem.

6.1.1 Monitoring Errors with Trace Files and the Alert Log

A trace file is a file that contains diagnostic data used to investigate problems. An alert log is a file that provides a chronological log of database messages and errors.

- About Monitoring Errors with Trace Files and the Alert Log
  The trace file and alert log contain information about errors.
- Controlling the Size of an Alert Log
  To control the size of an alert log, you must manually delete the file when you no longer need it. Otherwise the database continues to append to the file.
- Controlling the Size of Trace Files
  You can control the maximum size of all trace files (excluding the alert log) using the initialization parameter `MAX_DUMP_FILE_SIZE`.
- Controlling When Oracle Database Writes to Trace Files
  Background processes always write to a trace file when appropriate.
- Reading the Trace File for Shared Server Sessions
  If shared server is enabled, each session using a dispatcher is routed to a shared server process, and trace information is written to the server trace file only if the session has enabled tracing (or if an error is encountered). Therefore, to track tracing for a specific session that connects using a dispatcher, you might have to explore several shared server trace files.

6.1.1.1 About Monitoring Errors with Trace Files and the Alert Log

The trace file and alert log contain information about errors.

Each server and background process can write to an associated **trace file**. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, and other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

Note:

Critical errors also create incidents and incident dumps in the Automatic Diagnostic Repository. See Diagnosing and Resolving Problems for more information.

The **alert log** is a chronological log of messages and errors, and includes the following items:

- All internal errors (`ORA-00600`), block corruption errors (`ORA-01578`), and deadlock errors (`ORA-00060`) that occur
- Administrative operations, such as some `CREATE`, `ALTER`, and `DROP` statements and `STARTUP`, `SHUTDOWN`, and `ARCHIVELOG` statements
- Messages and errors relating to the functions of shared server and dispatcher processes
- Errors occurring during the automatic refresh of a materialized view

- The values of all initialization parameters that had nondefault values at the time the database and instance start

Oracle Database uses the alert log to record these operations as an alternative to displaying the information on an operator's console (although some systems also display information on the console). If an operation is successful, a "completed" message is written in the alert log, along with a timestamp.

The alert log is maintained as both an XML-formatted file and a text-formatted file. You can view either format of the alert log with any text editor or you can use the ADRCI utility to view the XML-formatted version of the file with the XML tags stripped.

Check the alert log and trace files of an instance periodically to learn whether the background processes have encountered errors. For example, when the log writer process (LGWR) cannot write to a member of a log group, an error message indicating the nature of the problem is written to the LGWR trace file and the alert log. Such an error message means that a media or I/O problem has occurred and should be corrected immediately.

Oracle Database also writes values of initialization parameters to the alert log, in addition to other important statistics.

The alert log and all trace files for background and server processes are written to the Automatic Diagnostic Repository, the location of which is specified by the `DIAGNOSTIC_DEST` initialization parameter. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file (such as LGWR and RECO).

See Also:

- "Diagnosing and Resolving Problems" for information about the Automatic Diagnostic Repository (ADR).
- "Alert Log" for additional information about the alert log.
- "Viewing the Alert Log"
- *Oracle Database Utilities* for information on the ADRCI utility.
- Your operating system specific Oracle documentation for information about the names of trace files

6.1.1.2 Controlling the Size of an Alert Log

To control the size of an alert log, you must manually delete the file when you no longer need it. Otherwise the database continues to append to the file.

You can safely delete the alert log while the instance is running, although you should consider making an archived copy of it first. This archived copy could prove valuable if you should have a future problem that requires investigating the history of an instance.

To control the size of an alert log:

- Delete the alert log file.

6.1.1.3 Controlling the Size of Trace Files

You can control the maximum size of all trace files (excluding the alert log) using the initialization parameter `MAX_DUMP_FILE_SIZE`.

You can set this parameter in the following ways:

- A numerical value specifies the maximum size in operating system blocks. The specified value is multiplied by the block size to obtain the limit.
- A number followed by a K, M, or G suffix specifies the file size in kilobytes, megabytes, or gigabytes.
- `UNLIMITED`, which is the default, specifies no limit.

- Trace File Segmentation and MAX_DUMP_FILE_SIZE
  Oracle Database can automatically segment trace files based on the limit you specify with the `MAX_DUMP_FILE_SIZE` initialization parameter. When a limit is reached, the database renames the current trace file using a sequential number, and creates an empty file with the original name.

See Also:

- *Oracle Database Reference* for more information about the `MAX_DUMP_FILE_SIZE` initialization parameter
- "About the Oracle Database Fault Diagnosability Infrastructure" for more information about IPS

6.1.1.3.1 Trace File Segmentation and MAX_DUMP_FILE_SIZE

Oracle Database can automatically segment trace files based on the limit you specify with the `MAX_DUMP_FILE_SIZE` initialization parameter. When a limit is reached, the database renames the current trace file using a sequential number, and creates an empty file with the original name.

The following table describes how trace files are segmented based on the `MAX_DUMP_FILE_SIZE` setting.

Table 6-1 The MAX_DUMP_FILE_SIZE Parameter and Trace File Segmentation

MAX_DUMP_FILE_SIZE Setting	Trace File Segmentation
`UNLIMITED`	Trace files are not segmented.
Larger than `15M`	Trace files are segmented on a boundary that is 1/5 of the `MAX_DUMP_FILE_SIZE` setting. Trace files with sizes that are less than this boundary in size are not segmented. For example, if the `MAX_DUMP_FILE_SIZE` setting is `100M`, then the boundary is 20 MB (1/5 of 100 MB).
`15M` or less	Trace files are not segmented.

There can be up to five segments, but the total combined size of the segments cannot exceed the `MAX_DUMP_FILE_SIZE` limit. When the combined size of all segments of the trace file exceeds the specified limit, the oldest segment after the first segment is deleted, and a new, empty segment is created. Therefore, the trace file always contains the most recent trace information. The first segment is not deleted because it might contain relevant information about the initial state of the process.

Segmentation improves space management for trace files. Specifically, segmentation enables you to manage trace files in the following ways:

- You can purge old trace files when they are no longer needed.
- You can diagnose problems with smaller trace files and isolate trace files that must be packaged for the incident packaging service (IPS).

Note:

Any segment that covers a time range that includes an incident is not deleted. It is kept in addition to the five default segments.

6.1.1.4 Controlling When Oracle Database Writes to Trace Files

Background processes always write to a trace file when appropriate.

In the case of the ARC*n* background process, it is possible, through the `LOG_ARCHIVE_TRACE` initialization parameter, to control the amount and type of trace information that is produced. To do so:

- Follow the instructions described in the section "Controlling Trace Output Generated by the Archivelog Process".

Other background processes do not have this flexibility.

Trace files are written on behalf of server processes whenever critical errors occur. Additionally, setting the initialization parameter `SQL_TRACE = TRUE` causes the SQL trace facility to generate performance statistics for the processing of all SQL statements for an instance and write them to the Automatic Diagnostic Repository.

Optionally, you can request that trace files be generated for server processes. Regardless of the current value of the `SQL_TRACE` initialization parameter, each session can enable or disable trace logging on behalf of the associated server process by using the SQL statement `ALTER SESSION SET SQL_TRACE`. This example enables the SQL trace facility for a specific session:

```
ALTER SESSION SET SQL_TRACE TRUE;
```

Use the `DBMS_SESSION` or the `DBMS_MONITOR` packages to control SQL tracing for a session.

Note:

The SQL trace facility for server processes can cause significant system overhead resulting in severe performance impact, so you should enable this feature only when collecting statistics.

See Also:

- "Diagnosing and Resolving Problems" for more information about how the database handles critical errors, otherwise known as *incidents*.

6.1.1.5 Reading the Trace File for Shared Server Sessions

If shared server is enabled, each session using a dispatcher is routed to a shared server process, and trace information is written to the server trace file only if the session has enabled tracing (or if an error is encountered). Therefore, to track tracing for a specific session that connects using a dispatcher, you might have to explore several shared server trace files.

To help you, Oracle provides a command line utility program, `trcsess`, which consolidates all trace information pertaining to a user session in one place and orders the information by time.

See Also:

Oracle Database SQL Tuning Guide for information about using the SQL trace facility and using `TKPROF` and `trcsess` to interpret the generated trace files

6.1.2 Monitoring a Database with Server-Generated Alerts

A server-generated alert is a notification from the Oracle Database server of an impending problem.

- About Monitoring a Database with Server-Generated Alerts
  A server-generated alert may contain suggestions for correcting the problem. Notifications are also provided when the problem condition has been cleared.
- Setting and Retrieving Thresholds for Server-Generated Alerts
  You can view and change threshold settings for the server alert metrics using the `SET_THRESHOLD` and `GET_THRESHOLD` procedures of the `DBMS_SERVER_ALERT` PL/SQL package.
- Viewing Server-Generated Alerts
  The easiest way to view server-generated alerts is by accessing the Database Home page of Cloud Control, but there are other methods of viewing these alerts.
- Server-Generated Alerts Data Dictionary Views
  You can query data dictionary views for information about server-generated alerts.

6.1.2.1 About Monitoring a Database with Server-Generated Alerts

A server-generated alert may contain suggestions for correcting the problem. Notifications are also provided when the problem condition has been cleared.

Alerts are automatically generated when a problem occurs or when data does not match expected values for metrics, such as the following:

- Physical Reads Per Second
- User Commits Per Second

- SQL Service Response Time

Server-generated alerts can be based on threshold levels or can issue simply because an event has occurred. Threshold-based alerts can be triggered at both threshold warning and critical levels. The value of these levels can be customer-defined or internal values, and some alerts have default threshold levels which you can change if appropriate. For example, by default a server-generated alert is generated for tablespace space usage when the percentage of space usage exceeds either the 85% warning or 97% critical threshold level. Examples of alerts not based on threshold levels are:

- `Snapshot Too Old`
- `Resumable Session Suspended`
- `Recovery Area Space Usage`

An alert message is sent to the predefined persistent queue `ALERT_QUE` owned by the user `SYS`. Cloud Control reads this queue and provides notifications about outstanding server alerts, and sometimes suggests actions for correcting the problem. The alerts are displayed on the Cloud Control Database Home page and can be configured to send email or pager notifications to selected administrators. If an alert cannot be written to the alert queue, a message about the alert is written to the Oracle Database alert log.

Background processes periodically flush the data to the Automatic Workload Repository to capture a history of metric values. The alert history table and `ALERT_QUE` are purged automatically by the system at regular intervals.

6.1.2.2 Setting and Retrieving Thresholds for Server-Generated Alerts

You can view and change threshold settings for the server alert metrics using the `SET_THRESHOLD` and `GET_THRESHOLD` procedures of the `DBMS_SERVER_ALERT` PL/SQL package.

Note:

The most convenient way to set and retrieve threshold values is to use the graphical interface of Cloud Control. See the Cloud Control online help about managing alerts for instructions.

- Setting Threshold Levels
  The `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can set threshold levels.
- Retrieving Threshold Information
  The `GET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can retrieve threshold information.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information about the `DBMS_SERVER_ALERT` package

6.1.2.2.1 Setting Threshold Levels

The `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can set threshold levels.

To set threshold levels:

- Run `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package, and specify the appropriate arguments.

The following example shows how to set thresholds with the `SET_THRESHOLD` procedure for CPU time for each user call for an instance:

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, DBMS_SERVER_ALERT.OPERATOR_GE, '8000',
 DBMS_SERVER_ALERT.OPERATOR_GE, '10000', 1, 2, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.example.com');
```

In this example, a warning alert is issued when CPU time exceeds 8000 microseconds for each user call and a critical alert is issued when CPU time exceeds 10,000 microseconds for each user call. The arguments include:

- `CPU_TIME_PER_CALL` specifies the metric identifier. For a list of support metrics, see *Oracle Database PL/SQL Packages and Types Reference*.
- The observation period is set to 1 minute. This period specifies the number of minutes that the condition must deviate from the threshold value before the alert is issued.
- The number of consecutive occurrences is set to 2. This number specifies how many times the metric value must violate the threshold values before the alert is generated.
- The name of the instance is set to `inst1`.
- The constant `DBMS_ALERT.OBJECT_TYPE_SERVICE` specifies the object type on which the threshold is set. In this example, the service name is `main.regress.rdbms.dev.us.example.com`.

6.1.2.2.2 Retrieving Threshold Information

The `GET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can retrieve threshold information.

To retrieve threshold values:

- Run the `GET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package and specify the appropriate arguments.

The following example retrieves threshold values:

```
DECLARE
 warning_operator        BINARY_INTEGER;
 warning_value           VARCHAR2(60);
 critical_operator       BINARY_INTEGER;
 critical_value          VARCHAR2(60);
 observation_period      BINARY_INTEGER;
 consecutive_occurrences BINARY_INTEGER;
BEGIN
 DBMS_SERVER_ALERT.GET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, warning_operator, warning_value,
    critical_operator, critical_value, observation_period,
    consecutive_occurrences, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.example.com');
 DBMS_OUTPUT.PUT_LINE('Warning operator:       ' || warning_operator);
 DBMS_OUTPUT.PUT_LINE('Warning value:          ' || warning_value);
 DBMS_OUTPUT.PUT_LINE('Critical operator:      ' || critical_operator);
 DBMS_OUTPUT.PUT_LINE('Critical value:         ' || critical_value);
 DBMS_OUTPUT.PUT_LINE('Observation_period:     ' || observation_period);
 DBMS_OUTPUT.PUT_LINE('Consecutive occurrences:' || consecutive_occurrences);
END;
/
```

You can also check specific threshold settings with the `DBA_THRESHOLDS` view. For example:

```
SELECT metrics_name, warning_value, critical_value, consecutive_occurrences
   FROM DBA_THRESHOLDS
   WHERE metrics_name LIKE '%CPU Time%';
```

6.1.2.3 Viewing Server-Generated Alerts

The easiest way to view server-generated alerts is by accessing the Database Home page of Cloud Control, but there are other methods of viewing these alerts.

If you use your own tool rather than Cloud Control to display alerts, then complete the following steps to view server-generated alerts:

1. Subscribe to the `ALERT_QUE`.
2. Read the `ALERT_QUE`.
3. Display an alert notification after setting the threshold levels for an alert

To create an agent and subscribe the agent to the `ALERT_QUE`, complete the following steps:

1. Run the `CREATE_AQ_AGENT` procedure of the `DBMS_AQADM` package.
2. Run the `ADD_SUBSCRIBER` procedure of the `DBMS_AQADM` package.
3. Associate a database user with the subscribing agent, because only a user associated with the subscribing agent can access queued messages in the secure `ALERT_QUE`.
4. Assign the enqueue privilege to the user by running the `ENABLE_DB_ACCESS` and `GRANT_QUEUE_PRIVILEGE` procedures of the `DBMS_AQADM` package.
5. Register with the `DBMS_AQ.REGISTER` procedure to receive an asynchronous notification when an alert is enqueued to `ALERT_QUE`. The notification can be in the form of email, HTTP post, or PL/SQL procedure.

To read an alert message, complete the following steps:

1. Use the `DBMS_AQ.DEQUEUE` procedure or `OCIAQDeq` call.
2. After the message has been dequeued, use the `DBMS_SERVER_ALERT.EXPAND_MESSAGE` procedure to expand the text of the message.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_AQ` package
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_AQADM` package

6.1.2.4 Server-Generated Alerts Data Dictionary Views

You can query data dictionary views for information about server-generated alerts.

View	Description
`DBA_THRESHOLDS`	Lists the threshold settings defined for the instance
`DBA_OUTSTANDING_ALERTS`	Describes the outstanding alerts in the database
`DBA_ALERT_HISTORY`	Lists a history of alerts that have been cleared
`V$ALERT_TYPES`	Provides information such as group and type for each alert
`V$METRICNAME`	Contains the names, identifiers, and other information about the system metrics
`V$METRIC`	Contains system-level metric values
`V$METRIC_HISTORY`	Contains a history of system-level metric values

6.2 Monitoring Performance

Monitoring performance includes monitoring locks and wait events and querying a set of data dictionary views.

Monitoring database performance is covered in detail in *Oracle Database Performance Tuning Guide* and *Oracle Database SQL Tuning Guide*.

- Monitoring Locks
  Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource. The resources can be either user objects, such as tables and rows, or system objects not visible to users, such as shared data structures in memory and data dictionary rows.
- About Monitoring Wait Events
  Wait events are statistics that are incremented by a server process to indicate that it had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write, or it could wait for a lock or latch.
- Performance Monitoring Data Dictionary Views
  You can query a set of data dictionary views to monitor an Oracle Database instance.

6.2.1 Monitoring Locks

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource. The resources can be either user objects, such as tables and rows, or system objects not visible to users, such as shared data structures in memory and data dictionary rows.

Oracle Database automatically obtains and manages necessary locks when executing SQL statements, so you need not be concerned with such details. However, the database also lets you lock data manually.

A deadlock can occur when two or more users are waiting for data locked by each other. Deadlocks prevent some transactions from continuing to work. Oracle Database automatically detects deadlock situations and resolves them by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks.

Oracle Database is designed to avoid deadlocks, and they are not common. Most often they occur when transactions explicitly override the default locking of the database. Deadlocks can affect the performance of your database, so Oracle provides some scripts and views that enable you to monitor locks.

To monitor locks:

1. Run the `catblock.sql`, which creates lock views.
2. Run the `utllockt.sql` script, which uses the views created by `catblock.sql` to display, in a tree fashion, the sessions in the system that are waiting for locks and the locks that they are waiting for.

The location of the script files is operating system dependent.

See Also:

- "Performance Monitoring Data Dictionary Views"
- *Oracle Database Concepts* contains more information about locks.

6.2.2 About Monitoring Wait Events

Wait events are statistics that are incremented by a server process to indicate that it had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write, or it could wait for a lock or latch.

When a session is waiting for resources, it is not doing any useful work. A large number of waits is a source of concern. Wait event data reveals various symptoms of problems that might be affecting performance, such as latch contention, buffer contention, and I/O contention.

Oracle provides several views that display wait event statistics. A discussion of these views and their role in instance tuning is contained in *Oracle Database Performance Tuning Guide*.

6.2.3 Performance Monitoring Data Dictionary Views

You can query a set of data dictionary views to monitor an Oracle Database instance.

These views are general in their scope. Other views, more specific to a process, are discussed in the section of this book where the process is described.

View	Description
`V$LOCK`	Lists the locks currently held by Oracle Database and outstanding requests for a lock or latch
`DBA_BLOCKERS`	Displays a session if it is holding a lock on an object for which another session is waiting
`DBA_WAITERS`	Displays a session if it is waiting for a locked object
`DBA_DDL_LOCKS`	Lists all DDL locks held in the database and all outstanding requests for a DDL lock
`DBA_DML_LOCKS`	Lists all DML locks held in the database and all outstanding requests for a DML lock

View	Description
`DBA_LOCK`	Lists all locks or latches held in the database and all outstanding requests for a lock or latch
`DBA_LOCK_INTERNAL`	Displays a row for each lock or latch that is being held, and one row for each outstanding request for a lock or latch
`V$LOCKED_OBJECT`	Lists all locks acquired by every transaction on the system
`V$SESSION_WAIT`	Lists the resources or events for which active sessions are waiting
`V$SYSSTAT`	Contains session statistics
`V$RESOURCE_LIMIT`	Provides information about current and maximum global resource utilization for some system resources
`V$SQLAREA`	Contains statistics about shared SQL area and contains one row for each SQL string. Also provides statistics about SQL statements that are in memory, parsed, and ready for execution
`V$LATCH`	Contains statistics for nonparent latches and summary statistics for parent latches

6.3 Monitoring Quarantined Objects

Object quarantine enables an Oracle database to function even when there are corrupted, unrecoverable objects. The `V$QUARANTINE` view contains information about quarantined objects.

- About Object Quarantine
  Object quarantine isolates an object that has raised an error and monitors the object for impacts on the system.
- Viewing Quarantined Objects
  The `V$QUARANTINE` view stores information about the objects that are currently quarantined.

6.3.1 About Object Quarantine

Object quarantine isolates an object that has raised an error and monitors the object for impacts on the system.

Some Oracle Database errors, such as `ORA-00600` and `ORA-07445`, typically cause the process to terminate, which can cause the database to terminate. When such an error is encountered, object quarantine attempts to isolate the resource that caused the error so that the database can continue to run. The resource is isolated in memory so that it does not affect the rest of the database. The `V$QUARANTINE` view stores information about the objects that are currently quarantined.

Most database resources can raise errors that can cause a database to terminate. For example, library cache memory objects can raise such errors.

In a multitenant environment, a multitenant container database (CDB) can, in some cases, use object quarantine to isolate and terminate a pluggable database (PDB) that has raised a serious error instead of terminating the CDB.

A quarantined resource typically remains quarantined until the database is restarted. If a resource is quarantined for a PDB in a CDB, then the resource is quarantined until the PDB is closed and re-opened.

6.3.2 Viewing Quarantined Objects

The `V$QUARANTINE` view stores information about the objects that are currently quarantined.

1. Connect to the database as an administrative user.
2. Query the `V$QUARANTINE` view.

Example 6-1 Querying the V$QUARANTINE View

This query shows the resources that are currently quarantined.

```
COLUMN OBJECT FORMAT A10
COLUMN ADDRESS FORMAT A10
COLUMN BYTES FORMAT 999999999
COLUMN ERROR FORMAT A20
COLUMN TIMESTAMP FORMAT A20

SELECT OBJECT, ADDRESS, BYTES, ERROR, TIMESTAMP
   FROM V$QUARANTINE;
```

Your output is similar to the following:

```
OBJECT     ADDRESS         BYTES ERROR                TIMESTAMP
---------- ---------- ---------- -------------------- --------------------
session    0000000078       9528 ORA-00600: internal  16-SEP-15 01.17.42.2
           B54BC8                error code, argument 85878 PM -07:00
                                 s: [12345], [], [],
                                 [], [], [], [], [],
                                 [], [], [], []
```

This output shows the following about the quarantined resource:

- The name of the resource is “session.”
- The start address of the memory region being quarantined is 0000000078B54BC8. Typically, this is the address of the resource, such as the session in this example.
- The resource is using 9528 bytes of memory in quarantine.
- The message of the error that caused the resource to be placed in quarantine is “ORA-00600 internal error code.”
- The timestamp shows the date and time of the error.

6.4 Automatically Monitoring Schema Objects

Oracle Database can automatically track the activities and usage of certain schema objects, such as tables and materialized views.

The Object Activity Tracking System (OATS) tracks various activities associated with database objects. Tracking can be performed both at the database level and pluggable database (PDB) level. The activities tracked include DML operations on tables, table and partition scans, partition maintenance operations (PMOPs), materialized view rewrite and refresh, and usage of auxiliary structures such as indexes. The statistics are used to support automated database functionality such as automatic materialized views.

Use procedures and functions in the `DBMS_ACTIVITY` package to control the information captured by OATS.

To enable Object Activity Tracking System:

- Set the `STATISTICS_LEVEL` initialization parameter to `TYPICAL` or `ALL`.

Statistics tracked by OATS can be viewed in the following data dictionary views: `DBA_ACTIVITY_CONFIG`, `DBA_ACTIVITY_SNAPSHOT_META`, `DBA_ACTIVITY_TABLE`, and `DBA_ACTIVITY_MVIEW`.

Related Topics

- *Oracle Database Data Warehousing Guide*

7

Diagnosing and Resolving Problems

Oracle Database includes an advanced fault diagnosability infrastructure for collecting and managing diagnostic data, so as to diagnose and resolve database problems. **Diagnostic data** includes the trace files, dumps, and core files that are also present in previous releases, plus new types of diagnostic data that enable customers and Oracle Support to identify, investigate, track, and resolve problems quickly and effectively.

- About the Oracle Database Fault Diagnosability Infrastructure
  Oracle Database includes a fault diagnosability infrastructure for preventing, detecting, diagnosing, and resolving database problems.
- About Investigating, Reporting, and Resolving a Problem
  You can use the Enterprise Manager Support Workbench (Support Workbench) to investigate and report a problem (critical error), and in some cases, resolve the problem. You can use a "roadmap" that summarizes the typical set of tasks that you must perform.
- Diagnosing Problems
  This section describes various methods to diagnose problems in an Oracle database.
- Reporting Problems
  Using the Enterprise Manager Support Workbench (Support Workbench), you can create, edit, and upload custom incident packages. With custom incident packages, you have fine control over the diagnostic data that you send to Oracle Support.
- Resolving Problems
  This section describes how to resolve database problems using advisor tools, such as SQL Repair Advisor and Data Recovery Advisor, and the resource management tools, such as the Resource Manager and related APIs.

7.1 About the Oracle Database Fault Diagnosability Infrastructure

Oracle Database includes a fault diagnosability infrastructure for preventing, detecting, diagnosing, and resolving database problems.

- Fault Diagnosability Infrastructure Overview
  The fault diagnosability infrastructure aids in preventing, detecting, diagnosing, and resolving problems. The problems that are targeted in particular are critical errors such as those caused by code bugs, metadata corruption, and customer data corruption.
- Incidents and Problems
  A **problem** is a critical error in a database instance, Oracle Automatic Storage Management (Oracle ASM) instance, or other Oracle product or component. An **incident** is a single occurrence of a problem.
- Fault Diagnosability Infrastructure Components
  The fault diagnosability infrastructure consists of several components, including the Automatic Diagnostic Repository (ADR), various logs, trace files, the Enterprise Manager Support Workbench, and the ADRCI Command-Line Utility.
- Structure, Contents, and Location of the Automatic Diagnostic Repository
  The Automatic Diagnostic Repository (ADR) is a directory structure that is stored outside of the database. It is therefore available for problem diagnosis when the database is down.

7.1.1 Fault Diagnosability Infrastructure Overview

The fault diagnosability infrastructure aids in preventing, detecting, diagnosing, and resolving problems. The problems that are targeted in particular are critical errors such as those caused by code bugs, metadata corruption, and customer data corruption.

When a critical error occurs, it is assigned an incident number, and diagnostic data for the error (such as trace files) are immediately captured and tagged with this number. The data is then stored in the Automatic Diagnostic Repository (ADR)—a file-based repository outside the database—where it can later be retrieved by incident number and analyzed.

The goals of the fault diagnosability infrastructure are the following:

- First-failure diagnosis
- Problem prevention
- Limiting damage and interruptions after a problem is detected
- Reducing problem diagnostic time
- Reducing problem resolution time
- Simplifying customer interaction with Oracle Support

The keys to achieving these goals are the following technologies:

- **Automatic capture of diagnostic data upon first failure**—For critical errors, the ability to capture error information at first-failure greatly increases the chance of a quick problem resolution and reduced downtime. An always-on memory-based tracing system proactively collects diagnostic data from many database components, and can help isolate root causes of problems. Such proactive diagnostic data is similar to the data collected by airplane "black box" flight recorders. When a problem is detected, alerts are generated and the fault diagnosability infrastructure is activated to capture and store diagnostic data. The data is stored in a repository that is outside the database (and therefore available when the database is down), and is easily accessible with command line utilities and Oracle Enterprise Manager Cloud Control (Cloud Control).
- **Standardized trace formats**—Standardizing trace formats across all database components enables DBAs and Oracle Support personnel to use a single set of tools for problem analysis. Problems are more easily diagnosed, and downtime is reduced.
- **Health checks**—Upon detecting a critical error, the fault diagnosability infrastructure can run one or more health checks to perform deeper analysis of a critical error. Health check results are then added to the other diagnostic data collected for the error. Individual health checks look for data block corruptions, undo and redo corruption, data dictionary corruption, and more. As a DBA, you can manually invoke these health checks, either on a regular basis or as required.
- **Incident packaging service (IPS) and incident packages**—The IPS enables you to automatically and easily gather the diagnostic data—traces, dumps, health check reports, and more—pertaining to a critical error and package the data into a zip file for transmission to Oracle Support. Because all diagnostic data relating to a critical error are tagged with that error's incident number, you do not have to search through trace files and other files to determine the files that are required for analysis; the incident packaging service identifies the required files automatically and adds them to the zip file. Before creating the zip file, the IPS first collects diagnostic data into an intermediate logical structure called an incident package (package). Packages are stored in the Automatic Diagnostic Repository. If you choose to, you can access this intermediate logical structure, view and modify its contents, add or remove additional diagnostic data at any time, and when you are ready, create the

zip file from the package. After these steps are completed, the zip file is ready to be uploaded to Oracle Support.

- **Data Recovery Advisor**—The Data Recovery Advisor integrates with database health checks and RMAN to display data corruption problems, assess the extent of each problem (critical, high priority, low priority), describe the impact of a problem, recommend repair options, conduct a feasibility check of the customer-chosen option, and automate the repair process.
- **SQL Test Case Builder**—For many SQL-related problems, obtaining a reproducible test case is an important factor in problem resolution speed. The SQL Test Case Builder automates the sometimes difficult and time-consuming process of gathering as much information as possible about the problem and the environment in which it occurred. After quickly gathering this information, you can upload it to Oracle Support to enable support personnel to easily and accurately reproduce the problem.

7.1.2 Incidents and Problems

A **problem** is a critical error in a database instance, Oracle Automatic Storage Management (Oracle ASM) instance, or other Oracle product or component. An **incident** is a single occurrence of a problem.

- About Incidents and Problems
  To facilitate diagnosis and resolution of critical errors, the fault diagnosability infrastructure introduces two concepts for Oracle Database: problems and incidents.
- Incident Flood Control
  It is conceivable that a problem could generate dozens or perhaps hundreds of incidents in a short period of time. This would generate too much diagnostic data, which would consume too much space in the ADR and could possibly slow down your efforts to diagnose and resolve the problem. For these reasons, the fault diagnosability infrastructure applies *flood control* to incident generation after certain thresholds are reached.
- Related Problems Across the Topology
  For any problem identified in a database instance, the diagnosability framework can identify related problems across the topology of your Oracle Database installation.

7.1.2.1 About Incidents and Problems

To facilitate diagnosis and resolution of critical errors, the fault diagnosability infrastructure introduces two concepts for Oracle Database: problems and incidents.

A **problem** is a critical error in a database instance, Oracle Automatic Storage Management (Oracle ASM) instance, or other Oracle product or component. Critical errors manifest as internal errors, such as `ORA-00600`, or other severe errors, such as `ORA-07445` (operating system exception) or `ORA-04031` (out of memory in the shared pool). Problems are tracked in the ADR. Each problem has a *problem key*, which is a text string that describes the problem. It includes an error code (such as `ORA 600`) and in some cases, one or more error parameters.

An **incident** is a single occurrence of a problem. When a problem (critical error) occurs multiple times, an incident is created for each occurrence. Incidents are timestamped and tracked in the Automatic Diagnostic Repository (ADR). Each incident is identified by a numeric incident ID, which is unique within the ADR. When an incident occurs, the database:

- Makes an entry in the alert log.
- Sends an *incident alert* to Cloud Control.
- Gathers first-failure diagnostic data about the incident in the form of dump files (incident dumps).

- Tags the incident dumps with the incident ID.
- Stores the incident dumps in an ADR subdirectory created for that incident.

Diagnosis and resolution of a critical error usually starts with an incident alert. Incident alerts are displayed on the Cloud Control Database Home page or Oracle Automatic Storage Management Home page. The Database Home page also displays in its Related Alerts section any critical alerts in the Oracle ASM instance or other Oracle products or components. After viewing an alert, you can then view the problem and its associated incidents with Cloud Control or with the ADRCI command-line utility.

See Also:

- "Viewing Problems with the Support Workbench"
- "About Investigating, Reporting, and Resolving a Problem"
- "ADRCI Command-Line Utility"

7.1.2.2 Incident Flood Control

It is conceivable that a problem could generate dozens or perhaps hundreds of incidents in a short period of time. This would generate too much diagnostic data, which would consume too much space in the ADR and could possibly slow down your efforts to diagnose and resolve the problem. For these reasons, the fault diagnosability infrastructure applies *flood control* to incident generation after certain thresholds are reached.

A **flood-controlled incident** is an incident that generates an alert log entry, is recorded in the ADR, but does not generate incident dumps. Flood-controlled incidents provide a way of informing you that a critical error is ongoing, without overloading the system with diagnostic data. You can choose to view or hide flood-controlled incidents when viewing incidents with Cloud Control or the ADRCI command-line utility.

Threshold levels for incident flood control are predetermined and cannot be changed. They are defined as follows:

- After five incidents occur for the same problem key in one hour, subsequent incidents for this problem key are flood-controlled. Normal (non-flood-controlled) recording of incidents for that problem key begins again in the next hour.
- After 25 incidents occur for the same problem key in one day, subsequent incidents for this problem key are flood-controlled. Normal recording of incidents for that problem key begins again on the next day.

In addition, after 50 incidents for the same problem key occur in one hour, or 250 incidents for the same problem key occur in one day, subsequent incidents for this problem key are not recorded at all in the ADR. In these cases, the database writes a message to the alert log indicating that no further incidents will be recorded. As long as incidents continue to be generated for this problem key, this message is added to the alert log every ten minutes until the hour or the day expires. Upon expiration of the hour or day, normal recording of incidents for that problem key begins again.

7.1.2.3 Related Problems Across the Topology

For any problem identified in a database instance, the diagnosability framework can identify related problems across the topology of your Oracle Database installation.

In a single instance environment, a related problem could be identified in the local Oracle ASM instance. In an Oracle RAC environment, a related problem could be identified in any database instance or Oracle ASM instance on any other node. When investigating problems, you are able to view and gather information on any related problems.

A problem is related to the original problem if it occurs within a designated time period or shares the same execution context identifier. An **execution context identifier (ECID)** is a globally unique identifier used to tag and track a single call through the Oracle software stack, for example, a call to Oracle Fusion Middleware that then calls into Oracle Database to retrieve data. The ECID is typically generated in the middle tier and is passed to the database as an Oracle Call Interface (OCI) attribute. When a single call has failures on multiple tiers of the Oracle software stack, problems that are generated are tagged with the same ECID so that they can be correlated. You can then determine the tier on which the originating problem occurred.

7.1.3 Fault Diagnosability Infrastructure Components

The fault diagnosability infrastructure consists of several components, including the Automatic Diagnostic Repository (ADR), various logs, trace files, the Enterprise Manager Support Workbench, and the ADRCI Command-Line Utility.

- Automatic Diagnostic Repository (ADR)
  The ADR is a file-based repository for database diagnostic data such as traces, dumps, the alert log, health monitor reports, and more. It has a unified directory structure across multiple instances and multiple products.
- Alert Log
  The alert log is an XML file that is a chronological log of messages and errors.
- Attention Log
  The attention log is a structured, externally modifiable file that contains information about critical and highly visible database events. Use the attention log to quickly access information about critical events that need action.
- Trace Files, Dumps, and Core Files
  Trace files, dumps, and core files contain diagnostic data that are used to investigate problems. They are stored in the ADR.
- DDL Log
  The data definition language (DDL) log is a file that has the same format and basic behavior as the alert log, but it only contains the DDL statements issued by the database.
- Debug Log
  An Oracle Database component can detect conditions, states, or events that are unusual, but which do not inhibit correct operation of the detecting component. The component can issue a warning about these conditions, states, or events. The debug log is a file that records these warnings.
- Other ADR Contents
  In addition to files mentioned in the previous sections, the ADR contains health monitor reports, data repair records, SQL test cases, incident packages, and more. These components are described later in the chapter.
- Enterprise Manager Support Workbench
  The Enterprise Manager Support Workbench (Support Workbench) is a facility that enables you to investigate, report, and in some cases, repair problems (critical errors), all with an easy-to-use graphical interface.

- ADRCI Command-Line Utility
  The ADR Command Interpreter (ADRCI) is a utility that enables you to investigate problems, view health check reports, and package first-failure diagnostic data, all within a command-line environment.

7.1.3.1 Automatic Diagnostic Repository (ADR)

The ADR is a file-based repository for database diagnostic data such as traces, dumps, the alert log, health monitor reports, and more. It has a unified directory structure across multiple instances and multiple products.

The database, Oracle Automatic Storage Management (Oracle ASM), the listener, Oracle Clusterware, and other Oracle products or components store all diagnostic data in the ADR. Each instance of each product stores diagnostic data underneath its own home directory within the ADR. For example, in an Oracle Real Application Clusters environment with shared storage and Oracle ASM, each database instance and each Oracle ASM instance has an ADR home directory. ADR's unified directory structure, consistent diagnostic data formats across products and instances, and a unified set of tools enable customers and Oracle Support to correlate and analyze diagnostic data across multiple instances. With Oracle Clusterware, each host node in the cluster has an ADR home directory.

Note:

Because all diagnostic data, including the alert log, are stored in the ADR, the initialization parameters `BACKGROUND_DUMP_DEST` and `USER_DUMP_DEST` are deprecated. They are replaced by the initialization parameter `DIAGNOSTIC_DEST`, which identifies the location of the ADR.

Related Topics

- Structure, Contents, and Location of the Automatic Diagnostic Repository
  The Automatic Diagnostic Repository (ADR) is a directory structure that is stored outside of the database. It is therefore available for problem diagnosis when the database is down.

7.1.3.2 Alert Log

The alert log is an XML file that is a chronological log of messages and errors.

There is one alert log in each ADR home. Each alert log is specific to its component type, such as database, Oracle ASM, listener, and Oracle Clusterware.

For the database, the alert log includes messages about the following:

- Critical errors (incidents)
- Administrative operations, such as starting up or shutting down the database, recovering the database, creating or dropping a tablespace, and others.
- Errors during automatic refresh of a materialized view
- Other database events

You can view the alert log in text format (with the XML tags stripped) with Cloud Control and with the ADRCI utility. There is also a text-formatted version of the alert log stored in the ADR for backward compatibility. However, Oracle recommends that any parsing of the alert log contents be done with the XML-formatted version, because the text format is unstructured and may change from release to release.

See Also:

- "ADRCI Command-Line Utility"
- "Viewing the Alert Log"

7.1.3.3 Attention Log

The attention log is a structured, externally modifiable file that contains information about critical and highly visible database events. Use the attention log to quickly access information about critical events that need action.

There is one attention log for each database instance. The attention log contains a pre-determined, translatable series of messages, with one message for each event. Some important attributes of a message are as follows:

- Attention ID: A unique identifier for the message.
- Attention type: The type of attention message. Possible values are Error, Warning, Notification, or Additional information. The attention type can be modified dynamically.
- Message text
- Urgency: Possible values are Immediate, Soon, Deferrable, or Information.
- Scope: Possible values are Session, Process, PDB Instance, CDB Instance, CDB Cluster, PDB (for issues in persistent storage that a database restart will not fix), or CDB (for issues in persistent storage that a database restart will not fix).
- Target user: The user who must act on this attention log message. Possible values are Clusterware Admin, CDB admin, or PDB admin.
- Cause
- Action

Contents of the Attention Log

The following is an example of an attention log message that needs immediate action.

```
{
IMMEDIATE : "PMON (ospid: 3565): terminating the instance due to ORA error
822"
CAUSE: "PMON detected fatal background process death"
ACTION: "Termination of fatal background is not recommended, Investigate
cause of process termination"
CLASS : CDB-INSTANCE / CDB_ADMIN / ERROR / DBAL-35782660
TIME : 2020-03-28T14:15:16.159-07:00
INFO : "Some additional data on error PMON error"
}
```

Related Topics

- Viewing Attention Log Information
  Access information stored in the attention log either by opening the file with any text editor or by querying the `V$DIAG_ALERT_EXT` view.

7.1.3.4 Trace Files, Dumps, and Core Files

Trace files, dumps, and core files contain diagnostic data that are used to investigate problems. They are stored in the ADR.

- Trace Files
  Each server and background process can write to an associated trace file. Trace files are updated periodically over the life of the process and can contain information on the process environment, status, activities, and errors. In addition, when a process detects a critical error, it writes information about the error to its trace file.
- Dumps
  A dump is a specific type of trace file. A dump is typically a one-time output of diagnostic data in response to an event (such as an incident), whereas a trace tends to be continuous output of diagnostic data.
- Core Files
  A core file contains a memory dump, in an all-binary, port-specific format.

7.1.3.4.1 Trace Files

Each server and background process can write to an associated trace file. Trace files are updated periodically over the life of the process and can contain information on the process environment, status, activities, and errors. In addition, when a process detects a critical error, it writes information about the error to its trace file.

The SQL trace facility also creates trace files, which provide performance information on individual SQL statements. You can enable SQL tracing for a session or an instance.

Trace file names are platform-dependent. Typically, database background process trace file names contain the Oracle SID, the background process name, and the operating system process number, while server process trace file names contain the Oracle SID, the string "ora", and the operating system process number. The file extension is `.trc`. An example of a server process trace file name is orcl_ora_344.trc. Trace files are sometimes accompanied by corresponding trace metadata (`.trm`) files, which contain structural information about trace files and are used for searching and navigation.

Starting with Oracle Database 21c, all trace file records contain a prefix that is used to classify records. The prefix indicates the level of sensitivity of each trace record. This helps in enhancing security.

Oracle Database includes tools that help you analyze trace files. For more information on application tracing, SQL tracing, and tracing tools, see *Oracle Database SQL Tuning Guide*.

Related Topics

- Finding Trace Files
  Trace files are stored in the Automatic Diagnostic Repository (ADR), in the `trace` directory under each ADR home. To help you locate individual trace files within this directory, you can use data dictionary views. For example, you can find the path to your current session's trace file or to the trace file for each Oracle Database process.

7.1.3.4.2 Dumps

A dump is a specific type of trace file. A dump is typically a one-time output of diagnostic data in response to an event (such as an incident), whereas a trace tends to be continuous output of diagnostic data.

When an incident occurs, the database writes one or more dumps to the incident directory created for the incident. Incident dumps also contain the incident number in the file name.

7.1.3.4.3 Core Files

A core file contains a memory dump, in an all-binary, port-specific format.

Core file names include the string "core" and the operating system process ID. Core files are useful to Oracle Support engineers only. Core files are not found on all platforms.

7.1.3.5 DDL Log

The data definition language (DDL) log is a file that has the same format and basic behavior as the alert log, but it only contains the DDL statements issued by the database.

The DDL log is created only for the RDBMS component and only if the `ENABLE_DDL_LOGGING` initialization parameter is set to `TRUE`. When this parameter is set to `FALSE`, DDL statements are not included in any log.

The DDL log contains one log record for each DDL statement issued by the database. The DDL log is included in IPS incident packages.

There are two DDL logs that contain the same information. One is an XML file, and the other is a text file. The DDL log is stored in the log/ddl subdirectory of the ADR home.

See Also:

Oracle Database Reference for more information about the `ENABLE_DDL_LOGGING` initialization parameter

7.1.3.6 Debug Log

An Oracle Database component can detect conditions, states, or events that are unusual, but which do not inhibit correct operation of the detecting component. The component can issue a warning about these conditions, states, or events. The debug log is a file that records these warnings.

These warnings recorded in the debug log are not serious enough to warrant an incident or a write to the alert log. They do warrant a record in a log file because they might be needed to diagnose a future problem.

The debug log has the same format and basic behavior as the alert log, but it only contains information about possible problems that might need to be corrected.

The debug log reduces the amount of information in the alert log and trace files. It also improves the visibility of debug information.

The debug log is included in IPS incident packages. The debug log's contents are intended for Oracle Support. Database administrators should not use the debug log directly.

Note:

Because there is a separate debug log starting with Oracle Database 12*c*, the alert log and the trace files are streamlined. They now contain fewer warnings of the type that are recorded in the debug log.

7.1.3.7 Other ADR Contents

In addition to files mentioned in the previous sections, the ADR contains health monitor reports, data repair records, SQL test cases, incident packages, and more. These components are described later in the chapter.

7.1.3.8 Enterprise Manager Support Workbench

The Enterprise Manager Support Workbench (Support Workbench) is a facility that enables you to investigate, report, and in some cases, repair problems (critical errors), all with an easy-to-use graphical interface.

The Support Workbench provides a self-service means for you to gather first-failure diagnostic data, obtain a support request number, and upload diagnostic data to Oracle Support with a minimum of effort and in a very short time, thereby reducing time-to-resolution for problems. The Support Workbench also recommends and provides easy access to Oracle advisors that help you repair SQL-related problems, data corruption problems, and more.

7.1.3.9 ADRCI Command-Line Utility

The ADR Command Interpreter (ADRCI) is a utility that enables you to investigate problems, view health check reports, and package first-failure diagnostic data, all within a command-line environment.

You can then upload the package to Oracle Support. ADRCI also enables you to view the names of the trace files in the ADR, and to view the alert log with XML tags stripped, with and without content filtering.

For more information on ADRCI, see *Oracle Database Utilities*.

7.1.4 Structure, Contents, and Location of the Automatic Diagnostic Repository

The Automatic Diagnostic Repository (ADR) is a directory structure that is stored outside of the database. It is therefore available for problem diagnosis when the database is down.

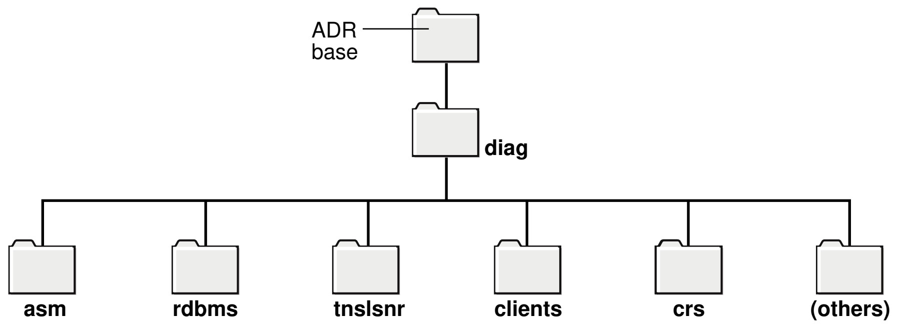
The ADR root directory is known as **ADR base**. Its location is set by the `DIAGNOSTIC_DEST` initialization parameter. If this parameter is omitted or left null, the database sets `DIAGNOSTIC_DEST` upon startup as follows:

- If environment variable `ORACLE_BASE` is set, `DIAGNOSTIC_DEST` is set to the directory designated by `ORACLE_BASE`.
- If environment variable `ORACLE_BASE` is not set, `DIAGNOSTIC_DEST` is set to *ORACLE_HOME*/log.

Within ADR base, there can be multiple ADR homes, where each ADR home is the root directory for all diagnostic data—traces, dumps, the alert log, and so on—for a particular instance of a particular Oracle product or component. For example, in an Oracle Real Application Clusters environment with Oracle ASM, each database instance, Oracle ASM instance, and listener has an ADR home.

ADR homes reside in ADR base subdirectories that are named according to the product or component type. Figure 7-1 illustrates these top-level subdirectories.

Figure 7-1 Product/Component Type Subdirectories in the ADR



> **Note:**
>
> Additional subdirectories might be created in the ADR depending on your configuration. Some products automatically purge expired diagnostic data from ADR. For other products, you can use the ADRCI utility `PURGE` command at regular intervals to purge expired diagnostic data.

The location of each ADR home is given by the following path, which starts at the ADR base directory:

```
diag/product_type/product_id/instance_id
```

As an example, Table 7-1 lists the values of the various path components for an Oracle Database instance.

Table 7-1 ADR Home Path Components for Oracle Database

Path Component	Value for Oracle Database
product_type	rdbms
product_id	*DB_UNIQUE_NAME*
instance_id	*SID*

For example, for a database with a SID and database unique name both equal to `orclbi`, the ADR home would be in the following location:

```
ADR_base/diag/rdbms/orclbi/orclbi/
```

Similarly, the ADR home path for the Oracle ASM instance in a single-instance environment would be:

```
ADR_base/diag/asm/+asm/+asm/
```

ADR Home Subdirectories

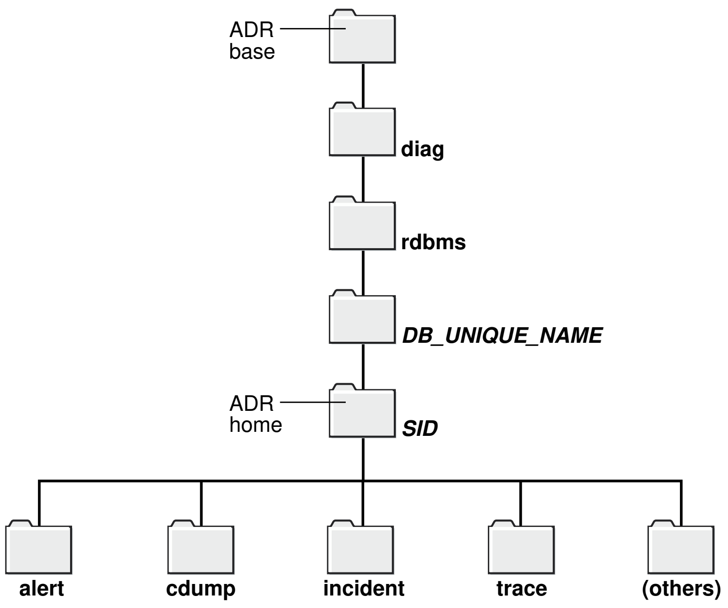
Within each ADR home directory are subdirectories that contain the diagnostic data. Table 7-2 lists some of these subdirectories and their contents.

Table 7-2 ADR Home Subdirectories

Subdirectory Name	Contents
alert	The XML-formatted alert log
cdump	Core files
incident	Multiple subdirectories, where each subdirectory is named for a particular incident, and where each contains dumps pertaining only to that incident
trace	Background and server process trace files, SQL trace files, and the text-formatted alert log
(others)	Other subdirectories of ADR home, which store incident packages, health monitor reports, logs other than the alert log (such as the DDL log and the debug log), and other information

Figure 7-2 illustrates the complete directory hierarchy of the ADR for a database instance.

Figure 7-2 ADR Directory Structure for a Database Instance



ADR in an Oracle Clusterware Environment

Oracle Clusterware uses ADR and has its own Oracle home and Oracle base. The ADR directory structure for Oracle Clusterware is different from that of a database instance. There is only one instance of Oracle Clusterware on a system, so Clusterware ADR homes use only a system's host name as a differentiator.

When Oracle Clusterware is configured, the ADR home uses `crs` for both the product type and the instance ID, and the system host name is used for the product ID. Thus, on a host named `dbprod01`, the CRS ADR home would be:

```
ADR_base/diag/crs/dbprod01/crs/
```

See Also:

Oracle Clusterware Administration and Deployment Guide

ADR in an Oracle Real Application Clusters Environment

In an Oracle Real Application Clusters (Oracle RAC) environment, each node can have ADR base on its own local storage, or ADR base can be set to a location on shared storage. You can use ADRCI to view aggregated diagnostic data from all instances on a single report.

ADR in Oracle Client

Each installation of Oracle Client includes an ADR for diagnostic data associated with critical failures in any of the Oracle Client components. The ADRCI utility is installed with Oracle Client so that you can examine diagnostic data and package it to enable it for upload to Oracle Support.

Viewing ADR Locations with the V$DIAG_INFO View

The `V$DIAG_INFO` view lists all important ADR locations for the current Oracle Database instance.

```
SELECT * FROM V$DIAG_INFO;

INST_ID NAME                  VALUE
------- --------------------- -------------------------------------------------------------
      1 Diag Enabled          TRUE
      1 ADR Base              /u01/oracle
      1 ADR Home              /u01/oracle/diag/rdbms/orclbi/orclbi
      1 Diag Trace            /u01/oracle/diag/rdbms/orclbi/orclbi/trace
      1 Diag Alert            /u01/oracle/diag/rdbms/orclbi/orclbi/alert
      1 Diag Incident         /u01/oracle/diag/rdbms/orclbi/orclbi/incident
      1 Diag Cdump            /u01/oracle/diag/rdbms/orclbi/orclbi/cdump
      1 Health Monitor        /u01/oracle/diag/rdbms/orclbi/orclbi/hm
      1 Default Trace File    /u01/oracle/diag/rdbms/orclbi/orclbi/trace/orcl_ora_22769.trc
      1 Active Problem Count  8
      1 Active Incident Count 20
```

The following table describes some of the information displayed by this view.

Table 7-3 Data in the V$DIAG_INFO View

Name	Description
ADR Base	Path of ADR base
ADR Home	Path of ADR home for the current database instance
Diag Trace	Location of background process trace files, server process trace files, SQL trace files, and the text-formatted version of the alert log
Diag Alert	Location of the XML-formatted version of the alert log
Default Trace File	Path to the trace file for the current session

Viewing Critical Errors with the V$DIAG_CRITICAL_ERROR View

The `V$DIAG_CRITICAL_ERROR` view lists all of the non-internal errors designated as critical errors for the current Oracle Database release. The view does not list internal errors because internal errors are always designated as critical errors.

The following example shows the output for the `V$DIAG_CRITICAL_ERROR` view:

```
SELECT * FROM V$DIAG_CRITICAL_ERROR;

FACILITY   ERROR
---------- ----------------------------------------------------------------
ORA        7445
ORA        4030
ORA        4031
ORA        29740
ORA        255
ORA        355
ORA        356
ORA        239
ORA        240
ORA        494
ORA        3137
ORA        227
ORA        353
ORA        1578
ORA        32701
ORA        32703
ORA        29770
ORA        29771
ORA        445
ORA        25319
OCI        3106
OCI        3113
OCI        3135
```

The following table describes the information displayed by this view.

Table 7-4 Data in the V$DIAG_CRITICAL_ERROR View

Column	Description
`FACILITY`	The facility that can report the error, such as Oracle Database (ORA) or Oracle Call Interface (OCI)
`ERROR`	The error number

See Also:

"About Incidents and Problems" for more information about internal errors

7.2 About Investigating, Reporting, and Resolving a Problem

You can use the Enterprise Manager Support Workbench (Support Workbench) to investigate and report a problem (critical error), and in some cases, resolve the problem. You can use a "roadmap" that summarizes the typical set of tasks that you must perform.

Note:

The tasks described in this section are all Cloud Control–based. You can also accomplish all of these tasks (or their equivalents) with the `ADRCI` command-line utility, with PL/SQL packages such as `DBMS_HM` and `DBMS_SQLDIAG`, and with other software tools. See *Oracle Database Utilities* for more information on the `ADRCI` utility, and see *Oracle Database PL/SQL Packages and Types Reference* for information on PL/SQL packages.

- Roadmap — Investigating, Reporting, and Resolving a Problem
  You can begin investigating a problem by starting from the Support Workbench home page in Cloud Control. However, the more typical workflow begins with a critical error alert on the Database Home page.
- Task 1: View Critical Error Alerts in Cloud Control
  You begin the process of investigating problems (critical errors) by reviewing critical error alerts on the Database Home page or Oracle Automatic Storage Management Home page.
- Task 2: View Problem Details
  You continue your investigation from the Incident Manager Problem Details page.
- Task 3: (Optional) Gather Additional Diagnostic Information
  You can perform the following activities to gather additional diagnostic information for a problem. This additional information is then automatically included in the diagnostic data uploaded to Oracle Support. If you are unsure about performing these activities, then check with your Oracle Support representative.
- Task 4: (Optional) Create a Service Request
  At this point, you can create an Oracle Support service request and record the service request number with the problem information.
- Task 5: Package and Upload Diagnostic Data to Oracle Support
  For this task, you use the quick packaging process of the Support Workbench to package and upload the diagnostic information for the problem to Oracle Support.
- Task 6: Track the Service Request and Implement Any Repairs
  After uploading diagnostic information to Oracle Support, you might perform various activities to track the service request, to collect additional diagnostic information, and to implement repairs.

See Also:

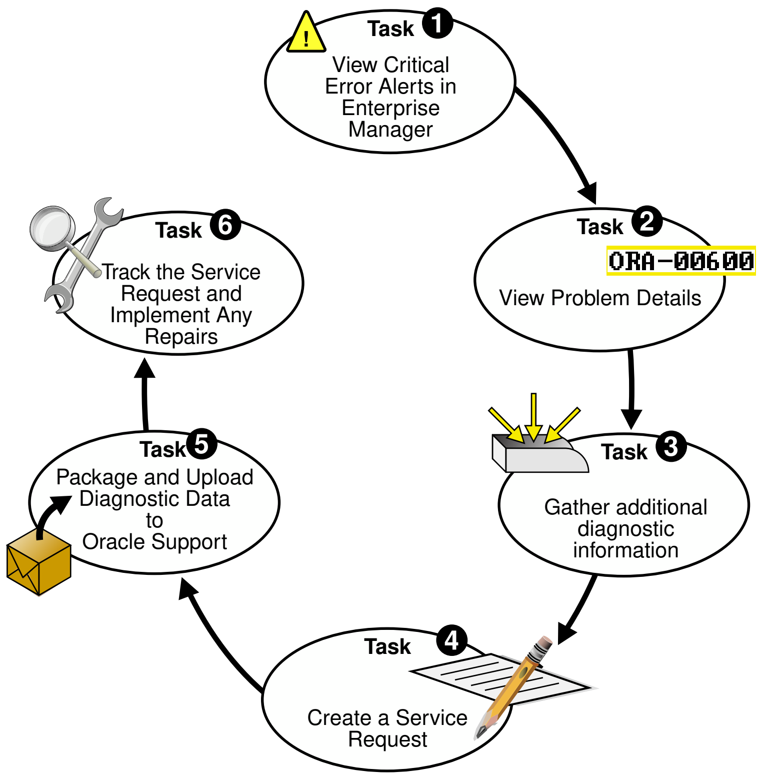
"About the Oracle Database Fault Diagnosability Infrastructure" for more information on problems and their diagnostic data

7.2.1 Roadmap — Investigating, Reporting, and Resolving a Problem

You can begin investigating a problem by starting from the Support Workbench home page in Cloud Control. However, the more typical workflow begins with a critical error alert on the Database Home page.

Figure 7-3 illustrates the tasks that you complete to investigate, report, and in some cases, resolve a problem.

Figure 7-3 Workflow for Investigating, Reporting, and Resolving a Problem


Task 1
View Critical Error Alerts in Enterprise Manager
Task 2
ORA-00600
View Problem Details
Task 3
Gather additional diagnostic information
Task 4
Create a Service Request
Task 5
Package and Upload Diagnostic Data to Oracle Support
Task 6
Track the Service Request and Implement Any Repairs

The following are task descriptions. Subsequent sections provide details for each task.

- **Task 1: View Critical Error Alerts in Cloud Control**

  Start by accessing the Database Home page in Cloud Control and reviewing critical error alerts. Select an alert for which to view details, and then go to the Problem Details page.

- **Task 2: View Problem Details**

  Examine the problem details and view a list of all incidents that were recorded for the problem. Display findings from any health checks that were automatically run.

- **Task 3: (Optional) Gather Additional Diagnostic Information**

  Optionally run additional health checks or other diagnostics. For SQL-related errors, optionally invoke the SQL Test Case Builder, which gathers all required data related to a SQL problem and packages the information in a way that enables the problem to be reproduced at Oracle Support.

- **Task 4: (Optional) Create a Service Request**

  Optionally create a service request with My Oracle Support and record the service request number with the problem information. If you skip this step, you can create a service request later, or the Support Workbench can create one for you.
- **Task 5: Package and Upload Diagnostic Data to Oracle Support**

  Invoke a guided workflow (a *wizard*) that automatically packages the gathered diagnostic data for a problem and uploads the data to Oracle Support.
- **Task 6: Track the Service Request and Implement Any Repairs**

  Optionally maintain an activity log for the service request in the Support Workbench. Run Oracle advisors to help repair SQL failures or corrupted data.

See Also:

"Viewing Problems with the Support Workbench"

7.2.2 Task 1: View Critical Error Alerts in Cloud Control

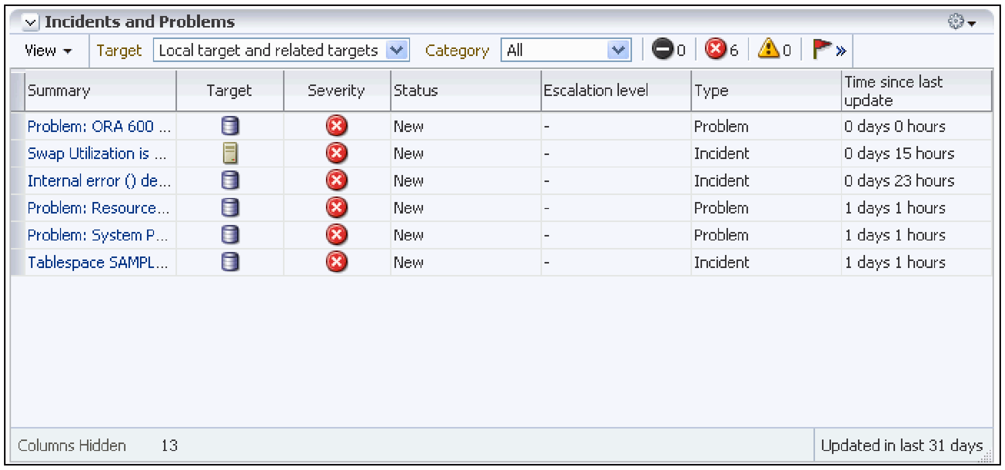
You begin the process of investigating problems (critical errors) by reviewing critical error alerts on the Database Home page or Oracle Automatic Storage Management Home page.

To view critical error alerts:

1. Access the Database Home page in Cloud Control.
2. View the alerts in the Incidents and Problems section.

   If necessary, click the hide/show icon next to the Alerts heading to display the alerts.

   Also, in the Category list, you can select a particular category to view alerts for only that category.



Incidents and Problems

View ▾ Target: Local target and related targets Category: All 0 6 0

Summary	Target	Severity	Status	Escalation level	Type	Time since last update
Problem: ORA 600 ...		✖	New	-	Problem	0 days 0 hours
Swap Utilization is ...		✖	New	-	Incident	0 days 15 hours
Internal error () de...		✖	New	-	Incident	0 days 23 hours
Problem: Resource...		✖	New	-	Problem	1 days 1 hours
Problem: System P...		✖	New	-	Problem	1 days 1 hours
Tablespace SAMPL...		✖	New	-	Incident	1 days 1 hours

Columns Hidden 13   Updated in last 31 days

3. In the Summary column, click the message of the critical error alert that you want to investigate.

   The General subpage of the Incident Manager Problem Details page appears. This page includes:

- Problem details
- Controls that allow you to acknowledge, clear, or record a comment about the alert in the Tracking section
- Links that enable you to diagnose the problem using Support Workbench and package the diagnostics in the Guided Resolution section.

Other sections might appear depending on the type of problem you are investigating.

To view more information about the problem, click the following subpages on the Incident Manager Problem Details page:

- The Incidents subpage contains information about individual incidents for the problem.
- The My Oracle Support Knowledge subpage provides access to My Oracle Support for more information about the problem.
- The Updates subpage shows any updates entered about the problem.
- The Related Problems subpage shows other open problems with the same problem key as the current problem.

4. Perform one of the following actions:
   - To view the details of the problem associated with the critical error alert that you are investigating, proceed with "Task 2: View Problem Details".
   - If there are several related problems and you want to view more information about them, then complete these steps:
     – View problems and incidents as described in "Viewing Problems with the Support Workbench".
     – Select a single problem and view problem details, as described in "Viewing Problems with the Support Workbench".
     – Continue with "Task 3: (Optional) Gather Additional Diagnostic Information".

7.2.3 Task 2: View Problem Details

You continue your investigation from the Incident Manager Problem Details page.

To view problem details:

1. On the General subpage of the Incident Manager Problem Details page, click **Support Workbench: Problem Details** in the Diagnostics subsection.

   The Support Workbench Problem Details page appears.
2. (Optional) Complete one or more of the following actions:
   - In the Investigate and Resolve section, under Diagnose, click Related Problems Across Topology.

     A page appears showing any related problems in the local Oracle Automatic Storage Management (Oracle ASM) instance, or in the database or Oracle ASM instances on other nodes in an Oracle Real Application Clusters environment. This step is recommended if any critical alerts appear in the Related Alerts section on the Cloud Control Database Home page.

     See "Related Problems Across the Topology" for more information.
   - To view incident details, in the Incidents subpage, select an incident, and then click **View**.

     The Incident Details page appears, showing the Dump Files subpage.

- On the Incident Details page, select **Checker Findings** to view the Checker Findings subpage.

  This page displays findings from any health checks that were automatically run when the critical error was detected.

7.2.4 Task 3: (Optional) Gather Additional Diagnostic Information

You can perform the following activities to gather additional diagnostic information for a problem. This additional information is then automatically included in the diagnostic data uploaded to Oracle Support. If you are unsure about performing these activities, then check with your Oracle Support representative.

- Manually invoke additional health checks.

  See "Identifying Problems Proactively with Health Monitor".
- Invoke the SQL Test Case Builder.

  See "Creating Test Cases with SQL Test Case Builder".

7.2.5 Task 4: (Optional) Create a Service Request

At this point, you can create an Oracle Support service request and record the service request number with the problem information.

If you choose to skip this task, then the Support Workbench will automatically create a draft service request for you in "Task 5: Package and Upload Diagnostic Data to Oracle Support".

To create a service request:

1. From the Enterprise menu, select **My Oracle Support**, then **Service Requests**.

   The My Oracle Support Login and Registration page appears.
2. Log in to My Oracle Support and create a service request in the usual manner.

   (Optional) Remember the service request number (SR#) for the next step.
3. (Optional) Return to the Problem Details page, and then do the following:
   a. In the Summary section, click the **Edit** button that is adjacent to the SR# label.
   b. Enter the SR#, and then click **OK**.

   The SR# is recorded in the Problem Details page. This is for your reference only. See "Viewing Problems with the Support Workbench" for information about returning to the Problem Details page.

7.2.6 Task 5: Package and Upload Diagnostic Data to Oracle Support

For this task, you use the quick packaging process of the Support Workbench to package and upload the diagnostic information for the problem to Oracle Support.

Quick packaging has a minimum of steps, organized in a guided workflow (a wizard). The wizard assists you with creating an incident package (package) for a single problem, creating a zip file from the package, and uploading the file. With quick packaging, you are not able to edit or otherwise customize the diagnostic information that is uploaded. However, quick packaging is the more direct, straightforward method to package and upload diagnostic data.

To edit or remove sensitive data from the diagnostic information, enclose additional user files (such as application configuration files or scripts), or perform other customizations before uploading, you must use the custom packaging process, which is a more manual process and has more steps. See "Reporting Problems" for instructions. If you choose to follow those instructions instead of the instructions here in Task 5, do so now and then continue with Task 6: Track the Service Request and Implement Any Repairs when you are finished.

To package and upload diagnostic data to Oracle Support:

1. On the Support Workbench Problem Details page, in the Investigate and Resolve section, click **Quick Package**.

   The Create New Package page of the Quick Packaging wizard appears.

   > **Note:**
   >
   > See "Viewing Problems with the Support Workbench" for instructions for returning to the Problem Details page if you are not already there.

2. (Optional) Enter a package name and description.
3. Fill in any remaining fields on the page. If you have created a service request for this problem, then select the **No** option button for Create new Service Request (SR).

   If you select the **Yes** option button for Create new Service Request (SR), then the Quick Packaging wizard creates a draft service request on your behalf. You must later log in to My Oracle Support and fill in the details of the service request.

   Click **Next**.

   The Quick Packaging wizard displays a page indicating that it is processing the command to create a new package. When it finished, the Quick Packaging: View Contents page is displayed.
4. Review the contents on the View Contents page, making a note of the size of the created package, then click **Next**.

   The Quick Packaging: View Manifest page appears.
5. Review the information on this page, making a note of the location of the manifest (listed next to the heading Path). After you have reviewed the information, click **Next**.

   The Quick Packaging: Schedule page appears.
6. Choose either **Immediately**, or **Later**. If you select **Later**, then you provide additional information about the time the package should be submitted to My Oracle Support. After you have made your choice and provided any necessary information, click **Submit**.

   The Processing: Packaging and Sending the Package progress page appears.

When the Quick Packaging wizard is complete, if a new draft service request was created, then the confirmation message contains a link to the draft service request in My Oracle Support in Cloud Control. You can review and edit the service request by clicking the link.

The package created by the Quick Packaging wizard remains available in the Support Workbench. You can then modify it with custom packaging operations (such as adding new incidents) and upload again at a later time. See "Viewing and Modifying Incident Packages".

7.2.7 Task 6: Track the Service Request and Implement Any Repairs

After uploading diagnostic information to Oracle Support, you might perform various activities to track the service request, to collect additional diagnostic information, and to implement repairs.

Among these activities are the following:

- Adding an Oracle bug number to the problem information.

  To do so, on the Problem Details page, click the **Edit** button that is adjacent to the Bug# label. This is for your reference only.
- Adding comments to the problem activity log.

  You may want to do this to share problem status or history information with other DBAs in your organization. For example, you could record the results of your conversations with Oracle Support. To add comments, complete the following steps:

  1. Access the Problem Details page for the problem, as described in "Viewing Problems with the Support Workbench".
  2. Click **Activity Log** to display the Activity Log subpage.
  3. In the Comment field, enter a comment, and then click **Add Comment**.

     Your comment is recorded in the activity log.
- As new incidents occur, adding them to the package and reuploading.

  For this activity, you must use the custom packaging method described in "Reporting Problems".
- Running health checks.

  See "Identifying Problems Proactively with Health Monitor".
- Running a suggested Oracle advisor to implement repairs.

  Access the suggested advisor in one of the following ways:

  – **Problem Details page**—In the Self-Service tab of the Investigate and Resolve section

  – **Support Workbench home page**—on the Checker Findings subpage

  – **Incident Details page**—on the Checker Findings subpage

  Table 7-5 lists the advisors that help repair critical errors.

Table 7-5 Oracle Advisors that Help Repair Critical Errors

Advisor	Critical Errors Addressed	See
Data Recovery Advisor	Corrupted blocks, corrupted or missing files, and other data failures	"Repairing Data Corruptions with the Data Recovery Advisor"
SQL Repair Advisor	SQL statement failures	"Repairing SQL Failures with the SQL Repair Advisor"

See Also:

"Viewing Problems with the Support Workbench" for instructions for viewing the Checker Findings subpage of the Incident Details page

7.3 Diagnosing Problems

This section describes various methods to diagnose problems in an Oracle database.

- Identifying Problems Reactively
  This section describes how to identify Oracle database problems reactively.
- Identifying Problems Proactively with Health Monitor
  You can run diagnostic checks on a database with Health Monitor.
- Gathering Additional Diagnostic Data
  This section describes how to gather additional diagnostic data using alert log and trace files.
- Creating Test Cases with SQL Test Case Builder
  SQL Test Case Builder is a tool that automatically gathers information needed to reproduce the problem in a different database instance.

7.3.1 Identifying Problems Reactively

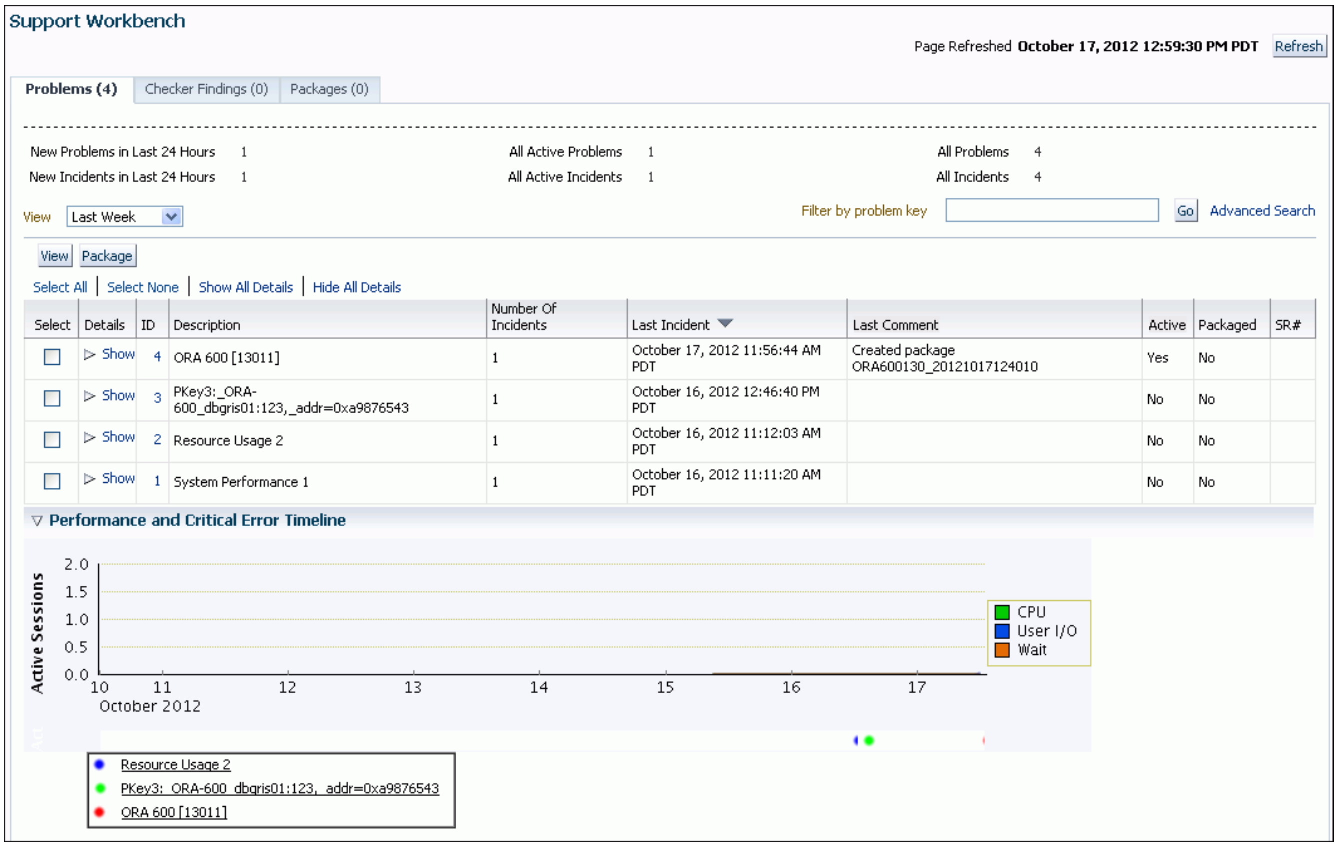
This section describes how to identify Oracle database problems reactively.

- Viewing Problems with the Support Workbench
  You can use the Support Workbench home page in Cloud Control to view all the problems or the problems in a specific time period.
- Adding Problems Manually to the Automatic Diagnostic Repository
  You can use Support Workbench in Cloud Control to manually add a problem to the ADR.
- Creating Incidents Manually
  You can create incidents manually by using the Automatic Diagnostic Repository Command Interpreter (ADRCI) utility.

7.3.1.1 Viewing Problems with the Support Workbench

You can use the Support Workbench home page in Cloud Control to view all the problems or the problems in a specific time period.

Figure 7-4 Support Workbench Home Page in Cloud Control



Support Workbench

Page Refreshed **October 17, 2012 12:59:30 PM PDT** Refresh

Problems (4) | Checker Findings (0) | Packages (0)

New Problems in Last 24 Hours	1	All Active Problems	1	All Problems	4
New Incidents in Last 24 Hours	1	All Active Incidents	1	All Incidents	4

View: Last Week | Filter by problem key | Go | Advanced Search

View | Package

Select All | Select None | Show All Details | Hide All Details

Select	Details	ID	Description	Number Of Incidents	Last Incident	Last Comment	Active	Packaged	SR#
☐	▷ Show	4	ORA 600 [13011]	1	October 17, 2012 11:56:44 AM PDT	Created package ORA600130_20121017124010	Yes	No	
☐	▷ Show	3	PKey3:_ORA-600_dbgris01:123,_addr=0xa9876543	1	October 16, 2012 12:46:40 PM PDT		No	No	
☐	▷ Show	2	Resource Usage 2	1	October 16, 2012 11:12:03 AM PDT		No	No	
☐	▷ Show	1	System Performance 1	1	October 16, 2012 11:11:20 AM PDT		No	No	

▽ Performance and Critical Error Timeline

● Resource Usage 2
● PKey3:_ORA-600_dbgris01:123,_addr=0xa9876543
● ORA 600 [13011]

To access the Support Workbench home page (database or Oracle ASM):

1. Access the Database Home page in Cloud Control.
2. From the Oracle Database menu, select **Diagnostics**, then **Support Workbench**.

   The Support Workbench home page for the database instance appears, showing the Problems subpage. By default the problems from the last 24 hours are displayed.
3. To view the Support Workbench home page for the Oracle ASM instance, click the link **Support Workbench (+ASM_*hostname*)** in the Related Links section.

To view problems and incidents:

1. On the Support Workbench home page, select the desired time period from the View list. To view all problems, select **All**.
2. (Optional) If the Performance and Critical Error Timeline section is hidden, click the **Show/Hide** icon adjacent to the section heading to show the section.

   This section enables you to view any correlation between performance changes and incident occurrences.

3. (Optional) Under the Details column, click **Show** to display a list of all incidents for a problem, and then click an incident ID to display the Incident Details page.

To view details for a particular problem:

1. On the Support Workbench home page, select the problem, and then click **View**.

   The Problem Details page appears, showing the Incidents subpage. The incidents subpage shows all incidents that are open and that generated dumps—that is, that were not flood-controlled.

2. (Optional) To view both normal and flood-controlled incidents, select **All** in the Data Dumped list.
3. (Optional) To view details for an incident, select the incident, and then click **View**.

   The Incident Details page appears.

4. (Optional) On the Incident Details page, to view checker findings for the incident, click **Checker Findings**.
5. (Optional) On the Incident Details page, to view the user actions that are available to you for the incident, click **Additional Diagnostics**. Each user action provides a way for you to gather additional diagnostics for the incident or its problem.

See Also:

"Incident Flood Control"

7.3.1.2 Adding Problems Manually to the Automatic Diagnostic Repository

You can use Support Workbench in Cloud Control to manually add a problem to the ADR.

System-generated problems, such as critical errors generated internally to the database are automatically added to the Automatic Diagnostic Repository (ADR) and tracked in the Support Workbench.

From the Support Workbench, you can gather additional diagnostic data on these problems, upload diagnostic data to Oracle Support, and in some cases, resolve the problems, all with the easy-to-use workflow that is explained in "About Investigating, Reporting, and Resolving a Problem".

There may be a situation in which you want to manually add a problem that you noticed to the ADR, so that you can put that problem through that same workflow. An example of such a situation might be a global database performance problem that was not diagnosed by Automatic Diagnostic Database Monitor (ADDM). The Support Workbench includes a mechanism for you to create and work with such a user-reported problem.

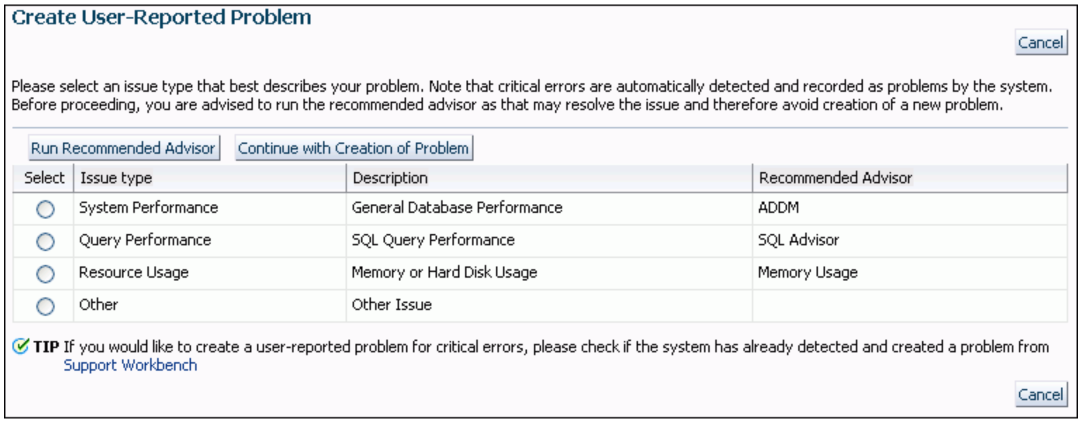
To create a user-reported problem:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Support Workbench" for instructions.

2. Under Related Links, click **Create User-Reported Problem**.

   The Create User-Reported Problem page appears.



Create User-Reported Problem

Please select an issue type that best describes your problem. Note that critical errors are automatically detected and recorded as problems by the system. Before proceeding, you are advised to run the recommended advisor as that may resolve the issue and therefore avoid creation of a new problem.

Select	Issue type	Description	Recommended Advisor
○	System Performance	General Database Performance	ADDM
○	Query Performance	SQL Query Performance	SQL Advisor
○	Resource Usage	Memory or Hard Disk Usage	Memory Usage
○	Other	Other Issue	

✓ TIP If you would like to create a user-reported problem for critical errors, please check if the system has already detected and created a problem from Support Workbench

3. If your problem matches one of the listed issue types, select the issue type, and then click **Run Recommended Advisor** to attempt to solve the problem with an Oracle advisor.
4. If the recommended advisor did not solve the problem, or if you did not run an advisor, do one of the following:
   - If your problem matches one of the listed issue types, select the issue type, and then click **Continue with Creation of Problem**.
   - If your problem does not match one of the listed issue types, select the issue type **Other** and then click **Continue with Creation of Problem**.

   The Problem Details page appears.
5. Follow the instructions on the Problem Details page.

   See "About Investigating, Reporting, and Resolving a Problem" for more information.

See Also:

"About the Oracle Database Fault Diagnosability Infrastructure" for more information on problems and the ADR

7.3.1.3 Creating Incidents Manually

You can create incidents manually by using the Automatic Diagnostic Repository Command Interpreter (ADRCI) utility.

To create an incident manually by using the ADRCI utility:

1. Ensure that the `ORACLE_HOME` and `PATH` environment variables are set properly. The `PATH` environment variable must include *`ORACLE_HOME`*`/bin` directory.
2. Start the ADRCI utility by running the following command at the operating system command prompt:

   ```
   ADRCI
   ```

   The ADRCI utility starts and displays the following prompt:

   ```
   adrci>
   ```

3. Run the ADRCI command having the following syntax to create an incident manually:

   ```
   adrci> dde create incident type incident_type
   ```

   Specify *incident_type* value for the type of incident that you want to create.

> **See Also:**
>
> - *Oracle Database Utilities* for more information about the ADRCI utility

7.3.2 Identifying Problems Proactively with Health Monitor

You can run diagnostic checks on a database with Health Monitor.

- About Health Monitor
  Oracle Database includes a framework called Health Monitor for running diagnostic checks on the database.
- Running Health Checks Manually
  Health Monitor can run health checks manually either by using the `DBMS_HM` PL/SQL package or by using the Cloud Control interface, found on the Checkers subpage of the Advisor Central page.
- Viewing Checker Reports
  After a checker has run, you can view a report of its execution. The report contains findings, recommendations, and other information. You can view reports using Cloud Control, the ADRCI utility, or the `DBMS_HM` PL/SQL package. The following table indicates the report formats available with each viewing method.
- Health Monitor Views
  Instead of requesting a checker report, you can view the results of a specific checker run by directly querying the ADR data from which reports are created.
- Health Check Parameters Reference
  Some health checks require parameters. Parameters with a default value of `(none)` are mandatory.

7.3.2.1 About Health Monitor

Oracle Database includes a framework called Health Monitor for running diagnostic checks on the database.

- About Health Monitor Checks
  Health Monitor checks (also known as checkers, health checks, or checks) examine various layers and components of the database.
- Types of Health Checks
  Health monitor runs several different types of checks.

7.3.2.1.1 About Health Monitor Checks

Health Monitor checks (also known as checkers, health checks, or checks) examine various layers and components of the database.

Health checks detect file corruptions, physical and logical block corruptions, undo and redo corruptions, data dictionary corruptions, and more. The health checks generate reports of their findings and, in many cases, recommendations for resolving problems. Health checks can be run in two ways:

- **Reactive**—The fault diagnosability infrastructure can run health checks automatically in response to a critical error.
- **Manual**—As a DBA, you can manually run health checks using either the `DBMS_HM` PL/SQL package or the Cloud Control interface. You can run checkers on a regular basis if desired, or Oracle Support may ask you to run a checker while working with you on a service request.

Health Monitor checks store findings, recommendations, and other information in the Automatic Diagnostic Repository (ADR).

Health checks can run in two modes:

- **DB-online** mode means the check can be run while the database is open (that is, in `OPEN` mode or `MOUNT` mode).
- **DB-offline** mode means the check can be run when the instance is available but the database itself is closed (that is, in `NOMOUNT` mode).

All the health checks can be run in DB-online mode. Only the Redo Integrity Check and the DB Structure Integrity Check can be used in DB-offline mode.

Note:

"Automatic Diagnostic Repository (ADR)"

7.3.2.1.2 Types of Health Checks

Health monitor runs several different types of checks.

Health monitor runs the following checks:

- **DB Structure Integrity Check**—This check verifies the integrity of database files and reports failures if these files are inaccessible, corrupt or inconsistent. If the database is in mount or open mode, this check examines the log files and data files listed in the control file. If the database is in `NOMOUNT` mode, only the control file is checked.
- **Data Block Integrity Check**—This check detects disk image block corruptions such as checksum failures, head/tail mismatch, and logical inconsistencies within the block. Most corruptions can be repaired using Block Media Recovery. Corrupted block information is also captured in the `V$DATABASE_BLOCK_CORRUPTION` view. This check does not detect inter-block or inter-segment corruption.
- **Redo Integrity Check**—This check scans the contents of the redo log for accessibility and corruption, as well as the archive logs, if available. The Redo Integrity Check reports failures such as archive log or redo corruption.
- **Undo Segment Integrity Check**—This check finds logical undo corruptions. After locating an undo corruption, this check uses PMON and SMON to try to recover the corrupted transaction. If this recovery fails, then Health Monitor stores information about the corruption in `V$CORRUPT_XID_LIST`. Most undo corruptions can be resolved by forcing a commit.

- **Transaction Integrity Check**—This check is identical to the Undo Segment Integrity Check except that it checks only one specific transaction.
- **Dictionary Integrity Check**—This check examines the integrity of core dictionary objects, such as `tab$` and `col$`. It performs the following operations:
  - Verifies the contents of dictionary entries for each dictionary object.
  - Performs a cross-row level check, which verifies that logical constraints on rows in the dictionary are enforced.
  - Performs an object relationship check, which verifies that parent-child relationships between dictionary objects are enforced.

  The Dictionary Integrity Check operates on the following dictionary objects:

  `tab$`, `clu$`, `fet$`, `uet$`, `seg$`, `undo$`, `ts$`, `file$`, `obj$`, `ind$`, `icol$`, `col$`, `user$`, `con$`, `cdef$`, `ccol$`, `bootstrap$`, `objauth$`, `ugroup$`, `tsq$`, `syn$`, `view$`, `typed_view$`, `superobj$`, `seq$`, `lob$`, `coltype$`, `subcoltype$`, `ntab$`, `refcon$`, `opqtype$`, `dependency$`, `access$`, `viewcon$`, `icoldep$`, `dual$`, `sysauth$`, `objpriv$`, `defrole$`, and `ecol$`.

7.3.2.2 Running Health Checks Manually

Health Monitor can run health checks manually either by using the `DBMS_HM` PL/SQL package or by using the Cloud Control interface, found on the Checkers subpage of the Advisor Central page.

- Running Health Checks Using the DBMS_HM PL/SQL Package
  The `DBMS_HM` procedure for running a health check is called `RUN_CHECK`.
- Running Health Checks Using Cloud Control
  Cloud Control provides an interface for running Health Monitor checkers.

7.3.2.2.1 Running Health Checks Using the DBMS_HM PL/SQL Package

The `DBMS_HM` procedure for running a health check is called `RUN_CHECK`.

1. To call `RUN_CHECK`, supply the name of the check and a name for the run, as follows:

```
BEGIN
  DBMS_HM.RUN_CHECK('Dictionary Integrity Check', 'my_run');
END;
/
```

2. To obtain a list of health check names, run the following query:

```
SELECT name FROM v$hm_check WHERE internal_check='N';
```

   Your output is similar to the following:

```
NAME
----------------------------------------------------------------
DB Structure Integrity Check
Data Block Integrity Check
Redo Integrity Check
Transaction Integrity Check
Undo Segment Integrity Check
Dictionary Integrity Check
```

Most health checks accept input parameters. You can view parameter names and descriptions with the `V$HM_CHECK_PARAM` view. Some parameters are mandatory while others are optional. If optional parameters are omitted, defaults are used. The following query displays parameter information for all health checks:

```
SELECT c.name check_name, p.name parameter_name, p.type,
p.default_value, p.description
FROM v$hm_check_param p, v$hm_check c
WHERE p.check_id = c.id and c.internal_check = 'N'
ORDER BY c.name;
```

Input parameters are passed in the `input_params` argument as name/value pairs separated by semicolons (;). The following example illustrates how to pass the transaction ID as a parameter to the Transaction Integrity Check:

```
BEGIN
  DBMS_HM.RUN_CHECK (
   check_name   => 'Transaction Integrity Check',
   run_name     => 'my_run',
   input_params => 'TXN_ID=7.33.2');
END;
/
```

See Also:

- "Health Check Parameters Reference"
- *Oracle Database PL/SQL Packages and Types Reference* for more examples of using `DBMS_HM`.

7.3.2.2.2 Running Health Checks Using Cloud Control

Cloud Control provides an interface for running Health Monitor checkers.

To run a Health Monitor Checker using Cloud Control:

1. Access the Database Home page.
2. From the Performance menu, select **Advisors Home**.
3. Click **Checkers** to view the Checkers subpage.
4. In the Checkers section, click the checker you want to run.
5. Enter values for input parameters or, for optional parameters, leave them blank to accept the defaults.
6. Click **OK**, confirm your parameters, and click **OK** again.

7.3.2.3 Viewing Checker Reports

After a checker has run, you can view a report of its execution. The report contains findings, recommendations, and other information. You can view reports using Cloud Control, the ADRCI utility, or the `DBMS_HM` PL/SQL package. The following table indicates the report formats available with each viewing method.

- About Viewing Checker Reports
  Results of checker runs (findings, recommendations, and other information) are stored in the ADR, but reports are not generated immediately.
- Viewing Reports Using Cloud Control
  You can also view Health Monitor reports and findings for a given checker run using Cloud Control.

- Viewing Reports Using DBMS_HM
  You can view Health Monitor checker reports with the `DBMS_HM` package function `GET_RUN_REPORT`.
- Viewing Reports Using the ADRCI Utility
  You can create and view Health Monitor checker reports using the ADRCI utility.

7.3.2.3.1 About Viewing Checker Reports

Results of checker runs (findings, recommendations, and other information) are stored in the ADR, but reports are not generated immediately.

Report Viewing Method	Report Formats Available
Cloud Control	HTML
`DBMS_HM` PL/SQL package	HTML, XML, and text
ADRCI utility	XML

When you request a report with the `DBMS_HM` PL/SQL package or with Cloud Control, if the report does not yet exist, it is first generated from the checker run data in the ADR, stored as a report file in XML format in the HM subdirectory of the ADR home for the current instance, and then displayed. If the report file already exists, it is just displayed. When using the ADRCI utility, you must first run a command to generate the report file if it does not exist, and then run another command to display its contents.

The preferred method to view checker reports is with Cloud Control.

Note:

"Automatic Diagnostic Repository (ADR)"

7.3.2.3.2 Viewing Reports Using Cloud Control

You can also view Health Monitor reports and findings for a given checker run using Cloud Control.

To view run findings using Cloud Control:

1. Access the Database Home page.
2. From the Performance menu, select **Advisors Home**.
3. Click **Checkers** to view the Checkers subpage.
4. Click the run name for the checker run that you want to view.

   The Run Detail page appears, showing the Findings subpage for that checker run.
5. Click **Runs** to display the Runs subpage.

   Cloud Control displays more information about the checker run.
6. Click **View Report** to view the report for the checker run.

   The report is displayed in a new browser window.

7.3.2.3.3 Viewing Reports Using DBMS_HM

You can view Health Monitor checker reports with the `DBMS_HM` package function `GET_RUN_REPORT`.

This function enables you to request HTML, XML, or text formatting. The default format is text, as shown in the following SQL*Plus example:

```
SET LONG 100000
SET LONGCHUNKSIZE 1000
SET PAGESIZE 1000
SET LINESIZE 512
SELECT DBMS_HM.GET_RUN_REPORT('HM_RUN_1061') FROM DUAL;

DBMS_HM.GET_RUN_REPORT('HM_RUN_1061')
-----------------------------------------------------------------------

 Run Name                     : HM_RUN_1061
 Run Id                       : 1061
 Check Name                   : Data Block Integrity Check
 Mode                         : REACTIVE
 Status                       : COMPLETED
 Start Time                   : 2007-05-12 22:11:02.032292 -07:00
 End Time                     : 2007-05-12 22:11:20.835135 -07:00
 Error Encountered            : 0
 Source Incident Id           : 7418
 Number of Incidents Created  : 0

Input Parameters for the Run
 BLC_DF_NUM=1
 BLC_BL_NUM=64349

Run Findings And Recommendations
 Finding
 Finding Name  : Media Block Corruption
 Finding ID    : 1065
 Type          : FAILURE
 Status        : OPEN
 Priority      : HIGH
 Message       : Block 64349 in datafile 1:
               '/u01/app/oracle/dbs/t_db1.f' is media corrupt
 Message       : Object BMRTEST1 owned by SYS might be unavailable
 Finding
 Finding Name  : Media Block Corruption
 Finding ID    : 1071
 Type          : FAILURE
 Status        : OPEN
 Priority      : HIGH
 Message       : Block 64351 in datafile 1:
               '/u01/app/oracle/dbs/t_db1.f' is media corrupt
 Message       : Object BMRTEST2 owned by SYS might be unavailable
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for details on the `DBMS_HM` package.

7.3.2.3.4 Viewing Reports Using the ADRCI Utility

You can create and view Health Monitor checker reports using the ADRCI utility.

To create and view a checker report using ADRCI:

1. Ensure that the `ORACLE_HOME` and `PATH` environment variables are set properly, and then start the ADRCI utility by running the following command at the operating system command prompt:

   ```
   ADRCI
   ```

   The ADRCI utility starts and displays the following prompt:

   ```
   adrci>
   ```

   Optionally, you can change the current ADR home. Use the `SHOW HOMES` command to list all ADR homes, and the `SET HOMEPATH` command to change the current ADR home. See *Oracle Database Utilities* for more information.
2. Enter the following command:

   ```
   show hm_run
   ```

   This command lists all the checker runs (stored in `V$HM_RUN`) registered in the ADR.
3. Locate the checker run for which you want to create a report and note the checker run name. The `REPORT_FILE` field contains a file name if a report already exists for this checker run. Otherwise, generate the report with the following command:

   ```
   create report hm_run run_name
   ```
4. To view the report, enter the following command:

   ```
   show report hm_run run_name
   ```

Note:

"Automatic Diagnostic Repository (ADR)"

7.3.2.4 Health Monitor Views

Instead of requesting a checker report, you can view the results of a specific checker run by directly querying the ADR data from which reports are created.

This data is available through the views `V$HM_RUN`, `V$HM_FINDING`, and `V$HM_RECOMMENDATION`.

The following example queries the `V$HM_RUN` view to determine a history of checker runs:

```
SELECT run_id, name, check_name, run_mode, src_incident FROM v$hm_run;

    RUN_ID NAME         CHECK_NAME                         RUN_MODE SRC_INCIDENT
---------- ------------ ---------------------------------- -------- ------------
         1 HM_RUN_1     DB Structure Integrity Check       REACTIVE            0
       101 HM_RUN_101   Transaction Integrity Check        REACTIVE         6073
       121 TXNCHK       Transaction Integrity Check        MANUAL              0
       181 HMR_tab$     Dictionary Integrity Check         MANUAL              0
         .
         .
```

```
            .
       981 Proct_ts$     Dictionary Integrity Check         MANUAL              0
      1041 HM_RUN_1041   DB Structure Integrity Check       REACTIVE            0
      1061 HM_RUN_1061   Data Block Integrity Check         REACTIVE         7418
```

The next example queries the `V$HM_FINDING` view to obtain finding details for the reactive data block check with `RUN_ID` 1061:

```
SELECT type, description FROM v$hm_finding WHERE run_id = 1061;

TYPE          DESCRIPTION
------------- -----------------------------------------
FAILURE       Block 64349 in datafile 1: '/u01/app/orac
              le/dbs/t_db1.f' is media corrupt

FAILURE       Block 64351 in datafile 1: '/u01/app/orac
              le/dbs/t_db1.f' is media corrupt
```

See Also:

- "Types of Health Checks"
- *Oracle Database Reference* for more information on the `V$HM_*` views

7.3.2.5 Health Check Parameters Reference

Some health checks require parameters. Parameters with a default value of `(none)` are mandatory.

Table 7-6 Parameters for Data Block Integrity Check

Parameter Name	Type	Default Value	Description
`BLC_DF_NUM`	Number	(none)	Block data file number
`BLC_BL_NUM`	Number	(none)	Data block number

Table 7-7 Parameters for Redo Integrity Check

Parameter Name	Type	Default Value	Description
`SCN_TEXT`	Text	0	SCN of the latest good redo (if known)

Table 7-8 Parameters for Undo Segment Integrity Check

Parameter Name	Type	Default Value	Description
`USN_NUMBER`	Text	(none)	Undo segment number

Table 7-9 Parameters for Transaction Integrity Check

Parameter Name	Type	Default Value	Description
`TXN_ID`	Text	(none)	Transaction ID

Table 7-10 Parameters for Dictionary Integrity Check

Parameter Name	Type	Default Value	Description
`CHECK_MASK`	Text	`ALL`	Possible values are: • `COLUMN_CHECKS`—Run column checks only. Verify column-level constraints in the core tables. • `ROW_CHECKS`—Run row checks only. Verify row-level constraints in the core tables. • `REFERENTIAL_CHECKS`—Run referential checks only. Verify referential constraints in the core tables. • `ALL`—Run all checks.
`TABLE_NAME`	Text	`ALL_CORE_TABLES`	Name of a single core table to check. If omitted, all core tables are checked.

7.3.3 Gathering Additional Diagnostic Data

This section describes how to gather additional diagnostic data using alert log and trace files.

- Viewing the Alert Log
  You can view the alert log with a text editor, with Cloud Control, or with the ADRCI utility.
- Finding Trace Files
  Trace files are stored in the Automatic Diagnostic Repository (ADR), in the `trace` directory under each ADR home. To help you locate individual trace files within this directory, you can use data dictionary views. For example, you can find the path to your current session's trace file or to the trace file for each Oracle Database process.

7.3.3.1 Viewing the Alert Log

You can view the alert log with a text editor, with Cloud Control, or with the ADRCI utility.

To view the alert log with Cloud Control:

1. Access the Database Home page in Cloud Control.
2. From the Oracle Database menu, select **Diagnostics**, then **Support Workbench**.
3. Under Related Links, click **Alert Log Contents**.

   The View Alert Log Contents page appears.
4. Select the number of entries to view, and then click **Go**.

To view the alert log with a text editor:

1. Connect to the database with SQL*Plus or another query tool, such as SQL Developer.
2. Query the `V$DIAG_INFO` view as shown in "Viewing ADR Locations with the V$DIAG_INFO View".
3. To view the text-only alert log, without the XML tags, complete these steps:
   a. In the `V$DIAG_INFO` query results, note the path that corresponds to the `Diag Trace` entry, and change directory to that path.
   b. Open file alert_*SID*.log with a text editor.
4. To view the XML-formatted alert log, complete these steps:
   a. In the `V$DIAG_INFO` query results, note the path that corresponds to the `Diag Alert` entry, and change directory to that path.
   b. Open the file log.xml with a text editor.

See Also:

Oracle Database Utilities for information about using the ADRCI utility to view a text version of the alert log (with XML tags stripped) and to run queries against the alert log

7.3.3.2 Finding Trace Files

Trace files are stored in the Automatic Diagnostic Repository (ADR), in the `trace` directory under each ADR home. To help you locate individual trace files within this directory, you can use data dictionary views. For example, you can find the path to your current session's trace file or to the trace file for each Oracle Database process.

To find the trace file for your current session:

- Submit the following query:

  ```
  SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
  ```

  The full path to the trace file is returned.

To find all trace files for the current instance:

- Submit the following query:

  ```
  SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Diag Trace';
  ```

  The path to the ADR trace directory for the current instance is returned.

To determine the trace file for each Oracle Database process:

- Submit the following query:

  ```
  SELECT PID, PROGRAM, TRACEFILE FROM V$PROCESS;
  ```

See Also:

- "Structure, Contents, and Location of the Automatic Diagnostic Repository"
- The ADRCI `SHOW TRACEFILE` command in *Oracle Database Utilities*

7.3.4 Creating Test Cases with SQL Test Case Builder

SQL Test Case Builder is a tool that automatically gathers information needed to reproduce the problem in a different database instance.

A **SQL test case** is a set of information that enables a developer to reproduce the execution plan for a specific SQL statement that has encountered a performance problem.

This section contains the following topics:

- Purpose of SQL Test Case Builder
  SQL Test Case Builder automates the process of gathering and reproducing information about a problem and the environment in which it occurred.
- Concepts for SQL Test Case Builder
  Key concepts for SQL Test Case Builder include SQL incidents, types of information recorded, and the form of the output.
- User Interfaces for SQL Test Case Builder
  You can access SQL Test Case Builder either through Cloud Control or using PL/SQL on the command line.
- Running SQL Test Case Builder
  You can run SQL Test Case Builder using Cloud Control.

7.3.4.1 Purpose of SQL Test Case Builder

SQL Test Case Builder automates the process of gathering and reproducing information about a problem and the environment in which it occurred.

For most SQL components, obtaining a reproducible test case is the most important factor in bug resolution speed. It is also the longest and most painful step for users. The goal of SQL Test Case Builder is to gather as much as information related to an SQL incident as possible, and then package it in a way that enables Oracle staff to reproduce the problem on a different system.

The output of SQL Test Case Builder is a set of scripts in a predefined directory. These scripts contain the commands required to re-create all the necessary objects and the environment on another database instance. After the test case is ready, you can create a zip file of the directory and move it to another database, or upload the file to Oracle Support.

7.3.4.2 Concepts for SQL Test Case Builder

Key concepts for SQL Test Case Builder include SQL incidents, types of information recorded, and the form of the output.

This section contains the following topics:

- SQL Incidents
  In the fault diagnosability infrastructure of Oracle Database, an **incident** is a single occurrence of a problem.
- What SQL Test Case Builder Captures
  SQL Test Case Builder captures permanent information about a SQL query and its environment.
- Output of SQL Test Case Builder
  The output of SQL Test Case Builder is a set of files that contains commands required to re-create the environment and all necessary objects.

7.3.4.2.1 SQL Incidents

In the fault diagnosability infrastructure of Oracle Database, an **incident** is a single occurrence of a problem.

A SQL incident is a SQL-related problem. When a problem (critical error) occurs multiple times, the database creates an incident for each occurrence. Incidents are timestamped and tracked in the Automatic Diagnostic Repository (ADR). Each incident has a numeric incident ID, which is unique within the ADR.

SQL Test Case Builder is accessible any time on the command line. In Oracle Enterprise Manager Cloud Control (Cloud Control), the SQL Test Case pages are only available after a SQL incident is found.

7.3.4.2.2 What SQL Test Case Builder Captures

SQL Test Case Builder captures permanent information about a SQL query and its environment.

The information includes the query being executed, table and index definitions (but not the actual data), PL/SQL packages and program units, optimizer statistics, SQL plan baselines, and initialization parameter settings. Starting with Oracle Database 12c, SQL Test Case Builder also captures and replays transient information, including information only available as part of statement execution.

SQL Test Case Builder supports the following:

- Adaptive plans

  SQL Test Case Builder captures inputs to the decisions made regarding adaptive plans, and replays them at each decision point. For adaptive plans, the final statistics value at each buffering statistics collector is sufficient to decide on the final plan.
- Automatic memory management

  The database automatically handles the memory requested for each SQL operation. Actions such as sorting can affect performance significantly. SQL Test Case Builder keeps track of the memory activities, for example, where the database allocated memory and how much it allocated.
- Dynamic statistics

  Dynamic statistics is an optimization technique in which the database executes a recursive SQL statement to scan a small random sample of a table's blocks to estimate predicate selectivities. Regathering dynamic statistics on a different database does not always generate the same results, for example, when data is missing. To reproduce the problem, SQL Test Case Builder exports the dynamic statistics result from the source database. In the testing database, SQL Test Case Builder reuses the same values captured from the source database instead of regathering dynamic statistics.

- Multiple execution support

  SQL Test Case Builder can capture dynamic information accumulated during multiple executions of the query. This capability is important for automatic reoptimization.
- Compilation environment and bind values replay

  The compilation environment setting is an important part of the query optimization context. SQL Test Case Builder captures nondefault settings altered by the user when running the problem query in the source database. If any nondefault parameter values are used, SQL Test Case Builder re-establishes the same values before running the query.
- Object statistics history

  The statistics history for objects is helpful to determine whether a plan change was caused by a change in statistics values. `DBMS_STATS` stores the history in the data dictionary. SQL Test Case Builder stores this statistics data into a staging table during export. During import, SQL Test Case Builder automatically reloads the statistics history data into the target database from the staging table.
- Statement history

  The statement history is important for diagnosing problems related to adaptive cursor sharing, statistics feedback, and cursor sharing bugs. The history includes execution plans and compilation and execution statistics.

See Also:

- *Oracle Database SQL Tuning Guide* for more information about adaptive query plans, supplemental dynamic statistics, automatic reoptimization, and SQL plan baselines
- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_STATS` package

7.3.4.2.3 Output of SQL Test Case Builder

The output of SQL Test Case Builder is a set of files that contains commands required to re-create the environment and all necessary objects.

By default, SQL Test Case Builder stores the files in the following directory, where *incnum* refers to the incident number and *runnum* refers to the run number:

```
$ADR_HOME/incident/incdir_incnum/SQLTCB_runnum
```

For example, a valid output file name could be as follows:

```
$ORACLE_HOME/log/diag/rdbms/dbsa/dbsa/incident/incdir_2657/SQLTCB_1
```

You can also specify a particular directory for storing the SQL Test Case Builder files by creating a directory object with the name `SQL_TCB_DIR` and running the procedure `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE` as shown in the following example:

```
CREATE OR REPLACE DIRECTORY SQL_TCB_DIR '/tmp';
```

```
DECLARE
tc CLOB;
BEGIN
  DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory => 'SQL_TCB_DIR',
    sql_text  => 'select * from hr_table',
    testcase  => tc);
END;
```

Note:

The database administrator must have read and write access permissions to the operating system directory specified in the directory object `SQL_TCB_DIR`.

You can also specify a name for a test case using the `testcase_name` parameter of the `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE` procedure. A test case name is used as a prefix for all the files generated by SQL Test Case Builder.

If you do not specify a test case name, then a default test case name having the following format is used by SQL Test Case Builder:

```
oratcb_connectionId_sqlId_sequenceNumber_sessionId
```

Here, *connectionId* is the database connection ID, *sqlId* is the SQL statement ID, *sequenceNumber* is the internal sequence number, and *sessionId* is the database session ID.

You can also specify any additional information to include in the output of SQL Test Case Builder using the `ctrlOptions` parameter of the `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE` procedure. The following are some of the options that you can specify in the `ctrlOptions` parameter:

- `compress`: This option is used to compress the SQL Test Case Builder output files into a zip file.
- `diag_event`: This option is used to specify the level of trace information to include in the SQL Test Case Builder output.
- `problem_type`: This option is used to assign an issue type for a SQL Test Case Builder test case. For example, if a test case is related to performance regression issue, then you can assign the value of `PERFORMANCE` to the `problem_type` option.

You can view the information about all the test cases generated by SQL Test Case Builder by querying the `V$SQL_TESTCASES` view as shown in the following example:

```
select testcase_name, sql_text from v$sql_testcases;

TESTCASE_NAME                         SQL_TEXT
-------------------------------------  ----------------------
oratcb_0_am8q8kudm02v9_1_00244CC50001  select * from hr_table
```

Note:

The `V$SQL_TESTCASES` view requires the existence of a SQL Test Case Builder root directory object named `SQL_TCB_DIR`. In Oracle Autonomous Database environments, this directory object is created automatically on each POD during provisioning. For on-premises databases, you must explicitly create the SQL Test Case Builder root directory object `SQL_TCB_DIR`, otherwise the `V$SQL_TESTCASES` view will not display any information. The database administrator must have read and write access permissions to the operating system directory specified in the directory object `SQL_TCB_DIR`.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SQLDIAG.EXPORT_SQL_TESTCASE` procedure
- *Oracle Database Reference* for more information about the `V$SQL_TESTCASES` view

7.3.4.3 User Interfaces for SQL Test Case Builder

You can access SQL Test Case Builder either through Cloud Control or using PL/SQL on the command line.

This section contains the following topics:

- Graphical Interface for SQL Test Case Builder
  Within Cloud Control, you can access SQL Test Case Builder from the Incident Manager page or the Support Workbench page.
- Command-Line Interface for SQL Test Case Builder
  The `DBMS_SQLDIAG` package performs tasks relating to SQL Test Case Builder.

7.3.4.3.1 Graphical Interface for SQL Test Case Builder

Within Cloud Control, you can access SQL Test Case Builder from the Incident Manager page or the Support Workbench page.

This section contains the following topics:

- Accessing the Incident Manager
  From the Incidents and Problems section on the Database Home page, you can navigate to the Incident Manager.
- Accessing the Support Workbench
  From the Oracle Database menu, you can navigate to the Support Workbench.

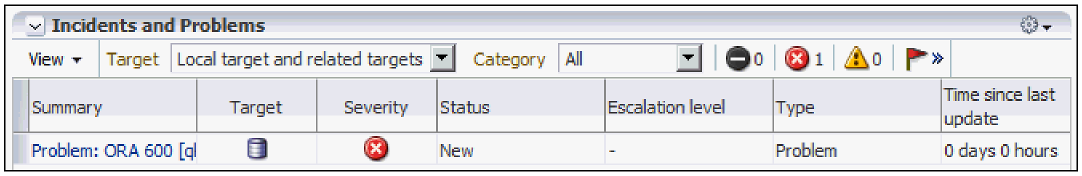
7.3.4.3.1.1 Accessing the Incident Manager

From the Incidents and Problems section on the Database Home page, you can navigate to the Incident Manager.

To access the Incident Manager:

1. Log in to Cloud Control with the appropriate credentials.
2. Under the **Targets** menu, select **Databases**.
3. In the list of database targets, select the target for the Oracle Database instance that you want to administer.
4. If prompted for database credentials, then enter the minimum credentials necessary for the tasks you intend to perform.
5. In the Incidents and Problems section, locate the SQL incident to be investigated.

   In the following example, the `ORA 600` error is a SQL incident.



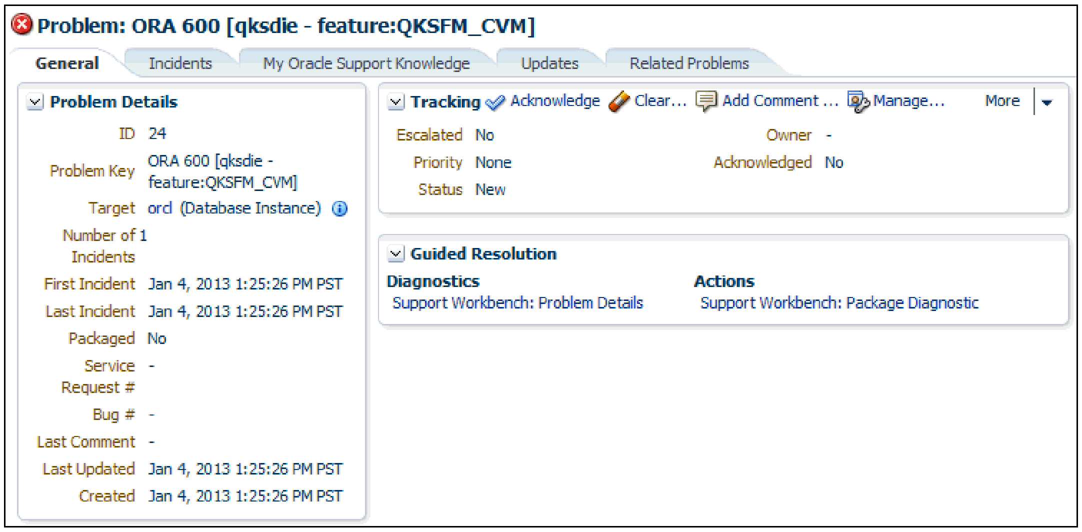
Incidents and Problems

View Target Local target and related targets Category All 0 1 0

Summary	Target	Severity	Status	Escalation level	Type	Time since last update
Problem: ORA 600 [ql		⊗	New	-	Problem	0 days 0 hours

6. Click the summary of the incident.

   The Problem Details page of the Incident Manager appears.



Problem: ORA 600 [qksdie - feature:QKSFM_CVM]

General | Incidents | My Oracle Support Knowledge | Updates | Related Problems

Problem Details

ID	24
Problem Key	ORA 600 [qksdie - feature:QKSFM_CVM]
Target	orcl (Database Instance)
Number of Incidents	1
First Incident	Jan 4, 2013 1:25:26 PM PST
Last Incident	Jan 4, 2013 1:25:26 PM PST
Packaged	No
Service Request #	-
Bug #	-
Last Comment	-
Last Updated	Jan 4, 2013 1:25:26 PM PST
Created	Jan 4, 2013 1:25:26 PM PST

Tracking Acknowledge Clear... Add Comment ... Manage... More

Escalated	No	Owner	-
Priority	None	Acknowledged	No
Status	New		

Guided Resolution

Diagnostics
Support Workbench: Problem Details

Actions
Support Workbench: Package Diagnostic

The Support Workbench page appears, with the incidents listed in a table.

7.3.4.3.1.2 Accessing the Support Workbench

From the Oracle Database menu, you can navigate to the Support Workbench.

To access the Support Workbench:

1. Log in to Cloud Control with the appropriate credentials.
2. Under the **Targets** menu, select **Databases**.
3. In the list of database targets, select the target for the Oracle Database instance that you want to administer.
4. If prompted for database credentials, then enter the minimum credentials necessary for the tasks you intend to perform.
5. From the **Oracle Database** menu, select **Diagnostics**, then **Support Workbench**.

   The Support Workbench page appears, with the incidents listed in a table.

7.3.4.3.2 Command-Line Interface for SQL Test Case Builder

The `DBMS_SQLDIAG` package performs tasks relating to SQL Test Case Builder.

This package consists of various subprograms for the SQL Test Case Builder, some of which are listed in the following table.

Table 7-11 SQL Test Case Functions in the DBMS_SQLDIAG Package

Procedure	Description
`EXPORT_SQL_TESTCASE`	Exports a SQL test case to a user-specified directory
`EXPORT_SQL_TESTCASE_DIR_BY_INC`	Exports a SQL test case corresponding to the incident ID passed as an argument
`EXPORT_SQL_TESTCASE_DIR_BY_TXT`	Exports a SQL test case corresponding to the SQL text passed as an argument
`IMPORT_SQL_TESTCASE`	Imports a SQL test case into a schema
`REPLAY_SQL_TESTCASE`	Automates reproduction of a SQL test case
`EXPLAIN_SQL_TESTCASE`	Explains a SQL test case

See Also:

Oracle Database PL/SQL Packages and Types Reference to learn more about the `DBMS_SQLDIAG` package

7.3.4.4 Running SQL Test Case Builder

You can run SQL Test Case Builder using Cloud Control.

Assumptions

This tutorial assumes the following:

- You ran the following `EXPLAIN PLAN` statement as user `sh`, which causes an internal error:

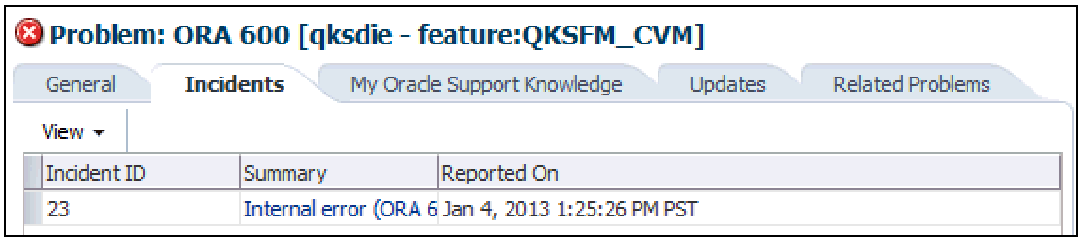
```
EXPLAIN PLAN FOR
  SELECT unit_cost, sold
  FROM   costs c,
         ( SELECT /*+ merge */ p.prod_id, SUM(quantity_sold) AS sold
           FROM   products p, sales s
           WHERE  p.prod_id = s.prod_id
           GROUP BY p.prod_id ) v
  WHERE  c.prod_id = v.prod_id;
```

- In the Incidents and Problems section on the Database Home page, a SQL incident generated by the internal error appears.
- You access the Incident Details page, as explained in "Accessing the Incident Manager".

To run SQL Test Case Builder:

1. Click the Incidents tab.

   The Problem Details page appears.

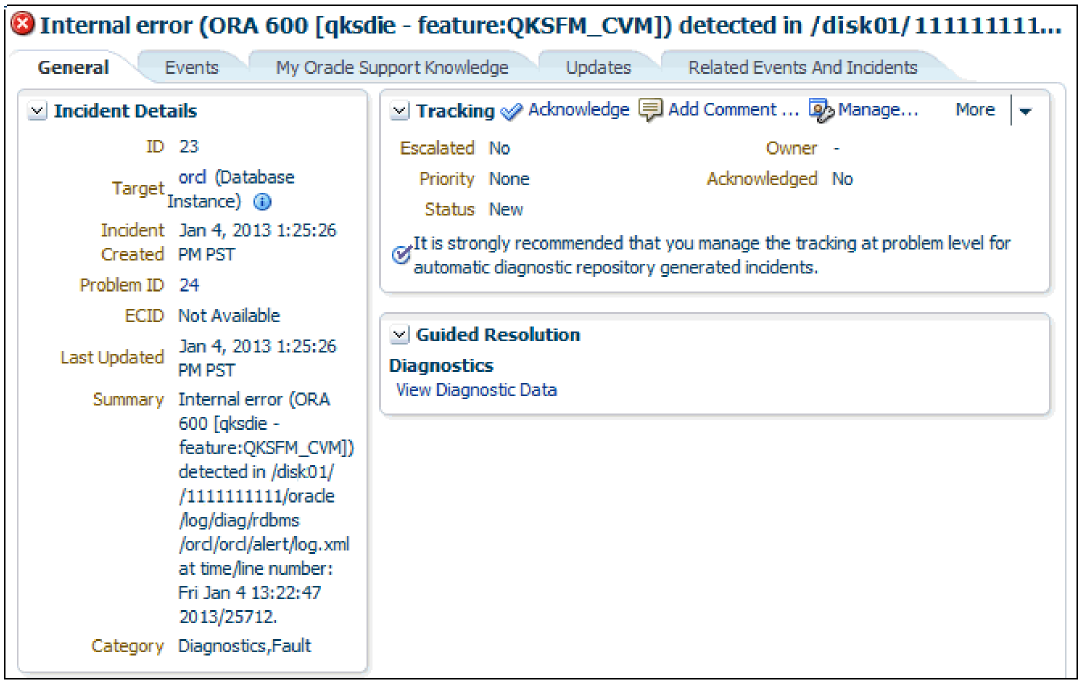


Problem: ORA 600 [qksdie - feature:QKSFM_CVM]

Incident ID	Summary	Reported On
23	Internal error (ORA 6	Jan 4, 2013 1:25:26 PM PST

2. Click the summary for the incident.

   The Incident Details page appears.



Internal error (ORA 600 [qksdie - feature:QKSFM_CVM]) detected in /disk01/111111111...

Incident Details

ID	23
Target	orcl (Database Instance)
Incident Created	Jan 4, 2013 1:25:26 PM PST
Problem ID	24
ECID	Not Available
Last Updated	Jan 4, 2013 1:25:26 PM PST
Summary	Internal error (ORA 600 [qksdie - feature:QKSFM_CVM]) detected in /disk01/ /1111111111/oracle /log/diag/rdbms /orcl/orcl/alert/log.xml at time/line number: Fri Jan 4 13:22:47 2013/25712.
Category	Diagnostics,Fault

Escalated	No	Owner	-
Priority	None	Acknowledged	No
Status	New		

It is strongly recommended that you manage the tracking at problem level for automatic diagnostic repository generated incidents.

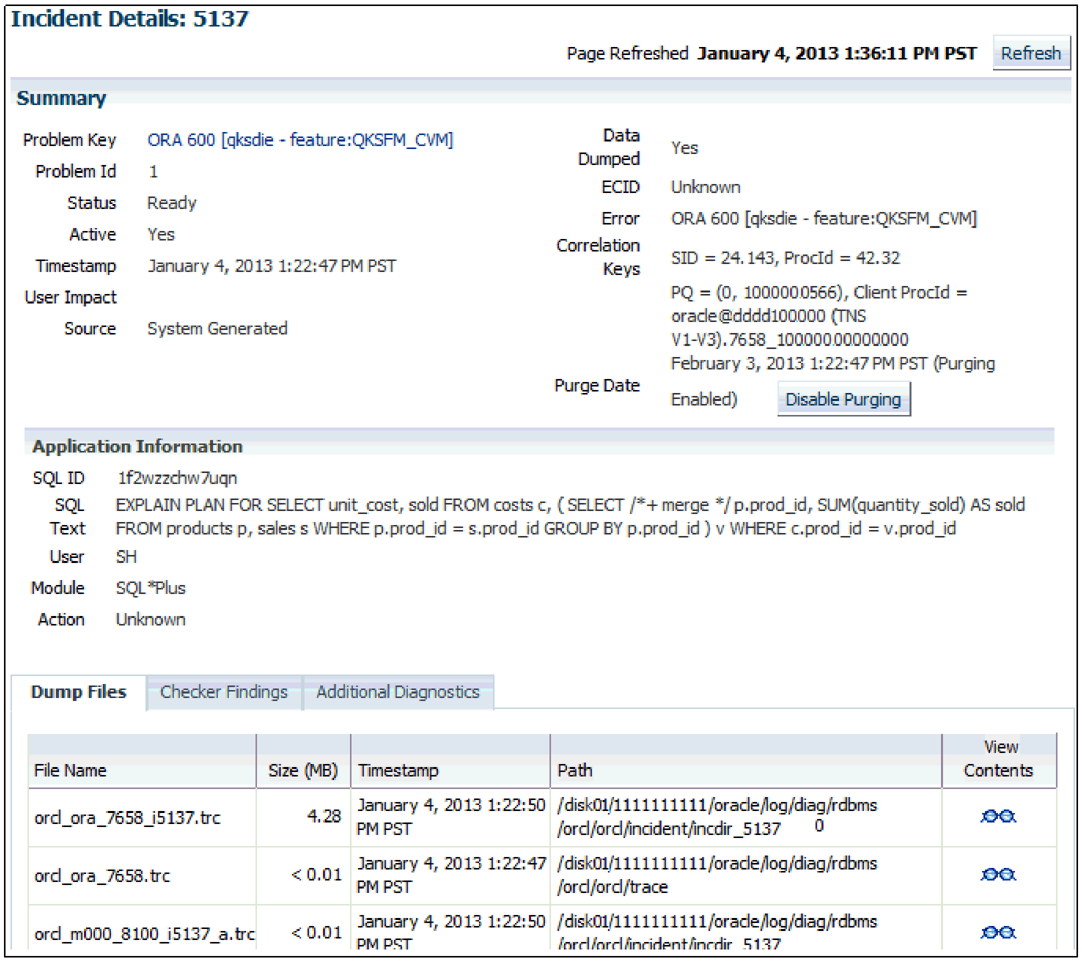
Guided Resolution

Diagnostics

View Diagnostic Data

3. In Guided Resolution, click **View Diagnostic Data**.

   The Incident Details: *incident_number* page appears.



Incident Details: 5137

Page Refreshed **January 4, 2013 1:36:11 PM PST** Refresh

Summary

Problem Key	ORA 600 [qksdie - feature:QKSFM_CVM]
Problem Id	1
Status	Ready
Active	Yes
Timestamp	January 4, 2013 1:22:47 PM PST
User Impact	
Source	System Generated
Data Dumped	Yes
ECID	Unknown
Error	ORA 600 [qksdie - feature:QKSFM_CVM]
Correlation Keys	SID = 24.143, ProcId = 42.32 PQ = (0, 1000000566), Client ProcId = oracle@dddd100000 (TNS V1-V3).7658_100000000000000
Purge Date	February 3, 2013 1:22:47 PM PST (Purging Enabled) Disable Purging

Application Information

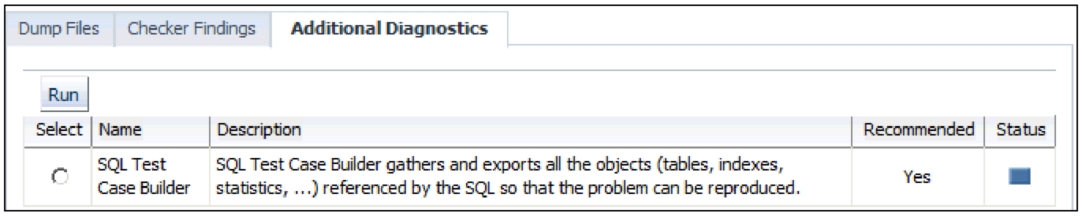
SQL ID	1f2wzzchw7uqn
SQL Text	EXPLAIN PLAN FOR SELECT unit_cost, sold FROM costs c, (SELECT /*+ merge */ p.prod_id, SUM(quantity_sold) AS sold FROM products p, sales s WHERE p.prod_id = s.prod_id GROUP BY p.prod_id) v WHERE c.prod_id = v.prod_id
User	SH
Module	SQL*Plus
Action	Unknown

Dump Files | Checker Findings | Additional Diagnostics

File Name	Size (MB)	Timestamp	Path	View Contents
orcl_ora_7658_i5137.trc	4.28	January 4, 2013 1:22:50 PM PST	/disk01/1111111111/oracle/log/diag/rdbms /orcl/orcl/incident/incdir_5137 0	👓
orcl_ora_7658.trc	< 0.01	January 4, 2013 1:22:47 PM PST	/disk01/1111111111/oracle/log/diag/rdbms /orcl/orcl/trace	👓
orcl_m000_8100_i5137_a.trc	< 0.01	January 4, 2013 1:22:50 PM PST	/disk01/1111111111/oracle/log/diag/rdbms /orcl/orcl/incident/incdir_5137	👓

4. In the Application Information section, click **Additional Diagnostics**.

   The Additional Diagnostics subpage appears.



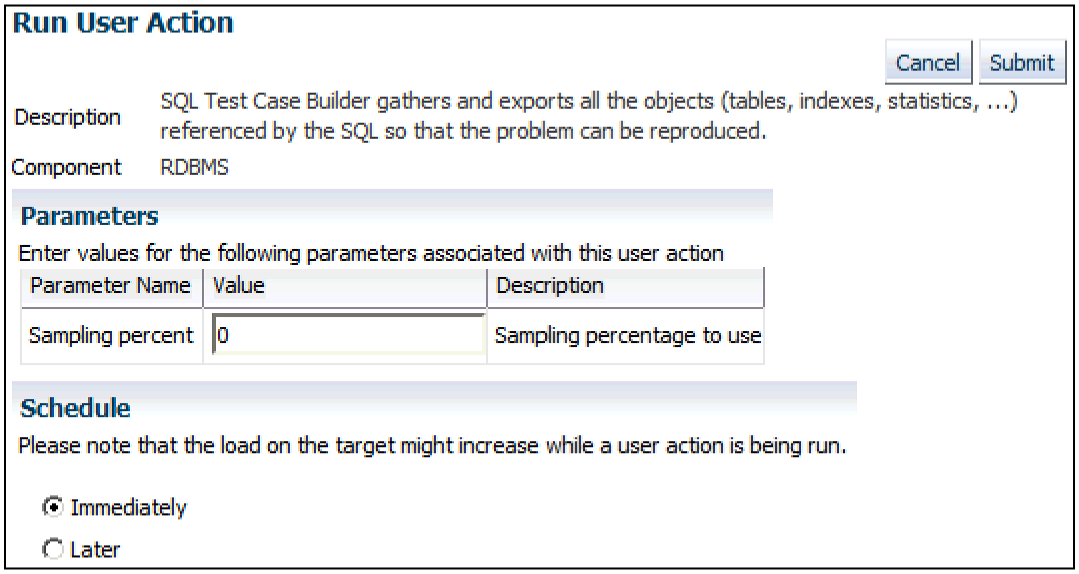
Dump Files | Checker Findings | **Additional Diagnostics**

Run

Select	Name	Description	Recommended	Status
○	SQL Test Case Builder	SQL Test Case Builder gathers and exports all the objects (tables, indexes, statistics, ...) referenced by the SQL so that the problem can be reproduced.	Yes	■

5. Select **SQL Test Case Builder**, and then click **Run**.

   The Run User Action page appears.



Run User Action

Description	SQL Test Case Builder gathers and exports all the objects (tables, indexes, statistics, ...) referenced by the SQL so that the problem can be reproduced.
Component	RDBMS

Parameters

Enter values for the following parameters associated with this user action

Parameter Name	Value	Description
Sampling percent	0	Sampling percentage to use

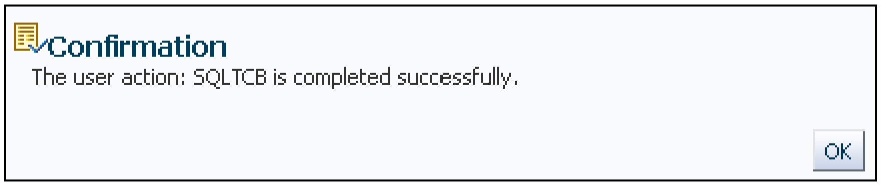
Schedule

Please note that the load on the target might increase while a user action is being run.

[✓ Immediately] Later

6. Select a sampling percentage (optional), and then click **Submit**.

   After processing completes, the Confirmation page appears.



Confirmation

The user action: SQLTCB is completed successfully.

7. Access the SQL Test Case files in the location described in "Output of SQL Test Case Builder".

7.4 Reporting Problems

Using the Enterprise Manager Support Workbench (Support Workbench), you can create, edit, and upload custom incident packages. With custom incident packages, you have fine control over the diagnostic data that you send to Oracle Support.

- Incident Packages
  You can collect diagnostic data into an intermediate logical structure called an incident package (package).
- Packaging and Uploading Problems with Custom Packaging
  You use Support Workbench (Support Workbench) to create and upload custom incident packages (packages). Before uploading, you can manually add, edit, and remove diagnostic data files from the package.
- Viewing and Modifying Incident Packages
  After creating an incident package with the custom packaging method, you can view or modify the contents of the package before uploading the package to Oracle Support.
- Creating, Editing, and Uploading Correlated Packages
  After you upload a package to Oracle Support, you can create and upload one or more correlated packages.
- Deleting Correlated Packages
  You delete a correlated package with the Support Workbench for the target for which you created the package.

- Setting Incident Packaging Preferences
  You can set incident packaging preferences. Examples of incident packaging preferences include the number of days to retain incident information, and the number of leading and trailing incidents to include in a package for each problem.

See Also:

"About the Oracle Database Fault Diagnosability Infrastructure"

7.4.1 Incident Packages

You can collect diagnostic data into an intermediate logical structure called an incident package (package).

- About Incident Packages
  For the customized approach to uploading diagnostic data to Oracle Support, you first collect the data into an intermediate logical structure called an incident package (package).
- About Correlated Diagnostic Data in Incident Packages
  To diagnose problem, it is sometimes necessary to examine not only diagnostic data that is directly related to the problem, but also diagnostic data that is *correlated* with the directly related data.
- About Quick Packaging and Custom Packaging
  The Support Workbench provides two methods for creating and uploading an incident package: the quick packaging method and the custom packaging method.
- About Correlated Packages
  Correlated packages provide a means of packaging and uploading diagnostic data for related problems.

7.4.1.1 About Incident Packages

For the customized approach to uploading diagnostic data to Oracle Support, you first collect the data into an intermediate logical structure called an incident package (package).

A **package** is a collection of metadata that is stored in the Automatic Diagnostic Repository (ADR) and that points to diagnostic data files and other files both in and out of the ADR. When you create a package, you select one or more problems to add to the package. The Support Workbench then automatically adds to the package the problem information, incident information, and diagnostic data (such as trace files and dumps) associated with the selected problems. Because a problem can have many incidents (many occurrences of the same problem), by default only the first three and last three incidents for each problem are added to the package, excluding any incidents that are over 90 days old. You can change these default numbers on the Incident Packaging Configuration page of the Support Workbench.

After the package is created, you can add any type of external file to the package, remove selected files from the package, or edit selected files in the package to remove sensitive data. As you add and remove package contents, only the package metadata is modified.

When you are ready to upload the diagnostic data to Oracle Support, you first create a zip file that contains all the files referenced by the package metadata. You then upload the zip file through My Oracle Support.

Related Topics

- Packaging and Uploading Problems with Custom Packaging
  You use Support Workbench (Support Workbench) to create and upload custom incident packages (packages). Before uploading, you can manually add, edit, and remove diagnostic data files from the package.
- Viewing and Modifying Incident Packages
  After creating an incident package with the custom packaging method, you can view or modify the contents of the package before uploading the package to Oracle Support.

7.4.1.2 About Correlated Diagnostic Data in Incident Packages

To diagnose problem, it is sometimes necessary to examine not only diagnostic data that is directly related to the problem, but also diagnostic data that is *correlated* with the directly related data.

Diagnostic data can be correlated by time, by process ID, or by other criteria. For example, when examining an incident, it may be helpful to also examine an incident that occurred five minutes after the original incident. Similarly, while it is clear that the diagnostic data for an incident should include the trace file for the Oracle Database process that was running when the incident occurred, it might be helpful to also include trace files for other processes that are related to the original process.

Thus, when problems and their associated incidents are added to a package, any correlated incidents are added at the same time, with their associated trace files.

During the process of creating the physical file for a package, the Support Workbench calls upon the Incident Packaging Service to finalize the package. **Finalizing** means adding to the package any additional trace files that are correlated by time to incidents in the package, and adding other diagnostic information such as the alert log, health checker reports, SQL test cases, configuration information, and so on. Therefore, the number of files in the zip file may be greater than the number of files that the Support Workbench had previously displayed as the package contents.

The Incident Packaging Service follows a set of rules to determine the trace files in the ADR that are correlated to existing package data. You can modify some of those rules in the Incident Packaging Configuration page in Cloud Control.

Because both initial package data and added correlated data may contain sensitive information, it is important to have an opportunity to remove or edit files that contain this information before uploading to Oracle Support. For this reason, the Support Workbench enables you to run a command that finalizes the package as a separate operation. After manually finalizing a package, you can examine the package contents, remove or edit files, and then generate and upload a zip file.

Note:

Finalizing a package does not mean closing it to further modifications. You can continue to add diagnostic data to a finalized package. You can also finalize the same package multiple times. Each time that you finalize, any new correlated data is added.

Related Topics

- Setting Incident Packaging Preferences
  You can set incident packaging preferences. Examples of incident packaging preferences include the number of days to retain incident information, and the number of leading and trailing incidents to include in a package for each problem.

7.4.1.3 About Quick Packaging and Custom Packaging

The Support Workbench provides two methods for creating and uploading an incident package: the quick packaging method and the custom packaging method.

Quick Packaging—This is the more automated method with a minimum of steps, organized in a guided workflow (a wizard). You select a single problem, provide a package name and description, and then schedule upload of the package contents, either immediately or at a specified date and time. The Support Workbench automatically places diagnostic data related to the problem into the package, finalizes the package, creates the zip file, and then uploads the file. With this method, you do not have the opportunity to add, edit, or remove package files or add other diagnostic data such as SQL test cases. However, it is the simplest and quickest way to get first-failure diagnostic data to Oracle Support. Quick packaging is the method used in the workflow described in "About Investigating, Reporting, and Resolving a Problem".

When quick packaging is complete, the package that was created by the wizard remains. You can then modify the package with custom packaging operations at a later time and manually reupload.

Custom Packaging—This is the more manual method, with more steps. It is intended for expert Support Workbench users who want more control over the packaging process. With custom packaging, you can create a new package with one or more problems, or you can add one or more problems to an existing package. You can then perform a variety of operations on the new or updated package, including:

- Adding or removing problems or incidents
- Adding, editing, or removing trace files in the package
- Adding or removing external files of any type
- Adding other diagnostic data such as SQL test cases
- Manually finalizing the package and then viewing package contents to determine if you must edit or remove sensitive data or remove files to reduce package size.

You might conduct these operations over several days, before deciding that you have enough diagnostic information to send to Oracle Support.

With custom packaging, you create the zip file and request the upload to Oracle Support as two separate steps. Each of these steps can be performed immediately or scheduled for a future date and time.

Related Topics

- About Investigating, Reporting, and Resolving a Problem
  You can use the Enterprise Manager Support Workbench (Support Workbench) to investigate and report a problem (critical error), and in some cases, resolve the problem. You can use a "roadmap" that summarizes the typical set of tasks that you must perform.
- Task 5: Package and Upload Diagnostic Data to Oracle Support
  For this task, you use the quick packaging process of the Support Workbench to package and upload the diagnostic information for the problem to Oracle Support.

7.4.1.4 About Correlated Packages

Correlated packages provide a means of packaging and uploading diagnostic data for related problems.

A database instance problem can have related problems in other database instances or in Oracle Automatic Storage Management instances. After you create and upload a package for one or more database instance problems (the "main package"), you can create and upload one or more correlated packages, each with one or more related problems. You can accomplish this only with the custom packaging workflow in Support Workbench.

Related Topics

- Related Problems Across the Topology
  For any problem identified in a database instance, the diagnosability framework can identify related problems across the topology of your Oracle Database installation.
- Creating, Editing, and Uploading Correlated Packages
  After you upload a package to Oracle Support, you can create and upload one or more correlated packages.

7.4.2 Packaging and Uploading Problems with Custom Packaging

You use Support Workbench (Support Workbench) to create and upload custom incident packages (packages). Before uploading, you can manually add, edit, and remove diagnostic data files from the package.

To package and upload problems with custom packaging:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Support Workbench" for instructions.

2. (Optional) For each problem that you want to include in the package, indicate the service request number (SR#) associated with the problem, if any. To do so, complete the following steps for each problem:

   a. In the Problems subpage at the bottom of the Support Workbench home page, select the problem, and then click **View**.

      Note:

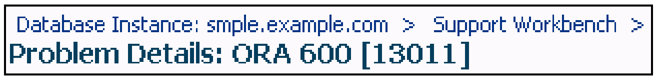
      If you do not see the desired problem in the list of problems, or if there are too many problems to scroll through, select a time period from the View list and click **Go**. You can then select the desired problem and click **View**.

      The Problem Details page appears.

   b. Next to the SR# label, click **Edit**, enter a service request number, and then click **OK**.

      The service request number is displayed on the Problem Details page.

   c. Return to the Support Workbench home page by clicking **Support Workbench** in the locator links at the top of the page.


Database Instance: smple.example.com > Support Workbench >
Problem Details: ORA 600 [13011]

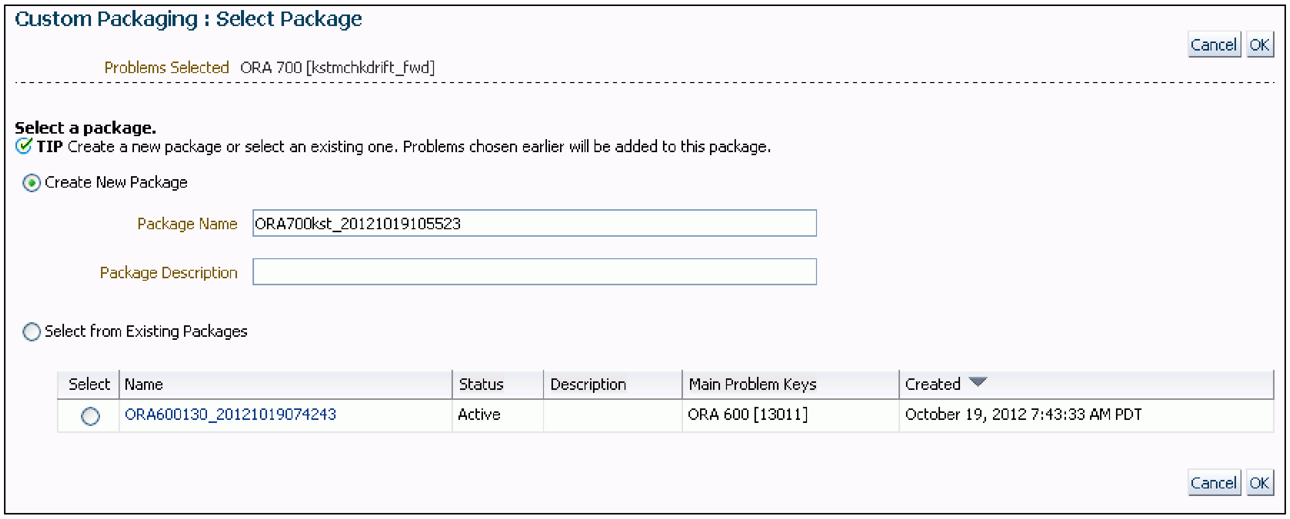
3. On the Support Workbench home page, select the problems that you want to package, and then click **Package**.

   The Select Packaging Mode page appears.

   > **Note:**
   >
   > The packaging process may automatically select additional correlated problems to add to the package. An example of a correlated problem is one that occurs within a few minutes of the selected problem. See "About Correlated Diagnostic Data in Incident Packages" for more information.

4. Select the **Custom packaging** option, and then click **Continue**.

   The Select Package page appears.



Custom Packaging : Select Package

Problems Selected ORA 700 [kstmchkdrift_fwd]

Select a package.
✓ **TIP** Create a new package or select an existing one. Problems chosen earlier will be added to this package.

● Create New Package

Package Name	ORA700kst_20121019105523
Package Description	

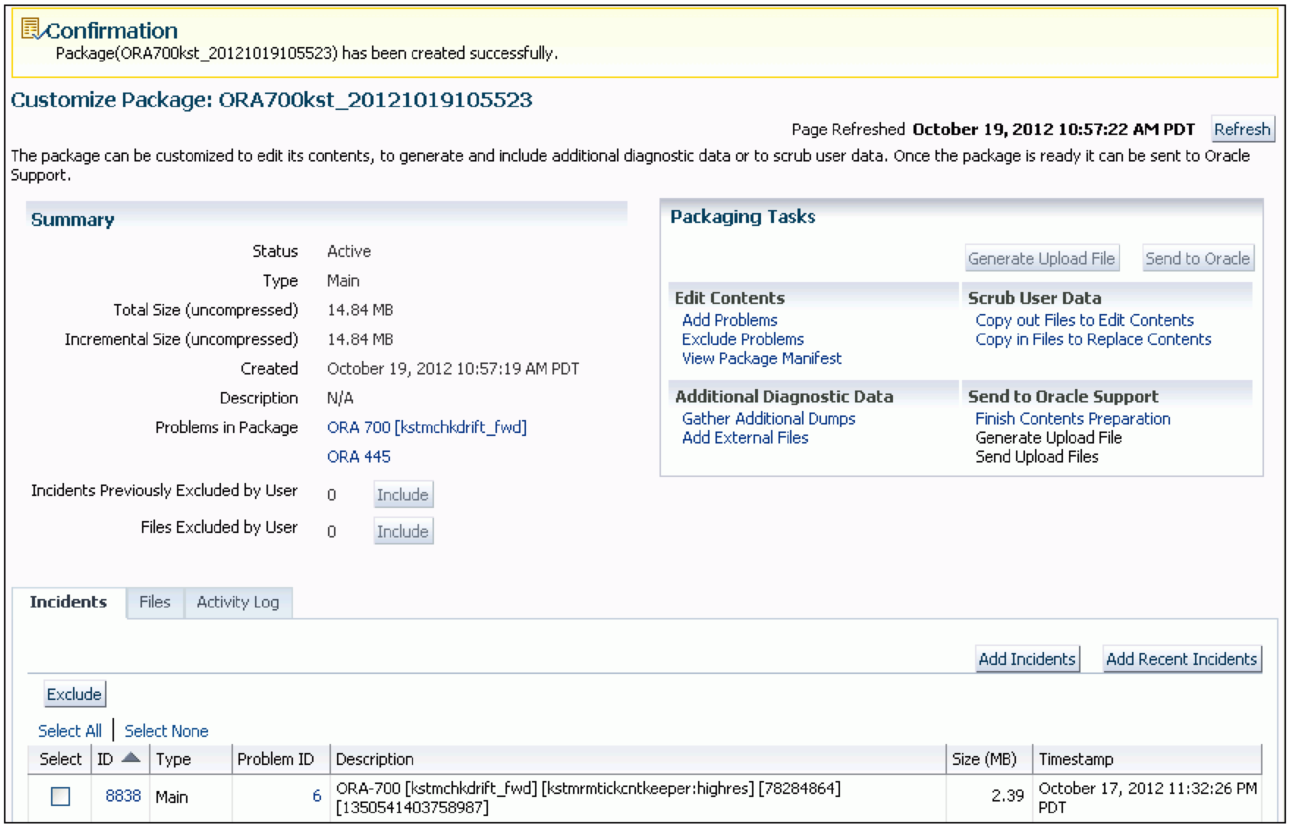
○ Select from Existing Packages

Select	Name	Status	Description	Main Problem Keys	Created
○	ORA600130_20121019074243	Active		ORA 600 [13011]	October 19, 2012 7:43:33 AM PDT

5. Do one of the following:

   - To create a new package, select the **Create new package** option, enter a package name and description, and then click **OK**.
   - To add the selected problems to an existing package, select the **Select from existing packages** option, select the package to update, and then click **OK**.

   The Customize Package page appears. It displays the problems and incidents that are contained in the package, plus a selection of packaging tasks to choose from. You run these tasks against the new package or the updated existing package.


Confirmation
Package(ORA700kst_20121019105523) has been created successfully.

Customize Package: ORA700kst_20121019105523

Page Refreshed **October 19, 2012 10:57:22 AM PDT** Refresh

The package can be customized to edit its contents, to generate and include additional diagnostic data or to scrub user data. Once the package is ready it can be sent to Oracle Support.

Summary

Status	Active
Type	Main
Total Size (uncompressed)	14.84 MB
Incremental Size (uncompressed)	14.84 MB
Created	October 19, 2012 10:57:19 AM PDT
Description	N/A
Problems in Package	ORA 700 [kstmchkdrift_fwd]
	ORA 445
Incidents Previously Excluded by User	0 Include
Files Excluded by User	0 Include

Packaging Tasks

Generate Upload File | Send to Oracle

Edit Contents
Add Problems
Exclude Problems
View Package Manifest

Scrub User Data
Copy out Files to Edit Contents
Copy in Files to Replace Contents

Additional Diagnostic Data
Gather Additional Dumps
Add External Files

Send to Oracle Support
Finish Contents Preparation
Generate Upload File
Send Upload Files

Select	ID	Type	Problem ID	Description	Size (MB)	Timestamp
☐	8838	Main	6	ORA-700 [kstmchkdrift_fwd] [kstmrmtickcntkeeper:highres] [78284864] [1350541403758987]	2.39	October 17, 2012 11:32:26 PM PDT

6. (Optional) In the Packaging Tasks section, click links to perform one or more packaging tasks. Or, use other controls on the Customize Package page and its subpages to manipulate the package. Return to the Customize Package page when you are finished.

   See "Viewing and Modifying Incident Packages" for instructions for some of the most common packaging tasks.

7. In the Packaging Tasks section of the Customize Package page, under the heading Send to Oracle Support, click **Finish Contents Preparation** to finalize the package.

   A list (or partial list) of files included in the package is displayed. (This may take a while.) The list includes files that were determined to contain correlated diagnostic information and added by the finalization process.

   See "About Correlated Diagnostic Data in Incident Packages" for a definition of package finalization.

8. Click **Files** to view all the files in the package. Examine the list to see if there are any files that might contain sensitive data that you do not want to expose. If you find such files, then exclude (remove) or edit them.

   See "Editing Incident Package Files (Copying Out and In)" and "Removing Incident Package Files" for instructions for editing and removing files.

   To view the contents of a file, click the eyeglasses icon in the rightmost column in the table of files. Enter host credentials, if prompted.

   Note:

   Trace files are generally for Oracle internal use only.

9. Click **Generate Upload File**.

   The Generate Upload File page appears.

10. Select the **Full** or **Incremental** option to generate a full package zip file or an incremental package zip file.

    For a full package zip file, all the contents of the package (original contents and all correlated data) are always added to the zip file.

    For an incremental package zip file, only the diagnostic information that is new or modified since the last time that you created a zip file for the same package is added to the zip file. For example, if trace information was appended to a trace file since that file was last included in the generated physical file for a package, the trace file is added to the incremental package zip file. Conversely, if no changes were made to a trace file since it was last uploaded for a package, that trace file is not included in the incremental package zip file.

    > **Note:**
    >
    > The Incremental option is dimmed (unavailable) if an upload file was never created for the package.

11. Schedule file creation either immediately or at a future date and time (select **Immediately** or **Later**), and then click **Submit**.

    File creation can use significant system resources, so it may be advisable to schedule it for a period of low system usage.

    A Processing page appears, and creation of the zip file proceeds. A confirmation page appears when processing is complete.

    > **Note:**
    >
    > The package is automatically finalized when the zip file is created.

12. Click **OK**.

    The Customize Package page returns.

13. Click **Send to Oracle**.

    The View/Send Upload Files page appears.

14. (Optional) Click the **Send Correlated Packages** link to create correlated packages and send them to Oracle.

    See "Creating, Editing, and Uploading Correlated Packages". When you are finished working with correlated packages, return to the View/Send Upload Files page by clicking the **Package Details** link at the top of the page, clicking **Customize Package**, and then clicking **Send to Oracle** again.

15. Select the zip files to upload, and then click **Send to Oracle**.

    The Send to Oracle page appears. The selected zip files are listed in a table.

16. Fill in the requested My Oracle Support information. Next to Create new Service Request (SR), select **Yes** or **No**. If you select Yes, a draft service request is created for you. You

must later log in to My Oracle Support and fill in the service request details. If you select No, enter an existing service request number.

17. Schedule the upload to take place immediately or at a future date and time, and then click **Submit**.

    A Processing page appears. If the upload is completed successfully, a confirmation page appears. If the upload could not complete, an error page appears. The error page may include a message that requests that you upload the zip file to Oracle manually. If so, contact your Oracle Support representative for instructions.

18. Click **OK**.

    The View/Send Upload Files page returns. Under the Time Sent column, check the status of the files that you attempted to upload.

19. (Optional) Create and upload correlated packages.

    See "Creating, Editing, and Uploading Correlated Packages" for instructions.

See Also:

- "About Incidents and Problems"
- "About Incident Packages"
- "About Quick Packaging and Custom Packaging"

7.4.3 Viewing and Modifying Incident Packages

After creating an incident package with the custom packaging method, you can view or modify the contents of the package before uploading the package to Oracle Support.

In addition, after using the quick packaging method to package and upload diagnostic data, you can view or modify the contents of the package that the Support Workbench created, and then reupload the package. To modify a package, you choose from among a selection of *packaging tasks*, most of which are available from the Customize Package page.

- Viewing Package Details
  The Package Details page contains information about the incidents, trace files, and other files in a package, and enables you to view and add to the package activity log.
- Accessing the Customize Package Page
  The Customize Package page is used to perform various packaging tasks, such as adding and removing problems; adding, removing, and scrubbing (editing) package files; and generating and uploading the package zip file.
- Editing Incident Package Files (Copying Out and In)
  The Support Workbench enables you to edit one or more files in an incident package.
- Adding an External File to an Incident Package
  You can add any type of external file to an incident package.
- Removing Incident Package Files
  You can remove one or more files of any type from the incident package.
- Viewing and Updating the Incident Package Activity Log
  The Support Workbench maintains an activity log for each incident package.

See Also:

- "About Incident Packages"
- "Packaging and Uploading Problems with Custom Packaging"

7.4.3.1 Viewing Package Details

The Package Details page contains information about the incidents, trace files, and other files in a package, and enables you to view and add to the package activity log.

To view package details:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Support Workbench" for instructions.
2. Click **Packages** to view the Packages subpage.

   A list of packages that are currently in the Automatic Diagnostic Repository (ADR) is displayed.
3. (Optional) To reduce the number of packages displayed, enter text into the **Search** field above the list, and then click **Go**.

   All packages that contain the search text anywhere in the package name are displayed. To view the full list of packages, remove the text from the **Search** field and click **Go** again.
4. Under the Package Name column, click the link for the desired package.

   The Package Details page appears.

7.4.3.2 Accessing the Customize Package Page

The Customize Package page is used to perform various packaging tasks, such as adding and removing problems; adding, removing, and scrubbing (editing) package files; and generating and uploading the package zip file.

To access the Customize Package page:

1. Access the Package Details page for the desired package, as described in "Viewing Package Details".
2. Click **Customize Package**.

   The Customize Package page appears.

7.4.3.3 Editing Incident Package Files (Copying Out and In)

The Support Workbench enables you to edit one or more files in an incident package.

You may want to do this to delete or overwrite sensitive data in the files. To edit package files, you must first copy the files out of the package into a designated directory, edit the files with a text editor or other utility, and then copy the files back into the package, overwriting the original package files.

The following procedure assumes that the package is already created and contains diagnostic data.

To edit incident package files:

1. Access the Customize Package page for the desired incident package.

   See "Accessing the Customize Package Page" for instructions.
2. In the Packaging Tasks section, under the Scrub User Data heading, click **Copy out Files to Edit contents**.

   If prompted for host credentials, enter credentials and then click **OK**.

   The Copy Out Files page appears. It displays the name of the host to which you can copy files.
3. Do one of the following to specify a destination directory for the files:
   - Enter a directory path in the **Destination Folder** field.
   - Click the magnifying glass icon next to the **Destination Folder** field, and then complete the following steps:
     a. If prompted for host credentials, enter credentials for the host to which you want to copy out the files, and then click **OK**. (Select **Save as Preferred Credential** to avoid the prompt for credentials next time.)

        The Browse and Select: File or Directory window appears.
     b. Select the desired destination directory, and then click **Select**.

        The Browse and Select: File or Directory window closes, and the path to the selected directory appears in the Destination Folder field of the Copy Out Files page.
4. Under Files to Copy Out, select the desired files, and then click **OK**.

   Note:

   If you do not see the desired files, then they may be on another page. Click the **Next** link to view the next page. Continue clicking **Next**, or select from the list of file numbers (to the left of the Next link) until you see the desired files. You can then select the files and click **OK**.

   The Customize Package page returns, displaying a confirmation message that lists the files that were copied out.
5. Using a text editor or other utility, edit the files.
6. On the Customize Package page, in the Packaging Tasks section, under the Scrub User Data heading, click **Copy in Files to Replace Contents**.

   The Copy In Files page appears. It displays the files that you copied out.
7. Select the files to copy in, and then click **OK**.

   The files are copied into the package, overwriting the existing files. The Customize Package page returns, displaying a confirmation message that lists the files that were copied in.

7.4.3.4 Adding an External File to an Incident Package

You can add any type of external file to an incident package.

To add an external file to an incident package:

1. Access the Customize Package page for the desired incident package.

   See "Accessing the Customize Package Page" for instructions.

2. Click the **Files** link to view the Files subpage.

   From this page, you can add and remove files to and from the package.

3. Click **Add external files**.

   The Add External File page appears. It displays the host name from which you may select a file.

4. Do one of the following to specify a file to add:

   - Enter the full path to the file in the **File Name** field.
   - Click the magnifying glass icon next to the **File Name** field, and then complete the following steps:

     a. If prompted for host credentials, enter credentials for the host on which the external file resides, and then click **OK**. (Select **Save as Preferred Credential** to avoid the prompt for credentials next time.)

     b. In the Browse and Select: File or Directory window, select the desired file and then click **Select**.

        The Browse and Select window closes, and the path to the selected file appears in the File Name field of the Add External File page.

5. Click **OK**.

   The Customize Package page returns, displaying the Files subpage. The selected file is now shown in the list of files.

7.4.3.5 Removing Incident Package Files

You can remove one or more files of any type from the incident package.

To remove incident package files:

1. Access the Customize Package page for the desired incident package.

   See "Accessing the Customize Package Page" for instructions.

2. Click the **Files** link to view the Files subpage.

   A list of files in the package is displayed.

   If you have not yet generated a physical file for this package, all package files are displayed in the list. If you have already generated a physical file, then a View list appears above the files list. It enables you to choose between viewing only incremental package contents or the full package contents. The default selection is incremental package contents. This default selection displays only those package files that were created or modified since the last time that a physical file was generated for the package. Select **Full package contents** from the View list to view all package files.

3. Select the files to remove, and then click **Exclude**.

Note:

If you do not see the desired files, then they may be on another page. Click the **Next** link to view the next page. Continue clicking **Next**, or select from the list of file numbers (to the left of the Next link) until you see the desired files. You can then select the files and click **Remove**.

7.4.3.6 Viewing and Updating the Incident Package Activity Log

The Support Workbench maintains an activity log for each incident package.

Most activities that you perform on a package, such as adding or removing files or creating a package zip file, are recorded in the log. You can also add your own notes to the log. This is especially useful if multiple database administrators are working with packages.

To view and update the incident package activity log:

1. Access the Package Details page for the desired incident package.

   See "Viewing Package Details" for instructions.
2. Click the **Activity Log** link to view the Activity Log subpage.

   The activity log is displayed.
3. To add your own comment to the activity log, enter text into the **Comment** field, and then click **Add Comment**.

   Your comment is appended to the list.

7.4.4 Creating, Editing, and Uploading Correlated Packages

After you upload a package to Oracle Support, you can create and upload one or more correlated packages.

This is recommended if critical alerts appeared in the Related Alerts section of the Database Home page. The correlated packages are associated with the original package, also known as the **main package**. The main package contains problems that occurred in a database instance. Correlated packages contain problems that occurred on other instances (Oracle ASM instances or other database instances) and that are related problems for the problems in the main package. There can be only one correlated package for each related instance.

To create, edit, and upload a correlated package:

1. View the Package Details page for the main package.

   See "Viewing Package Details" for instructions.
2. On the Package Details page, click **Customize Package**.
3. On the Customize Package page, in the Packaging Tasks section, under Additional Diagnostic Data, click **Create/Update Correlated Packages**.
4. On the Correlated Packages page, under Correlated Packages, select one or more instances that have incidents and click **Create**.

   A confirmation message appears, and the package IDs of the newly created correlated packages appear in the ID column.
5. Select the instance on which you created the correlated package, and click **Finish Contents Preparation**.

A confirmation message appears.

6. (Optional) View and edit a correlated package by completing these steps:
   a. Click the package ID to view the package.

      If prompted for credentials, enter them and click **Login**.
   b. On the Package Details page, click **Files** to view the files in the package.
   c. Click **Customize Package** and perform any desired customization tasks, as described in "Viewing and Modifying Incident Packages".
7. For each correlated package to upload, click **Generate Upload File**.
8. For each correlated package to send to Oracle, select the package and click **Send to Oracle**.

> **Note:**
>
> If **Send to Oracle** is unavailable (dimmed), then there were no correlated incidents for the instance.

> **See Also:**
>
> - "About Correlated Packages"
> - "Related Problems Across the Topology"

7.4.5 Deleting Correlated Packages

You delete a correlated package with the Support Workbench for the target for which you created the package.

For example, if you created a correlated package for an Oracle ASM instance target, access the Support Workbench for that Oracle ASM instance.

To delete a correlated package:

1. Access the Support Workbench for the target on which you created the correlated package.

   > **Tip:**
   >
   > See the Related Links section at the bottom of any Support Workbench page. Or, see "Viewing Problems with the Support Workbench"
2. Click **Packages** to view the Packages subpage.
3. Locate the correlated package in the list. Verify that it is a correlated package by viewing the package description.
4. Select the package and click **Delete**.
5. On the confirmation page, click **Yes**.

See Also:

- "About Correlated Packages"
- "Related Problems Across the Topology"

7.4.6 Setting Incident Packaging Preferences

You can set incident packaging preferences. Examples of incident packaging preferences include the number of days to retain incident information, and the number of leading and trailing incidents to include in a package for each problem.

By default, if a problem has many incidents, only the first three and last three incidents are packaged. You can change these and other incident packaging preferences with Cloud Control or with the ADRCI utility.

To set incident packaging preferences with Cloud Control:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Support Workbench" for instructions.
2. In the Related Links section at the bottom of the page, click **Incident Packaging Configuration**.

   The View Incident Packaging Configuration page appears. Click **Help** to view descriptions of the settings on this page.
3. Click **Edit**.

   The Edit Incident Packaging Configuration page appears.
4. Edit settings, and then click **OK** to apply changes.

See Also:

- "About Incident Packages"
- "About Incidents and Problems"
- "Task 5: Package and Upload Diagnostic Data to Oracle Support"
- *Oracle Database Utilities* for information on ADRCI

7.5 Resolving Problems

This section describes how to resolve database problems using advisor tools, such as SQL Repair Advisor and Data Recovery Advisor, and the resource management tools, such as the Resource Manager and related APIs.

- Repairing SQL Failures with the SQL Repair Advisor
  In the rare case that a SQL statement fails with a critical error, you can run the SQL Repair Advisor to try to repair the failed statement.

- Repairing Data Corruptions with the Data Recovery Advisor
  You use the Data Recovery Advisor to repair data block corruptions, undo corruptions, data dictionary corruptions, and more.
- Quarantine for Execution Plans for SQL Statements Consuming Excessive System Resources
  Starting with Oracle Database 19c, you can use the SQL Quarantine infrastructure (SQL Quarantine) to quarantine execution plans for SQL statements that are terminated by the Resource Manager for consuming excessive system resources in an Oracle database. An individual SQL statement may have multiple execution plans, and if it attempts to use the execution plan that is quarantined, then that SQL statement is not allowed to run, thus preventing database performance degradation.
- Viewing Attention Log Information
  Access information stored in the attention log either by opening the file with any text editor or by querying the `V$DIAG_ALERT_EXT` view.

7.5.1 Repairing SQL Failures with the SQL Repair Advisor

In the rare case that a SQL statement fails with a critical error, you can run the SQL Repair Advisor to try to repair the failed statement.

- About the SQL Repair Advisor
  You run the SQL Repair Advisor after a SQL statement fails with a critical error.
- Running the SQL Repair Advisor Using Cloud Control
  You can run the SQL Repair Advisor from the Problem Details page of the Support Workbench of Cloud Control.
- Running the SQL Repair Advisor Using the DBMS_SQLDIAG Package Subprograms
  You can run the SQL Repair Advisor using the `DBMS_SQLDIAG` package subprograms.
- Viewing, Disabling, or Removing a SQL Patch Using Cloud Control
  After you apply a SQL patch with the SQL Repair Advisor, you can view it to confirm its presence, disable it, or remove it using Cloud Control. One reason to disable or remove a patch is if you install a later release of Oracle Database that fixes the bug that caused the failure in the patched SQL statement.
- Disabling or Removing a SQL Patch Using DBMS_SQLDIAG Package Subprograms
  After you apply a SQL patch with the SQL Repair Advisor, you can disable or remove it using the `DBMS_SQLDIAG` package subprograms. One reason to disable or remove a patch is if you install a later release of Oracle Database that fixes the bug that caused the failure in the patched SQL statement.
- Exporting and Importing a Patch Using DBMS_SQLDIAG Package Subprograms
  A patch created using the SQL Repair Advisor can be exported out of one system and imported into another system using `DBMS_SQLDIAG` package subprograms.

7.5.1.1 About the SQL Repair Advisor

You run the SQL Repair Advisor after a SQL statement fails with a critical error.

The advisor analyzes the statement and in many cases recommends a patch to repair the statement. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternate execution plan for future executions.

You can run the SQL Repair Advisor using either Cloud Control or `DBMS_SQLDIAG` package subprograms.

7.5.1.2 Running the SQL Repair Advisor Using Cloud Control

You can run the SQL Repair Advisor from the Problem Details page of the Support Workbench of Cloud Control.

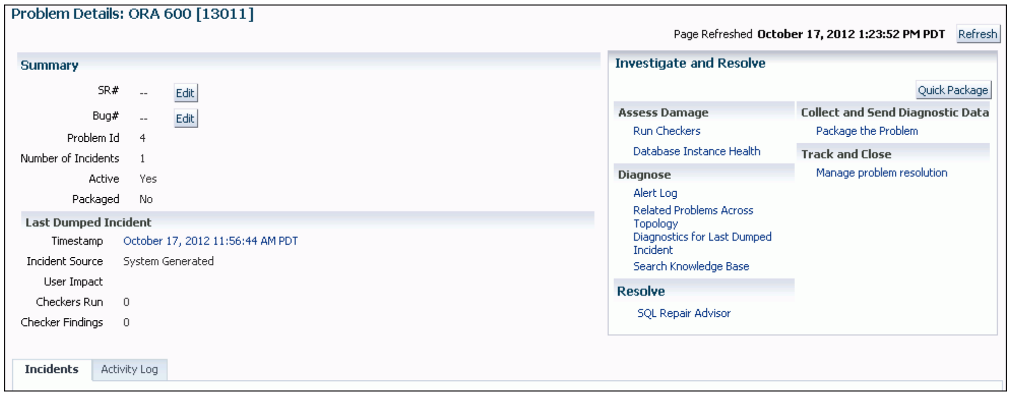
Typically, you do so when you were already notified of a critical error caused by your SQL statement and that you followed the workflow described in "About Investigating, Reporting, and Resolving a Problem".

To run the SQL Repair Advisor using Cloud Control:

1. Access the Problem Details page for the problem that pertains to the failed SQL statement.

   See "Viewing Problems with the Support Workbench" for instructions.

2. In the Investigate and Resolve section, under the Resolve heading, click **SQL Repair Advisor**.



Problem Details: ORA 600 [13011]

Page Refreshed **October 17, 2012 1:23:52 PM PDT**

Summary

SR#	--	Edit
Bug#	--	Edit
Problem Id	4	
Number of Incidents	1	
Active	Yes	
Packaged	No	

Last Dumped Incident

Timestamp	October 17, 2012 11:56:44 AM PDT
Incident Source	System Generated
User Impact	
Checkers Run	0
Checker Findings	0

Investigate and Resolve

Assess Damage
- Run Checkers
- Database Instance Health

Diagnose
- Alert Log
- Related Problems Across Topology
- Diagnostics for Last Dumped Incident
- Search Knowledge Base

Resolve
- SQL Repair Advisor

Collect and Send Diagnostic Data
- Package the Problem

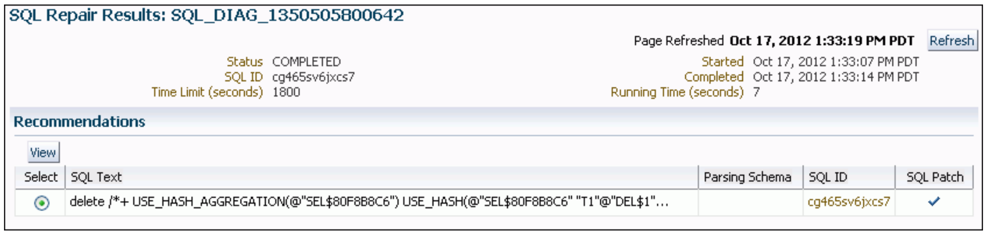
Track and Close
- Manage problem resolution

3. On the SQL Repair Advisor page, complete these steps:

   a. Modify the preset task name if desired, optionally enter a task description, modify or clear the optional time limit for the advisor task, and adjust settings to schedule the advisor to run either immediately or at a future date and time.

   b. Click **Submit**.

   A "Processing" page appears. After a short delay, the SQL Repair Results page appears.



SQL Repair Results: SQL_DIAG_1350505800642

Page Refreshed **Oct 17, 2012 1:33:19 PM PDT**

Status	COMPLETED	Started	Oct 17, 2012 1:33:07 PM PDT
SQL ID	cg465sv6jxcs7	Completed	Oct 17, 2012 1:33:14 PM PDT
Time Limit (seconds)	1800	Running Time (seconds)	7

Recommendations

Select	SQL Text	Parsing Schema	SQL ID	SQL Patch
●	delete /*+ USE_HASH_AGGREGATION(@"SEL$80F8B8C6") USE_HASH(@"SEL$80F8B8C6" "T1"@"DEL$1"...		cg465sv6jxcs7	✓

A check mark in the SQL Patch column indicates that a recommendation is present. The absence of a check mark in this column means that the SQL Repair Advisor was unable to devise a patch for the SQL statement.

Note:

If the SQL Repair Results page fails to appear, then complete these steps to display it:

a. Go to the Database Home page.

b. From the Performance menu, select **Advisors Home**.

c. On the Advisor Central page, in the Results list, locate the most recent entry for the SQL Repair Advisor.

d. Select the entry and click **View Result**.

4. If a recommendation is present (there is a check mark in the SQL Patch column), then click **View** to view the recommendation.

   The Repair Recommendations page appears, showing the recommended patch for the statement.

5. Click **Implement**.

   The SQL Repair Results page returns, showing a confirmation message.

6. (Optional) Click **Verify using SQL Worksheet** to run the statement in the SQL worksheet and verify that the patch successfully repaired the statement.

7.5.1.3 Running the SQL Repair Advisor Using the DBMS_SQLDIAG Package Subprograms

You can run the SQL Repair Advisor using the `DBMS_SQLDIAG` package subprograms.

Typically, you do so when you are notified of a critical error caused by your SQL statement and that you followed the workflow described in "About Investigating, Reporting, and Resolving a Problem".

You run the SQL Repair Advisor by creating and executing a diagnostic task using the `DBMS_SQLDIAG` package subprograms `CREATE_DIAGNOSIS_TASK` and `EXECUTE_DIAGNOSIS_TASK` respectively. The SQL Repair Advisor first reproduces the critical error and then tries to produce a workaround in the form of a SQL patch, which you can apply using the `ACCEPT_SQL_PATCH` subprogram.

Note:

Starting with Oracle Database 19c, you can also use a single subprogram `SQL_DIAGNOSE_AND_REPAIR` to create a diagnostic task, execute it, and accept SQL patch recommendation for a given SQL statement. Thus, the `SQL_DIAGNOSE_AND_REPAIR` subprogram achieves the functionality of all the following subprograms – `CREATE_DIAGNOSIS_TASK`, `EXECUTE_DIAGNOSIS_TASK`, and `ACCEPT_SQL_PATCH`.

To run the SQL Repair Advisor using the DBMS_SQLDIAG package subprograms:

1. **Identify the problematic SQL statement**

   Consider the SQL statement that gives a critical error:

```
DELETE FROM t t1
WHERE t1.a = 'a' AND
      ROWID <> (SELECT MAX(ROWID)
                FROM t t2
                WHERE t1.a = t2.a AND
                      t1.b = t2.b AND
                      t1.d = t2.d)
```

You use the SQL Repair Advisor to repair this critical error.

2. **Create a diagnosis task**

   Run `DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK`. You can specify an optional task name, an optional time limit for the advisor task, and problem type. In the example below, we specify the SQL text, the task name as '`error_task`' and a problem type as '`DBMS_SQLDIAG.PROBLEM_TYPE_COMPILATION_ERROR`'.

```
DECLARE
  rep_out   CLOB;
  t_id      VARCHAR2(50);
BEGIN
  t_id := DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
           sql_text => 'DELETE FROM t t1
                        WHERE t1.a = ''a'' AND
                              ROWID <> (SELECT MAX(ROWID)
                                        FROM t t2
                                        WHERE t1.a = t2.a AND
                                              t1.b = t2.b AND
                                              t1.d = t2.d)',
           task_name => 'error_task',
           problem_type => DBMS_SQLDIAG.PROBLEM_TYPE_COMPILATION_ERROR);
```

3. **Execute the diagnosis task**

   To execute the workaround generation and analysis phase of the SQL Repair Advisor, you run `DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK` with the task ID returned by the `CREATE_DIAGNOSIS_TASK`. After a short delay, the SQL Repair Advisor returns. As part of its execution, the SQL Repair Advisor keeps a record of its findings which can be accessed through the reporting facilities of SQL Repair Advisor.

```
DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK (t_id);
```

4. **Generate a report for the diagnosis task**

   The analysis of the diagnosis task is accessed using `DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK`. If the SQL Repair Advisor was able to find a workaround, it recommends a SQL Patch. A SQL Patch is similar to a SQL profile, but unlike the SQL Profile, it is used to workaround compilation or execution errors.

```
rep_out := DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK (t_id, DBMS_SQLDIAG.TYPE_TEXT);
DBMS_OUTPUT.PUT_LINE ('Report : ' ||  rep_out);
END;
/
```

5. **Apply the patch**

   If a patch recommendation is present in the report, you can run `DBMS_SQLDIAG.ACCEPT_SQL_PATCH` to accept the patch. This procedure takes task name as an argument.

```
EXECUTE DBMS_SQLDIAG.ACCEPT_SQL_PATCH(task_name => 'error_task', task_owner =>
'SYS', replace => TRUE);
```

6. **Test the patch**

Now that you have accepted the patch, you can rerun the SQL statement. This time, it will not give you the critical error. If you run *explain plan* for this statement, you will see that a SQL patch was used to generate the plan.

```
DELETE FROM t t1
WHERE t1.a = 'a' AND
      ROWID <> (SELECT max(rowid)
                FROM t t2
                WHERE t1.a = t2.a AND
                      t1.b = t2.b AND
                      t1.d = t2.d);
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SQLDIAG` package subprograms

7.5.1.4 Viewing, Disabling, or Removing a SQL Patch Using Cloud Control

After you apply a SQL patch with the SQL Repair Advisor, you can view it to confirm its presence, disable it, or remove it using Cloud Control. One reason to disable or remove a patch is if you install a later release of Oracle Database that fixes the bug that caused the failure in the patched SQL statement.

To view, disable, or remove a SQL patch using Cloud Control:

1. Access the Database Home page in Cloud Control.
2. From the Performance menu, select **SQL**, then **SQL Plan Control**.

   The SQL Plan Control page appears.
3. Click **SQL Patch** to display the SQL Patch subpage.

   The SQL Patch subpage displays all SQL patches in the database.
4. Locate the desired patch by examining the associated SQL text.

   Click the SQL text to view the complete text of the statement. After viewing the SQL text, click **Return**.
5. To disable the patch on the SQL Patch subpage, select it, and then click **Disable**.

   A confirmation message appears, and the patch status changes to `DISABLED`. You can later reenable the patch by selecting it and clicking **Enable**.
6. To remove the patch, select it, and then click **Drop**.

   A confirmation message appears.

See Also:

"About the SQL Repair Advisor"

7.5.1.5 Disabling or Removing a SQL Patch Using DBMS_SQLDIAG Package Subprograms

After you apply a SQL patch with the SQL Repair Advisor, you can disable or remove it using the `DBMS_SQLDIAG` package subprograms. One reason to disable or remove a patch is if you install a later release of Oracle Database that fixes the bug that caused the failure in the patched SQL statement.

To disable a SQL patch using DBMS_SQLDIAG package subprogram:

Run the procedure `DBMS_SQLDIAG.ALTER_SQL_PATCH` by specifying the patch name to disable with the status value of `DISABLED`.

The following example disables the SQL patch `sql_patch_12345`.

```
EXEC DBMS_SQLDIAG.ALTER_SQL_PATCH('sql_patch_12345', 'STATUS', 'DISABLED');
```

To remove a SQL patch using DBMS_SQLDIAG package subprogram:

Run the procedure `DBMS_SQLDIAG.DROP_SQL_PATCH` by specifying the patch name to remove. The patch name can be obtained from the explain plan section or by querying the view `DBA_SQL_PATCHES`.

The following example removes the SQL patch `sql_patch_12345`.

```
EXEC DBMS_SQLDIAG.DROP_SQL_PATCH('sql_patch_12345');
```

See Also:

- "About the SQL Repair Advisor"
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SQLDIAG` package subprograms

7.5.1.6 Exporting and Importing a Patch Using DBMS_SQLDIAG Package Subprograms

A patch created using the SQL Repair Advisor can be exported out of one system and imported into another system using `DBMS_SQLDIAG` package subprograms.

Patches can be exported out of one system and imported into another by using a staging table. Like with SQL diagnosis sets, the operation of inserting into the staging table is called as "pack", and the operation of creating patches from staging table data is called as "unpack".

To export and import a patch using the DBMS_SQLDIAG package subprograms:

1. Create a staging table owned by user '`SH`' by calling `DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH`:

```
EXEC DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH(
    table_name          =>  'STAGING_TABLE',
    schema_name         =>  'SH');
```

2. Call `DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH` one or more times to write SQL patch data into the staging table. In this case, copy data for all SQL patches in the `DEFAULT` category into a staging table owned by the current schema owner:

```
EXEC DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH(
    staging_table_name  =>  'STAGING_TABLE');
```

3. In this case, only a single SQL patch `SP_FIND_EMPLOYEE` is copied into a staging table owned by the current schema owner:

```
EXEC DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH(
    patch_name          =>  'SP_FIND_EMPLOYEE',
    staging_table_name  =>  'STAGING_TABLE');
```

   The staging table can then be moved to another system using either data pump, import/export commands or using a database link.

4. Call `DBMS_SQLDIAG.UNPACK_STGTAB_SQLPATCH` to create SQL patches on the new system from the patch data in the staging table. In this case, change the name in the data for the `SP_FIND_EMPLOYEE` patch stored in the staging table to '`SP_FIND_EMP_PROD`':

```
exec dbms_sqldiag.remap_stgtab_sqlpatch(
   old_patch_name      =>  'SP_FIND_EMPLOYEE',
   new_patch_name      =>  'SP_FIND_EMP_PROD',
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SQLDIAG` package subprograms

7.5.2 Repairing Data Corruptions with the Data Recovery Advisor

You use the Data Recovery Advisor to repair data block corruptions, undo corruptions, data dictionary corruptions, and more.

The Data Recovery Advisor integrates with the Enterprise Manager Support Workbench (Support Workbench), with the Health Monitor, and with the RMAN utility to display data corruption problems, assess the extent of each problem (critical, high priority, low priority), describe the impact of a problem, recommend repair options, conduct a feasibility check of the customer-chosen option, and automate the repair process.

The Cloud Control online help provides details on how to use the Data Recovery Advisor. This section describes how to access the advisor from the Support Workbench.

The Data Recovery Advisor is automatically recommended by and accessible from the Support Workbench when you are viewing health checker findings that are related to a data corruption or other data failure. The Data Recovery Advisor is also available from the Advisor Central page.

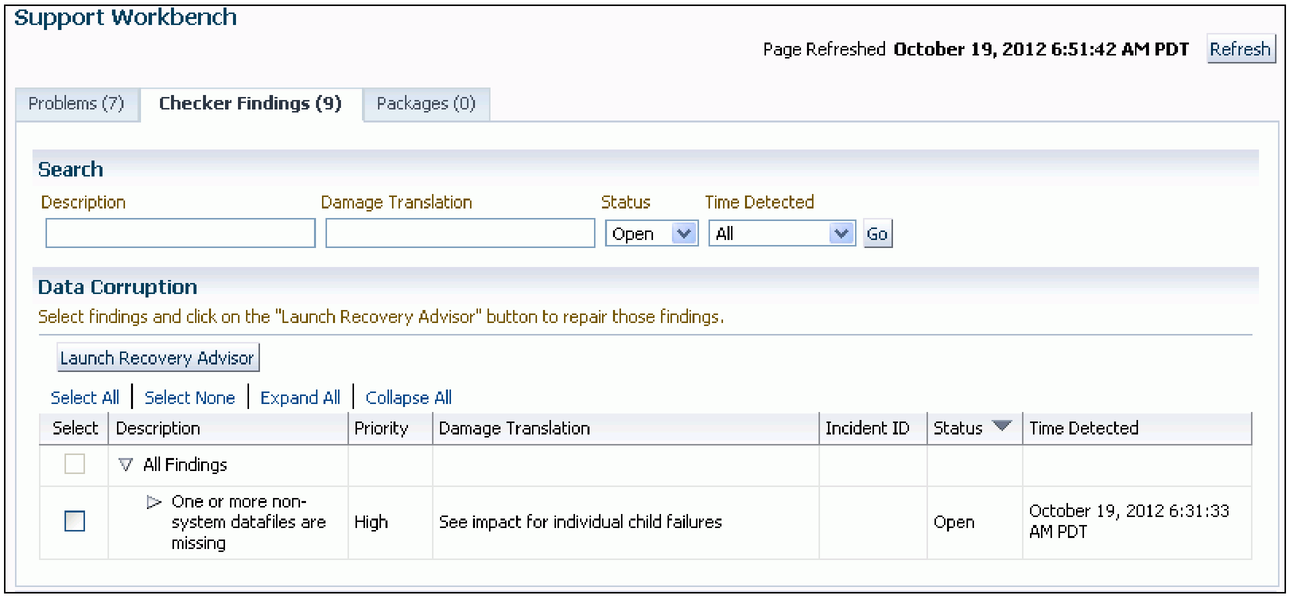
To access the Data Recovery Advisor in Cloud Control:

1. Access the Database Home page in Cloud Control.

   The Data Recovery Advisor is available only when you are connected as `SYSDBA`.

2. From the Oracle Database menu, select **Diagnostics**, then **Support Workbench**.
3. Click **Checker Findings**.

   The Checker Findings subpage appears.



Support Workbench

Page Refreshed **October 19, 2012 6:51:42 AM PDT** Refresh

Problems (7) | **Checker Findings (9)** | Packages (0)

Search

Description	Damage Translation	Status	Time Detected
		Open	All

Data Corruption

Select findings and click on the "Launch Recovery Advisor" button to repair those findings.

Launch Recovery Advisor

Select All | Select None | Expand All | Collapse All

Select	Description	Priority	Damage Translation	Incident ID	Status	Time Detected
☐	▽ All Findings					
☐	▷ One or more non-system datafiles are missing	High	See impact for individual child failures		Open	October 19, 2012 6:31:33 AM PDT

4. Select one or more data corruption findings and then click **Launch Recovery Advisor**.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information about the Data Recovery Advisor

7.5.3 Quarantine for Execution Plans for SQL Statements Consuming Excessive System Resources

Starting with Oracle Database 19c, you can use the SQL Quarantine infrastructure (SQL Quarantine) to quarantine execution plans for SQL statements that are terminated by the Resource Manager for consuming excessive system resources in an Oracle database. An individual SQL statement may have multiple execution plans, and if it attempts to use the execution plan that is quarantined, then that SQL statement is not allowed to run, thus preventing database performance degradation.

- About Quarantine for Execution Plans for SQL Statements
  You can use the SQL Quarantine infrastructure (SQL Quarantine) to quarantine execution plans for SQL statements that are terminated by the Resource Manager for consuming excessive system resources in an Oracle database. The quarantined execution plans for such SQL statements are not allowed to run again, thus preventing database performance degradation.
- Creating a Quarantine Configuration for an Execution Plan of a SQL Statement
  You can create a quarantine configuration for an execution plan of a SQL statement using any of these `DBMS_SQLQ` package functions – `CREATE_QUARANTINE_BY_SQL_ID` or `CREATE_QUARANTINE_BY_SQL_TEXT`.

- Specifying Quarantine Thresholds in a Quarantine Configuration
  After creating a quarantine configuration for an execution plan for a SQL statement, you can specify quarantine thresholds for it using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure. When any of the Resource Manager thresholds is equal to or less than a quarantine threshold specified in a SQL statement's quarantine configuration, then the SQL statement is not allowed to run, if it uses the execution plan specified in its quarantine configuration.
- Enabling and Disabling a Quarantine Configuration
  You can enable or disable a quarantine configuration using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure. A quarantine configuration is enabled by default when it is created.
- Viewing the Details of a Quarantine Configuration
  You can query the `DBA_SQL_QUARANTINE` view to get details of all the quarantine configurations.
- Deleting a Quarantine Configuration
  The unused quarantine configurations are automatically purged or deleted after 53 weeks. You can also delete a quarantine configuration using the `DBMS_SQLQ.DROP_QUARANTINE` procedure. You can disable automatic deletion of a quarantine configuration using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure.
- Viewing the Details of Quarantined Execution Plans of SQL Statements
  You can query the `V$SQL` and `GV$SQL` views to get details about the quarantined execution plans of SQL statements.
- Transferring Quarantine Configurations from One Database to Another Database
  You can transfer quarantine configurations from one database to another database using the `DBMS_SQLQ` package subprograms – `CREATE_STGTAB_QUARANTINE`, `PACK_STGTAB_QUARANTINE`, and `UNPACK_STGTAB_QUARANTINE`.
- Example: Quarantine for an Execution Plan of a SQL Statement Consuming Excessive System Resources
  This example shows how an execution plan of a SQL statement is quarantined when it exceeds a resource consumption limit configured using the Resource Manager.

7.5.3.1 About Quarantine for Execution Plans for SQL Statements

You can use the SQL Quarantine infrastructure (SQL Quarantine) to quarantine execution plans for SQL statements that are terminated by the Resource Manager for consuming excessive system resources in an Oracle database. The quarantined execution plans for such SQL statements are not allowed to run again, thus preventing database performance degradation.

Using the Resource Manager, you can configure limits for SQL statements for consuming system resources (*Resource Manager thresholds*). The Resource Manager terminates SQL statements that exceed the Resource Manager thresholds. In the earlier Oracle Database releases, if a SQL statement that is terminated by the Resource Manager runs again, the Resource Manager allows it to run again and terminates it again when it exceeds the Resource Manager thresholds. Thus, it is a waste of system resources to allow such SQL statements to run again.

Starting with Oracle Database 19c, you can use SQL Quarantine to automatically quarantine execution plans of SQL statements terminated by the Resource Manager, so that they are not allowed to run again. SQL Quarantine information is periodically persisted to the data dictionary. When resource manager terminates a SQL statement, it may be several minutes before the statement is quarantined.

Note:

Oracle Database Licensing Information User Manual for details on which features are supported for different editions and services

Additionally, SQL Quarantine can also be used to create *quarantine configurations* for execution plans of SQL statements by specifying thresholds for consuming various system resources (similar to the Resource Manager thresholds) using the `DBMS_SQLQ` package subprograms. These thresholds are known as *quarantine thresholds*. If any of the Resource Manager thresholds is equal to or less than a quarantine threshold specified in a SQL statement's quarantine configuration, then the SQL statement is not allowed to run, if it uses the execution plan specified in its quarantine configuration.

The following are the steps to manually set quarantine thresholds for an execution plan for a SQL statement using the `DBMS_SQLQ` package subprograms:

1. Create a quarantine configuration for an execution plan for a SQL statement
2. Specify quarantine thresholds in the quarantine configuration

You can also perform the following operations related to quarantine configurations using the `DBMS_SQLQ` package subprograms:

- Enable or disable a quarantine configuration
- Delete a quarantine configuration
- Transfer quarantine configurations from one database to another

Note:

- A quarantine configuration is specific to an execution plan for a SQL statement. If two different SQL statements use the same execution plan, they do not share the same quarantine configuration.
- An execution plan is quarantined specific to a SQL statement that is terminated by the Resource Manager. Thus, an execution plan that is quarantined for a SQL statement will not be quarantined for a different SQL statement that is not yet terminated by the Resource Manager.
- If there is no quarantine configuration created for an execution plan for a SQL statement, or if no quarantine thresholds are specified in its quarantine configuration, the execution plan for a SQL statement still gets automatically quarantined, if the Resource Manager terminates it for exceeding any of the Resource Manager thresholds.

For example, consider a resource plan of the Resource Manager that limits execution time for SQL statements to be 10 seconds (Resource Manager threshold). Consider a SQL statement Q1 for which this resource plan is application. When Q1 exceeds execution time of 10 seconds, it gets terminated by the Resource Manager. SQL Quarantine then creates a quarantine configuration for Q1 specific to that execution plan and stores this execution time of 10 seconds as a quarantine threshold in the quarantine configuration.

If Q1 is executed again with the same execution plan and the Resource Manager threshold is still 10 seconds, then SQL Quarantine does not allow Q1 to execute, because it refers to the quarantine threshold of 10 seconds to determine that Q1 will be eventually terminated by the Resource Manager as Q1 takes at least 10 seconds to execute.

If the Resource Manager threshold is changed to 5 seconds and Q1 is executed again with the same execution plan, then SQL Quarantine does not allow Q1 to execute, because it refers to the quarantine threshold of 10 seconds to determine that Q1 will be eventually terminated by the Resource Manager as Q1 takes at least 10 seconds to execute.

If the Resource Manager threshold is changed to 15 seconds and Q1 is executed again with the same execution plan, then SQL quarantine allows Q1 to execute, because it refers to the quarantine threshold of 10 seconds to determine that Q1 takes at least 10 seconds to execute, but there is a possibility that Q1 may complete its execution within 15 seconds.

Note:

A quarantine threshold is specific to an execution plan for a SQL statement, and it is automatically set by SQL Quarantine based on the Resource Manager threshold that is exceeded by the SQL statement and its execution plan. You can also manually set a quarantine threshold for a specific execution plan for a SQL statement by using the `DBMS_SQLQ` package subprograms.

See Also:

- "Creating a Quarantine Configuration for an Execution Plan of a SQL Statement"
- "Specifying Quarantine Thresholds in a Quarantine Configuration"
- "Enabling and Disabling a Quarantine Configuration"
- "Viewing the Details of a Quarantine Configuration"
- "Deleting a Quarantine Configuration"
- "Specifying Automatic Switching by Setting Resource Limits" for information about how to configure resource consumption limits for SQL statements using the Resource Manager

7.5.3.2 Creating a Quarantine Configuration for an Execution Plan of a SQL Statement

You can create a quarantine configuration for an execution plan of a SQL statement using any of these `DBMS_SQLQ` package functions – `CREATE_QUARANTINE_BY_SQL_ID` or `CREATE_QUARANTINE_BY_SQL_TEXT`.

The following example creates a quarantine configuration for an execution plan having the hash value of `3488063716` for a SQL statement having the SQL ID of `8vu7s907prbgr`:

```
DECLARE
    quarantine_config VARCHAR2(30);
BEGIN
    quarantine_config := DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_ID(
                             SQL_ID => '8vu7s907prbgr',
                            PLAN_HASH_VALUE => '3488063716');
END;
/
```

If you do not specify an execution plan or specify it as `NULL`, then the quarantine configuration is applied to all the execution plans of a SQL statement, except for those execution plans for which the execution plan-specific quarantine configurations are already created.

The following example creates a quarantine configuration for all the execution plans for a SQL statement having the SQL ID of `152sukb473gsk`:

```
DECLARE
    quarantine_config VARCHAR2(30);
BEGIN
    quarantine_config := DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_ID(
                            SQL_ID => '152sukb473gsk');
END;
/
```

The following example creates a quarantine configuration for all the execution plans for a SQL statement `'select count(*) from emp'`:

```
DECLARE
    quarantine_config VARCHAR2(30);
BEGIN
    quarantine_config := DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_TEXT(
                            SQL_TEXT => to_clob('select count(*) from emp'));
END;
/
```

The `CREATE_QUARANTINE_BY_SQL_ID` and `CREATE_QUARANTINE_BY_SQL_TEXT` functions return the name for the quarantine configuration, which can be used for specifying *quarantine thresholds* for an execution plan for a SQL statement using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure.

See Also:

- "Specifying Quarantine Thresholds in a Quarantine Configuration"
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SQLQ` package subprograms

7.5.3.3 Specifying Quarantine Thresholds in a Quarantine Configuration

After creating a quarantine configuration for an execution plan for a SQL statement, you can specify quarantine thresholds for it using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure. When any of the Resource Manager thresholds is equal to or less than a quarantine threshold specified in a SQL statement's quarantine configuration, then the SQL statement is not allowed to run, if it uses the execution plan specified in its quarantine configuration.

You can specify quarantine thresholds for the following resources in a quarantine configuration using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure:

- CPU time
- Elapsed time
- I/O in megabytes
- Number of physical I/O requests
- Number of logical I/O requests

In the following example, the quarantine threshold specified for CPU time is 5 seconds and elapsed time is 10 seconds for the quarantine configuration `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`.

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'CPU_TIME',
       PARAMETER_VALUE => '5');

    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'ELAPSED_TIME',
       PARAMETER_VALUE => '10');
END;
/
```

When the SQL statement is executed using the execution plan specified in this quarantine configuration, and if the Resource Manager threshold for CPU time is 5 seconds or less, or elapsed time is 10 seconds or less, then the SQL statement is not allowed to run.

Note:

If any of the Resource Manager thresholds is equal to or less than a quarantine threshold specified in a SQL statement's quarantine configuration, then that SQL statement is not allowed to run, if it uses the execution plan specified in its quarantine configuration.

Querying quarantine thresholds for a quarantine configuration

You can query a quarantine threshold for a quarantine configuration using the `DBMS_SQLQ.GET_PARAM_VALUE_QUARANTINE` function. The following example returns the quarantine threshold for CPU time consumption for the quarantine configuration `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
DECLARE
    quarantine_config_setting_value VARCHAR2(30);
BEGIN
    quarantine_config_setting_value :=
        DBMS_SQLQ.GET_PARAM_VALUE_QUARANTINE(
              QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
              PARAMETER_NAME  => 'CPU_TIME');
END;
/
```

Deleting quarantine thresholds from a quarantine configuration

You can delete a quarantine threshold from a quarantine configuration by specifying `DBMS_SQLQ.DROP_THRESHOLD` as the value for `PARAMETER_VALUE`. The following example deletes the quarantine threshold for CPU time consumption from the quarantine configuration `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'CPU_TIME',
       PARAMETER_VALUE => DBMS_SQLQ.DROP_THRESHOLD);
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SQLQ.ALTER_QUARANTINE` procedure

7.5.3.4 Enabling and Disabling a Quarantine Configuration

You can enable or disable a quarantine configuration using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure. A quarantine configuration is enabled by default when it is created.

The following example disables the quarantine configuration having the name `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'ENABLED',
       PARAMETER_VALUE => 'NO');
END;
/
```

The following example enables the quarantine configuration having the name `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'ENABLED',
       PARAMETER_VALUE => 'YES');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SQLQ.ALTER_QUARANTINE` procedure

7.5.3.5 Viewing the Details of a Quarantine Configuration

You can query the `DBA_SQL_QUARANTINE` view to get details of all the quarantine configurations.

The `DBA_SQL_QUARANTINE` view contains the following information about each quarantine configuration:

- Quarantine configuration name
- SQL statement for which the quarantine configuration is applicable
- Hash value of the execution plan for which the quarantine configuration is applicable
- Status of the quarantine configuration (enabled or disabled)
- Status of automatic purging of the quarantine configuration (yes or no)
- Quarantine thresholds specified for the quarantine configuration:
  - CPU time
  - Elapsed time
  - I/O in megabytes
  - Number of physical I/O requests
  - Number of logical I/O requests
- Date and time when the quarantine configuration was created
- Date and time when the quarantine configuration was last executed

See Also:

Oracle Database Reference for details of the `DBA_SQL_QUARANTINE` view

7.5.3.6 Deleting a Quarantine Configuration

The unused quarantine configurations are automatically purged or deleted after 53 weeks. You can also delete a quarantine configuration using the `DBMS_SQLQ.DROP_QUARANTINE` procedure. You can disable automatic deletion of a quarantine configuration using the `DBMS_SQLQ.ALTER_QUARANTINE` procedure.

The following example disables automatic deletion of the quarantine configuration `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'AUTOPURGE',
        PARAMETER_VALUE => 'NO');
END;
/
```

The following example enables automatic deletion of the quarantine configuration `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4',
       PARAMETER_NAME  => 'AUTOPURGE',
       PARAMETER_VALUE => 'YES');
END;
/
```

The following example deletes the quarantine configuration `SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4`:

```
BEGIN
    DBMS_SQLQ.DROP_QUARANTINE('SQL_QUARANTINE_3z0mwuq3aqsm8cfe7a0e4');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SQLQ` package subprograms

7.5.3.7 Viewing the Details of Quarantined Execution Plans of SQL Statements

You can query the `V$SQL` and `GV$SQL` views to get details about the quarantined execution plans of SQL statements.

The following columns of the `V$SQL` and `GV$SQL` views show the quarantine information of execution plans of SQL statements:

- `SQL_QUARANTINE`: This column shows the name of the quarantine configuration for an execution plan of a SQL statement.
- `AVOIDED_EXECUTIONS`: This column shows the number of times an execution plan of a SQL statement was prevented from running after it was quarantined.

See Also:

Oracle Database Reference for details of the `V$SQL` and `GV$SQL` views

7.5.3.8 Transferring Quarantine Configurations from One Database to Another Database

You can transfer quarantine configurations from one database to another database using the `DBMS_SQLQ` package subprograms – `CREATE_STGTAB_QUARANTINE`, `PACK_STGTAB_QUARANTINE`, and `UNPACK_STGTAB_QUARANTINE`.

For example, you may have tested the quarantine configurations on a *test* database and confirmed that they have performed well. You may then want to load these quarantine configurations into a *production* database.

The following example describes the steps to transfer quarantine configurations from one database *(source database)* to another database *(destination database)* using the `DBMS_SQLQ` package subprograms:

1. Using SQL*Plus, connect to the source database as a user with the administrative privileges, and create a staging table using the `DBMS_SQLQ.CREATE_STGTAB_QUARANTINE` procedure.

   The following example creates a staging table named `TBL_STG_QUARANTINE`:

   ```
   BEGIN
     DBMS_SQLQ.CREATE_STGTAB_QUARANTINE (
       staging_table_name => 'TBL_STG_QUARANTINE');
   END;
   /
   ```

2. Add the quarantine configurations into the staging table, which you want to transfer to the destination database.

   The following example adds all the quarantine configurations starting with the name `QUARANTINE_CONFIG_` into the staging table `TBL_STG_QUARANTINE`:

   ```
   DECLARE
     quarantine_configs NUMBER;
   BEGIN
     quarantine_configs := DBMS_SQLQ.PACK_STGTAB_QUARANTINE(
                               staging_table_name => 'TBL_STG_QUARANTINE',
                               name => 'QUARANTINE_CONFIG_%');
   END;
   /
   ```

   The `DBMS_SQLQ.PACK_STGTAB_QUARANTINE` function returns the number of quarantine configurations added to the staging table.

3. Export the staging table `TBL_STG_QUARANTINE` to a dump file using the Oracle Data Pump Export utility.
4. Transfer the dump file from the source database system to the destination database system.
5. On the destination database system, import the staging table `TBL_STG_QUARANTINE` from the dump file into the destination database using the Oracle Data Pump Import utility.
6. Using SQL*Plus, connect to the destination database as a user with the administrative privileges, and create the quarantine configurations from the imported staging table.

The following example creates the quarantine configurations on the destination database based on all the quarantine configurations stored in the imported staging table `TBL_STG_QUARANTINE`:

```
DECLARE
  quarantine_configs NUMBER;
BEGIN
  quarantine_configs := DBMS_SQLQ.UNPACK_STGTAB_QUARANTINE(
                          staging_table_name => 'TBL_STG_QUARANTINE');
END;
/
```

The `DBMS_SQLQ.UNPACK_STGTAB_QUARANTINE` function returns the number of quarantine configurations created in the destination database.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SQLQ` package subprograms

7.5.3.9 Example: Quarantine for an Execution Plan of a SQL Statement Consuming Excessive System Resources

This example shows how an execution plan of a SQL statement is quarantined when it exceeds a resource consumption limit configured using the Resource Manager.

1. Using the Resource Manager, specify the execution time limit of 3 seconds for SQL statements executed by the user `HR`.

   The following code performs these operations by creating a complex resource plan using the `DBMS_RESOURCE_MANAGER` package subprograms:

   - creates a consumer group `TEST_RUNAWAY_GROUP`.
   - assigns the user `HR` to the `TEST_RUNAWAY_GROUP` consumer group.
   - creates a resource plan `LIMIT_RESOURCE` that terminates SQL statements when they exceed the execution time of 3 seconds.
   - assigns the `LIMIT_RESOURCE` resource plan to the `TEST_RUNAWAY_GROUP` consumer group.

```
connect / as sysdba

begin

  -- Create a pending area
  dbms_resource_manager.create_pending_area();

  -- Create a consumer group 'TEST_RUNAWAY_GROUP'
  dbms_resource_manager.create_consumer_group (
    consumer_group => 'TEST_RUNAWAY_GROUP',
    comment        => 'This consumer group limits execution time for SQL statements'
  );

  -- Map the sessions of the user 'HR' to the consumer group 'TEST_RUNAWAY_GROUP'
  dbms_resource_manager.set_consumer_group_mapping(
    attribute      => DBMS_RESOURCE_MANAGER.ORACLE_USER,
    value          => 'HR',
    consumer_group => 'TEST_RUNAWAY_GROUP'
  );

  -- Create a resource plan 'LIMIT_RESOURCE'
  dbms_resource_manager.create_plan(
    plan    => 'LIMIT_RESOURCE',
    comment => 'Terminate SQL statements after exceeding total execution time'
  );

  -- Create a resource plan directive by assigning the 'LIMIT_RESOURCE' plan to
  -- the 'TEST_RUNAWAY_GROUP' consumer group
  -- Specify the execution time limit of 3 seconds for SQL statements belonging to
  -- the 'TEST_RUNAWAY_GROUP' group
  dbms_resource_manager.create_plan_directive(
    plan             => 'LIMIT_RESOURCE',
    group_or_subplan => 'TEST_RUNAWAY_GROUP',
    comment          => 'Terminate SQL statements when they exceed the' ||
                        'execution time of 3 seconds',
    switch_group     => 'CANCEL_SQL',
    switch_time      => 3,
    switch_estimate  => false
  );

  -- Allocate resources to the sessions not covered by the currently active plan
  -- according to the OTHER_GROUPS directive
  dbms_resource_Manager.create_plan_directive(
    plan              => 'LIMIT_RESOURCE',
    group_or_subplan  => 'OTHER_GROUPS',
    comment           => 'Ignore'
  );

  -- Validate and submit the pending area
  dbms_resource_manager.validate_pending_area();
  dbms_resource_manager.submit_pending_area();

  -- Grant switch privilege to the 'HR' user to switch to the 'TEST_RUNAWAY_GROUP'
  -- consumer group
  dbms_resource_manager_privs.grant_switch_consumer_group('HR',
                                                          'TEST_RUNAWAY_GROUP',
                                                          false);

  -- Set the initial consumer group of the 'HR' user to 'TEST_RUNAWAY_GROUP'
  dbms_resource_manager.set_initial_consumer_group('HR',
                                                   'TEST_RUNAWAY_GROUP');

end;
/

-- Set the 'LIMIT_RESOURCE' plan as the top plan for the Resource Manager
alter system set RESOURCE_MANAGER_PLAN = 'LIMIT_RESOURCE' scope = memory;

-- Unlock the HR user and assign it the DBA role
alter user hr identified by hr_user_password account unlock;
grant dba to hr;

-- Flush the shared pool
alter system flush shared_pool;
```

2. Connect to the Oracle database as the `HR` user and run the SQL statement that exceeds the execution time limit of 3 seconds:

```
select count(*)
from employees emp1, employees emp2,
     employees emp3, employees emp4,
     employees emp5, employees emp6,
     employees emp7, employees emp8,
     employees emp9, employees emp10
where rownum <= 100000000;
```

The SQL statement is terminated by the Resource Manager as it exceeds the execution time limit of 3 seconds and the following error message is displayed:

```
ORA-00040: active time limit exceeded - call aborted
```

The execution plan for the SQL statement is now added to the quarantine list, so that it is not allowed to run again.

3. Run the SQL statement again.

   Now the SQL statement should terminate immediately with the following error message, because its execution plan is quarantined:

   ```
   ORA-56955: quarantined plan used
   ```

4. View the details of the quarantined execution plan of the SQL statement by querying the `v$sql` and `dba_sql_quarantine` views.

   - Query the `v$sql` view. The `v$sql` view contains information about various statistics of the SQL statements including the quarantine statistics.

     ```
     select sql_text, plan_hash_value, avoided_executions, sql_quarantine
     from v$sql
     where sql_quarantine is not null;
     ```

     The output of this query is similar to the following:

```
SQL_TEXT                             PLAN_HASH_VALUE  AVOIDED_EXECUTIONS  SQL_QUARANTINE
------------------------------------ ---------------  ------------------  ------------------------------------
select count(*)                      3719017987       1                   SQL_QUARANTINE_3uuhv1u5day0yf6ed7f0c
from employees emp1, employees emp2,
     employees emp3, employees emp4,
     employees emp5, employees emp6,
     employees emp7, employees emp8,
     employees emp9, employees emp10
where rownum <= 100000000;
```

     The `sql_quarantine` column shows the auto-generated name for the quarantine configuration for the execution plan of the SQL statement.

   - Query the `dba_sql_quarantine` view. The `dba_sql_quarantine` view contains information about the quarantine configurations of execution plans of the SQL statements.

     ```
     select sql_text, name, plan_hash_value, last_executed, enabled
     from dba_sql_quarantine;
     ```

     The output of this query is similar to the following:

```
SQL_TEXT                             NAME                                 PLAN_HASH_VALUE
LAST_EXECUTED                  ENABLED
------------------------------------ ------------------------------------ ---------------
----------------------------   -------
select count(*)                      SQL_QUARANTINE_3uuhv1u5day0yf6ed7f0c 3719017987      14-JAN-19 02.19.01.000000
AM   YES
from employees emp1, employees emp2,
     employees emp3, employees emp4,
     employees emp5, employees emp6,
     employees emp7, employees emp8,
```

```
      employees emp9, employees emp10
where rownum <= 100000000;
```

The `name` column shows the auto-generated name for the quarantine configuration for the execution plan of the SQL statement.

5. Clean up the example environment.

   The following code deletes all the database objects created for this example:

```
connect / as sysdba

begin
  for quarantineObj in (select name from dba_sql_quarantine) loop
    sys.dbms_sqlq.drop_quarantine(quarantineObj.name);
  end loop;
end;
/

alter system set RESOURCE_MANAGER_PLAN = '' scope = memory;

execute dbms_resource_manager.clear_pending_area();
execute dbms_resource_manager.create_pending_area();
execute dbms_resource_manager.delete_plan('LIMIT_RESOURCE');
execute dbms_resource_manager.delete_consumer_group('TEST_RUNAWAY_GROUP');
execute dbms_resource_manager.validate_pending_area();
execute dbms_resource_manager.submit_pending_area();
```

See Also:

- "Creating a Complex Resource Plan" for more information about creating a complex resource plan using the `DBMS_RESOURCE_MANAGER` package subprograms
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_RESOURCE_MANAGER` package subprograms
- *Oracle Database Reference* for more information about the `V$SQL` view

7.5.4 Viewing Attention Log Information

Access information stored in the attention log either by opening the file with any text editor or by querying the `V$DIAG_ALERT_EXT` view.

To view the attention log by using a text editor:

1. Navigate to the `*$ORACLE_HOME*/diag/rdbms/*database_name*/*instance_id*/trace` directory.
2. Open the `attention.log` file.

To view attention log information stored in the data dictionary:

1. Connect to the database with SQL*Plus or a query tool such as SQL Developer.
2. Query the `V$DIAG_ALERT_EXT` view using the required filters.

   For example, the following query displays attention messages for which urgent action must be taken. A `message_level` of 1 corresponds to critical errors that need immediate action.

```
SELECT container_name, message_id, message_type, message_text, version,
host_id, component_id
```

```
FROM V$DIAG_ALERT_EXT
WHERE message_level = 1;
```

Part II
Oracle Database Structure and Storage

You can create and manage database structures and storage components.

- Managing Control Files
  You can create, back up, and drop control files.
- Managing the Redo Log
  You manage the redo log by completing tasks such as creating redo log groups and members, relocating and renaming redo log members, dropping redo log groups and members, and forcing log switches.
- Managing Archived Redo Log Files
  You manage the archived redo log files by completing tasks such as choosing between `NOARCHIVELOG` or `ARCHIVELOG` mode and specifying archive destinations.
- Managing Tablespaces
  A tablespace is a database storage unit that groups related logical structures together. The database data files are stored in tablespaces.
- Managing Data Files and Temp Files
  You can manage data files and temp files by performing tasks such as creating them, altering them, and dropping them.
- Transporting Data
  Transporting data moves the data from one database to another.
- Managing Undo
  For a default installation, Oracle Database automatically manages undo. There is typically no need for DBA intervention. However, if your installation uses Oracle Flashback operations, you may need to perform some undo management tasks to ensure the success of these operations.
- Using Oracle Managed Files
  Oracle Database can manage the files that comprise the database.
- Using Persistent Memory Database
  Mapping the database directly into persistent memory (PMEM) provides significant performance enhancements.

8

Managing Control Files

You can create, back up, and drop control files.

- What Is a Control File?
  Every Oracle Database has a **control file**, which is a small binary file that records the physical structure of the database.
- Guidelines for Control Files
  You can follow guidelines to manage the control files for a database.
- Creating Control Files
  You can create, copy, rename, and relocate control files.
- Troubleshooting After Creating Control Files
  After issuing the `CREATE CONTROLFILE` statement, you may encounter some errors.
- Backing Up Control Files
  Use the `ALTER DATABASE BACKUP CONTROLFILE` statement to back up your control files.
- Recovering a Control File Using a Current Copy
  You can recover your control file from a current backup or from a multiplexed copy.
- Dropping Control Files
  You can drop control files, but the database should have at least two control files at all times.
- Control Files Data Dictionary Views
  You can query a set of data dictionary views for information about control files.

See Also:

- *Oracle Database Concepts* for an overview of control files
- Using Oracle Managed Files for information about creating control files that are both created and managed by the Oracle Database server

8.1 What Is a Control File?

Every Oracle Database has a **control file**, which is a small binary file that records the physical structure of the database.

The control file includes:

- The database name
- Names and locations of associated data files and redo log files
- The timestamp of the database creation
- The current log sequence number
- Checkpoint information

The control file must be available for writing by the Oracle Database server whenever the database is open. Without the control file, the database cannot be mounted and recovery is difficult.

The control file of an Oracle Database is created at the same time as the database. By default, at least one copy of the control file is created during database creation. On some operating systems the default is to create multiple copies. You should create two or more copies of the control file during database creation. You can also create control files later, if you lose control files or want to change particular settings in the control files.

8.2 Guidelines for Control Files

You can follow guidelines to manage the control files for a database.

- Provide File Names for the Control Files
  You specify control file names using the CONTROL_FILES initialization parameter in the database initialization parameter file. The instance recognizes and opens all the listed file during startup, and the instance writes to and maintains all listed control files during database operation.
- Multiplex Control Files on Different Disks
  Every Oracle Database should have at least two control files, each stored on a different physical disk.
- Back Up Control Files
  It is very important that you back up your control files. This is true initially, and every time you change the physical structure of your database.
- Manage the Size of Control Files
  The main determinants of the size of a control file are the values set for the MAXDATAFILES, MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES parameters in the CREATE DATABASE statement that created the associated database.

8.2.1 Provide File Names for the Control Files

You specify control file names using the CONTROL_FILES initialization parameter in the database initialization parameter file. The instance recognizes and opens all the listed file during startup, and the instance writes to and maintains all listed control files during database operation.

If you do not specify files for CONTROL_FILES before database creation:

- If you are not using Oracle Managed Files, then the database creates a control file and uses a default file name. The default name is operating system specific.
- If you are using Oracle Managed Files, then the initialization parameters you set to enable that feature determine the name and location of the control files.
- If you are using Oracle Automatic Storage Management (Oracle ASM), you can place incomplete Oracle ASM file names in the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters. Oracle ASM then automatically creates control files in the appropriate places.

Related Topics

- Creating Initial Control Files
  The initial control files of an Oracle Database are created when you issue the CREATE DATABASE statement.

- Using Oracle Managed Files
  Oracle Database can manage the files that comprise the database.
- *Oracle Automatic Storage Management Administrator's Guide*

8.2.2 Multiplex Control Files on Different Disks

Every Oracle Database should have at least two control files, each stored on a different physical disk.

If a control file is damaged due to a disk failure, the associated instance must be shut down. Once the disk drive is repaired, the damaged control file can be restored using the intact copy of the control file from the other disk and the instance can be restarted. In this case, no media recovery is required.

The behavior of multiplexed control files is this:

- The database writes to all file names listed for the initialization parameter `CONTROL_FILES` in the database initialization parameter file.
- The database reads only the first file listed in the `CONTROL_FILES` parameter during database operation.
- If any of the control files become unavailable during database operation, the instance becomes inoperable and should be terminated.

> **Note:**
>
> Oracle strongly recommends that your database has a minimum of two control files and that they are located on separate physical disks.

One way to multiplex control files is to store a control file copy on every disk drive that stores members of redo log groups, if the redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the redo log will be lost in a single disk failure.

8.2.3 Back Up Control Files

It is very important that you back up your control files. This is true initially, and every time you change the physical structure of your database.

Such structural changes include:

- Adding, dropping, or renaming data files
- Adding or dropping a tablespace, or altering the read/write state of the tablespace
- Adding or dropping redo log files or groups

The methods for backing up control files are discussed in "Backing Up Control Files".

8.2.4 Manage the Size of Control Files

The main determinants of the size of a control file are the values set for the `MAXDATAFILES`, `MAXLOGFILES`, `MAXLOGMEMBERS`, `MAXLOGHISTORY`, and `MAXINSTANCES` parameters in the `CREATE DATABASE` statement that created the associated database.

Increasing the values of these parameters increases the size of a control file of the associated database.

See Also:

- Your operating system specific Oracle documentation contains more information about the maximum control file size.
- *Oracle Database SQL Language Reference* for a description of the `CREATE DATABASE` statement

8.3 Creating Control Files

You can create, copy, rename, and relocate control files.

- Creating Initial Control Files
  The initial control files of an Oracle Database are created when you issue the `CREATE DATABASE` statement.
- Creating Additional Copies, Renaming, and Relocating Control Files
  You can create an additional control file copy for multiplexing by copying an existing control file to a new location and adding the file name to the list of control files.
- Creating New Control Files
  You can create new control files when all of the control files for the database have been permanently damaged and you do not have a control file backup or when you want to change the database name.

8.3.1 Creating Initial Control Files

The initial control files of an Oracle Database are created when you issue the `CREATE DATABASE` statement.

The names of the control files are specified by the `CONTROL_FILES` parameter in the initialization parameter file used during database creation. The file names specified in `CONTROL_FILES` should be fully specified and are operating system specific. The following is an example of a `CONTROL_FILES` initialization parameter:

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                 /u02/oracle/prod/control02.ctl,
                 /u03/oracle/prod/control03.ctl)
```

If files with the specified names currently exist at the time of database creation, you must specify the `CONTROLFILE REUSE` clause in the `CREATE DATABASE` statement, or else an error occurs. Also, if the size of the old control file differs from the `SIZE` parameter of the new one, you cannot use the `REUSE` clause.

The size of the control file changes between some releases of Oracle Database, as well as when the number of files specified in the control file changes. Configuration parameters such as `MAXLOGFILES`, `MAXLOGMEMBERS`, `MAXLOGHISTORY`, `MAXDATAFILES`, and `MAXINSTANCES` affect control file size.

You can subsequently change the value of the `CONTROL_FILES` initialization parameter to add more control files or to change the names or locations of existing control files.

See Also:

Your operating system specific Oracle documentation contains more information about specifying control files.

8.3.2 Creating Additional Copies, Renaming, and Relocating Control Files

You can create an additional control file copy for multiplexing by copying an existing control file to a new location and adding the file name to the list of control files.

Similarly, you rename an existing control file by copying the file to its new name or location, and changing the file name in the control file list. In both cases, to guarantee that control files do not change during the procedure, shut down the database before copying the control file.

To add a multiplexed copy of the current control file or to rename a control file:

1. Shut down the database.
2. Copy an existing control file to a new location, using operating system commands.
3. Edit the `CONTROL_FILES` parameter in the database initialization parameter file to add the new control file name, or to change the existing control file name.
4. Restart the database.

8.3.3 Creating New Control Files

You can create new control files when all of the control files for the database have been permanently damaged and you do not have a control file backup or when you want to change the database name.

- When to Create New Control Files
  You must create new control files in certain situations.
- The CREATE CONTROLFILE Statement
  You can create a new control file for a database using the `CREATE CONTROLFILE` statement.
- Creating New Control Files
  You can create new control files for your database.

8.3.3.1 When to Create New Control Files

You must create new control files in certain situations.

It is necessary for you to create new control files in the following situations:

- All control files for the database have been permanently damaged and you do not have a control file backup.
- You want to change the database name.

  For example, you would change a database name if it conflicted with another database name in a distributed environment.

Note:

You can change the database name and DBID (internal database identifier) using the DBNEWID utility. See *Oracle Database Utilities* for information about using this utility.

8.3.3.2 The CREATE CONTROLFILE Statement

You can create a new control file for a database using the `CREATE CONTROLFILE` statement.

The following statement creates a new control file for the `prod` database (a database that formerly used a different database name):

```
CREATE CONTROLFILE
   SET DATABASE prod
   LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',
                    '/u01/oracle/prod/redo01_02.log'),
           GROUP 2 ('/u01/oracle/prod/redo02_01.log',
                    '/u01/oracle/prod/redo02_02.log'),
           GROUP 3 ('/u01/oracle/prod/redo03_01.log',
                    '/u01/oracle/prod/redo03_02.log')
   RESETLOGS
   DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,
            '/u01/oracle/prod/rbs01.dbs' SIZE 5M,
            '/u01/oracle/prod/users01.dbs' SIZE 5M,
            '/u01/oracle/prod/temp01.dbs' SIZE 5M
   MAXLOGFILES 50
   MAXLOGMEMBERS 3
   MAXLOGHISTORY 400
   MAXDATAFILES 200
   MAXINSTANCES 6
   ARCHIVELOG;
```

Note:

- The `CREATE CONTROLFILE` statement can potentially damage specified data files and redo log files. Omitting a file name can cause loss of the data in that file, or loss of access to the entire database. Use caution when issuing this statement and be sure to follow the instructions in "Creating New Control Files".
- If the database had forced logging enabled before creating the new control file, and you want it to continue to be enabled, then you must specify the `FORCE LOGGING` clause in the `CREATE CONTROLFILE` statement. See "*Oracle Database SQL Language Reference*".

See Also:

Oracle Database SQL Language Reference describes the complete syntax of the `CREATE CONTROLFILE` statement

8.3.3.3 Creating New Control Files

You can create new control files for your database.

Complete the following steps to create a new control file.

1. Make a list of all data files and redo log files of the database.

   If you follow recommendations for control file backups as discussed in "Backing Up Control Files" , you will already have a list of data files and redo log files that reflect the current structure of the database. However, if you have no such list, executing the following statements will produce one.

   ```
   SELECT MEMBER FROM V$LOGFILE;
   SELECT NAME FROM V$DATAFILE;
   SELECT VALUE FROM V$PARAMETER WHERE NAME = 'control_files';
   ```

   If you have no such lists and your control file has been damaged so that the database cannot be opened, try to locate all of the data files and redo log files that constitute the database. Any files not specified in step 5 are not recoverable once a new control file has been created. Moreover, if you omit any of the files that comprise the `SYSTEM` tablespace, you might not be able to recover the database.

2. Shut down the database.

   If the database is open, shut down the database normally if possible. Use the `IMMEDIATE` or `ABORT` clauses only as a last resort.

3. Back up all data files and redo log files of the database.

4. Start up a new instance, but do not mount or open the database:

   ```
   STARTUP NOMOUNT
   ```

5. Create a new control file for the database using the `CREATE CONTROLFILE` statement.

   When creating a new control file, specify the `RESETLOGS` clause if you have lost any redo log groups in addition to control files. In this case, you will need to recover from the loss of the redo logs (step 8). You must specify the `RESETLOGS` clause if you have renamed the database. Otherwise, select the `NORESETLOGS` clause.

6. Store a backup of the new control file on an offline storage device. See "Backing Up Control Files" for instructions for creating a backup.

7. Edit the `CONTROL_FILES` initialization parameter for the database to indicate all of the control files now part of your database as created in step 5 (not including the backup control file). If you are renaming the database, edit the `DB_NAME` parameter in your instance parameter file to specify the new name.

8. Recover the database if necessary. If you are not recovering the database, skip to step 9.

   If you are creating the control file as part of recovery, recover the database. If the new control file was created using the `NORESETLOGS` clause, you can recover the database with complete, closed database recovery.

   If the new control file was created using the `RESETLOGS` clause, you must specify `USING BACKUP CONTROL FILE`. If you have lost online redo logs, archived redo log files, or data files, use the procedures for recovering those files.

See Also:

Oracle Database Backup and Recovery User's Guide for information about recovering your database and methods of recovering a lost control file

9. Open the database using one of the following methods:
   - If you did not perform recovery, or you performed complete, closed database recovery in step 8, open the database normally.

     ```
     ALTER DATABASE OPEN;
     ```

   - If you specified `RESETLOGS` when creating the control file, use the `ALTER DATABASE` statement, indicating `RESETLOGS`.

     ```
     ALTER DATABASE OPEN RESETLOGS;
     ```

The database is now open and available for use.

8.4 Troubleshooting After Creating Control Files

After issuing the `CREATE CONTROLFILE` statement, you may encounter some errors.

- Checking for Missing or Extra Files
  After creating a new control file and using it to open the database, check the alert log to see if the database has detected inconsistencies between the data dictionary and the control file, such as a data file in the data dictionary includes that the control file does not list.
- Handling Errors During CREATE CONTROLFILE
  If Oracle Database sends you an error when you attempt to mount and open the database after creating a new control file, the most likely cause is that you omitted a file from the `CREATE CONTROLFILE` statement or included one that should not have been listed.

8.4.1 Checking for Missing or Extra Files

After creating a new control file and using it to open the database, check the alert log to see if the database has detected inconsistencies between the data dictionary and the control file, such as a data file in the data dictionary includes that the control file does not list.

If a data file exists in the data dictionary but not in the new control file, the database creates a placeholder entry in the control file under the name `MISSING`*`nnnn`*, where *`nnnn`* is the file number in decimal. `MISSING`*`nnnn`* is flagged in the control file as being offline and requiring media recovery.

If the actual data file corresponding to `MISSING`*`nnnn`* is read-only or offline normal, then you can make the data file accessible by renaming `MISSING`*`nnnn`* to the name of the actual data file. If `MISSING`*`nnnn`* corresponds to a data file that was not read-only or offline normal, then you cannot use the rename operation to make the data file accessible, because the data file requires media recovery that is precluded by the results of `RESETLOGS`. In this case, you must drop the tablespace containing the data file.

Conversely, if a data file listed in the control file is not present in the data dictionary, then the database removes references to it from the new control file. In both cases, the database includes an explanatory message in the alert log to let you know what was found.

8.4.2 Handling Errors During CREATE CONTROLFILE

If Oracle Database sends you an error when you attempt to mount and open the database after creating a new control file, the most likely cause is that you omitted a file from the `CREATE CONTROLFILE` statement or included one that should not have been listed.

Typically, the error is `ORA-01173`, `ORA-01176`, `ORA-01177`, `ORA-01215`, or `ORA-01216`. In this case, you should restore the files you backed up in "Creating New Control Files" and repeat the procedure in that task, using the correct file names.

8.5 Backing Up Control Files

Use the `ALTER DATABASE BACKUP CONTROLFILE` statement to back up your control files.

You have two options:

- Back up the control file to a binary file (duplicate of existing control file) using the following statement:

  ```
  ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/control.bkp';
  ```

- Produce SQL statements that can later be used to re-create your control file:

  ```
  ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
  ```

  This command writes a SQL script to a trace file where it can be captured and edited to reproduce the control file. View the alert log to determine the name and location of the trace file.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information on backing up your control files
- "Viewing the Alert Log"

8.6 Recovering a Control File Using a Current Copy

You can recover your control file from a current backup or from a multiplexed copy.

- Recovering from Control File Corruption Using a Control File Copy
  If a control file becomes corrupted, then you can recover it using a control file copy.
- Recovering from Permanent Media Failure Using a Control File Copy
  If there is permanent media failure, then you can recover by using a control file copy.

8.6.1 Recovering from Control File Corruption Using a Control File Copy

If a control file becomes corrupted, then you can recover it using a control file copy.

This method assumes that one of the control files specified in the `CONTROL_FILES` parameter is corrupted, that the control file directory is still accessible, and that you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to overwrite the bad control file with a good copy:

   ```
   % cp /u03/oracle/prod/control03.ctl  /u02/oracle/prod/control02.ctl
   ```

2. Start SQL*Plus and open the database:

   ```
   SQL> STARTUP
   ```

8.6.2 Recovering from Permanent Media Failure Using a Control File Copy

If there is permanent media failure, then you can recover by using a control file copy.

This method assumes that one of the control files specified in the `CONTROL_FILES` parameter is inaccessible due to a permanent media failure and that you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to copy the current copy of the control file to a new, accessible location:

   ```
   % cp /u01/oracle/prod/control01.ctl  /u04/oracle/prod/control03.ctl
   ```

2. Edit the `CONTROL_FILES` parameter in the initialization parameter file to replace the bad location with the new location:

   ```
   CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                    /u02/oracle/prod/control02.ctl,
                    /u04/oracle/prod/control03.ctl)
   ```

3. Start SQL*Plus and open the database:

   ```
   SQL> STARTUP
   ```

If you have multiplexed control files, you can get the database started up quickly by editing the `CONTROL_FILES` initialization parameter. Remove the bad control file from `CONTROL_FILES` setting and you can restart the database immediately. Then you can perform the reconstruction of the bad control file and at some later time shut down and restart the database after editing the `CONTROL_FILES` initialization parameter to include the recovered control file.

8.7 Dropping Control Files

You can drop control files, but the database should have at least two control files at all times.

You want to drop control files from the database, for example, if the location of a control file is no longer appropriate.

1. Shut down the database.
2. Edit the `CONTROL_FILES` parameter in the database initialization parameter file to delete the old control file name.
3. Restart the database.

> **Note:**
>
> This operation does not physically delete the unwanted control file from the disk. Use operating system commands to delete the unnecessary file after you have dropped the control file from the database.

8.8 Control Files Data Dictionary Views

You can query a set of data dictionary views for information about control files.

The following views display information about control files:

View	Description
`V$DATABASE`	Displays database information from the control file
`V$CONTROLFILE`	Lists the names of control files
`V$CONTROLFILE_RECORD_SECTION`	Displays information about control file record sections
`V$PARAMETER`	Displays the names of control files as specified in the `CONTROL_FILES` initialization parameter

This example lists the names of the control files.

```
SQL> SELECT NAME FROM V$CONTROLFILE;

NAME
-------------------------------------
/u01/oracle/prod/control01.ctl
/u02/oracle/prod/control02.ctl
/u03/oracle/prod/control03.ctl
```

9

Managing the Redo Log

You manage the redo log by completing tasks such as creating redo log groups and members, relocating and renaming redo log members, dropping redo log groups and members, and forcing log switches.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- What Is the Redo Log?
  The most crucial structure for recovery operations is the **redo log**, which consists of two or more preallocated files that store all changes made to the database as they occur. Every instance of an Oracle Database has an associated redo log to protect the database in case of an instance failure.
- Planning the Redo Log
  You can follow guidelines when configuring a database instance redo log.
- Creating Redo Log Groups and Members
  Plan the redo log for a database and create all required groups and members of redo log files during database creation. However, there are situations where you might want to create additional groups or members. For example, adding groups to a redo log can correct redo log group availability problems.
- Relocating and Renaming Redo Log Members
  You can use operating system commands to relocate redo logs, then use the `ALTER DATABASE` statement to make their new names (locations) known to the database.
- Dropping Redo Log Groups and Members
  In some cases, you may want to drop an entire group of redo log members.
- Forcing Log Switches
  A log switch occurs when LGWR stops writing to one redo log group and starts writing to another. By default, a log switch occurs automatically when the current redo log file group fills.
- Verifying Blocks in Redo Log Files
  You can configure the database to use checksums to verify blocks in the redo log files.
- Clearing a Redo Log File
  A redo log file might become corrupted while the database is open, and ultimately stop database activity because archiving cannot continue.
- Precedence of FORCE LOGGING Settings
  You can set `FORCE LOGGING` and `NOLOGGING` at various levels, such as for a database, pluggable database (PDB), tablespace, or database object. When `FORCE LOGGING` is set at

one or more levels, the precedence of `FORCE LOGGING` settings determines what is logged in the redo log.

- Redo Log Data Dictionary Views
  You can query a set of data dictionary views for information about the redo log.

> **See Also:**
>
> Using Oracle Managed Files for information about redo log files that are both created and managed by the Oracle Database server

9.1 What Is the Redo Log?

The most crucial structure for recovery operations is the **redo log**, which consists of two or more preallocated files that store all changes made to the database as they occur. Every instance of an Oracle Database has an associated redo log to protect the database in case of an instance failure.

- Redo Threads
  When speaking in the context of multiple database instances, the redo log for each database instance is also referred to as a *redo thread*.
- Redo Log Contents
  Redo log files are filled with **redo records**.
- How Oracle Database Writes to the Redo Log
  The redo log for a database consists of two or more redo log files. The database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in `ARCHIVELOG` mode).

9.1.1 Redo Threads

When speaking in the context of multiple database instances, the redo log for each database instance is also referred to as a *redo thread*.

In typical configurations, only one database instance accesses an Oracle Database, so only one thread is present. In an Oracle Real Application Clusters environment, however, two or more instances concurrently access a single database and each instance has its own thread of redo. A separate redo thread for each instance avoids contention for a single set of redo log files, thereby eliminating a potential performance bottleneck.

This chapter describes how to configure and manage the redo log on a standard single-instance Oracle Database. The thread number can be assumed to be 1 in all discussions and examples of statements. For information about redo log groups in an Oracle Real Application Clusters environment, see *Oracle Real Application Clusters Administration and Deployment Guide*.

9.1.2 Redo Log Contents

Redo log files are filled with **redo records**.

A redo record, also called a **redo entry**, is made up of a group of **change vectors**, each of which is a description of a change made to a single block in the database. For example, if you change a salary value in an employee table, you generate a redo record containing change vectors that describe changes to the data segment block for the table, the undo segment data block, and the transaction table of the undo segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. Therefore, the redo log also protects rollback data. When you recover the database using redo data, the database reads the change vectors in the redo records and applies the changes to the relevant blocks.

Redo records are buffered in a circular fashion in the redo log buffer of the SGA (see "How Oracle Database Writes to the Redo Log") and are written to one of the redo log files by the Log Writer (LGWR) database background process. Whenever a transaction is committed, LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file, and assigns a **system change number** (SCN) to identify the redo records for each committed transaction. Only when all redo records associated with a given transaction are safely on disk in the online logs is the user process notified that the transaction has been committed.

Redo records can also be written to a redo log file before the corresponding transaction is committed. If the redo log buffer fills, or another transaction commits, LGWR flushes all of the redo log entries in the redo log buffer to a redo log file, even though some redo records may not be committed. If necessary, the database can roll back these changes.

9.1.3 How Oracle Database Writes to the Redo Log

The redo log for a database consists of two or more redo log files. The database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in `ARCHIVELOG` mode).

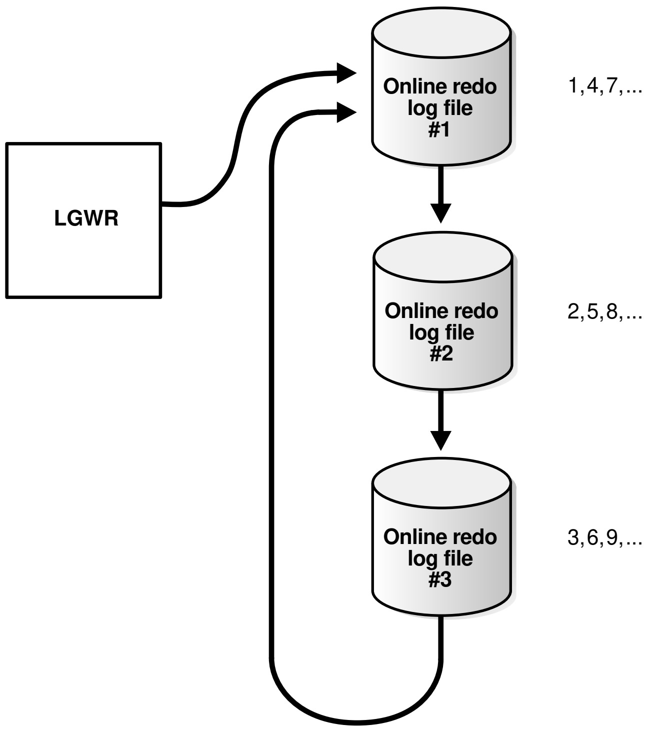
See " Managing Archived Redo Log Files" for more information.

LGWR writes to redo log files in a circular fashion. When the current redo log file fills, LGWR begins writing to the next available redo log file. When the last available redo log file is filled, LGWR returns to the first redo log file and writes to it, starting the cycle again. Figure 9-1 illustrates the circular writing of the redo log file. The numbers next to each line indicate the sequence in which LGWR writes to each redo log file.

Filled redo log files are available to LGWR for reuse depending on whether archiving is enabled.

- If archiving is disabled (the database is in `NOARCHIVELOG` mode), a filled redo log file is available after the changes recorded in it have been written to the data files.
- If archiving is enabled (the database is in `ARCHIVELOG` mode), a filled redo log file is available to LGWR after the changes recorded in it have been written to the data files *and* the file has been archived.

Figure 9-1 Reuse of Redo Log Files by LGWR



- Active (Current) and Inactive Redo Log Files
  Oracle Database uses only one redo log file at a time to store redo records written from the redo log buffer. The redo log file that LGWR is actively writing to is called the **current** redo log file.
- Log Switches and Log Sequence Numbers
  A **log switch** is the point at which the database stops writing to one redo log file and begins writing to another. Normally, a log switch occurs when the current redo log file is completely filled and writing must continue to the next redo log file.

9.1.3.1 Active (Current) and Inactive Redo Log Files

Oracle Database uses only one redo log file at a time to store redo records written from the redo log buffer. The redo log file that LGWR is actively writing to is called the **current** redo log file.

Redo log files that are required for instance recovery are called **active** redo log files. Redo log files that are no longer required for instance recovery are called **inactive** redo log files.

If you have enabled archiving (the database is in `ARCHIVELOG` mode), then the database cannot reuse or overwrite an active online log file until one of the archiver background processes (ARC*n*) has archived its contents. If archiving is disabled (the database is in `NOARCHIVELOG` mode), then when the last redo log file is full, LGWR continues by overwriting the next log file in the sequence when it becomes inactive.

9.1.3.2 Log Switches and Log Sequence Numbers

A **log switch** is the point at which the database stops writing to one redo log file and begins writing to another. Normally, a log switch occurs when the current redo log file is completely filled and writing must continue to the next redo log file.

However, you can configure log switches to occur at regular intervals, regardless of whether the current redo log file is completely filled. You can also force log switches manually.

Oracle Database assigns each redo log file a new **log sequence number** every time a log switch occurs and LGWR begins writing to it. When the database archives redo log files, the archived log retains its log sequence number. A redo log file that is cycled back for use is given the next available log sequence number.

Each online or archived redo log file is uniquely identified by its log sequence number. During crash, instance, or media recovery, the database properly applies redo log files in ascending order by using the log sequence number of the necessary archived and redo log files.

9.2 Planning the Redo Log

You can follow guidelines when configuring a database instance redo log.

Starting with Oracle Database Release 21c, on non-Exadata Linux systems, redo logs can be stored in a persistent memory (PMEM) DAX file system. The tasks and commands for managing redo logs in PMEM are the same as those for managing redo logs on disk.

Oracle Database requires the persistent memory to be configured with App Direct Mode, preferably with interleaved configuration. A DAX-enabled file system such as XFS or EXT4 must be configured on the namespaces used by the database. On compute nodes, the database only supports FSDAX configuration and does not support Device DAX (devdax) configuration.

- Multiplexing Redo Log Files
  To protect against a failure involving the redo log itself, Oracle Database allows a **multiplexed** redo log, meaning that two or more identical copies of the redo log can be automatically maintained in separate locations.
- Placing Redo Log Members on Different Disks
  When setting up a multiplexed redo log, place members of a group on different physical disks. If a single disk fails, then only one member of a group becomes unavailable to LGWR and other members remain accessible to LGWR, so the instance can continue to function.
- Planning the Size of Redo Log Files
  When setting the size of redo log files, consider whether you will be archiving the redo log. Redo log files should be sized so that a filled group can be archived to a single unit of offline storage media (such as a tape or disk), with the least amount of space on the medium left unused.
- Planning the Block Size of Redo Log Files
  Unlike the database block size, which can be between 2K and 32K, redo log files always default to a block size that is equal to the physical sector size of the disk. Historically, this has typically been 512 bytes (512B).
- Choosing the Number of Redo Log Files
  The best way to determine the appropriate number of redo log files for a database instance is to test different configurations. The optimum configuration has the fewest groups possible without hampering LGWR from writing redo log information.
- Controlling Archive Lag
  You can force all enabled redo log threads to switch their current logs at regular time intervals.

See Also:

Persistent Memory documentation for information about provisioning Intel Optane DC persistent memory

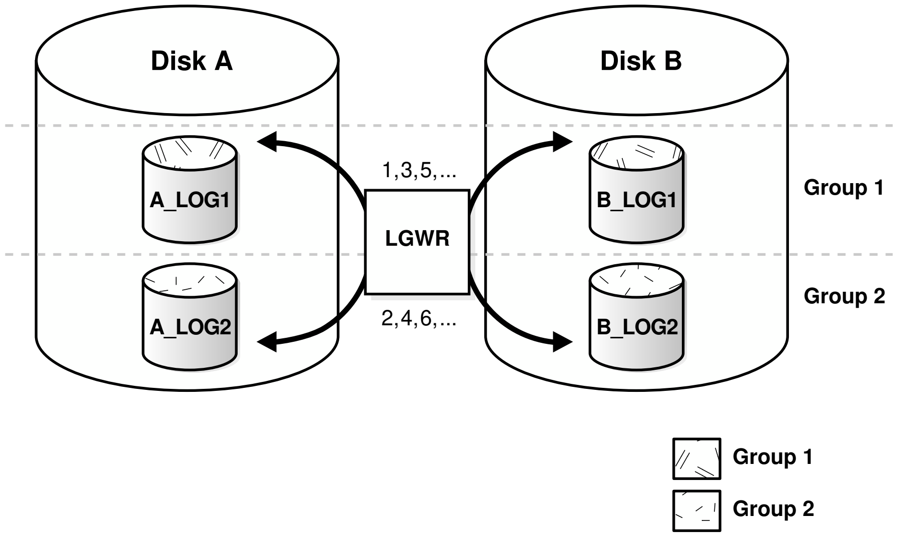
9.2.1 Multiplexing Redo Log Files

To protect against a failure involving the redo log itself, Oracle Database allows a **multiplexed** redo log, meaning that two or more identical copies of the redo log can be automatically maintained in separate locations.

For the most benefit, these locations should be on separate disks. Even if all copies of the redo log are on the same disk, however, the redundancy can help protect against I/O errors, file corruption, and so on. When redo log files are multiplexed, LGWR concurrently writes the same redo log information to multiple identical redo log files, thereby eliminating a single point of redo log failure.

Multiplexing is implemented by creating *groups* of redo log files. A **group** consists of a redo log file and its multiplexed copies. Each identical copy is said to be a **member** of the group. Each redo log group is defined by a number, such as group 1, group 2, and so on.

Figure 9-2 Multiplexed Redo Log Files



In Figure 9-2, `A_LOG1` and `B_LOG1` are both members of Group 1, `A_LOG2` and `B_LOG2` are both members of Group 2, and so forth. Each member in a group must be the same size.

Each member of a log file group is concurrently active—that is, concurrently written to by LGWR—as indicated by the identical log sequence numbers assigned by LGWR. In Figure 9-2, first LGWR writes concurrently to both `A_LOG1` and `B_LOG1`. Then it writes concurrently to both `A_LOG2` and `B_LOG2`, and so on. LGWR never writes concurrently to members of different groups (for example, to `A_LOG1` and `B_LOG2`).

Note:

Oracle recommends that you multiplex your redo log files. The loss of the log file data can be catastrophic if recovery is required. Note that when you multiplex the redo log, the database must increase the amount of I/O that it performs. Depending on your configuration, this may impact overall database performance.

- Responding to Redo Log Failure
  Whenever LGWR cannot write to a member of a group, the database marks that member as `INVALID` and writes an error message to the LGWR trace file and to the database alert log to indicate the problem with the inaccessible files.
- Legal and Illegal Configurations
  In most cases, a multiplexed redo log should be symmetrical: all groups of the redo log should have the same number of members. However, the database does not require that a multiplexed redo log be symmetrical.

9.2.1.1 Responding to Redo Log Failure

Whenever LGWR cannot write to a member of a group, the database marks that member as `INVALID` and writes an error message to the LGWR trace file and to the database alert log to indicate the problem with the inaccessible files.

The specific reaction of LGWR when a redo log member is unavailable depends on the reason for the lack of availability, as summarized in the table that follows.

Condition	LGWR Action
LGWR can successfully write to at least one member in a group	Writing proceeds as normal. LGWR writes to the available members of a group and ignores the unavailable members.
LGWR cannot access the next group at a log switch because the group must be archived	Database operation temporarily halts until the group becomes available or until the group is archived.
All members of the next group are inaccessible to LGWR at a log switch because of media failure	Oracle Database returns an error, and the database instance shuts down. In this case, you may need to perform media recovery on the database from the loss of a redo log file. If the database checkpoint has moved beyond the lost redo log, media recovery is not necessary, because the database has saved the data recorded in the redo log to the data files. You need only drop the inaccessible redo log group. If the database did not archive the bad log, use `ALTER DATABASE CLEAR LOGFILE UNARCHIVED` to disable archiving before the log can be dropped.
All members of a group suddenly become inaccessible to LGWR while it is writing to them	Oracle Database returns an error and the database instance immediately shuts down. In this case, you may need to perform media recovery. If the media containing the log is not actually lost--for example, if the drive for the log was inadvertently turned off--media recovery may not be needed. In this case, you need only turn the drive back on and let the database perform automatic instance recovery.

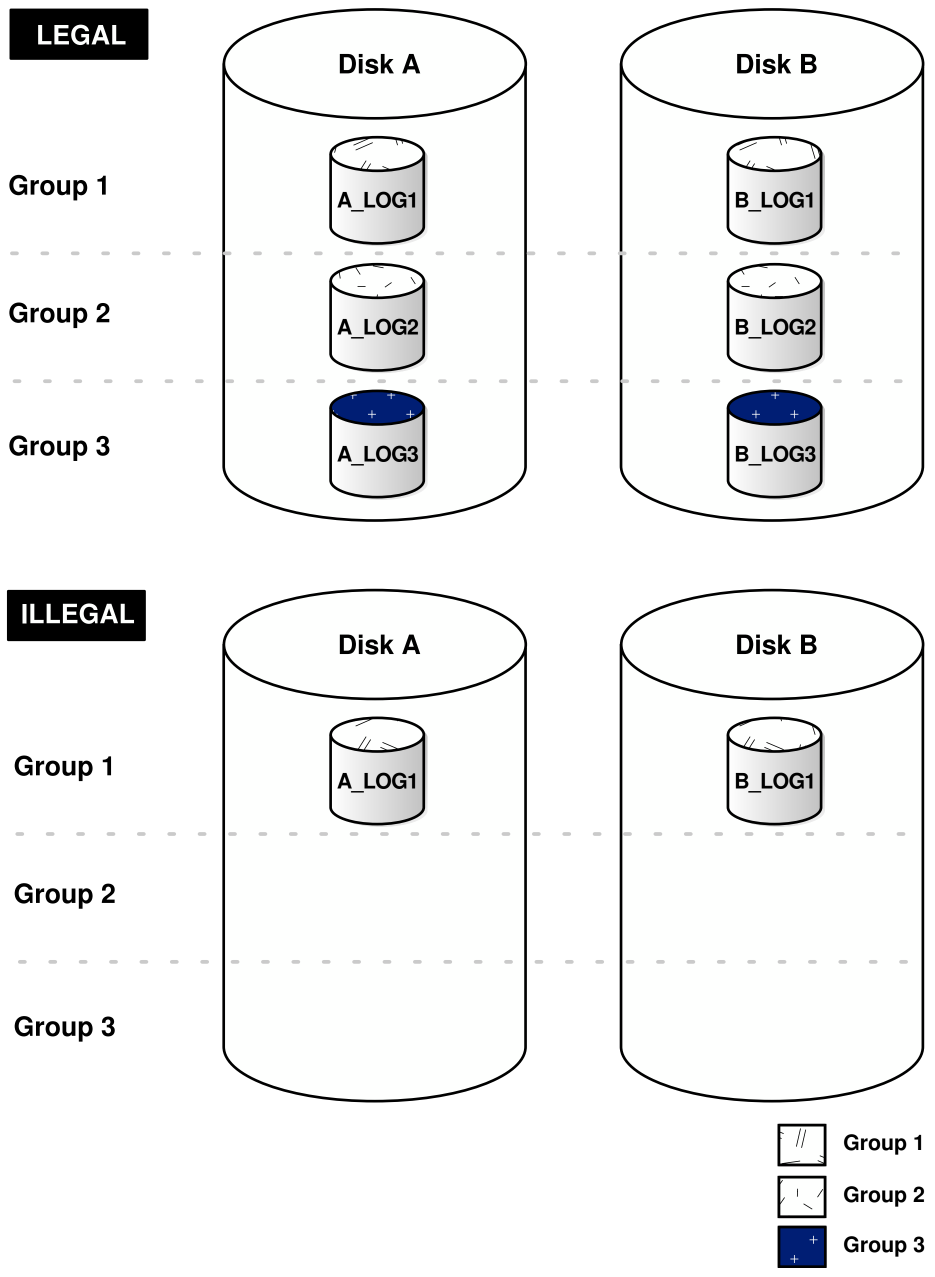
9.2.1.2 Legal and Illegal Configurations

In most cases, a multiplexed redo log should be symmetrical: all groups of the redo log should have the same number of members. However, the database does not require that a multiplexed redo log be symmetrical.

For example, one group can have only one member, and other groups can have two members. This configuration protects against disk failures that temporarily affect some redo log members but leave others intact.

The only requirement for an instance redo log is that it have at least two groups. Figure 9-3 shows legal and illegal multiplexed redo log configurations. The second configuration is illegal because it has only one group.

Figure 9-3 Legal and Illegal Multiplexed Redo Log Configuration



9.2.2 Placing Redo Log Members on Different Disks

When setting up a multiplexed redo log, place members of a group on different physical disks. If a single disk fails, then only one member of a group becomes unavailable to LGWR and other members remain accessible to LGWR, so the instance can continue to function.

If you archive the redo log, spread redo log members across disks to eliminate contention between the LGWR and ARC*n* background processes. For example, if you have two groups of multiplexed redo log members (a *duplexed* redo log), place each member on a different disk and set your archiving destination to a fifth disk. Doing so will avoid contention between LGWR (writing to the members) and ARC*n* (reading the members).

Data files should also be placed on different disks from redo log files to reduce contention in writing data blocks and redo records.

9.2.3 Planning the Size of Redo Log Files

When setting the size of redo log files, consider whether you will be archiving the redo log. Redo log files should be sized so that a filled group can be archived to a single unit of offline storage media (such as a tape or disk), with the least amount of space on the medium left unused.

For example, suppose only one filled redo log group can fit on a tape and 49% of the tape storage capacity remains unused. In this case, it is better to decrease the size of the redo log files slightly, so that two log groups could be archived on each tape.

All members of the same multiplexed redo log group must be the same size. Members of different groups can have different sizes. However, there is no advantage in varying file size between groups. If checkpoints are not set to occur between log switches, make all groups the same size to guarantee that checkpoints occur at regular intervals.

The minimum size permitted for a redo log file is 4 MB.

See Also:

Your operating system–specific Oracle documentation. The default size of redo log files is operating system dependent.

9.2.4 Planning the Block Size of Redo Log Files

Unlike the database block size, which can be between 2K and 32K, redo log files always default to a block size that is equal to the physical sector size of the disk. Historically, this has typically been 512 bytes (512B).

Some newer high-capacity disk drives offer 4K byte (4K) sector sizes for both increased ECC capability and improved format efficiency. Most Oracle Database platforms are able to detect this larger sector size. The database then automatically creates redo log files with a 4K block size on those disks.

However, with a block size of 4K, there is increased redo wastage. In fact, the amount of redo wastage in 4K blocks versus 512B blocks is significant. You can determine the amount of redo wastage by viewing the statistics stored in the `V$SESSTAT` and `V$SYSSTAT` views.

```
SQL> SELECT name, value FROM v$sysstat WHERE name = 'redo wastage';

NAME                                VALUE
-------------------------------- ----------
redo wastage                       17941684
```

To avoid the additional redo wastage, if you are using emulation-mode disks—4K sector size disk drives that emulate a 512B sector size at the disk interface—you can override the default 4K block size for redo logs by specifying a 512B block size or, for some platforms, a 1K block size. However, you will incur a significant performance degradation when a redo log write is not aligned with the beginning of the 4K physical sector. Because seven out of eight 512B slots in a 4K physical sector are not aligned, performance degradation typically does occur. Thus, you must evaluate the trade-off between performance and disk wastage when planning the redo log block size on 4K sector size emulation-mode disks.

You can specify the block size of online redo log files with the `BLOCKSIZE` keyword in the `CREATE DATABASE`, `ALTER DATABASE`, and `CREATE CONTROLFILE` statements. On some platforms, the permissible block sizes are 512 and 4096. On other platforms, the permissible block sizes are 1024 and 4096.

The following statement adds a redo log file group with a block size of 512B. The `BLOCKSIZE 512` clause is valid but not required for 512B sector size disks. For 4K sector size emulation-mode disks, the `BLOCKSIZE 512` clause overrides the default 4K size.

```
ALTER DATABASE orcl ADD LOGFILE
  GROUP 4 ('/u01/logs/orcl/redo04a.log','/u01/logs/orcl/redo04b.log')
  SIZE 100M BLOCKSIZE 512 REUSE;
```

To ascertain the redo log file block size, run the following query:

```
SQL> SELECT BLOCKSIZE FROM V$LOG;

BLOCKSIZE
---------
      512
```

See Also:

- *Oracle Database SQL Language Reference* for information about the `ALTER DATABASE` command.
- *Oracle Database Reference* for information about the `V$SESSTAT` view
- *Oracle Database Reference* for information about the `V$SYSSTAT` view

9.2.5 Choosing the Number of Redo Log Files

The best way to determine the appropriate number of redo log files for a database instance is to test different configurations. The optimum configuration has the fewest groups possible without hampering LGWR from writing redo log information.

In some cases, a database instance may require only two groups. In other situations, a database instance may require additional groups to guarantee that a recycled group is always available to LGWR. During testing, the easiest way to determine whether the current redo log configuration is satisfactory is to examine the contents of the LGWR trace file and the database alert log. If messages indicate that LGWR frequently has to wait for a group because a checkpoint has not completed or a group has not been archived, add groups.

Consider the parameters that can limit the number of redo log files before setting up or altering the configuration of an instance redo log. The following parameters limit the number of redo log files that you can add to a database:

- The `MAXLOGFILES` parameter used in the `CREATE DATABASE` statement determines the maximum number of groups of redo log files for each database. Group values can range from 1 to `MAXLOGFILES`. You can exceed the `MAXLOGFILES` limit, and the control files expand as needed. If `MAXLOGFILES` is not specified for the `CREATE DATABASE` statement, then the database uses an operating system specific default value.
- The `MAXLOGMEMBERS` parameter used in the `CREATE DATABASE` statement determines the maximum number of members for each group. As with `MAXLOGFILES`, the only way to override this upper limit is to re-create the database or control file. Therefore, it is important to consider this limit before creating a database. If no `MAXLOGMEMBERS` parameter is specified for the `CREATE DATABASE` statement, then the database uses an operating system specific default value.

See Also:

Your operating system specific Oracle documentation for the default and legal values of the `MAXLOGFILES` and `MAXLOGMEMBERS` parameters

9.2.6 Controlling Archive Lag

You can force all enabled redo log threads to switch their current logs at regular time intervals.

In a primary/standby database configuration, changes are made available to the standby database by archiving redo logs at the primary site and then shipping them to the standby database. The changes that are being applied by the standby database can lag behind the changes that are occurring on the primary database, because the standby database must wait for the changes in the primary database redo log to be archived (into the archived redo log) and then shipped to it. To limit this lag, you can set the `ARCHIVE_LAG_TARGET` initialization parameter. Setting this parameter lets you specify in seconds how long that lag can be.

- Setting the ARCHIVE_LAG_TARGET Initialization Parameter
  When you set the `ARCHIVE_LAG_TARGET` initialization parameter, you cause the database to examine the current redo log for the instance periodically and determine when to switch the log.
- Factors Affecting the Setting of ARCHIVE_LAG_TARGET
  There are several factors to consider when you are setting the `ARCHIVE_LAG_TARGET` initialization parameter.

9.2.6.1 Setting the ARCHIVE_LAG_TARGET Initialization Parameter

When you set the `ARCHIVE_LAG_TARGET` initialization parameter, you cause the database to examine the current redo log for the instance periodically and determine when to switch the log.

If the following conditions are met, then the instance will switch the log:

- The current log was created before *n* seconds ago, and the estimated archival time for the current log is *m* seconds (proportional to the number of redo blocks used in the current log), where *n* + *m* exceeds the value of the `ARCHIVE_LAG_TARGET` initialization parameter.
- The current log contains redo records.

In an Oracle Real Application Clusters environment, the instance also causes other threads to switch and archive their logs if they are falling behind. This can be particularly useful when one instance in the cluster is more idle than the other instances (as when you are running a 2-node primary/secondary configuration of Oracle Real Application Clusters).

The `ARCHIVE_LAG_TARGET` initialization parameter provides an upper limit for how long (in seconds) the current log of the database can span. Because the estimated archival time is also considered, this is not the exact log switch time.

- Set the `ARCHIVE_LAG_TARGET` initialization parameter.

The following initialization parameter setting sets the log switch interval to 30 minutes (a typical value).

```
ARCHIVE_LAG_TARGET = 1800
```

A value of 0 disables this time-based log switching functionality. This is the default setting.

You can set the `ARCHIVE_LAG_TARGET` initialization parameter even if there is no standby database. For example, the `ARCHIVE_LAG_TARGET` parameter can be set specifically to force logs to be switched and archived.

`ARCHIVE_LAG_TARGET` is a dynamic parameter and can be set with the `ALTER SYSTEM SET` statement.

Note:

The `ARCHIVE_LAG_TARGET` parameter must be set to the same value in all instances of an Oracle Real Application Clusters environment. Failing to do so results in unpredictable behavior.

9.2.6.2 Factors Affecting the Setting of ARCHIVE_LAG_TARGET

There are several factors to consider when you are setting the `ARCHIVE_LAG_TARGET` initialization parameter.

Consider the following factors when determining if you want to set the `ARCHIVE_LAG_TARGET` initialization parameter and in determining the value for this parameter.

- Overhead of switching (as well as archiving) logs
- How frequently normal log switches occur as a result of log full conditions
- How much redo loss is tolerated in the standby database

Setting `ARCHIVE_LAG_TARGET` may not be very useful if natural log switches already occur more frequently than the interval specified. However, in the case of irregularities of redo generation speed, the interval does provide an upper limit for the time range each current log covers.

If the `ARCHIVE_LAG_TARGET` initialization parameter is set to a very low value, there can be a negative impact on performance. This can force frequent log switches. Set the parameter to a reasonable value so as not to degrade the performance of the primary database.

9.3 Creating Redo Log Groups and Members

Plan the redo log for a database and create all required groups and members of redo log files during database creation. However, there are situations where you might want to create additional groups or members. For example, adding groups to a redo log can correct redo log group availability problems.

To create new redo log groups and members, you must have the `ALTER DATABASE` system privilege. A database can have up to `MAXLOGFILES` groups.

- Creating Redo Log Groups
  To create a new group of redo log files, use the SQL statement `ALTER DATABASE` with the `ADD LOGFILE` clause.
- Creating Redo Log Members
  In some cases, it might not be necessary to create a complete group of redo log files. A group could already exist, but not be complete because one or more members of the group were dropped (for example, because of a disk failure). In this case, you can add new members to an existing group.

See Also:

Oracle Database SQL Language Reference for a complete description of the `ALTER DATABASE` statement

9.3.1 Creating Redo Log Groups

To create a new group of redo log files, use the SQL statement `ALTER DATABASE` with the `ADD LOGFILE` clause.

- Run the SQL statement `ALTER DATABASE` with the `ADD LOGFILE` clause.

For example, the following statement adds a new group of redo logs to the database:

```
ALTER DATABASE
  ADD LOGFILE ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 100M;
```

Note:

Provide full path names of new log members to specify their location. Otherwise, the files are created in either the default or current directory of the database server, depending upon your operating system.

You can also specify the number that identifies the group using the `GROUP` clause:

```
ALTER DATABASE
  ADD LOGFILE GROUP 10 ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo')
      SIZE 100M BLOCKSIZE 512;
```

Using group numbers can make administering redo log groups easier. However, the group number must be between 1 and `MAXLOGFILES`. Do not skip redo log file group numbers (that is, do not number your groups 10, 20, 30, and so on), or you will consume unnecessary space in the control files of the database.

In the preceding statement, the `BLOCKSIZE` clause is optional. See "Planning the Block Size of Redo Log Files" for more information.

9.3.2 Creating Redo Log Members

In some cases, it might not be necessary to create a complete group of redo log files. A group could already exist, but not be complete because one or more members of the group were dropped (for example, because of a disk failure). In this case, you can add new members to an existing group.

To create new redo log members for an existing group:

- Run the SQL statement `ALTER DATABASE` with the `ADD LOGFILE MEMBER` clause.

For example, the following statement adds a new redo log member to redo log group number 2:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2b.rdo' TO GROUP 2;
```

Notice that file names must be specified, but sizes need not be. The size of the new members is determined from the size of the existing members of the group.

When using the `ALTER DATABASE` statement, you can alternatively identify the target group by specifying all of the other members of the group in the `TO` clause, as shown in the following example:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.rdo'
    TO ('/oracle/dbs/log2a.rdo', '/oracle/dbs/log2b.rdo');
```

Note:

Fully specify the file names of new log members to indicate where the operating system files should be created. Otherwise, the files will be created in either the default or current directory of the database server, depending upon your operating system. You may also note that the status of the new log member is shown as `INVALID`. This is normal and it will change to active (blank) when it is first used.

9.4 Relocating and Renaming Redo Log Members

You can use operating system commands to relocate redo logs, then use the `ALTER DATABASE` statement to make their new names (locations) known to the database.

This procedure is necessary, for example, if the disk currently used for some redo log files is going to be removed, or if data files and several redo log files are stored on the same disk and should be separated to reduce contention.

To rename redo log members, you must have the `ALTER DATABASE` system privilege. Additionally, you might also need operating system privileges to copy files to the desired location and privileges to open and back up the database.

Before relocating your redo logs, or making any other structural changes to the database, completely back up the database in case you experience problems while performing the operation. As a precaution, after renaming or relocating a set of redo log files, immediately back up the database control file.

Use the following steps for relocating redo logs. The example used to illustrate these steps assumes:

- The log files are located on two disks: `diska` and `diskb`.
- The redo log is duplexed: one group consists of the members `/diska/logs/log1a.rdo` and `/diskb/logs/log1b.rdo`, and the second group consists of the members `/diska/logs/log2a.rdo` and `/diskb/logs/log2b.rdo`.
- The redo log files located on `diska` must be relocated to `diskc`. The new file names will reflect the new location: `/diskc/logs/log1c.rdo` and `/diskc/logs/log2c.rdo`.

To rename redo log members:

1. Shut down the database.

   ```
   SHUTDOWN
   ```

2. Copy the redo log files to the new location.

   Operating system files, such as redo log members, must be copied using the appropriate operating system commands. See your operating system specific documentation for more information about copying files.

   Note:

   You can execute an operating system command to copy a file (or perform other operating system commands) without exiting SQL*Plus by using the `HOST` command. Some operating systems allow you to use a character in place of the word `HOST`. For example, you can use an exclamation point (!) in UNIX.

   The following example uses operating system commands (UNIX) to move the redo log members to a new location:

   ```
   mv /diska/logs/log1a.rdo /diskc/logs/log1c.rdo
   mv /diska/logs/log2a.rdo /diskc/logs/log2c.rdo
   ```

3. Startup the database, mount, but do not open it.

   ```
   CONNECT / as SYSDBA
   STARTUP MOUNT
   ```

4. Rename the redo log members.

   Use the `ALTER DATABASE` statement with the `RENAME FILE` clause to rename the database redo log files.

   ```
   ALTER DATABASE
     RENAME FILE '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
              TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
   ```

5. Open the database for normal operation.

   The redo log alterations take effect when the database is opened.

   ```
   ALTER DATABASE OPEN;
   ```

9.5 Dropping Redo Log Groups and Members

In some cases, you may want to drop an entire group of redo log members.

For example, you want to reduce the number of groups in an instance redo log. In a different case, you may want to drop one or more specific redo log members. For example, if a disk failure occurs, you may need to drop all the redo log files on the failed disk so that the database does not try to write to the inaccessible files. In other situations, particular redo log files become unnecessary. For example, a file might be stored in an inappropriate location.

- Dropping Log Groups
  You can drop a redo log group.
- Dropping Redo Log Members
  You can drop redo log members.

9.5.1 Dropping Log Groups

You can drop a redo log group.

To drop a redo log group, you must have the `ALTER DATABASE` system privilege. Before dropping a redo log group, consider the following restrictions and precautions:

- An instance requires at least two groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.)
- You can drop a redo log group only if it is inactive. If you must drop the current group, then first force a log switch to occur.
- Make sure a redo log group is archived (if archiving is enabled) before dropping it. To see whether this has happened, use the `V$LOG` view.

```
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;

   GROUP# ARC STATUS
--------- --- ----------------
        1 YES ACTIVE
        2 NO  CURRENT
        3 YES INACTIVE
        4 YES INACTIVE
```

To drop a redo log group:

- Run the SQL statement `ALTER DATABASE` with the `DROP LOGFILE` clause.

For example, the following statement drops redo log group number 3:

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

When a redo log group is dropped from the database, and you are not using the Oracle Managed Files feature, the operating system files are not deleted from disk. Rather, the control files of the associated database are updated to drop the members of the group from the database structure. After dropping a redo log group, ensure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log files.

When using Oracle Managed Files, the cleanup of operating systems files is done automatically for you.

9.5.2 Dropping Redo Log Members

You can drop redo log members.

To drop a redo log member, you must have the `ALTER DATABASE` system privilege. Consider the following restrictions and precautions before dropping individual redo log members:

- It is permissible to drop redo log files so that a multiplexed redo log becomes temporarily asymmetric. For example, if you use duplexed groups of redo log files, you can drop one member of one group, even though all other groups have two members each. However, you should rectify this situation immediately so that all groups have at least two members, and thereby eliminate the single point of failure possible for the redo log.
- An instance always requires at least two valid groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.) If the member you want to drop is the last valid member of the group, you cannot drop the member until the other members become valid. To see a redo log file status, use the `V$LOGFILE` view. A redo log file becomes `INVALID` if the database cannot access it. It becomes `STALE` if the database suspects that it is not complete or correct. A stale log file becomes valid again the next time its group is made the active group.
- You can drop a redo log member only if it is *not* part of an active or current group. To drop a member of an active group, first force a log switch to occur.
- Make sure the group to which a redo log member belongs is archived (if archiving is enabled) before dropping the member. To see whether this has happened, use the `V$LOG` view.

To drop specific inactive redo log members:

- Run the `ALTER DATABASE` statement with the `DROP LOGFILE MEMBER` clause.

The following statement drops the redo log `/oracle/dbs/log3c.rdo`:

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log3c.rdo';
```

When a redo log member is dropped from the database, the operating system file is not deleted from disk. Rather, the control files of the associated database are updated to drop the member from the database structure. After dropping a redo log file, ensure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log file.

To drop a member of an active group, you must first force a log switch.

9.6 Forcing Log Switches

A log switch occurs when LGWR stops writing to one redo log group and starts writing to another. By default, a log switch occurs automatically when the current redo log file group fills.

You can force a log switch to make the currently active group inactive and available for redo log maintenance operations. For example, you want to drop the currently active group, but are not able to do so until the group is inactive. You may also want to force a log switch if the currently active group must be archived at a specific time before the members of the group are completely filled. This option is useful in configurations with large redo log files that take a long time to fill.

To force a log switch, you must have the `ALTER SYSTEM` privilege.

To force a log switch,

- Run the `ALTER SYSTEM` statement with the `SWITCH LOGFILE` clause.

For example, the following statement forces a log switch:

```
ALTER SYSTEM SWITCH LOGFILE;
```

9.7 Verifying Blocks in Redo Log Files

You can configure the database to use checksums to verify blocks in the redo log files.

If you set the initialization parameter `DB_BLOCK_CHECKSUM` to `TYPICAL` (the default), then the database computes a checksum for each database block when it is written to disk, including each redo log block as it is being written to the current log. The checksum is stored the header of the block.

Oracle Database uses the checksum to detect corruption in a redo log block. The database verifies the redo log block when the block is read from an archived log during recovery and when it writes the block to an archive log file. An error is raised and written to the alert log if corruption is detected.

If corruption is detected in a redo log block while trying to archive it, the system attempts to read the block from another member in the group. If the block is corrupted in all members of the redo log group, then archiving cannot proceed.

The value of the `DB_BLOCK_CHECKSUM` parameter can be changed dynamically using the `ALTER SYSTEM` statement.

Note:

There is a slight overhead and decrease in database performance with `DB_BLOCK_CHECKSUM` enabled. Monitor your database performance to decide if the benefit of using data block checksums to detect corruption outweighs the performance impact.

See Also:

Oracle Database Reference for a description of the `DB_BLOCK_CHECKSUM` initialization parameter

9.8 Clearing a Redo Log File

A redo log file might become corrupted while the database is open, and ultimately stop database activity because archiving cannot continue.

In this situation, to reinitialize the file without shutting down the database:

- Run the `ALTER DATABASE CLEAR LOGFILE` SQL statement.

The following statement clears the log files in redo log group number 3:

```
ALTER DATABASE CLEAR LOGFILE GROUP 3;
```

This statement overcomes two situations where dropping redo logs is not possible:

- If there are only two log groups
- The corrupt redo log file belongs to the current group

If the corrupt redo log file has not been archived, use the `UNARCHIVED` keyword in the statement.

```
ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;
```

This statement clears the corrupted redo logs and avoids archiving them. The cleared redo logs are available for use even though they were not archived.

If you clear a log file that is needed for recovery of a backup, then you can no longer recover from that backup. The database writes a message in the alert log describing the backups from which you cannot recover.

> **Note:**
>
> If you clear an unarchived redo log file, you should make another backup of the database.

To clear an unarchived redo log that is needed to bring an offline tablespace online, use the `UNRECOVERABLE DATAFILE` clause in the `ALTER DATABASE CLEAR LOGFILE` statement.

If you clear a redo log needed to bring an offline tablespace online, you will not be able to bring the tablespace online again. You will have to drop the tablespace or perform an incomplete recovery. Note that tablespaces taken offline normal do not require recovery.

9.9 Precedence of FORCE LOGGING Settings

You can set `FORCE LOGGING` and `NOLOGGING` at various levels, such as for a database, pluggable database (PDB), tablespace, or database object. When `FORCE LOGGING` is set at one or more levels, the precedence of `FORCE LOGGING` settings determines what is logged in the redo log.

You can put a multitenant container database (CDB) into `FORCE LOGGING` mode. In this mode, the database logs all changes in the database except for changes in temporary tablespaces and temporary segments. This setting takes precedence over and is independent of any `NOLOGGING` or `FORCE LOGGING` settings you specify for individual tablespaces and any `NOLOGGING` settings you specify for individual database objects.

You can also put a tablespace into `FORCE LOGGING` mode. The database logs all changes to all objects in the tablespace except changes to temporary segments, overriding any `NOLOGGING` setting for individual objects.

In addition, you can specify a logging attribute with the *logging_clause* for various types of database objects that determines whether certain DML operations will be logged in the redo log file (`LOGGING`) or not (`NOLOGGING`). You can specify a logging attribute for the following types of database objects:

- Tables
- Indexes
- Materialized views

The following table summarizes the logging settings at each level and shows the result for a CDB.

Table 9-1 Precedence of FORCE LOGGING Settings for a CDB

CDB	PDB	Tablespace	Database Object LOGGING Attribute	Result
FORCE LOGGING	Ignored	Ignored	Ignored	Logged
NO FORCE LOGGING	ENABLE FORCE LOGGING	Ignored	Ignored	Logged
NO FORCE LOGGING	ENABLE FORCE NOLOGGING	Ignored	Ignored	Not Logged
NO FORCE LOGGING	DISABLE FORCE [NO]LOGGING (no setting)	FORCE LOGGING	Ignored	Logged
NO FORCE LOGGING	DISABLE FORCE [NO]LOGGING (no setting)	NO FORCE LOGGING	LOGGING	Logged
NO FORCE LOGGING	DISABLE FORCE [NO]LOGGING (no setting)	NO FORCE LOGGING	NOLOGGING	Not Logged

9.10 Redo Log Data Dictionary Views

You can query a set of data dictionary views for information about the redo log.

The following views provide information on redo logs.

View	Description
V$LOG	Displays the redo log file information from the control file
V$LOGFILE	Identifies redo log groups and members and member status
V$LOG_HISTORY	Contains log history information

The following query returns the control file information about the redo log for a database.

```
SELECT GROUP#, THREAD#, SEQUENCE#, BYTES, MEMBERS, ARCHIVED,
       STATUS, FIRST_CHANGE#, FIRST_TIME
  FROM V$LOG;

GROUP# THREAD#   SEQ   BYTES  MEMBERS  ARC STATUS     FIRST_CHANGE# FIRST_TIM
------ ------- ----- -------  -------  --- ---------  ------------- ---------
     1       1 10605 1048576        1  YES ACTIVE          11515628 16-APR-00
     2       1 10606 1048576        1  NO  CURRENT         11517595 16-APR-00
     3       1 10603 1048576        1  YES INACTIVE        11511666 16-APR-00
     4       1 10604 1048576        1  YES INACTIVE        11513647 16-APR-00
```

To see the names of all of the member of a group, use a query similar to the following:

```
SELECT GROUP#, STATUS, MEMBER FROM V$LOGFILE;

GROUP#   STATUS  MEMBER
------  -------  ----------------------------------
     1           D:\ORANT\ORADATA\IDDB2\REDO04.LOG
```

```
    2           D:\ORANT\ORADATA\IDDB2\REDO03.LOG
    3           D:\ORANT\ORADATA\IDDB2\REDO02.LOG
    4           D:\ORANT\ORADATA\IDDB2\REDO01.LOG
```

If `STATUS` is blank for a member, then the file is in use.

10

Managing Archived Redo Log Files

You manage the archived redo log files by completing tasks such as choosing between NOARCHIVELOG or ARCHIVELOG mode and specifying archive destinations.

See Also:

- Using Oracle Managed Files for information about creating an archived redo log that is both created and managed by the Oracle Database server
- *Oracle Real Application Clusters Administration and Deployment Guide* for information specific to archiving in the Oracle Real Application Clusters environment

10.1 What Is the Archived Redo Log?

Oracle Database lets you save filled groups of redo log files to one or more offline destinations, known collectively as the **archived redo log**.

The process of turning redo log files into archived redo log files is called **archiving**. This process is only possible if the database is running in **`ARCHIVELOG` mode**. You can choose automatic or manual archiving.

An archived redo log file is a copy of one of the filled members of a redo log group. It includes the redo entries and the unique log sequence number of the identical member of the redo log group. For example, if you are multiplexing your redo log, and if group 1 contains identical member files `a_log1` and `b_log1`, then the archiver process (ARC*n*) will archive one of these member files. Should `a_log1` become corrupted, then ARC*n* can still archive the identical `b_log1`. The archived redo log contains a copy of every group created since you enabled archiving.

When the database is running in `ARCHIVELOG` mode, the log writer process (LGWR) cannot reuse and hence overwrite a redo log group until it has been archived. The background process ARC*n* automates archiving operations when automatic archiving is enabled. The database starts multiple archiver processes as needed to ensure that the archiving of filled redo logs does not fall behind.

You can use archived redo log files to:

- Recover a database
- Update a standby database
- Get information about the history of a database using the LogMiner utility

See Also:

The following sources document the uses for archived redo log files:

- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Data Guard Concepts and Administration* discusses setting up and maintaining a standby database
- *Oracle Database Utilities* contains instructions for using the LogMiner PL/SQL package

10.2 Choosing Between NOARCHIVELOG and ARCHIVELOG Mode

You must choose between running your database in `NOARCHIVELOG` or `ARCHIVELOG` mode.

The choice of whether to enable the archiving of filled groups of redo log files depends on the availability and reliability requirements of the application running on the database. If you cannot afford to lose any data in your database in the event of a disk failure, use `ARCHIVELOG` mode. The archiving of filled redo log files can require you to perform extra administrative operations.

- Running a Database in NOARCHIVELOG Mode
  When you run your database in `NOARCHIVELOG` mode, you disable the archiving of the redo log.
- Running a Database in ARCHIVELOG Mode
  When you run a database in `ARCHIVELOG` mode, you enable the archiving of the redo log.

10.2.1 Running a Database in NOARCHIVELOG Mode

When you run your database in `NOARCHIVELOG` mode, you disable the archiving of the redo log.

The database control file indicates that filled groups are not required to be archived. Therefore, when a filled group becomes inactive after a log switch, the group is available for reuse by LGWR.

`NOARCHIVELOG` mode protects a database from instance failure but not from media failure. Only the most recent changes made to the database, which are stored in the online redo log groups, are available for instance recovery. If a media failure occurs while the database is in `NOARCHIVELOG` mode, you can only restore the database to the point of the most recent full database backup. You cannot recover transactions subsequent to that backup.

In `NOARCHIVELOG` mode you cannot perform online tablespace backups, nor can you use online tablespace backups taken earlier while the database was in `ARCHIVELOG` mode. To restore a database operating in `NOARCHIVELOG` mode, you can use only whole database backups taken while the database is closed. Therefore, if you decide to operate a database in `NOARCHIVELOG` mode, take whole database backups at regular, frequent intervals.

10.2.2 Running a Database in ARCHIVELOG Mode

When you run a database in `ARCHIVELOG` mode, you enable the archiving of the redo log.

The database control file indicates that a group of filled redo log files cannot be reused by LGWR until the group is archived. A filled group becomes available for archiving immediately after a redo log switch occurs.

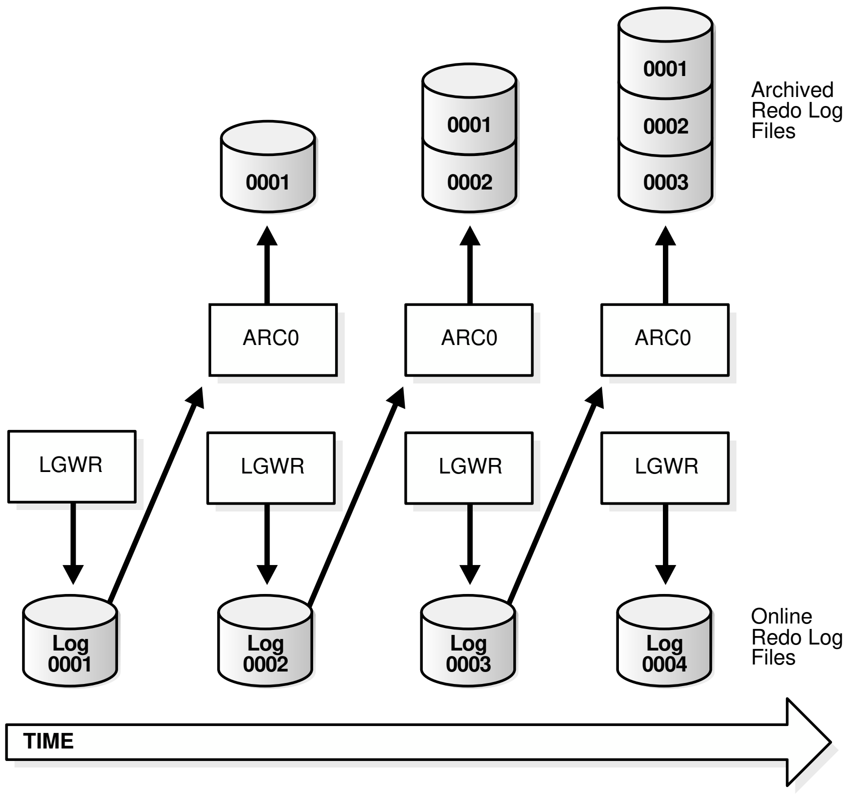
The archiving of filled groups has these advantages:

- A database backup, together with online and archived redo log files, guarantees that you can recover all committed transactions in the event of an operating system or disk failure.
- If you keep archived logs available, you can use a backup taken while the database is open and in normal system use.
- You can keep a standby database current with its original database by continuously applying the original archived redo log files to the standby.

You can configure an instance to archive filled redo log files automatically, or you can archive manually. For convenience and efficiency, automatic archiving is usually best. Figure 10-1 illustrates how the archiver process (ARC0 in this illustration) writes filled redo log files to the database archived redo log.

If all databases in a distributed database operate in `ARCHIVELOG` mode, you can perform coordinated distributed database recovery. However, if any database in a distributed database is in `NOARCHIVELOG` mode, recovery of a global distributed database (to make all databases consistent) is limited by the last full backup of any database operating in `NOARCHIVELOG` mode.

Figure 10-1 Redo Log File Use in ARCHIVELOG Mode



Tip:

It is good practice to move archived redo log files and corresponding database backups from the local disk to permanent offline storage media such as tape. A primary value of archived logs is database recovery, so you want to ensure that these logs are safe should disaster strike your primary database.

10.3 Controlling Archiving

You can set the archiving mode for your database and adjust the number of archiver processes.

- Setting the Initial Database Archiving Mode
  You set the initial archiving mode as part of database creation in the `CREATE DATABASE` statement.
- Changing the Database Archiving Mode
  To change the archiving mode of the database, use the `ALTER DATABASE` statement with the `ARCHIVELOG` or `NOARCHIVELOG` clause.
- Performing Manual Archiving
  For convenience and efficiency, automatic archiving is usually best. However, you can configure your database for manual archiving only.

- Adjusting the Number of Archiver Processes
  The `LOG_ARCHIVE_MAX_PROCESSES` initialization parameter specifies the number of ARC*n* processes that the database initially starts. The default is four processes.

> **See Also:**
>
> Your Oracle operating system specific documentation for additional information on controlling archiving modes

10.3.1 Setting the Initial Database Archiving Mode

You set the initial archiving mode as part of database creation in the `CREATE DATABASE` statement.

Usually, you can use the default of `NOARCHIVELOG` mode at database creation because there is no need to archive the redo information generated by that process. After creating the database, decide whether to change the initial archiving mode.

If you specify `ARCHIVELOG` mode, you must have initialization parameters set that specify the destinations for the archived redo log files (see "Setting Initialization Parameters for Archive Destinations").

10.3.2 Changing the Database Archiving Mode

To change the archiving mode of the database, use the `ALTER DATABASE` statement with the `ARCHIVELOG` or `NOARCHIVELOG` clause.

To change the archiving mode, you must be connected to the database with administrator privileges (`AS SYSDBA`).

The following steps switch the database archiving mode from `NOARCHIVELOG` to `ARCHIVELOG`:

1. Shut down the database instance.

   ```
   SHUTDOWN IMMEDIATE
   ```

   An open database must first be closed and any associated instances shut down before you can switch the database archiving mode. You cannot change the mode from `ARCHIVELOG` to `NOARCHIVELOG` if any data files need media recovery.
2. Back up the database.

   Before making any major change to a database, always back up the database to protect against any problems. This will be your final backup of the database in `NOARCHIVELOG` mode and can be used if something goes wrong during the change to `ARCHIVELOG` mode. See *Oracle Database Backup and Recovery User's Guide* for information about taking database backups.
3. Edit the initialization parameter file to include the initialization parameters that specify the destinations for the archived redo log files (see "Setting Initialization Parameters for Archive Destinations").
4. Start a new instance and mount, but do not open, the database.

   ```
   STARTUP MOUNT
   ```

   To enable or disable archiving, the database must be mounted but not open.

5. Change the database archiving mode. Then open the database for normal operations.

   ```
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

6. Shut down the database.

   ```
   SHUTDOWN IMMEDIATE
   ```

7. Back up the database.

   Changing the database archiving mode updates the control file. After changing the database archiving mode, you must back up all of your database files and control file. Any previous backup is no longer usable because it was taken in `NOARCHIVELOG` mode.

   See Also:

   Oracle Real Application Clusters Administration and Deployment Guide for more information about switching the archiving mode when using Real Application Clusters

10.3.3 Performing Manual Archiving

For convenience and efficiency, automatic archiving is usually best. However, you can configure your database for manual archiving only.

When you operate your database in manual `ARCHIVELOG` mode, you must archive inactive groups of filled redo log files or your database operation can be temporarily suspended.

To operate your database in manual archiving mode:

1. Follow the procedure described in "Changing the Database Archiving Mode ", but replace the `ALTER DATABASE` statement with the following statement:

   ```
   ALTER DATABASE ARCHIVELOG MANUAL;
   ```

2. Connect to the database as a user with administrator privileges.
3. Ensure that the database is either mounted or open.
4. Use the `ALTER SYSTEM` statement with the `ARCHIVE LOG` clause to manually archive filled redo log files. For example, the following statement archives all unarchived redo log files:

   ```
   ALTER SYSTEM ARCHIVE LOG ALL;
   ```

When you use manual archiving mode, you cannot specify any standby databases in the archiving destinations.

Even when automatic archiving is enabled, you can use manual archiving for such actions as rearchiving an inactive group of filled redo log members to another location. In this case, it is possible for the instance to reuse the redo log group before you have finished manually archiving, and thereby overwrite the files. If this happens, the database writes an error message to the alert log.

Related Topics

- Running a Database in ARCHIVELOG Mode
  When you run a database in `ARCHIVELOG` mode, you enable the archiving of the redo log.

10.3.4 Adjusting the Number of Archiver Processes

The `LOG_ARCHIVE_MAX_PROCESSES` initialization parameter specifies the number of ARC*n* processes that the database initially starts. The default is four processes.

To avoid any run-time overhead of starting additional ARC*n* processes:

- Set the `LOG_ARCHIVE_MAX_PROCESSES` initialization parameter to specify that up to 30 ARC*n* processes be started at instance startup.

The `LOG_ARCHIVE_MAX_PROCESSES` parameter is dynamic, so you can change it using the `ALTER SYSTEM` statement.

The following statement configures the database to start six ARC*n* processes upon startup:

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=6;
```

The statement also has an immediate effect on the currently running instance. It increases or decreases the current number of running ARC*n* processes to six.

10.4 Specifying Archive Destinations

Before you can archive redo logs, you must determine the destination to which you will archive, and familiarize yourself with the various destination states.

The dynamic performance (V$) views, listed in "Viewing Information About the Archived Redo Log", provide all needed archive information.

- Setting Initialization Parameters for Archive Destinations
  You can choose to archive redo logs to a single destination or to multiple destinations.
- Expanding Alternate Destinations with Log Archive Destination Groups
  You can expand the number of alternate archive destinations by using log archive destination groups.
- Understanding Archive Destination Status
  Several variables determine an archive destination’s status.
- Specifying Alternate Destinations
  To specify that a location be an archive destination only in the event of a failure of another destination, you can make it an alternate destination. Both local and remote destinations can be alternates.

10.4.1 Setting Initialization Parameters for Archive Destinations

You can choose to archive redo logs to a single destination or to multiple destinations.

Destinations can be local—within the local file system or an Oracle Automatic Storage Management (Oracle ASM) disk group—or remote (on a standby database). When you archive to multiple destinations, a copy of each filled redo log file is written to each destination. These redundant copies help ensure that archived logs are always available in the event of a failure at one of the destinations.

To archive to only a single destination:

- Specify that destination using the `LOG_ARCHIVE_DEST` initialization parameter.

To archive to multiple destinations:

- Choose to archive to two or more locations using the `LOG_ARCHIVE_DEST_n` initialization parameters, or to archive only to a primary and secondary destination using the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` initialization parameters.

For local destinations, in addition to the local file system or an Oracle ASM disk group, you can archive to the Fast Recovery Area. The database uses the Fast Recovery Area to store and automatically manage disk space for a variety of files related to backup and recovery. See *Oracle Database Backup and Recovery User's Guide* for details about the Fast Recovery Area.

Typically, you determine archive log destinations during database planning, and you set the initialization parameters for archive destinations during database installation. However, you can use the `ALTER SYSTEM` command to dynamically add or change archive destinations after your database is running. Any destination changes that you make take effect at the next log switch (automatic or manual).

The following table summarizes the archive destination alternatives, which are further described in the sections that follow.

Method	Initialization Parameter	Host	Example

1	`LOG_ARCHIVE_DEST_n`
	where:
	n is an integer from 1 to 31. Archive destinations 1 to 10 are available for local or remote locations. Archive destinations 11 to 31 are available for remote locations only.
		Local or remote	`LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc'`
			`LOG_ARCHIVE_DEST_2 = 'LOCATION=/disk2/arc'`
			`LOG_ARCHIVE_DEST_3 = 'SERVICE=standby1'`

2	`LOG_ARCHIVE_DEST` and
	`LOG_ARCHIVE_DUPLEX_DEST`
		Local only	`LOG_ARCHIVE_DEST = '/disk1/arc'`
			`LOG_ARCHIVE_DUPLEX_DEST = '/disk2/arc'`

- Method 1: Using the LOG_ARCHIVE_DEST_n Parameter
  You can use the `LOG_ARCHIVE_DEST_n` initialization parameter to specify different destinations for archived logs.
- Method 2: Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST
  To specify a maximum of two locations, use the `LOG_ARCHIVE_DEST` parameter to specify a primary archive destination and the `LOG_ARCHIVE_DUPLEX_DEST` to specify an optional secondary archive destination.

10.4.1.1 Method 1: Using the LOG_ARCHIVE_DEST_*n* Parameter

You can use the `LOG_ARCHIVE_DEST_n` initialization parameter to specify different destinations for archived logs.

Set the `LOG_ARCHIVE_DEST_n` initialization parameter (where *n* is an integer from 1 to 31) to specify from one to 31. Each numerically suffixed parameter uniquely identifies an individual destination.

You specify the location for `LOG_ARCHIVE_DEST_n` using the keywords explained in the following table:

Keyword	Indicates	Example

`LOCATION`	A local file system location or Oracle ASM disk group	`LOG_ARCHIVE_DEST_n = 'LOCATION=/disk1/arc'`
		`LOG_ARCHIVE_DEST_n = 'LOCATION=+DGROUP1/orcl/arc_1'`

`LOCATION`	The Fast Recovery Area	`LOG_ARCHIVE_DEST_n = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'`

Keyword	Indicates	Example
`SERVICE`	Remote archival through Oracle Net service name.	`LOG_ARCHIVE_DEST_n = 'SERVICE=standby1'`

If you use the `LOCATION` keyword, specify one of the following:

- A valid path name in your operating system's local file system
- An Oracle ASM disk group
- The keyword `USE_DB_RECOVERY_FILE_DEST` to indicate the Fast Recovery Area

If you specify `SERVICE`, supply a net service name that Oracle Net can resolve to a connect descriptor for a standby database. The connect descriptor contains the information necessary for connecting to the remote database.

Perform the following steps to set the destination for archived redo log files using the `LOG_ARCHIVE_DEST_n` initialization parameter:

1. Set the `LOG_ARCHIVE_DEST_n` initialization parameter to specify from one to 31 archiving locations. For example, enter:

   ```
   LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
   LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
   LOG_ARCHIVE_DEST_3 = 'LOCATION = +RECOVERY/orcl/arc_3'
   ```

   If you are archiving to a standby database, then use the `SERVICE` keyword to specify a valid net service name. For example, enter:

   ```
   LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'
   ```

2. (Optional) Set the `LOG_ARCHIVE_FORMAT` initialization parameter, using `%t` to include the thread number as part of the file name, `%s` to include the log sequence number, and `%r` to include the resetlogs ID (a timestamp value represented in ub4). Use capital letters (`%T`, `%S`, and `%R`) to pad the file name to the left with zeroes.

   Note:

   The database requires the specification of resetlogs ID (`%r`) when you include the `LOG_ARCHIVE_FORMAT` parameter. The default for this parameter is operating system dependent.

   The incarnation of a database changes when you open it with the `RESETLOGS` option. Specifying `%r` causes the database to capture the resetlogs ID in the archived redo log file name. See *Oracle Database Backup and Recovery User's Guide* for more information about this method of recovery.

   The following example shows a setting of `LOG_ARCHIVE_FORMAT`:

   ```
   LOG_ARCHIVE_FORMAT = arch_%t_%s_%r.arc
   ```

   This setting will generate archived logs as follows for thread 1; log sequence numbers 100, 101, and 102; resetlogs ID 509210197. The identical resetlogs ID indicates that the files are all from the same database incarnation:

   ```
   /disk1/archive/arch_1_100_509210197.arc,
   /disk1/archive/arch_1_101_509210197.arc,
   ```

```
/disk1/archive/arch_1_102_509210197.arc

/disk2/archive/arch_1_100_509210197.arc,
/disk2/archive/arch_1_101_509210197.arc,
/disk2/archive/arch_1_102_509210197.arc

/disk3/archive/arch_1_100_509210197.arc,
/disk3/archive/arch_1_101_509210197.arc,
/disk3/archive/arch_1_102_509210197.arc
```

The `LOG_ARCHIVE_FORMAT` initialization parameter is ignored in some cases. See *Oracle Database Reference* for more information about this parameter.

10.4.1.2 Method 2: Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST

To specify a maximum of two locations, use the `LOG_ARCHIVE_DEST` parameter to specify a primary archive destination and the `LOG_ARCHIVE_DUPLEX_DEST` to specify an optional secondary archive destination.

All locations must be local. Whenever the database archives a redo log, it archives it to every destination specified by either set of parameters.

Perform the following steps the use method 2:

1. Specify destinations for the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` parameter (you can also specify `LOG_ARCHIVE_DUPLEX_DEST` dynamically using the `ALTER SYSTEM` statement). For example, enter:

   ```
   LOG_ARCHIVE_DEST = '/disk1/archive'
   LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'
   ```

2. Set the `LOG_ARCHIVE_FORMAT` initialization parameter as described in step 2 for method 1.

Note:

If you configure a Fast Recovery Area (by setting the `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE` parameters) and do not specify any local archive destinations, the database automatically selects the Fast Recovery Area as a local archive destination and sets `LOG_ARCHIVE_DEST_1` to `USE_DB_RECOVERY_FILE_DEST`.

WARNING:

You must ensure that there is sufficient disk space at all times for archive log destinations. If the database encounters a disk full error as it attempts to archive a log file, a fatal error occurs and the database stops responding. You can check the alert log for a disk full message.

See Also:

- *Oracle Database Reference* for additional information about the initialization parameters used to control the archiving of redo logs
- *Oracle Data Guard Concepts and Administration* for information about using the `LOG_ARCHIVE_DEST_n` initialization parameter for specifying a standby destination. There are additional keywords that can be specified with this initialization parameter that are not discussed in this book.
- *Oracle Database Net Services Administrator's Guide* for a discussion of net service names and connect descriptors.
- *Oracle Database Backup and Recovery User's Guide* for information about the Fast Recovery Area

10.4.2 Expanding Alternate Destinations with Log Archive Destination Groups

You can expand the number of alternate archive destinations by using log archive destination groups.

- About Log Archive Destination Groups
  A log archive destination group specifies multiple archive destinations, and the destinations in the group can be prioritized. You can specify multiple groups to expand the number of possible archive destinations for your database.
- Specifying Log Archive Destination Groups
  Use the `GROUP` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter to specify log archive destination groups.

10.4.2.1 About Log Archive Destination Groups

A log archive destination group specifies multiple archive destinations, and the destinations in the group can be prioritized. You can specify multiple groups to expand the number of possible archive destinations for your database.

To specify a log archive destination group, use the `GROUP` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter. There can be up to 30 log archive destinations included in a group. One member of each group is active, and the others are available for use in the event of a failure of the active destination. If the active destination becomes inactive, then Oracle Database switches to an available destination as long as one or more are available in the group. You can indicate which destinations to use first by prioritizing the destinations with the `PRIORITY` attribute.

A log archive destination group is referenced by a group number, which is specified when the group is created. There can be up to eight groups. To specify where to archive the redo data within a group, all of the log archive destinations must specify the `SERVICE` attribute.

To prioritize the destinations in a group, set the `PRIORITY` attribute for a destination to an integer in the range of 1 through 8. The lower number indicates the higher priority. The priority determines which destination within a group to make active when the database is mounted or when the active destination fails. For example, a `PRIORITY` value of `2` is higher priority than a `PRIORITY` value of `7`. Therefore, if the currently active destination with the `PRIORITY` value of `1` in the group becomes inactive, then the destination with the PRIORITY value of 2 is used before the destination with the PRIORITY value of 7. If the PRIORITY attribute is not set for a destination, then the default value is 1.

The priority is also considered when a previously failed destination becomes available. If an active destination fails, and Oracle Database switches to a destination with a lower priority, then Oracle Database switches back to the destination with higher priority when it becomes available again. For example, if an active destination with priority 1 becomes inactive, and Oracle Database switches to a destination with priority 2, then Oracle Database switches back to the destination with priority 1 when it becomes available again, even if the priority 2 destination did not fail.

However, more than one destination assigned to the same group can have the same priority. For example, there can be three destinations with priority 1. In such a group, a failure of the active destination results in a switch to another member with the same priority. In this case, there is no switch back to the original destination when it becomes available again because both destinations have the same priority. If the second destination fails after the first destination has become available again, then the database will switch to the first destination or to another destination in the group with the same priority.

See Also:

Oracle Data Guard Concepts and Administration

10.4.2.2 Specifying Log Archive Destination Groups

Use the GROUP attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to specify log archive destination groups.

You can create up to eight log archive destination groups, and each group can have up to 30 destinations specified.

To specify log archive destination groups, the database must be running in ARCHIVELOG mode.

- Set the LOG_ARCHIVE_DEST_*n* initialization parameter, and include the GROUP attribute to specify log archive destination groups.

  Optionally, include the PRIORITY attribute to specify which log archive destination within a group to make active when the system is started or when a destination fails.

Example 10-1   Specifying Two Log Archive Destination Groups

This example specifies two log archive destination groups (1 and 2). Each group has three log archive destinations specified.

```
LOG_ARCHIVE_DEST_1 = 'SERVICE=SITEa VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
GROUP=1'
LOG_ARCHIVE_DEST_2 = 'SERVICE=SITEb VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
GROUP=1'
LOG_ARCHIVE_DEST_3 = 'SERVICE=SITEc VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
GROUP=1'

LOG_ARCHIVE_DEST_4 = 'SERVICE=SITE1 VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
GROUP=2'
LOG_ARCHIVE_DEST_5 = 'SERVICE=SITE2 VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
GROUP=2'
```

```
LOG_ARCHIVE_DEST_6 = 'SERVICE=SITE3 VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
GROUP=2'
```

Example 10-2 Specifying Priority Within a Log Archive Destination Group

This example specifies different priority levels for destinations within a single log archive destination group. Specifically, destination 1 and 2 are both at priority level 1, destination 3 is at priority level 2, and destination 4 is at priority level 3.

```
LOG_ARCHIVE_DEST_1 = 'SERVICE=SITE1 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'
LOG_ARCHIVE_DEST_2 = 'SERVICE=SITE2 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'
LOG_ARCHIVE_DEST_3 = 'SERVICE=SITE3 ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=2'
LOG_ARCHIVE_DEST_4 = 'SERVICE=SITE4 ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=3'
```

In this example, sites 1, 2 and 3 could be Oracle Data Guard far sync instances that only forward the redo, and site 4 is the actual remote standby database. Alternatively, sites 1, 2, 3, and 4 could all be standby databases which are configured to cascade the redo to the other sites when they are the active destination.

The following priority rules are followed:

- The default active destination can be destination 1 or destination 2 because both are at priority level 1.
- If destination 1 is active but then becomes unavailable, then Oracle Database switches to destination 2. Similarly, if destination 2 is active but then becomes unavailable, then Oracle Database switches to destination 1. When either destination 1 or 2 is available, one of them is used.
- If both destination 1 and destination 2 become unavailable, then destination 3 is used.
- If, when destination 3 is active, destination 1 or destination 2 becomes available, Oracle Database switches to the available priority 1 destination.
- If destination 1, 2, and 3 all become unavailable, then destination 4 is used.
- If, when destination 4 is active, destination 1, 2, or 3 becomes available, Oracle Database switches to the available priority 1 destination first and then to the available priority 2 destination.

See Also:

Oracle Data Guard Concepts and Administration

10.4.3 Understanding Archive Destination Status

Several variables determine an archive destination’s status.

Each archive destination has the following variable characteristics that determine its status:

- **Valid/Invalid**: indicates whether the disk location or service name information is specified and valid

- **Enabled/Disabled**: indicates the availability state of the location and whether the database can use the destination
- **Active/Inactive**: indicates whether there was a problem accessing the destination

Several combinations of these characteristics are possible. To obtain the current status and other information about each destination for an instance, query the `V$ARCHIVE_DEST` view.

The `LOG_ARCHIVE_DEST_STATE_n` (where *n* is an integer from 1 to 31) initialization parameter lets you control the availability state of the specified destination (*n*).

- `ENABLE` indicates that the database can use the destination.
- `DEFER` indicates that the location is temporarily disabled.
- `ALTERNATE` indicates that the destination is an alternate. The availability state of an alternate destination is `DEFER`. If its parent destination fails, the availability state of the alternate becomes `ENABLE`. `ALTERNATE` cannot be specified for destinations `LOG_ARCHIVE_DEST_11` to `LOG_ARCHIVE_DEST_31`.

10.4.4 Specifying Alternate Destinations

To specify that a location be an archive destination only in the event of a failure of another destination, you can make it an alternate destination. Both local and remote destinations can be alternates.

The following example makes `LOG_ARCHIVE_DEST_4` an alternate for `LOG_ARCHIVE_DEST_3`:

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_4 = 'LOCATION=/disk4/arch';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_3 = 'LOCATION=/disk3/arch MAX_FAILURE=1
   ALTERNATE=LOG_ARCHIVE_DEST_4';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=ALTERNATE;

SQL> SELECT dest_name, status, destination FROM v$archive_dest;

DEST_NAME               STATUS    DESTINATION
----------------------- --------- ----------------------------------------------
LOG_ARCHIVE_DEST_1      VALID     /disk1/arch
LOG_ARCHIVE_DEST_2      VALID     /disk2/arch
LOG_ARCHIVE_DEST_3      VALID     /disk3/arch
LOG_ARCHIVE_DEST_4      ALTERNATE /disk4/arch
```

10.5 About Log Transmission Modes

The two modes of transmitting archived logs to their destination are **normal archiving transmission** and **standby transmission** mode. Normal transmission involves transmitting files to a local disk. Standby transmission involves transmitting files through a network to either a local or remote standby database.

- Normal Transmission Mode
  In normal transmission mode, the archiving destination is another disk drive of the database server.
- Standby Transmission Mode
  In standby transmission mode, the archiving destination is either a local or remote standby database.

10.5.1 Normal Transmission Mode

In normal transmission mode, the archiving destination is another disk drive of the database server.

In this configuration archiving does not contend with other files required by the instance and can complete more quickly. Specify the destination with either the `LOG_ARCHIVE_DEST_n` or `LOG_ARCHIVE_DEST` parameters.

10.5.2 Standby Transmission Mode

In standby transmission mode, the archiving destination is either a local or remote standby database.

Note:

You can maintain a standby database on a local disk, but Oracle strongly encourages you to maximize disaster protection by maintaining your standby database at a remote site.

See Also:

- *Oracle Data Guard Concepts and Administration*
- *Oracle Database Net Services Administrator's Guide* for information about connecting to a remote database using a service name

10.6 Managing Archive Destination Failure

Sometimes archive destinations can fail, causing problems when you operate in automatic archiving mode. Oracle Database provides procedures to help you minimize the problems associated with destination failure.

- Specifying the Minimum Number of Successful Destinations
  The optional initialization parameter `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` determines the minimum number of destinations to which the database must successfully archive a redo log group before it can reuse online log files. The default value is 1. Valid values for *n* are 1 to 2 if you are using duplexing, or 1 to 31 if you are multiplexing.
- Rearchiving to a Failed Destination
  Use the `REOPEN` attribute of the `LOG_ARCHIVE_DEST_n` parameter to specify whether and when ARC*n* should attempt to rearchive to a failed destination following an error. `REOPEN` applies to all errors, not just `OPEN` errors.

10.6.1 Specifying the Minimum Number of Successful Destinations

The optional initialization parameter `LOG_ARCHIVE_MIN_SUCCEED_DEST=`*n* determines the minimum number of destinations to which the database must successfully archive a redo log group before it can reuse online log files. The default value is 1. Valid values for *n* are 1 to 2 if you are using duplexing, or 1 to 31 if you are multiplexing.

- Specifying Mandatory and Optional Destinations
  The `LOG_ARCHIVE_DEST_`*n* initialization parameter lets you specify whether a destination is `OPTIONAL` (the default) or `MANDATORY`.
- Specifying the Number of Successful Destinations: Scenarios
  You can see the relationship between the `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_MIN_SUCCEED_DEST` initialization parameters most easily through sample scenarios.

10.6.1.1 Specifying Mandatory and Optional Destinations

The `LOG_ARCHIVE_DEST_`*n* initialization parameter lets you specify whether a destination is `OPTIONAL` (the default) or `MANDATORY`.

- Set the destination as `OPTIONAL` (the default) or `MANDATORY` in the `LOG_ARCHIVE_DEST_`*n* initialization parameter.

The `LOG_ARCHIVE_MIN_SUCCEED_DEST=`*n* parameter uses all `MANDATORY` destinations plus some number of non-standby `OPTIONAL` destinations to determine whether LGWR can overwrite the online log. The following rules apply:

- Omitting the `MANDATORY` attribute for a destination is the same as specifying `OPTIONAL`.
- You must have at least one local destination, which you can declare `OPTIONAL` or `MANDATORY`.
- The `MANDATORY` attribute can only be specified for destinations `LOG_ARCHIVE_DEST_1` through `LOG_ARCHIVE_DEST_10`.
- When you specify a value for `LOG_ARCHIVE_MIN_SUCCEED_DEST=`*n*, Oracle Database will treat at least one local destination as `MANDATORY`, because the minimum value for `LOG_ARCHIVE_MIN_SUCCEED_DEST` is 1.
- The `LOG_ARCHIVE_MIN_SUCCEED_DEST` value cannot be greater than the number of destinations, nor can it be greater than the number of `MANDATORY` destinations plus the number of `OPTIONAL` local destinations.
- If you `DEFER` a `MANDATORY` destination, and the database overwrites the online log without transferring the archived log to the standby site, then you must transfer the log to the standby manually.

If you are duplexing the archived logs, you can establish which destinations are mandatory or optional by using the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` parameters. The following rules apply:

- Any destination declared by `LOG_ARCHIVE_DEST` is mandatory.
- Any destination declared by `LOG_ARCHIVE_DUPLEX_DEST` is optional if `LOG_ARCHIVE_MIN_SUCCEED_DEST = 1` and mandatory if `LOG_ARCHIVE_MIN_SUCCEED_DEST = 2`.

10.6.1.2 Specifying the Number of Successful Destinations: Scenarios

You can see the relationship between the `LOG_ARCHIVE_DEST_n` and `LOG_ARCHIVE_MIN_SUCCEED_DEST` initialization parameters most easily through sample scenarios.

- Scenario for Archiving to Optional Local Destinations
  In this scenario, you archive to three local destinations, each of which you declare as `OPTIONAL`.
- Scenario for Archiving to Both Mandatory and Optional Destinations
  In this scenario, you archive to `MANDATORY` and `OPTIONAL` local destinations.

10.6.1.2.1 Scenario for Archiving to Optional Local Destinations

In this scenario, you archive to three local destinations, each of which you declare as `OPTIONAL`.

Table 10-1 illustrates the possible values for `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` in this case.

Table 10-1 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 1

Value	Meaning
1	The database can reuse log files only if at least one of the `OPTIONAL` destinations succeeds.
2	The database can reuse log files only if at least two of the `OPTIONAL` destinations succeed.
3	The database can reuse log files only if all of the `OPTIONAL` destinations succeed.
4 or greater	`ERROR`: The value is greater than the number of destinations.

This scenario shows that even though you do not explicitly set any of your destinations to `MANDATORY` using the `LOG_ARCHIVE_DEST_n` parameter, the database must successfully archive to one or more of these locations when `LOG_ARCHIVE_MIN_SUCCEED_DEST` is set to 1, 2, or 3.

10.6.1.2.2 Scenario for Archiving to Both Mandatory and Optional Destinations

In this scenario, you archive to `MANDATORY` and `OPTIONAL` local destinations.

Consider a case in which:

- You specify two `MANDATORY` destinations.
- You specify two `OPTIONAL` destinations.
- No destination is a standby database.

Table 10-2 shows the possible values for `LOG_ARCHIVE_MIN_SUCCEED_DEST=n`.

Table 10-2 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2

Value	Meaning
1	The database ignores the value and uses the number of `MANDATORY` destinations (in this example, 2).

Table 10-2 (Cont.) LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2

Value	Meaning
2	The database can reuse log files even if no `OPTIONAL` destination succeeds.
3	The database can reuse logs only if at least one `OPTIONAL` destination succeeds.
4	The database can reuse logs only if both `OPTIONAL` destinations succeed.
5 or greater	`ERROR`: The value is greater than the number of destinations.

This case shows that the database must archive to the destinations you specify as `MANDATORY`, regardless of whether you set `LOG_ARCHIVE_MIN_SUCCEED_DEST` to archive to a smaller number of destinations.

10.6.2 Rearchiving to a Failed Destination

Use the `REOPEN` attribute of the `LOG_ARCHIVE_DEST_n` parameter to specify whether and when ARC*n* should attempt to rearchive to a failed destination following an error. `REOPEN` applies to all errors, not just `OPEN` errors.

`REOPEN=n` sets the minimum number of seconds before ARC*n* should try to reopen a failed destination. The default value for *n* is 300 seconds. A value of 0 is the same as turning off the `REOPEN` attribute; ARC*n* will not attempt to archive after a failure. If you do not specify the `REOPEN` keyword, ARC*n* will never reopen a destination following an error.

You cannot use `REOPEN` to specify the number of attempts ARC*n* should make to reconnect and transfer archived logs. The `REOPEN` attempt either succeeds or fails.

When you specify `REOPEN` for an `OPTIONAL` destination, the database can overwrite online logs if there is an error. If you specify `REOPEN` for a `MANDATORY` destination, the database stalls the production database when it cannot successfully archive. In this situation, consider the following options:

- Archive manually to the failed destination.
- Change the destination by deferring the destination, specifying the destination as optional, or changing the service.
- Drop the destination.

When using the `REOPEN` keyword, note the following:

- ARC*n* reopens a destination only when *starting* an archive operation from the beginning of the log file, never *during* a current operation. ARC*n* always retries the log copy from the beginning.
- If you specified `REOPEN`, either with a specified time the default, ARC*n* checks to see whether the time of the recorded error plus the `REOPEN` interval is less than the current time. If it is, ARC*n* retries the log copy.
- The `REOPEN` clause successfully affects the `ACTIVE=TRUE` destination state. The `VALID` and `ENABLED` states are not changed.

  Something wrong here. A destination can be inactive, or valid, or disabled. There is no ACTIVE status. So I think maybe it should say, "The REOPEN clause sets the destination status to VALID" ...? DL

10.7 Controlling Trace Output Generated by the Archivelog Process

Background processes always write to a trace file when appropriate. In the case of the archivelog process, you can control the output that is generated to the trace file.

To control the output that is generated to the trace file for the archivelog process:

- Set the `LOG_ARCHIVE_TRACE` initialization parameter to specify a **trace level**, such as 0, 1, 2, 4, 8, and so on.

You can combine tracing levels by specifying a value equal to the sum of the individual levels that you would like to trace. For example, setting `LOG_ARCHIVE_TRACE=12` will generate trace level 8 and 4 output. You can set different values for the primary and any standby database.

The default value for the `LOG_ARCHIVE_TRACE` parameter is 0. At this level, the archivelog process generates appropriate alert and trace entries for error conditions.

You can change the value of this parameter dynamically using the `ALTER SYSTEM` statement. For example:

```
ALTER SYSTEM SET LOG_ARCHIVE_TRACE=12;
```

Changes initiated in this manner will take effect at the start of the next archiving operation.

See Also:

- "Monitoring Errors with Trace Files and the Alert Log"
- *Oracle Database Reference* for more information about the `LOG_ARCHIVE_TRACE` initialization parameter, including descriptions of the valid values for this parameter
- *Oracle Data Guard Concepts and Administration* for information about using this parameter with a standby database

10.8 Viewing Information About the Archived Redo Log

You can display information about the archived redo log using dynamic performance views or the `ARCHIVE LOG LIST` command.

- Archived Redo Log Files Views
  You can query a set of dynamic performance views for information about archived redo log files.
- Using the ARCHIVE LOG LIST Command
  The SQL*Plus command `ARCHIVE LOG LIST` displays archiving information for the connected instance.

10.8.1 Archived Redo Log Files Views

You can query a set of dynamic performance views for information about archived redo log files.

Several dynamic performance views contain useful information about archived redo log files, as summarized in the following table.

Dynamic Performance View	Description
`V$DATABASE`	Shows if the database is in `ARCHIVELOG` or `NOARCHIVELOG` mode and if `MANUAL` (archiving mode) has been specified.
`V$ARCHIVED_LOG`	Displays historical archived log information from the control file. If you use a recovery catalog, the `RC_ARCHIVED_LOG` view contains similar information.
`V$ARCHIVE_DEST`	Describes the current instance, all archive destinations, and the current value, mode, and status of these destinations.
`V$ARCHIVE_PROCESSES`	Displays information about the state of the various archive processes for an instance.
`V$BACKUP_REDOLOG`	Contains information about any backups of archived logs. If you use a recovery catalog, the `RC_BACKUP_REDOLOG` contains similar information.
`V$LOG`	Displays all redo log groups for the database and indicates which need to be archived.
`V$LOG_HISTORY`	Contains log history information such as which logs have been archived and the SCN range for each archived log.

For example, the following query displays which redo log group requires archiving:

```
SELECT GROUP#, ARCHIVED
   FROM SYS.V$LOG;

GROUP#     ARC
--------   ---
       1   YES
       2   NO
```

To see the current archiving mode, query the `V$DATABASE` view:

```
SELECT LOG_MODE FROM SYS.V$DATABASE;

LOG_MODE
------------
NOARCHIVELOG
```

10.8.2 Using the ARCHIVE LOG LIST Command

The SQL*Plus command `ARCHIVE LOG LIST` displays archiving information for the connected instance.

For example:

```
SQL> ARCHIVE LOG LIST

Database log mode              Archive Mode
Automatic archival             Enabled
```

```
Archive destination            D:\oracle\oradata\IDDB2\archive
Oldest online log sequence     11160
Next log sequence to archive   11163
Current log sequence           11163
```

This display tells you all the necessary information regarding the archived redo log settings for the current instance:

- The database is currently operating in `ARCHIVELOG` mode.
- Automatic archiving is enabled.
- The archived redo log destination is D:\oracle\oradata\IDDB2\archive.
- The oldest filled redo log group has a sequence number of 11160.
- The next filled redo log group to archive has a sequence number of 11163.
- The current redo log file has a sequence number of 11163.

See Also:

*SQL*Plus User's Guide and Reference* for more information on the `ARCHIVE LOG LIST` command

11 Managing Tablespaces

A tablespace is a database storage unit that groups related logical structures together. The database data files are stored in tablespaces.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- Guidelines for Managing Tablespaces
  You can follow guidelines for working with tablespaces.
- Creating Tablespaces
  You create a tablespace to group related logical structures, such as tables and indexes, together. The database data files are stored in tablespaces.
- Consider Storing Tablespaces in the In-Memory Column Store
  You can enable a tablespace for the In-Memory Column Store during tablespace creation or by altering a tablespace. When this enable a tablespace for the In-Memory Column Store, all tables in the tablespace are enabled for the In-Memory Column Store by default.
- Specifying Nonstandard Block Sizes for Tablespaces
  You can create tablespaces with block sizes different from the standard database block size, which is specified by the `DB_BLOCK_SIZE` initialization parameter. This feature lets you transport tablespaces with unlike block sizes between databases.
- Controlling the Writing of Redo Records
  For some database operations, you can control whether the database generates redo records.
- Altering Tablespace Availability
  You can take an online tablespace offline so that it is temporarily unavailable for general use. The rest of the database remains open and available for users to access data. Conversely, you can bring an offline tablespace online to make the schema objects within the tablespace available to database users. The database must be open to alter the availability of a tablespace.
- Using Read-Only Tablespaces
  A tablespace can be put into read-only mode. This prevents any data stored in it from being updated.
- Altering and Maintaining Tablespaces
  You can alter and maintain tablespaces by performing such tasks as adding data files and temp files to them.

- Renaming Tablespaces
  Using the `RENAME TO` clause of the `ALTER TABLESPACE`, you can rename a permanent or temporary tablespace.
- Dropping Tablespaces
  You can drop a tablespace and its contents (the segments contained in the tablespace) from the database if the tablespace and its contents are no longer required.
- Managing Lost Write Protection with Shadow Tablespaces
  A data block lost write occurs when an I/O subsystem acknowledges the completion of the block write, but the write did not occur in the persistent storage. Shadow lost write protection can protect against lost writes.
- Managing the SYSAUX Tablespace
  The `SYSAUX` tablespace was installed as an auxiliary tablespace to the `SYSTEM` tablespace when you created your database. Some database components that formerly created and used separate tablespaces now occupy the `SYSAUX` tablespace.
- Correcting Problems with Locally Managed Tablespaces
  Oracle Database includes aids for correcting problems with locally managed tablespaces.
- Migrating the SYSTEM Tablespace to a Locally Managed Tablespace
  Use the `DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL` procedure to migrate the `SYSTEM` tablespace from dictionary-managed to locally managed.
- Viewing Information About Tablespaces
  Oracle Database includes data dictionary views that you can query for information about tablespaces.

See Also:

- *Oracle Database Concepts*
- Using Oracle Managed Files for information about creating data files and temp files that are both created and managed by the Oracle Database server
- "Transporting Tablespaces Between Databases"

11.1 Guidelines for Managing Tablespaces

You can follow guidelines for working with tablespaces.

- Use Multiple Tablespaces
  Using multiple tablespaces allows you more flexibility in performing database operations.
- Assign Tablespace Quotas to Users
  Grant to users who will be creating tables, clusters, materialized views, indexes, and other objects the privilege to create the object and a **quota** (space allowance or limit) in the tablespace intended to hold the object segment.

11.1.1 Use Multiple Tablespaces

Using multiple tablespaces allows you more flexibility in performing database operations.

When a database has multiple tablespaces, you can:

- Separate user data from data dictionary data to reduce I/O contention.

- Separate data of one application from the data of another to prevent multiple applications from being affected if a tablespace must be taken offline.
- Store the data files of different tablespaces on different disk drives to reduce I/O contention.
- Take individual tablespaces offline while others remain online, providing better overall availability.
- Optimizing tablespace use by reserving a tablespace for a particular type of database use, such as high update activity, read-only activity, or temporary segment storage.
- Back up individual tablespaces.

Some operating systems set a limit on the number of files that can be open simultaneously. Such limits can affect the number of tablespaces that can be simultaneously online. To avoid exceeding your operating system limit, plan your tablespaces efficiently. Create only enough tablespaces to fulfill your needs, and create these tablespaces with as few files as possible. If you must increase the size of a tablespace, then add one or two large data files, or create data files with autoextension enabled, rather than creating many small data files.

Review your data in light of these factors and decide how many tablespaces you need for your database design.

11.1.2 Assign Tablespace Quotas to Users

Grant to users who will be creating tables, clusters, materialized views, indexes, and other objects the privilege to create the object and a **quota** (space allowance or limit) in the tablespace intended to hold the object segment.

Note:

For PL/SQL objects such as packages, procedures, and functions, users only need the privileges to create the objects. No explicit tablespace quota is required to create these PL/SQL objects.

See Also:

Oracle Database Security Guide for information about creating users and assigning tablespace quotas.

11.2 Creating Tablespaces

You create a tablespace to group related logical structures, such as tables and indexes, together. The database data files are stored in tablespaces.

- About Creating Tablespaces
  To create a new tablespace, use the SQL statement `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE`. You must have the `CREATE TABLESPACE` system privilege to create a tablespace.

- Locally Managed Tablespaces
  A locally managed tablespace uses a bitmap stored in each data file to manage the extents.
- Bigfile Tablespaces
  Bigfile tablespaces can increase the storage capacity of a database and reduce the burden of managing many data files and temp files.
- Tablespaces with Default Compression Attributes
  When you create a tablespace, you can specify that all tables and indexes, or their partitions, created in a tablespace are compressed by default.
- Encrypted Tablespaces
  You can encrypt any permanent tablespace to protect sensitive data.
- Temporary Tablespaces
  Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory. These tablespaces also improve the efficiency of space management operations during sorts.
- Temporary Tablespace Groups
  A temporary tablespace group is a tablespace group that is assigned as the default temporary tablespace for the database.

11.2.1 About Creating Tablespaces

To create a new tablespace, use the SQL statement `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE`. You must have the `CREATE TABLESPACE` system privilege to create a tablespace.

Before you can create a tablespace, you must create a database to contain it. The primary tablespace in any database is the `SYSTEM` tablespace, which contains information basic to the functioning of the database server, such as the data dictionary and the system rollback segment. The `SYSTEM` tablespace is the first tablespace created at database creation. It is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the `SYSTEM` tablespace or take it offline.

The `SYSAUX` tablespace, which acts as an auxiliary tablespace to the `SYSTEM` tablespace, is also always created when you create a database. It contains the schemas used by various Oracle products and features, so that those products do not require their own tablespaces. As for the `SYSTEM` tablespace, management of the `SYSAUX` tablespace requires a higher level of security and you cannot rename or drop it. The management of the `SYSAUX` tablespace is discussed separately in "Managing the SYSAUX Tablespace".

The steps for creating tablespaces vary by operating system, but the first step is always to use your operating system to create a directory structure in which your data files will be allocated. On most operating systems, you specify the size and fully specified file names of data files when you create a new tablespace or alter an existing tablespace by adding data files. Whether you are creating a new tablespace or modifying an existing one, the database automatically allocates and formats the data files as specified.

You can also use the `CREATE UNDO TABLESPACE` statement to create a special type of tablespace called an **undo tablespace**, which is specifically designed to contain undo records. These are records generated by the database that are used to roll back, or undo, changes to the database for recovery, read consistency, or as requested by a `ROLLBACK` statement. Creating and managing undo tablespaces is the subject of Managing Undo .

You can use the `ALTER TABLESPACE` or `ALTER DATABASE` statements to alter the tablespace. You must have the `ALTER TABLESPACE` or `ALTER DATABASE` system privilege, correspondingly.

See Also:

- *Oracle Multitenant Administrator's Guide* and your Oracle Database installation documentation for your operating system for information about tablespaces that are created at database creation
- *Oracle Database SQL Language Reference* for more information about the syntax and semantics of the `CREATE TABLESPACE`, `CREATE TEMPORARY TABLESPACE`, `ALTER TABLESPACE`, and `ALTER DATABASE` statements.
- *Oracle Multitenant Administrator's Guide* for information about initialization parameters necessary to create tablespaces with nonstandard block sizes

11.2.2 Locally Managed Tablespaces

A locally managed tablespace uses a bitmap stored in each data file to manage the extents.

- About Locally Managed Tablespaces
  Locally managed tablespaces track all extent information in the tablespace itself by using bitmaps.
- Creating a Locally Managed Tablespace
  Create a locally managed tablespace by specifying `LOCAL` in the `EXTENT MANAGEMENT` clause of the `CREATE TABLESPACE` statement.
- Specifying Segment Space Management in Locally Managed Tablespaces
  In a locally managed tablespace, there are two methods that Oracle Database can use to manage segment space: automatic and manual.

11.2.2.1 About Locally Managed Tablespaces

Locally managed tablespaces track all extent information in the tablespace itself by using bitmaps.

Locally managed tablespaces provide the following benefits:

- Fast, concurrent space operations. Space allocations and deallocations modify locally managed resources (bitmaps stored in header files).
- Enhanced performance
- Readable standby databases are allowed, because locally managed temporary tablespaces do not generate any undo or redo.
- Space allocation is simplified, because when the `AUTOALLOCATE` clause is specified, the database automatically selects the appropriate extent size.
- User reliance on the data dictionary is reduced, because the necessary information is stored in file headers and bitmap blocks.
- Coalescing free extents is unnecessary for locally managed tablespaces.

All tablespaces, including the `SYSTEM` tablespace, can be locally managed.

The `DBMS_SPACE_ADMIN` package provides maintenance procedures for locally managed tablespaces.

See Also:

- *Oracle Multitenant Administrator's Guide*, "Migrating the SYSTEM Tablespace to a Locally Managed Tablespace", and "Diagnosing and Repairing Locally Managed Tablespace Problems"
- "Bigfile Tablespaces" for information about creating another type of locally managed tablespace that contains only a single data file or temp file.
- *Oracle Database PL/SQL Packages and Types Reference* for information on the `DBMS_SPACE_ADMIN` package

11.2.2.2 Creating a Locally Managed Tablespace

Create a locally managed tablespace by specifying `LOCAL` in the `EXTENT MANAGEMENT` clause of the `CREATE TABLESPACE` statement.

This is the default for new permanent tablespaces, but you must specify the `EXTENT MANAGEMENT LOCAL` clause to specify either the `AUTOALLOCATE` clause or the `UNIFORM` clause. You can have the database manage extents for you automatically with the `AUTOALLOCATE` clause (the default), or you can specify that the tablespace is managed with uniform extents of a specific size (`UNIFORM`).

If you expect the tablespace to contain objects of varying sizes requiring many extents with different extent sizes, then `AUTOALLOCATE` is the best choice. `AUTOALLOCATE` is also a good choice if it is not important for you to have a lot of control over space allocation and deallocation, because it simplifies tablespace management. Some space may be wasted with this setting, but the benefit of having Oracle Database manage your space most likely outweighs this drawback.

If you want exact control over unused space, and you can predict exactly the space to be allocated for an object or objects and the number and size of extents, then `UNIFORM` is a good choice. This setting ensures that you will never have unusable space in your tablespace.

When you do not explicitly specify the type of extent management, Oracle Database determines extent management as follows:

- If the `CREATE TABLESPACE` statement omits the `DEFAULT` storage clause, then the database creates a locally managed autoallocated tablespace.
- If the `CREATE TABLESPACE` statement includes a `DEFAULT` storage clause, then the database considers the following:
  - If you specified the `MINIMUM EXTENT` clause, the database evaluates whether the values of `MINIMUM EXTENT`, `INITIAL`, and `NEXT` are equal and the value of `PCTINCREASE` is 0. If so, the database creates a locally managed uniform tablespace with extent size = `INITIAL`. If the `MINIMUM EXTENT`, `INITIAL`, and `NEXT` parameters are not equal, or if `PCTINCREASE` is not 0, then the database ignores any extent storage parameters you may specify and creates a locally managed, autoallocated tablespace.
  - If you did not specify `MINIMUM EXTENT` clause, then the database evaluates only whether the storage values of `INITIAL` and `NEXT` are equal and `PCTINCREASE` is 0. If so, the tablespace is locally managed and uniform. Otherwise, the tablespace is locally managed and autoallocated.

For example, the following statement creates a locally managed tablespace named `lmtbsb` and specifies `AUTOALLOCATE`:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

`AUTOALLOCATE` causes the tablespace to be system managed with a minimum extent size of 64K.

The alternative to `AUTOALLOCATE` is `UNIFORM`. which specifies that the tablespace is managed with extents of uniform size. You can specify that size in the `SIZE` clause of `UNIFORM`. If you omit `SIZE`, then the default size is 1M.

The following example creates a tablespace with uniform 128K extents. (In a database with 2K blocks, each extent would be equivalent to 64 database blocks). Each 128K extent is represented by a bit in the extent bitmap for this file.

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

You cannot specify the `DEFAULT` storage clause, `MINIMUM EXTENT`, or `TEMPORARY` when you explicitly specify `EXTENT MANAGEMENT LOCAL`. To create a temporary locally managed tablespace, use the `CREATE TEMPORARY TABLESPACE` statement.

Note:

When you allocate a data file for a locally managed tablespace, you should allow space for metadata used for space management (the extent bitmap or space header segment) which are part of user space. For example, if you specify the `UNIFORM` clause in the extent management clause but you omit the `SIZE` parameter, then the default extent size is 1MB. In that case, the size specified for the data file must be larger (at least one block plus space for the bitmap) than 1MB.

11.2.2.3 Specifying Segment Space Management in Locally Managed Tablespaces

In a locally managed tablespace, there are two methods that Oracle Database can use to manage segment space: automatic and manual.

Manual segment space management uses linked lists called "freelists" to manage free space in the segment, while automatic segment space management uses bitmaps. Automatic segment space management is the more efficient method, and is the default for all new permanent, locally managed tablespaces.

Automatic segment space management delivers better space utilization than manual segment space management. It is also self-tuning, in that it scales with increasing number of users or instances. In an Oracle Real Application Clusters environment, automatic segment space management allows for a dynamic affinity of space to instances. In addition, for many standard workloads, application performance with automatic segment space management is better than the performance of a well-tuned application using manual segment space management.

Although automatic segment space management is the default for all new permanent, locally managed tablespaces, you can explicitly enable it with the `SEGMENT SPACE MANAGEMENT AUTO` clause.

For example, the following statement creates tablespace `lmtbsb` with automatic segment space management:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO;
```

The `SEGMENT SPACE MANAGEMENT MANUAL` clause disables automatic segment space management.

The segment space management that you specify at tablespace creation time applies to all segments subsequently created in the tablespace. You cannot change the segment space management mode of a tablespace.

Note:

- If you set extent management to `LOCAL UNIFORM`, then you must ensure that each extent contains at least 5 database blocks.
- If you set extent management to `LOCAL AUTOALLOCATE`, and if the database block size is 16K or greater, then Oracle manages segment space by creating extents with a minimum size of 5 blocks rounded up to 64K.
- You cannot specify automatic segment space management for the `SYSTEM` tablespace.

Locally managed tablespaces using automatic segment space management can be created as single-file or bigfile tablespaces, as described in "Bigfile Tablespaces".

11.2.3 Bigfile Tablespaces

Bigfile tablespaces can increase the storage capacity of a database and reduce the burden of managing many data files and temp files.

- About Bigfile Tablespaces
  A **bigfile tablespace** is a tablespace with a single, but potentially very large (up to 4G blocks) data file. Traditional smallfile tablespaces, in contrast, can contain multiple data files, but the files cannot be as large.
- Creating a Bigfile Tablespace
  To create a bigfile tablespace, specify the `BIGFILE` keyword of the `CREATE TABLESPACE` statement (`CREATE BIGFILE TABLESPACE` ...).
- Identifying a Bigfile Tablespace
  You can query a set of data dictionary views for information about bigfile tablespaces.

11.2.3.1 About Bigfile Tablespaces

A **bigfile tablespace** is a tablespace with a single, but potentially very large (up to 4G blocks) data file. Traditional smallfile tablespaces, in contrast, can contain multiple data files, but the files cannot be as large.

The benefits of bigfile tablespaces are the following:

- A bigfile tablespace with 8K blocks can contain a 32 terabyte data file. A bigfile tablespace with 32K blocks can contain a 128 terabyte data file. The maximum number of data files in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.

- Bigfile tablespaces can reduce the number of data files needed for a database. An additional benefit is that the `DB_FILES` initialization parameter and `MAXDATAFILES` parameter of the `CREATE DATABASE` and `CREATE CONTROLFILE` statements can be adjusted to reduce the amount of SGA space required for data file information and the size of the control file.
- Bigfile tablespaces simplify database management by providing data file transparency. SQL syntax for the `ALTER TABLESPACE` statement lets you perform operations on tablespaces, rather than the underlying individual data files.

Bigfile tablespaces are supported only for locally managed tablespaces with automatic segment space management, with three exceptions: locally managed undo tablespaces, temporary tablespaces, and the `SYSTEM` tablespace.

Note:

- Bigfile tablespaces are intended to be used with Automatic Storage Management (Oracle ASM) or other logical volume managers that supports striping or RAID, and dynamically extensible logical volumes.
- Avoid creating bigfile tablespaces on a system that does not support striping because of negative implications for parallel query execution and RMAN backup parallelization.
- Using bigfile tablespaces on platforms that do not support large file sizes is not recommended and can limit tablespace capacity. See your operating system specific documentation for information about maximum supported file sizes.

11.2.3.2 Creating a Bigfile Tablespace

To create a bigfile tablespace, specify the `BIGFILE` keyword of the `CREATE TABLESPACE` statement (`CREATE BIGFILE TABLESPACE` ...).

Oracle Database automatically creates a locally managed tablespace with automatic segment space management. You can, but need not, specify `EXTENT MANAGEMENT LOCAL` and `SEGMENT SPACE MANAGEMENT AUTO` in this statement. However, the database returns an error if you specify `EXTENT MANAGEMENT DICTIONARY` or `SEGMENT SPACE MANAGEMENT MANUAL`. The remaining syntax of the statement is the same as for the `CREATE TABLESPACE` statement, but you can only specify one data file. For example:

```
CREATE BIGFILE TABLESPACE bigtbs
    DATAFILE '/u02/oracle/data/bigtbs01.dbf' SIZE 50G
...
```

You can specify `SIZE` in kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T).

If the default tablespace type was set to `BIGFILE` at database creation, you need not specify the keyword `BIGFILE` in the `CREATE TABLESPACE` statement. A bigfile tablespace is created by default.

If the default tablespace type was set to `BIGFILE` at database creation, but you want to create a traditional (smallfile) tablespace, then specify a `CREATE SMALLFILE TABLESPACE` statement to override the default tablespace type for the tablespace that you are creating.

See Also:

Oracle Multitenant Administrator's Guide

11.2.3.3 Identifying a Bigfile Tablespace

You can query a set of data dictionary views for information about bigfile tablespaces.

The following views contain a `BIGFILE` column that identifies a tablespace as a bigfile tablespace:

- `DBA_TABLESPACES`
- `USER_TABLESPACES`
- `V$TABLESPACE`

Query these views for information about bigfile tablespaces.

You can also identify a bigfile tablespace by the relative file number of its single data file. That number is 1024 on most platforms, but 4096 on OS/390.

11.2.4 Tablespaces with Default Compression Attributes

When you create a tablespace, you can specify that all tables and indexes, or their partitions, created in a tablespace are compressed by default.

- About Tablespaces with Default Compression Attributes
  When you create a tablespace, you can specify the default compression of data for all tables and indexes created in the tablespace. The default compression level also applies to the partitions that comprise the tablespace. Compressing this data can reduce disk use.
- Creating Tablespaces with Default Compression Attributes
  When you create a tablespace, you can specify the type of table compression using the `DEFAULT` keyword, followed by the table compression clause including the compression type. You can also specify the type of index compression using the `DEFAULT` keyword, followed by index compression clause and the index compression type.

11.2.4.1 About Tablespaces with Default Compression Attributes

When you create a tablespace, you can specify the default compression of data for all tables and indexes created in the tablespace. The default compression level also applies to the partitions that comprise the tablespace. Compressing this data can reduce disk use.

11.2.4.2 Creating Tablespaces with Default Compression Attributes

When you create a tablespace, you can specify the type of table compression using the `DEFAULT` keyword, followed by the table compression clause including the compression type. You can also specify the type of index compression using the `DEFAULT` keyword, followed by index compression clause and the index compression type.

The following statement indicates that all tables and partitions created in the tablespace are to use advanced row compression, unless otherwise specified:

```
CREATE TABLESPACE ... DEFAULT ROW STORE COMPRESS ADVANCED ... ;
```

You can override the default tablespace compression specification when you create a table or partition in that tablespace.

The following statement indicates that all indexes created in the tablespace are to use high level advanced index compression, unless otherwise specified:

```
CREATE TABLESPACE ... DEFAULT INDEX COMPRESS ADVANCED HIGH ... ;
```

You can override the default tablespace compression specification when you create an index in that tablespace.

11.2.5 Encrypted Tablespaces

You can encrypt any permanent tablespace to protect sensitive data.

- About Encrypted Tablespaces
  Encrypted tablespaces primarily protect your data from unauthorized access by means other than through the database. For example, when encrypted tablespaces are written to backup media for travel from one Oracle database to another or for travel to an off-site facility for storage, they remain encrypted.
- Creating Encrypted Tablespaces
  You can create encrypted tablespaces to protect your data from unauthorized access.
- Viewing Information About Encrypted Tablespaces
  You can query the `DBA_TABLESPACES` and `USER_TABLESPACES` data dictionary views for information about encrypted tablespaces.

11.2.5.1 About Encrypted Tablespaces

Encrypted tablespaces primarily protect your data from unauthorized access by means other than through the database. For example, when encrypted tablespaces are written to backup media for travel from one Oracle database to another or for travel to an off-site facility for storage, they remain encrypted.

Also, encrypted tablespaces protect data from users who try to circumvent the security features of the database and access database files directly through the operating system file system. Tablespace encryption is completely transparent to your applications, so no application modification is necessary.

Tablespace encryption does not address all security issues. It does not, for example, provide access control from within the database. Any user who is granted privileges on objects stored in an encrypted tablespace can access those objects without providing any kind of additional password or key.

When you encrypt a tablespace, all tablespace blocks are encrypted. All segment types are supported for encryption, including tables, clusters, indexes, LOBs (`BASICFILE` and `SECUREFILE`), table and index partitions, and so on.

> **Note:**
>
> There is no need to use LOB encryption on `SECUREFILE` LOBs stored in an encrypted tablespace.

To maximize security, data from an encrypted tablespace is automatically encrypted when written to the undo tablespace, to the redo logs, and to any temporary tablespace. However, starting with Oracle Database 12*c* Release 2 (12.2), you can optionally encrypt undo tablespaces and temporary tablespaces.

For partitioned tables and indexes that have different partitions in different tablespaces, it is permitted to use both encrypted and non-encrypted tablespaces in the same table or index.

Tablespace encryption uses the Transparent Data Encryption feature of Oracle Database, which requires that you create a keystore to store the master encryption key for the database. The keystore must be open before you can create the encrypted tablespace and before you can store or retrieve encrypted data. When you open the keystore, it is available to all session, and it remains open until you explicitly close it or until the database is shut down.

Transparent Data Encryption supports industry-standard encryption algorithms, including the following types of encryption algorithms Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms:

- Advanced Encryption Standard (AES)
- ARIA
- GHOST
- SEED
- Triple Data Encryption Standard (3DES)

See *Oracle Database Advanced Security Guide* for detailed information about the supported encryption algorithms.

The encryption key length is implied by the algorithm name. For example, the AES128 algorithm uses 128-bit keys. You specify the algorithm to use when you create the tablespace, and different tablespaces can use different algorithms. Although longer key lengths theoretically provide greater security, there is a trade-off in CPU overhead. If you do not specify the algorithm in your `CREATE TABLESPACE` statement, then AES128 is the default. There is no disk space overhead for encrypting a tablespace.

After an encrypted table is created, you can use an `ALTER TABLESPACE` statement to decrypt it or change its key. You can also use an `ALTER TABLESPACE` statement to encrypt a tablespace that is not encrypted.

Restrictions

The following are restrictions for encrypted tablespaces:

- Encrypted tablespaces are subject to restrictions when they are transported to another database. See "General Limitations on Transporting Data".
- When recovering a database with encrypted tablespaces (for example after a `SHUTDOWN ABORT` or a catastrophic error that brings down the database instance), you must open the keystore after database mount and before database open, so that the recovery process can decrypt data blocks and redo.

In addition, see *Oracle Database Advanced Security Guide* for general restrictions for Transparent Data Encryption.

See Also:

- *Oracle Database Advanced Security Guide* for more information about Transparent Data Encryption
- "Consider Encrypting Columns That Contain Sensitive Data" for an alternative to encrypting an entire tablespace
- *Oracle Real Application Clusters Administration and Deployment Guide* for information on using a keystore in an Oracle Real Application Clusters environment
- *Oracle Database SQL Language Reference* for information about the `CREATE TABLESPACE` statement

11.2.5.2 Creating Encrypted Tablespaces

You can create encrypted tablespaces to protect your data from unauthorized access.

To encrypt a tablespace, you must open the database with the `COMPATIBLE` initialization parameter set to `11.2.0` or higher. Any user who can create a tablespace can create an encrypted tablespace.

To create an encrypted tablespace:

- Run a `CREATE TABLESPACE` statement with an `ENCRYPTION` clause.

Starting with Oracle Database Release 21c, use the `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` initialization parameter to specify the default encryption algorithm. You can set this parameter either in the initialization parameter file or by using the `ALTER SYSTEM` statement. With wallet-based TDE, set this parameter before the first `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE` command. With OKV-based TDE deployments, set this parameter before the first `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` command.

Examples

The following statement sets the default encryption algorithm to AES192:

```
ALTER SYSTEM SET TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM=AES192;
```

The following statement creates an encrypted tablespace with the default encryption algorithm:

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION ENCRYPT;
```

If the default encryption algorithm was not explicitly set using an `ALTER SYSTEM` command, then the default encryption algorithm used is AES128.

The following statement creates the same tablespace with the AES256 algorithm:

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION USING 'AES256' ENCRYPT;
```

See Also:

Oracle Database Advanced Security Guide

11.2.5.3 Viewing Information About Encrypted Tablespaces

You can query the `DBA_TABLESPACES` and `USER_TABLESPACES` data dictionary views for information about encrypted tablespaces.

The `DBA_TABLESPACES` and `USER_TABLESPACES` data dictionary views include a column named `ENCRYPTED`. This column contains `YES` for encrypted tablespaces.

The view `V$ENCRYPTED_TABLESPACES` lists all currently encrypted tablespaces. The following query displays the name and encryption algorithm of encrypted tablespaces:

```
SELECT t.name, e.encryptionalg algorithm
FROM  v$tablespace t, v$encrypted_tablespaces e
WHERE t.ts# = e.ts#;

NAME                           ALGORITHM
------------------------------ ---------
SECURESPACE                    AES256
```

Note:

You can convert an existing tablespace to an encrypted tablespace.

See Also:

Oracle Database Advanced Security Guide for information about convert an existing tablespace to an encrypted tablespace

11.2.6 Temporary Tablespaces

Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory. These tablespaces also improve the efficiency of space management operations during sorts.

- About Temporary Tablespaces
  A **temporary tablespace** contains transient data that persists only for the duration of the session. Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory and can improve the efficiency of space management operations during sorts.
- Creating a Locally Managed Temporary Tablespace
  Because space management is much simpler and more efficient in locally managed tablespaces, they are ideally suited for temporary tablespaces.
- Creating a Bigfile Temporary Tablespace
  Just as for regular tablespaces, you can create single-file (bigfile) temporary tablespaces.

- Viewing Space Usage for Temporary Tablespaces
  The `DBA_TEMP_FREE_SPACE` dictionary view contains information about space usage for each temporary tablespace.

11.2.6.1 About Temporary Tablespaces

A **temporary tablespace** contains transient data that persists only for the duration of the session. Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory and can improve the efficiency of space management operations during sorts.

Temporary tablespaces are used to store the following:

- Intermediate sort results
- Temporary tables and temporary indexes
- Temporary LOBs
- Temporary B-trees

Within a temporary tablespace, all sort operations for a particular instance share a single *sort segment*, and sort segments exist for every instance that performs sort operations that require temporary space. A sort segment is created by the first statement after startup that uses the temporary tablespace for sorting, and is released only at shutdown.

By default, a single temporary tablespace named `TEMP` is created for each new Oracle Database installation. You can create additional temporary tablespaces with the `CREATE TABLESPACE` statement. You can assign a temporary tablespace to each database user with the `CREATE USER` or `ALTER USER` statement. A single temporary tablespace can be shared by multiple users.

You cannot explicitly create objects in a temporary tablespace.

Note:

The exception to the preceding statement is a temporary table. When you create a temporary table, its rows are stored in your default temporary tablespace, unless you create the table in a new temporary tablespace. See "Creating a Temporary Table" for more information.

Starting with Oracle Database 12*c* Release 2 (12.2), local temporary tablespaces are available. A local temporary tablespace stores separate, non-shared temp files for every database instance. A local temporary tablespace is used only for spilling temporary results of SQL statements, such as queries that involve sorts, hash aggregations, and joins. These results are only accessible within an instance. In contrast, a shared temporary tablespace resides on a shared disk and is available to all instances. To create a local temporary tablespace, use a `CREATE LOCAL TEMPORARY TABLESPACE` statement. Shared temporary tablespaces were available in prior releases of Oracle Database and were called "temporary tablespaces." In this *Oracle Database Administrator's Guide*, the term "temporary tablespace" refers to a shared temporary tablespace unless specified otherwise.

Default Temporary Tablespace

Users who are not explicitly assigned a temporary tablespace use the database default temporary tablespace, which for new installations is `TEMP`. You can change the default temporary tablespace for the database with the following command:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tablespace_name;
```

To determine the current default temporary tablespace for the database, run the following query:

```
SELECT PROPERTY_NAME, PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
   PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';

PROPERTY_NAME              PROPERTY_VALUE
-------------------------- ------------------------------
DEFAULT_TEMP_TABLESPACE    TEMP
```

Space Allocation in a Temporary Tablespace

You can view the allocation and deallocation of space in a temporary tablespace sort segment using the `V$SORT_SEGMENT` view. The `V$TEMPSEG_USAGE` view identifies the current sort users in those segments.

When a sort operation that uses temporary space completes, allocated extents in the sort segment are not deallocated; they are just marked as free and available for reuse. The `DBA_TEMP_FREE_SPACE` view displays the total allocated and free space in each temporary tablespace. See "Viewing Space Usage for Temporary Tablespaces" for more information. You can manually shrink a locally managed temporary tablespace that has a large amount of unused space. See "Shrinking a Locally Managed Temporary Tablespace" for details.

Automatic Temporary Tablespace Shrink and Extension

Queries, sorts, hash joins, query transformations, and other operations can cause a temporary tablespace to grow very large due to spikes in usage. The Automatic Temporary Tablespace Shrink and Extension feature can automatically shrink the tablespace in the background after the usage has subsided. In addition, if the database detects that temporary tablespace use is increasing, it will preemptively grow the temporary tablespace to ensure performance is not impacted. This feature requires no intervention of the database administrator.

See Also:

- *Oracle Database Security Guide* for information about creating users and assigning temporary tablespaces
- *Oracle Database Concepts* for more information about local temporary tablespaces, shared temporary tablespaces, and the default temporary tablespace
- *Oracle Database Reference* for more information about the `V$SORT_SEGMENT`, `V$TEMPSEG_USAGE`, and `DBA_TEMP_FREE_SPACE` views
- *Oracle Database Performance Tuning Guide* for a discussion on tuning sorts
- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about local temporary tablespace

11.2.6.2 Creating a Locally Managed Temporary Tablespace

Because space management is much simpler and more efficient in locally managed tablespaces, they are ideally suited for temporary tablespaces.

Locally managed temporary tablespaces use **temp files**, which do not modify data outside of the temporary tablespace or generate any redo for temporary tablespace data. Because of this, they enable you to perform on-disk sorting operations in a read-only or standby database.

You also use different views for viewing information about temp files than you would for data files. The `V$TEMPFILE` and `DBA_TEMP_FILES` views are analogous to the `V$DATAFILE` and `DBA_DATA_FILES` views.

To create a locally managed temporary tablespace, you use the `CREATE TEMPORARY TABLESPACE` statement, which requires that you have the `CREATE TABLESPACE` system privilege.

The following statement creates a temporary tablespace in which each extent is 16M. Each 16M extent (which is the equivalent of 8000 blocks when the standard block size is 2K) is represented by a bit in the bitmap for the file.

```
CREATE TEMPORARY TABLESPACE lmtemp TEMPFILE '/u02/oracle/data/lmtemp01.dbf'
     SIZE 20M REUSE
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

The extent management clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. If an extent size is specified in the `EXTENT SIZE` clause, then it is used. If it is not specified, then, Oracle Database uses the tablespace size and file sizes to determine the default extent size.

Note:

On some operating systems, the database does not allocate space for the temp file until the temp file blocks are actually accessed. This delay in space allocation results in faster creation and resizing of temp files, but it requires that sufficient disk space is available when the temp files are later used. See your operating system documentation to determine whether the database allocates temp file space in this way on your system.

11.2.6.3 Creating a Bigfile Temporary Tablespace

Just as for regular tablespaces, you can create single-file (bigfile) temporary tablespaces.

To create a bigfile temporary tablespace:

- Run the `CREATE BIGFILE TEMPORARY TABLESPACE` statement to create a single-temp file tablespace.

See the sections "Creating a Bigfile Tablespace" and "Altering a Bigfile Tablespace" for information about bigfile tablespaces, but consider that you are creating temporary tablespaces that use temp files instead of data files.

11.2.6.4 Viewing Space Usage for Temporary Tablespaces

The `DBA_TEMP_FREE_SPACE` dictionary view contains information about space usage for each temporary tablespace.

The information includes the space allocated and the free space. You can query this view for these statistics using the following statement:

```
SELECT * from DBA_TEMP_FREE_SPACE;

TABLESPACE_NAME                    TABLESPACE_SIZE ALLOCATED_SPACE FREE_SPACE
---------------------------------- --------------- --------------- ----------
TEMP                               250609664       250609664       249561088
```

11.2.7 Temporary Tablespace Groups

A temporary tablespace group is a tablespace group that is assigned as the default temporary tablespace for the database.

- Multiple Temporary Tablespaces: Using Tablespace Groups
  A **tablespace group** enables a user to consume temporary space from multiple tablespaces. Using a tablespace group, rather than a single temporary tablespace, can alleviate problems caused where one tablespace is inadequate to hold the results of a sort, particularly on a table that has many partitions. A tablespace group enables parallel execution servers in a single parallel operation to use multiple temporary tablespaces.
- Creating a Tablespace Group
  You create a tablespace group implicitly when you include the `TABLESPACE GROUP` clause in the `CREATE TEMPORARY TABLESPACE` or `ALTER TABLESPACE` statement and the specified tablespace group does not currently exist.
- Changing Members of a Tablespace Group
  You can add a tablespace to an existing tablespace group by specifying the existing group name in the `TABLESPACE GROUP` clause of the `CREATE TEMPORARY TABLESPACE` or `ALTER TABLESPACE` statement.
- Assigning a Tablespace Group as the Default Temporary Tablespace
  Use the `ALTER DATABASE...DEFAULT TEMPORARY TABLESPACE` statement to assign a tablespace group as the default temporary tablespace for the database.

11.2.7.1 Multiple Temporary Tablespaces: Using Tablespace Groups

A **tablespace group** enables a user to consume temporary space from multiple tablespaces. Using a tablespace group, rather than a single temporary tablespace, can alleviate problems caused where one tablespace is inadequate to hold the results of a sort, particularly on a table that has many partitions. A tablespace group enables parallel execution servers in a single parallel operation to use multiple temporary tablespaces.

A tablespace group has the following characteristics:

- It contains at least one tablespace. There is no explicit limit on the maximum number of tablespaces that are contained in a group.
- It shares the namespace of tablespaces, so its name cannot be the same as any tablespace.

- You can specify a tablespace group name wherever a tablespace name would appear when you assign a default temporary tablespace for the database or a temporary tablespace for a user.

You do not explicitly create a tablespace group. Rather, it is created implicitly when you assign the first temporary tablespace to the group. The group is deleted when the last temporary tablespace it contains is removed from it.

The view `DBA_TABLESPACE_GROUPS` lists tablespace groups and their member tablespaces.

See Also:

Oracle Database Security Guide for more information about assigning a temporary tablespace or tablespace group to a user

11.2.7.2 Creating a Tablespace Group

You create a tablespace group implicitly when you include the `TABLESPACE GROUP` clause in the `CREATE TEMPORARY TABLESPACE` or `ALTER TABLESPACE` statement and the specified tablespace group does not currently exist.

For example, if neither `group1` nor `group2` exists, then the following statements create those groups, each of which has only the specified tablespace as a member:

```
CREATE TEMPORARY TABLESPACE lmtemp2 TEMPFILE '/u02/oracle/data/lmtemp201.dbf'
     SIZE 50M
     TABLESPACE GROUP group1;

ALTER TABLESPACE lmtemp TABLESPACE GROUP group2;
```

11.2.7.3 Changing Members of a Tablespace Group

You can add a tablespace to an existing tablespace group by specifying the existing group name in the `TABLESPACE GROUP` clause of the `CREATE TEMPORARY TABLESPACE` or `ALTER TABLESPACE` statement.

For example, the following statement adds a tablespace to an existing group. It creates and adds tablespace `lmtemp3` to `group1`, so that `group1` contains tablespaces `lmtemp2` and `lmtemp3`.

```
CREATE TEMPORARY TABLESPACE lmtemp3 TEMPFILE '/u02/oracle/data/lmtemp301.dbf'
     SIZE 25M
     TABLESPACE GROUP group1;
```

The following statement also adds a tablespace to an existing group, but in this case because tablespace `lmtemp2` already belongs to `group1`, it is in effect moved from `group1` to `group2`:

```
ALTER TABLESPACE lmtemp2 TABLESPACE GROUP group2;
```

Now `group2` contains both `lmtemp` and `lmtemp2`, while `group1` consists of only `tmtemp3`.

You can remove a tablespace from a group as shown in the following statement:

```
ALTER TABLESPACE lmtemp3 TABLESPACE GROUP '';
```

Tablespace `lmtemp3` no longer belongs to any group. Further, since there are no longer any members of `group1`, this results in the implicit deletion of `group1`.

11.2.7.4 Assigning a Tablespace Group as the Default Temporary Tablespace

Use the `ALTER DATABASE...DEFAULT TEMPORARY TABLESPACE` statement to assign a tablespace group as the default temporary tablespace for the database.

For example:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE group2;
```

Any user who has not explicitly been assigned a temporary tablespace will now use tablespaces `lmtemp` and `lmtemp2`.

If a tablespace group is specified as the default temporary tablespace, you cannot drop any of its member tablespaces. You must first remove the tablespace from the tablespace group. Likewise, you cannot drop a single temporary tablespace as long as it is the default temporary tablespace.

11.3 Consider Storing Tablespaces in the In-Memory Column Store

You can enable a tablespace for the In-Memory Column Store during tablespace creation or by altering a tablespace. When this enable a tablespace for the In-Memory Column Store, all tables in the tablespace are enabled for the In-Memory Column Store by default.

Note:

This feature is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

The In-Memory Column Store is an optional portion of the system global area (SGA) that stores copies of tables, table partitions, and other database objects that is optimized for rapid scans. In the In-Memory Column Store, table data is stored by column rather than row in the SGA.

See Also:

"Improving Query Performance with Oracle Database In-Memory"

11.4 Specifying Nonstandard Block Sizes for Tablespaces

You can create tablespaces with block sizes different from the standard database block size, which is specified by the `DB_BLOCK_SIZE` initialization parameter. This feature lets you transport tablespaces with unlike block sizes between databases.

To create a tablespace with a block size different from the database standard block size:

- Use the `BLOCKSIZE` clause of the `CREATE TABLESPACE` statement.

In order for the BLOCKSIZE clause to succeed, you must have already set the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE initialization parameter. Further, and the integer you specify in the BLOCKSIZE clause must correspond with the setting of one DB_nK_CACHE_SIZE parameter setting. Although redundant, specifying a BLOCKSIZE equal to the standard block size, as specified by the DB_BLOCK_SIZE initialization parameter, is allowed.

The following statement creates tablespace lmtbsb, but specifies a block size that differs from the standard database block size (as specified by the DB_BLOCK_SIZE initialization parameter):

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K
    BLOCKSIZE 8K;
```

See Also:

- "*Oracle Database SQL Language Reference*"
- "Setting the Buffer Cache Initialization Parameters" for information about the DB_CACHE_SIZE and DB_nK_CACHE_SIZE parameter settings
- "Transporting Tablespaces Between Databases"

11.5 Controlling the Writing of Redo Records

For some database operations, you can control whether the database generates redo records.

Without redo, no media recovery is possible. However, suppressing redo generation can improve performance, and may be appropriate for easily recoverable operations. An example of such an operation is a CREATE TABLE...AS SELECT statement, which can be repeated in case of database or instance failure.

To suppress redo when these operations are performed for objects within the tablespace:

- Specify the NOLOGGING clause in the CREATE TABLESPACE statement.

If you do not include this clause, or if you specify LOGGING instead, then the database generates redo when changes are made to objects in the tablespace. Redo is never generated for temporary segments or in temporary tablespaces, regardless of the logging attribute.

The logging attribute specified at the tablespace level is the default attribute for objects created within the tablespace. You can override this default logging attribute by specifying LOGGING or NOLOGGING at the schema object level--for example, in a CREATE TABLE statement.

If you have a standby database, NOLOGGING mode causes problems with the availability and accuracy of the standby database. To overcome this problem, you can specify FORCE LOGGING mode. When you include the FORCE LOGGING clause in the CREATE TABLESPACE statement, you force the generation of redo records for all operations that make changes to objects in a tablespace. This overrides any specification made at the object level.

If you transport a tablespace that is in FORCE LOGGING mode to another database, the new tablespace will not maintain the FORCE LOGGING mode.

See Also:

- *Oracle Database SQL Language Reference* for information about operations that can be done in `NOLOGGING` mode
- "*Oracle Database SQL Language Reference*" for more information about `FORCE LOGGING` mode and for information about the effects of the `FORCE LOGGING` clause used with the `CREATE DATABASE` statement

11.6 Altering Tablespace Availability

You can take an online tablespace offline so that it is temporarily unavailable for general use. The rest of the database remains open and available for users to access data. Conversely, you can bring an offline tablespace online to make the schema objects within the tablespace available to database users. The database must be open to alter the availability of a tablespace.

To alter the availability of a tablespace, use the `ALTER TABLESPACE` statement. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

- Taking Tablespaces Offline
  Taking a tablespace offline makes it unavailable for normal access.
- Bringing Tablespaces Online
  You can bring any tablespace in an Oracle Database online whenever the database is open. A tablespace is normally online so that the data contained within it is available to database users.

See Also:

"Altering Data File Availability" for information about altering the availability of individual data files within a tablespace

11.6.1 Taking Tablespaces Offline

Taking a tablespace offline makes it unavailable for normal access.

You may want to take a tablespace offline for any of the following reasons:

- To make a portion of the database unavailable while allowing normal access to the remainder of the database
- To perform an offline tablespace backup (even though a tablespace can be backed up while online and in use)
- To make an application and its group of tables temporarily unavailable while updating or maintaining the application
- To rename or relocate tablespace data files

  See "Renaming and Relocating Data Files" for details.

To take a tablespace offline:

- Run an `ALTER TABLESPACE` statement with the `OFFLINE` clause.

When a tablespace is taken offline, the database takes all the associated files offline.

You cannot take the following tablespaces offline:

- `SYSTEM`
- The undo tablespace
- Temporary tablespaces

Before taking a tablespace offline, consider altering the tablespace allocation of any users who have been assigned the tablespace as a default tablespace. Doing so is advisable because those users will not be able to access objects in the tablespace while it is offline.

You can specify any of the following parameters as part of the `ALTER TABLESPACE...OFFLINE` statement:

Clause	Description
`NORMAL`	A tablespace can be taken offline normally if no error conditions exist for any of the data files of the tablespace. No data file in the tablespace can be currently offline as the result of a write error. When you specify `OFFLINE NORMAL`, the database takes a checkpoint for all data files of the tablespace as it takes them offline. `NORMAL` is the default.
`TEMPORARY`	A tablespace can be taken offline temporarily, even if there are error conditions for one or more files of the tablespace. When you specify `OFFLINE TEMPORARY`, the database takes offline the data files that are not already offline, checkpointing them as it does so. If no files are offline, but you use the temporary clause, media recovery is not required to bring the tablespace back online. However, if one or more files of the tablespace are offline because of write errors, and you take the tablespace offline temporarily, the tablespace requires recovery before you can bring it back online.
`IMMEDIATE`	A tablespace can be taken offline immediately, without the database taking a checkpoint on any of the data files. When you specify `OFFLINE IMMEDIATE`, media recovery for the tablespace is required before the tablespace can be brought online. You cannot take a tablespace offline immediately if the database is running in `NOARCHIVELOG` mode.

Note:

If you must take a tablespace offline, use the `NORMAL` clause (the default) if possible. This setting guarantees that the tablespace will not require recovery to come back online, even if after incomplete recovery you reset the redo log sequence using an `ALTER DATABASE OPEN RESETLOGS` statement.

Specify `TEMPORARY` only when you cannot take the tablespace offline normally. In this case, only the files taken offline because of errors need to be recovered before the tablespace can be brought online. Specify `IMMEDIATE` only after trying both the normal and temporary settings.

The following example takes the `users` tablespace offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

11.6.2 Bringing Tablespaces Online

You can bring any tablespace in an Oracle Database online whenever the database is open. A tablespace is normally online so that the data contained within it is available to database users.

To bring a tablespace online:

- Run an `ALTER TABLESPACE` statement with the `ONLINE` clause.

If a tablespace to be brought online was not taken offline "cleanly" (that is, using the `NORMAL` clause of the `ALTER TABLESPACE OFFLINE` statement), you must first perform media recovery on the tablespace before bringing it online. Otherwise, the database returns an error and the tablespace remains offline.

For example, the following statement brings the `users` tablespace online:

```
ALTER TABLESPACE users ONLINE;
```

See Also:

Oracle Database Backup and Recovery User's Guide for information about performing media recovery

11.7 Using Read-Only Tablespaces

A tablespace can be put into read-only mode. This prevents any data stored in it from being updated.

- About Read-Only Tablespaces
  Making a tablespace read-only prevents write operations on the data files in the tablespace.
- Making a Tablespace Read-Only
  You can make a tablespace read-only using the `ALTER TABLESPACE` statement with the `READ ONLY` clause.
- Making a Read-Only Tablespace Writable
  Making a read-only tablespace writable allows write operations on the data files in the tablespace.
- Creating a Read-Only Tablespace on a WORM Device
  You can create a read-only tablespace on a CD-ROM or WORM (Write Once-Read Many) device.
- Delaying the Opening of Data Files in Read-Only Tablespaces
  You can delay the opening of data files for read-only tablespaces until there is an attempt to access them.

11.7.1 About Read-Only Tablespaces

Making a tablespace read-only prevents write operations on the data files in the tablespace.

The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Read-only tablespaces also provide a way to protecting historical data so that users cannot modify it. Making a tablespace read-only prevents updates on all tables in the tablespace, regardless of a user's update privilege level.

> **Note:**
>
> Making a tablespace read-only cannot in itself be used to satisfy archiving or data publishing requirements, because the tablespace can only be brought online in the database in which it was created. However, you can meet such requirements by using the transportable tablespace feature, as described in "Transporting Tablespaces Between Databases".

You can drop items, such as tables or indexes, from a read-only tablespace, but you cannot create or alter objects in a read-only tablespace. You can execute statements that update the file description in the data dictionary, such as `ALTER TABLE...ADD` or `ALTER TABLE...MODIFY`, but you will not be able to use the new description until the tablespace is made read/write. Note that you cannot add a column of data type `BLOB` when you alter a table definition.

Read-only tablespaces can be transported to other databases. And, since read-only tablespaces can never be updated, they can reside on CD-ROM or WORM (Write Once-Read Many) devices.

> **See Also:**
>
> "Transporting Tablespaces Between Databases"

11.7.2 Making a Tablespace Read-Only

You can make a tablespace read-only using the `ALTER TABLESPACE` statement with the `READ ONLY` clause.

All tablespaces are initially created as read/write. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

Before you can make a tablespace read-only, the following conditions must be met.

- The tablespace must be online. This is necessary to ensure that there is no undo information that must be applied to the tablespace.
- The tablespace cannot be the active undo tablespace or `SYSTEM` tablespace.
- The tablespace must not currently be involved in an online backup, because the end of a backup updates the header file of all data files in the tablespace.
- The tablespace cannot be a temporary tablespace.

To change a tablespace to read-only:

- Use the `READ ONLY` clause in the `ALTER TABLESPACE` statement.

For example the following statement makes the `flights` tablespace read-only:

```
ALTER TABLESPACE flights READ ONLY;
```

For better performance while accessing data in a read-only tablespace, you can issue a query that accesses all of the blocks of the tables in the tablespace just before making it read-only. A simple query, such as `SELECT COUNT (*)`, executed against each table ensures that the data blocks in the tablespace can be subsequently accessed most efficiently. This eliminates the need for the database to check the status of the transactions that most recently modified the blocks.

You can issue the `ALTER TABLESPACE...READ ONLY` statement while the database is processing transactions. After the statement is issued, the tablespace is put into a transitional read-only mode, and the `ALTER` command waits for existing transactions to complete by committing or by rolling back. No further DML operations are allowed to the tablespace, and if a DML statement attempts further changes, then an error is returned.

The `ALTER TABLESPACE...READ ONLY` statement waits for the following transactions to either commit or roll back before returning: transactions that have pending or uncommitted changes to the tablespace and that were started before you issued the statement. If a transaction started before the statement remains active, but rolls back to a savepoint, rolling back its changes to the tablespace, then the statement no longer waits for this active transaction.

If you find it is taking a long time for the `ALTER TABLESPACE` statement to complete, then you can identify the transactions that are preventing the read-only state from taking effect. You can then notify the owners of those transactions and decide whether to terminate the transactions, if necessary.

The following example identifies the transaction entry for the `ALTER TABLESPACE...READ ONLY` statement and displays its session address (`saddr`):

```
SELECT SQL_TEXT, SADDR
    FROM V$SQLAREA,V$SESSION
    WHERE V$SQLAREA.ADDRESS = V$SESSION.SQL_ADDRESS
        AND SQL_TEXT LIKE 'alter tablespace%';

SQL_TEXT                                  SADDR
---------------------------------------- --------
alter tablespace tbs1 read only           80034AF0
```

The start SCN of each active transaction is stored in the `V$TRANSACTION` view. Displaying this view sorted by ascending start SCN lists the transactions in execution order. From the preceding example, you already know the session address of the transaction entry for the read-only statement, and you can now locate it in the `V$TRANSACTION` view. All transactions with smaller start SCN, which indicates an earlier execution, can potentially hold up the quiesce and subsequent read-only state of the tablespace.

```
SELECT SES_ADDR, START_SCNB
    FROM V$TRANSACTION
    ORDER BY START_SCNB;

SES_ADDR START_SCNB
-------- ----------
800352A0       3621   --> waiting on this txn
80035A50       3623   --> waiting on this txn
80034AF0       3628   --> this is the ALTER TABLESPACE statement
80037910       3629   --> don't care about this txn
```

You can now find the owners of the blocking transactions.

```
SELECT T.SES_ADDR, S.USERNAME, S.MACHINE
   FROM V$SESSION S, V$TRANSACTION T
   WHERE T.SES_ADDR = S.SADDR
   ORDER BY T.SES_ADDR
```

```
SES_ADDR USERNAME             MACHINE
-------- -------------------- --------------------
800352A0 DAVIDB               DAVIDBLAP            --> Contact this user
80035A50 MIKEL                LAB61                --> Contact this user
80034AF0 DBA01                STEVEFLAP
80037910 NICKD                NICKDLAP
```

After making the tablespace read-only, it is advisable to back it up immediately. As long as the tablespace remains read-only, no further backups of the tablespace are necessary, because no changes can be made to it.

See Also:

Oracle Database Backup and Recovery User's Guide

11.7.3 Making a Read-Only Tablespace Writable

Making a read-only tablespace writable allows write operations on the data files in the tablespace.

You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

To change a tablespace to allow write operations:

- Use the `READ WRITE` keywords in the `ALTER TABLESPACE` statement

A prerequisite to making the tablespace read/write is that all of the data files in the tablespace, as well as the tablespace itself, must be online. Use the `DATAFILE...ONLINE` clause of the `ALTER DATABASE` statement to bring a data file online. The `V$DATAFILE` view lists the current status of data files.

For example, the following statement makes the `flights` tablespace writable:

```
ALTER TABLESPACE flights READ WRITE;
```

Making a read-only tablespace writable updates the control file entry for the data files, so that you can use the read-only version of the data files as a starting point for recovery.

11.7.4 Creating a Read-Only Tablespace on a WORM Device

You can create a read-only tablespace on a CD-ROM or WORM (Write Once-Read Many) device.

Follow these steps to create a read-only tablespace on a CD-ROM or WORM device.

1. Create a writable tablespace on another device. Create the objects that belong in the tablespace and insert your data.
2. Alter the tablespace to make it read-only.
3. Copy the data files of the tablespace onto the WORM device. Use operating system commands to copy the files.
4. Take the tablespace offline.

5. Rename the data files to coincide with the names of the data files you copied onto your WORM device. Use `ALTER TABLESPACE` with the `RENAME DATAFILE` clause. Renaming the data files changes their names in the control file.
6. Bring the tablespace back online.

11.7.5 Delaying the Opening of Data Files in Read-Only Tablespaces

You can delay the opening of data files for read-only tablespaces until there is an attempt to access them.

When substantial portions of a very large database are stored in read-only tablespaces that are located on slow-access devices or hierarchical storage, you should consider setting the `READ_ONLY_OPEN_DELAYED` initialization parameter to `TRUE`. This speeds certain operations, primarily opening the database, by causing data files in read-only tablespaces to be accessed for the first time only when an attempt is made to read data stored within them.

Setting `READ_ONLY_OPEN_DELAYED=TRUE` has the following side-effects:

- A missing or bad read-only file is not detected at open time. It is only discovered when there is an attempt to access it.
- `ALTER SYSTEM CHECK DATAFILES` does not check read-only files.
- `ALTER TABLESPACE...ONLINE` and `ALTER DATABASE DATAFILE...ONLINE` do not check read-only files. They are checked only upon the first access.
- `V$RECOVER_FILE`, `V$BACKUP`, and `V$DATAFILE_HEADER` do not access read-only files. Read-only files are indicated in the results list with the error "`DELAYED OPEN`", with zeroes for the values of other columns.
- `V$DATAFILE` does not access read-only files. Read-only files have a size of "0" listed.
- `V$RECOVERY_LOG` does not access read-only files. Logs they could need for recovery are not added to the list.
- `ALTER DATABASE NOARCHIVELOG` does not access read-only files. It proceeds even if there is a read-only file that requires recovery.

Note:

– `RECOVER DATABASE` and `ALTER DATABASE OPEN RESETLOGS` continue to access all read-only data files regardless of the parameter value. To avoid accessing read-only files for these operations, take those files offline.

– If a backup control file is used, the read-only status of some files may be inaccurate. This can cause some of these operations to return unexpected results. Care should be taken in this situation.

11.8 Altering and Maintaining Tablespaces

You can alter and maintain tablespaces by performing such tasks as adding data files and temp files to them.

- Increasing the Size of a Tablespace
  You can increase the size of a tablespace by either increasing the size of a data file in the tablespace or adding one.

- Altering a Locally Managed Tablespace
  You can add a data file to a locally managed tablespace, alter its availability, make it read-only or read/write, rename it, or enable/disable autoextension.
- Altering a Bigfile Tablespace
  You can resize or autoextend a bigfile tablespace.
- Altering a Locally Managed Temporary Tablespace
  You can alter a locally managed temporary tablespace to add a temp file, take a temp file offline, or bring a temp file online.
- Shrinking a Locally Managed Temporary Tablespace
  You can shrink locally managed temporary tablespaces and release unused space.

11.8.1 Increasing the Size of a Tablespace

You can increase the size of a tablespace by either increasing the size of a data file in the tablespace or adding one.

See "Changing Data File Size" and "Creating Data Files and Adding Data Files to a Tablespace " for more information.

Additionally, you can enable automatic file extension (`AUTOEXTEND`) to data files and bigfile tablespaces. See "Enabling and Disabling Automatic Extension for a Data File".

11.8.2 Altering a Locally Managed Tablespace

You can add a data file to a locally managed tablespace, alter its availability, make it read-only or read/write, rename it, or enable/disable autoextension.

You cannot alter a locally managed tablespace to a locally managed temporary tablespace, nor can you change its method of segment space management. Coalescing free extents is unnecessary for locally managed tablespaces. However, you can use the `ALTER TABLESPACE` statement on locally managed tablespaces for some operations, including the following:

- Adding a data file. For example:

```
ALTER TABLESPACE lmtbsb
   ADD DATAFILE '/u02/oracle/data/lmtbsb02.dbf' SIZE 1M;
```

- Altering tablespace availability (`ONLINE`/`OFFLINE`).
- Making a tablespace read-only or read/write.
- Renaming a data file, or enabling or disabling the autoextension of the size of a data file in the tablespace.

See Also:

- "Altering Tablespace Availability "
- "About Read-Only Tablespaces"
- "Managing Data Files and Temp Files"

11.8.3 Altering a Bigfile Tablespace

You can resize or autoextend a bigfile tablespace.

Two clauses of the `ALTER TABLESPACE` statement support data file transparency when you are using bigfile tablespaces:

- `RESIZE`: The `RESIZE` clause lets you resize the single data file in a bigfile tablespace to an absolute size, without referring to the data file. For example:

  ```
  ALTER TABLESPACE bigtbs RESIZE 80G;
  ```

- `AUTOEXTEND` (used outside of the `ADD DATAFILE` clause):

  With a bigfile tablespace, you can use the `AUTOEXTEND` clause outside of the `ADD DATAFILE` clause. For example:

  ```
  ALTER TABLESPACE bigtbs AUTOEXTEND ON NEXT 20G;
  ```

An error is raised if you specify an `ADD DATAFILE` clause for a bigfile tablespace.

11.8.4 Altering a Locally Managed Temporary Tablespace

You can alter a locally managed temporary tablespace to add a temp file, take a temp file offline, or bring a temp file online.

Note:

You cannot use the `ALTER TABLESPACE` statement, with the `TEMPORARY` keyword, to change a locally managed permanent tablespace into a locally managed temporary tablespace. You must use the `CREATE TEMPORARY TABLESPACE` statement to create a locally managed temporary tablespace.

You can use `ALTER TABLESPACE` to add a temp file, take a temp file offline, or bring a temp file online, as illustrated in the following examples:

```
ALTER TABLESPACE lmtemp
   ADD TEMPFILE '/u02/oracle/data/lmtemp02.dbf' SIZE 18M REUSE;

ALTER TABLESPACE lmtemp TEMPFILE OFFLINE;
ALTER TABLESPACE lmtemp TEMPFILE ONLINE;
```

Note:

You cannot take a temporary tablespace offline. Instead, you take its temp file offline. The view `V$TEMPFILE` displays online status for a temp file.

The `ALTER DATABASE` statement can be used to alter temp files.

The following statements take offline and bring online temp files. They behave identically to the last two `ALTER TABLESPACE` statements in the previous example.

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' ONLINE;
```

The following statement resizes a temp file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' RESIZE 18M;
```

The following statement drops a temp file and deletes its operating system file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
    INCLUDING DATAFILES;
```

The tablespace to which this temp file belonged remains. A message is written to the alert log for the temp file that was deleted. If an operating system error prevents the deletion of the file, the statement still succeeds, but a message describing the error is written to the alert log.

It is also possible to use the `ALTER DATABASE` statement to enable or disable the automatic extension of an existing temp file, and to rename a temp file. See *Oracle Database SQL Language Reference* for the required syntax.

Note:

To rename a temp file, you take the temp file offline, use operating system commands to rename or relocate the temp file, and then use the `ALTER DATABASE RENAME FILE` command to update the database control files.

11.8.5 Shrinking a Locally Managed Temporary Tablespace

You can shrink locally managed temporary tablespaces and release unused space.

Large sort operations performed by the database may result in a temporary tablespace growing and occupying a considerable amount of disk space. After the sort operation completes, the extra space is not released; it is just marked as free and available for reuse. Therefore, a single large sort operation might result in a large amount of allocated temporary space that remains unused after the sort operation is complete. For this reason, the database enables you to shrink locally managed temporary tablespaces and release unused space.

To shrink a temporary tablespace:

- Use the `SHRINK SPACE` clause of the `ALTER TABLESPACE` statement.

To shrink a specific temp file of a temporary tablespace:

- Use the `SHRINK TEMPFILE` clause of the `ALTER TABLESPACE` statement .

Shrinking frees as much space as possible while maintaining the other attributes of the tablespace or temp file. The optional `KEEP` clause defines a minimum size for the tablespace or temp file.

Shrinking is an online operation, which means that user sessions can continue to allocate sort extents if needed, and already-running queries are not affected.

The following example shrinks the locally managed temporary tablespace `lmtmp1` while ensuring a minimum size of 20M.

```
ALTER TABLESPACE lmtemp1 SHRINK SPACE KEEP 20M;
```

The following example shrinks the temp file `lmtemp02.dbf` of the locally managed temporary tablespace `lmtmp2`. Because the `KEEP` clause is omitted, the database attempts to shrink the temp file to the minimum possible size.

```
ALTER TABLESPACE lmtemp2 SHRINK TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

11.9 Renaming Tablespaces

Using the `RENAME TO` clause of the `ALTER TABLESPACE`, you can rename a permanent or temporary tablespace.

For example, the following statement renames the `users` tablespace:

```
ALTER TABLESPACE users RENAME TO usersts;
```

When you rename a tablespace the database updates all references to the tablespace name in the data dictionary, control file, and (online) data file headers. The database does not change the tablespace ID so if this tablespace were, for example, the default tablespace for a user, then the renamed tablespace would show as the default tablespace for the user in the `DBA_USERS` view.

The following affect the operation of this statement:

- If the tablespace being renamed is the `SYSTEM` tablespace or the `SYSAUX` tablespace, then it will not be renamed and an error is raised.
- If any data file in the tablespace is offline, or if the tablespace is offline, then the tablespace is not renamed and an error is raised.
- If the tablespace is read only, then data file headers are not updated. This should not be regarded as corruption; instead, it causes a message to be written to the alert log indicating that data file headers have not been renamed. The data dictionary and control file are updated.
- If the tablespace is the default temporary tablespace, then the corresponding entry in the database properties table is updated and the `DATABASE_PROPERTIES` view shows the new name.
- If the tablespace is an undo tablespace and if the following conditions are met, then the tablespace name is changed to the new tablespace name in the server parameter file (`SPFILE`).
  - The server parameter file was used to start up the database.
  - The tablespace name is specified as the `UNDO_TABLESPACE` for any instance.

  If a traditional initialization parameter file (`PFILE`) is being used then a message is written to the alert log stating that the initialization parameter file must be manually changed.

11.10 Dropping Tablespaces

You can drop a tablespace and its contents (the segments contained in the tablespace) from the database if the tablespace and its contents are no longer required.

You must have the `DROP TABLESPACE` system privilege to drop a tablespace.

Note:

Once a tablespace has been dropped, the data in the tablespace is not recoverable. Therefore, ensure that all data contained in a tablespace to be dropped will not be required in the future. Also, immediately before and after dropping a tablespace from a database, back up the database completely. This is *strongly recommended* so that you can recover the database if you mistakenly drop a tablespace, or if the database experiences a problem in the future after the tablespace has been dropped.

When you drop a tablespace, the file pointers in the control file of the associated database are removed. You can optionally direct Oracle Database to delete the operating system files (data files) that constituted the dropped tablespace. If you do not direct the database to delete the data files at the same time that it deletes the tablespace, you must later use the appropriate commands of your operating system to delete them.

You cannot drop a tablespace that contains any active segments. For example, if a table in the tablespace is currently being used or the tablespace contains undo data needed to roll back uncommitted transactions, you cannot drop the tablespace. The tablespace can be online or offline, but it is best to take the tablespace offline before dropping it.

To drop a tablespace:

- Use the `DROP TABLESPACE` statement.

The following statement drops the `users` tablespace, including the segments in the tablespace:

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

If the tablespace is empty (does not contain any tables, views, or other structures), you do not need to specify the `INCLUDING CONTENTS` clause. Use the `CASCADE CONSTRAINTS` clause to drop all referential integrity constraints from tables outside the tablespace that refer to primary and unique keys of tables inside the tablespace.

To delete the data files associated with a tablespace at the same time that the tablespace is dropped, use the `INCLUDING CONTENTS AND DATAFILES` clause. The following statement drops the `users` tablespace and its associated data files:

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```

A message is written to the alert log for each data file that is deleted. If an operating system error prevents the deletion of a file, the `DROP TABLESPACE` statement still succeeds, but a message describing the error is written to the alert log.

See Also:

"Dropping Data Files"

11.11 Managing Lost Write Protection with Shadow Tablespaces

A data block lost write occurs when an I/O subsystem acknowledges the completion of the block write, but the write did not occur in the persistent storage. Shadow lost write protection can protect against lost writes.

- About Shadow Lost Write Protection
  A data block lost write occurs when an I/O subsystem acknowledges the completion of the block write even though the write did not occur or when a former image of the block overwrites the current image. Shadow lost write protection can protect against lost writes for tablespaces or for individual data files.
- Creating Shadow Tablespaces for Shadow Lost Write Protection
  To create a shadow tablespace for shadow lost write protection, issue a `CREATE BIGFILE TABLESPACE` statement with the `LOST WRITE PROTECTION` clause.
- Enabling Shadow Lost Write Protection for a Database
  To enable shadow lost write protection for a multitenant container database (CDB), use the `ALTER DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause. To enable shadow lost write protection for a pluggable database (PDB), use the `ALTER PLUGGABLE DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause.
- Enabling Shadow Lost Write Protection for Tablespaces and Data Files
  You can enable shadow lost write protection for tablespaces and data files.
- Disabling Shadow Lost Write Protection for a Database
  To disable shadow lost write protection for a multitenant container database (CDB), issue an `ALTER DATABASE` statement with the `DISABLE LOST WRITE PROTECTION` clause. To disable shadow lost write protection for a pluggable database (PDB), issue an `ALTER PLUGGABLE DATABASE` statement with the `DISABLE LOST WRITE PROTECTION` clause.
- Removing or Suspending Shadow Lost Write Protection
  You can remove or suspend shadow lost write protection for a tablespace or a data file.
- Dropping a Shadow Tablespace
  You can drop a shadow tablespace using the `DROP TABLESPACE` statement. If you use the `DROP TABLESPACE` statement with the `INCLUDING CONTENTS` clause, then the shadow tablespace is dropped along with its contents. If you use the `DROP TABLESPACE` statement without the `INCLUDING CONTENTS` clause, then before dropping the shadow tablespace, its contents are moved to another shadow tablespace, if it exists and has a sufficient free space.

11.11.1 About Shadow Lost Write Protection

A data block lost write occurs when an I/O subsystem acknowledges the completion of the block write even though the write did not occur or when a former image of the block overwrites the current image. Shadow lost write protection can protect against lost writes for tablespaces or for individual data files.

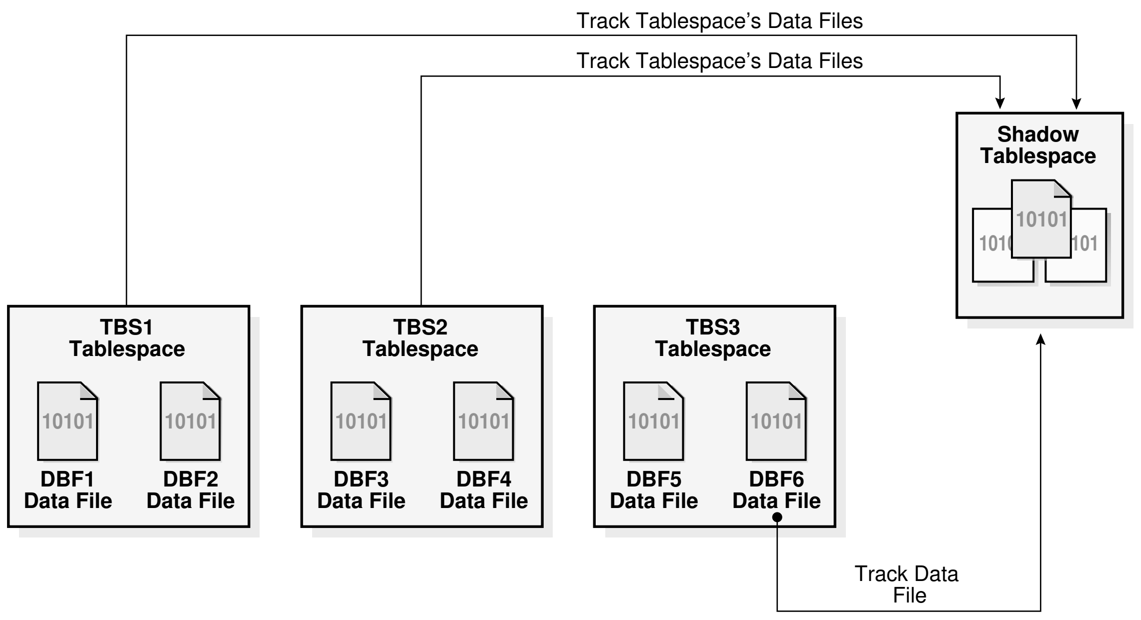
Shadow lost write protection provides fast detection and immediate response to a lost write. Using shadow lost write protection can minimize data loss and the time required to repair a database.

To use shadow lost write protection, you must enable it for the database and create one or more shadow tablespaces. A shadow tablespace is a special-purpose bigfile tablespace that contains only system change numbers (SCNs) for tracked data files. You create a shadow tablespace by including the `LOST WRITE PROTECTION` clause in the `CREATE TABLESPACE` statement.

When a tracked data block is read from disk, shadow lost write protection can detect a lost write by comparing the SCN for the block in the shadow tablespace with the SCN of the most recent write in the block being read. If the shadow entry has an SCN greater than the data block being read, then a lost write has occurred. When a lost write is detected, an error is returned.

An undetected lost write can result in data corruption because the incorrect data can be used for other DML transactions. Shadow lost write protection detects a lost write before it is consumed to prevent data corruption. You can enable shadow lost write protection for specific tablespaces and data files. Therefore, you can choose to enable it only for your most important data. You do not need to use it to track all of your data. In addition, shadow tablespaces are flexible. You can replace one shadow tablespace with another to change its configuration or location.

Figure 11-1 Shadow Lost Write Protection



When shadow lost write protection is enabled, it is enabled for normal DML operations and SQL*Loader conventional path load and direct path load operations. It is also enabled for Recovery Manager (RMAN) backups. An RMAN backup checks the blocks being read for lost writes and raises an error if such a block is found.

After shadow lost write protection is enabled for a tablespace or data file, you can suspend it if you want to stop collecting new lost write information and checking for lost writes for them. When shadow lost write protection is suspended. the tracking data is preserved in the shadow tablespace, and you can re-enable shadow lost write protection. If you remove shadow lost write protection for a data file or a tablespace, then its tracking data is deleted and is no longer reusable.

You enable a tablespace for shadow lost write protection by including the `LOST WRITE PROTECTION` clause in an `ALTER TABLESPACE` statement, and you enable a data file for shadow lost write protection by including the `LOST WRITE PROTECTION` clause in an `ALTER DATABASE` *`data_file_name`* statement. When shadow lost write protection is enabled for a tablespace, all of the tablespace's current and future data files are enabled for shadow lost write protection.

Oracle Database assigns a tracked data file to a specific shadow tablespace automatically. You cannot specify which shadow tablespace is used for a particular data file. The amount of space in shadow tablespaces should be at least 2% of the space used by the data files enabled for shadow lost write protection.

Note:

- If you increase the size of a tracked data file, then shadow lost write protection attempts to resize the tracking data in the corresponding shadow tablespace. If there is insufficient space to track all of the data, then shadow lost write protection inserts a warning message into the log and continues to track the data that it can using the available shadow space.
- A flashback of a database causes any shadow lost write protection data to be removed. After the flashback, shadow lost write protection tracks the data as it is repopulated and updates are made to the shadow tracking data as block updates occur.
- Shadow lost write protection is not related to lost write protection that is configured with the `DB_LOST_WRITE_PROTECT` initialization parameter and a standby database.

11.11.2 Creating Shadow Tablespaces for Shadow Lost Write Protection

To create a shadow tablespace for shadow lost write protection, issue a `CREATE BIGFILE TABLESPACE` statement with the `LOST WRITE PROTECTION` clause.

A shadow tablespace can be used by any tablespace or data file enabled for shadow lost write protection. The amount of space in shadow tablespaces should be at least 2% of the space used by data files enabled for shadow lost write protection. A shadow tablespace must be a bigfile tablespace.

Note:

For creating shadow tablespaces, the database compatibility level must be 18.0.0 or higher.

To create a shadow tablespace in a database:

1. In SQL*Plus, connect to the database as a user with `CREATE TABLESPACE` system privilege.
2. Issue a `CREATE BIGFILE TABLESPACE` statement with the `LOST WRITE PROTECTION` clause.

Example 11-1 Creating a Shadow Tablespace for Shadow Lost Write Protection

This example creates the `shadow_lwp1` tablespace as a shadow tablespace for shadow lost write protection.

```
CREATE BIGFILE TABLESPACE shadow_lwp1 DATAFILE 'shadow_lwp1.df'
   SIZE 10M LOST WRITE PROTECTION;
```

11.11.3 Enabling Shadow Lost Write Protection for a Database

To enable shadow lost write protection for a multitenant container database (CDB), use the `ALTER DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause. To enable shadow lost write protection for a pluggable database (PDB), use the `ALTER PLUGGABLE DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause.

Before you can enable individual tablespaces and data files for shadow lost write protection, you must create at least one shadow tablespace, and you must enable the database that contains it for shadow lost write protection. After doing so, you can use `ALTER TABLESPACE` statements to enable tablespaces for shadow lost write protection, and you can use `ALTER DATABASE` statements to enable data files for shadow lost write protection.

Note:

- For enabling shadow lost write protection for a database, the database compatibility level must be 18.0.0 or higher, and at least one shadow tablespace must exist.
- Enabling or disabling shadow lost write protection for a CDB root does not impact the shadow lost write protection for the PDBs. Therefore, shadow lost write protection can be enabled for a PDB even if it is disabled for the CDB root.
- When you enable shadow lost write protection for a database, a shadow tablespace is automatically assigned to it.

To enable shadow lost write protection for a database:

1. In SQL*Plus, connect to a user with the required privileges:
   - In a CDB root, connect as a user with `ALTER DATABASE` system privilege.
   - In an application root, PDB, or application PDB, connect as a user with `ALTER PLUGGABLE DATABASE` system privilege.
2. Do one of the following:
   - For a CDB root, issue an `ALTER DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause.
   - For an application root, PDB, or application PDB, issue an `ALTER PLUGGABLE DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause.

Example 11-2 Enabling Shadow Lost Write Protection for a CDB Root

```
ALTER DATABASE ENABLE LOST WRITE PROTECTION;
```

Example 11-3 Enabling Shadow Lost Write Protection for a PDB

```
ALTER PLUGGABLE DATABASE ENABLE LOST WRITE PROTECTION;
```

Related Topics

- Creating Shadow Tablespaces for Shadow Lost Write Protection
  To create a shadow tablespace for shadow lost write protection, issue a `CREATE BIGFILE TABLESPACE` statement with the `LOST WRITE PROTECTION` clause.

11.11.4 Enabling Shadow Lost Write Protection for Tablespaces and Data Files

You can enable shadow lost write protection for tablespaces and data files.

To enable shadow lost write protection for a tablespace, issue an `ALTER TABLESPACE` statement with the `ENABLE LOST WRITE PROTECTION` clause. To enable shadow lost write protection for a data file, issue an `ALTER DATABASE` *`data_file_name`* statement with the `ENABLE LOST WRITE PROTECTION` clause. When you enable shadow lost write protection for a tablespace, all of the data files of the tablespace are enabled for shadow lost write protection, and any data files added to the tablespace are enabled for shadow lost write protection.

Note:

- To enable shadow lost write protection for a tablespace or data file, shadow lost write protection must be enabled for the database and at least one shadow tablespace must exist.
- When you enable shadow lost write protection for a tablespace or data file, a shadow tablespace is automatically assigned to it.

To enable shadow lost write protection for a tablespace or a data file:

1. In SQL*Plus, connect to the database as a user with the required privileges:
   - If you are enabling shadow lost write protection for a tablespace, then connect as a user with `ALTER TABLESPACE` privilege.
   - If you are enabling shadow lost write protection for a data file used by a CDB root, then connect as a user with `ALTER DATABASE` privilege.
   - If you are enabling shadow lost write protection for a data file used by an application root, PDB, or application PDB, then connect as a user with `ALTER PLUGGABLE DATABASE` privilege.
2. Perform one of the following actions:
   - To enable shadow lost write protection for a tablespace, issue an `ALTER TABLESPACE` statement with the `ENABLE LOST WRITE PROTECTION` clause.
   - To enable shadow lost write protection for a data file that is used by a CDB root, issue an `ALTER DATABASE DATAFILE` *`data_file_name`* statement with the `ENABLE LOST WRITE PROTECTION` clause, and replace *`data_file_name`* with the name of the data file.
   - To enable shadow lost write protection for a data file that is used by an application root, PDB, or application PDB, issue an `ALTER PLUGGABLE DATABASE DATAFILE` *`data_file_name`* statement with the `ENABLE LOST WRITE PROTECTION` clause, and replace *`data_file_name`* with the name of the data file.

Example 11-4   Enabling Shadow Lost Write Protection for a Tablespace

This example enables lost write protection for the `tbsu1` tablespace.

```
ALTER TABLESPACE tbsu1 ENABLE LOST WRITE PROTECTION;
```

Example 11-5 Enabling Shadow Lost Write Protection for a Data File Used by a CDB Root

This example enables shadow lost write protection for the `dfile1.df` data file.

```
ALTER DATABASE DATAFILE 'dfile1.df' ENABLE LOST WRITE PROTECTION;
```

Example 11-6 Enabling Shadow Lost Write Protection for a Data File Used by an Application Root, a PDB, or an Application PDB

This example enables shadow lost write protection for the `dfile2.df` data file.

```
ALTER PLUGGABLE DATABASE DATAFILE 'dfile2.df' ENABLE LOST WRITE PROTECTION;
```

Related Topics

- Enabling Shadow Lost Write Protection for a Database
  To enable shadow lost write protection for a multitenant container database (CDB), use the `ALTER DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause. To enable shadow lost write protection for a pluggable database (PDB), use the `ALTER PLUGGABLE DATABASE` statement with the `ENABLE LOST WRITE PROTECTION` clause.
- Creating Shadow Tablespaces for Shadow Lost Write Protection
  To create a shadow tablespace for shadow lost write protection, issue a `CREATE BIGFILE TABLESPACE` statement with the `LOST WRITE PROTECTION` clause.

11.11.5 Disabling Shadow Lost Write Protection for a Database

To disable shadow lost write protection for a multitenant container database (CDB), issue an `ALTER DATABASE` statement with the `DISABLE LOST WRITE PROTECTION` clause. To disable shadow lost write protection for a pluggable database (PDB), issue an `ALTER PLUGGABLE DATABASE` statement with the `DISABLE LOST WRITE PROTECTION` clause.

When you disable shadow lost write protection for a database, no tablespaces or data files in the database can be protected by shadow lost write protection.

Note:

- Disabling shadow lost write protection does not remove the data in the existing shadow tablespace, but this data is no longer updated or checked. If you want to remove the data in the shadow tablespace, then you can drop the shadow tablespace using the `DROP TABLESPACE` statement with the `INCLUDING CONTENTS` clause.
- Enabling or disabling shadow lost write protection for a CDB root does not impact the shadow lost write protection for the PDBs.

To disable shadow lost write protection for a database:

1. In SQL*Plus, connect to a user with the required privileges:
   - In a CDB root, connect as a user with `ALTER DATABASE` system privilege.

- In an application root, PDB, or application PDB, connect as a user with `ALTER PLUGGABLE DATABASE` system privilege.

2. Do one of the following:
   - For a CDB root, issue an `ALTER DATABASE` statement with the `DISABLE LOST WRITE PROTECTION` clause.
   - For an application root, PDB, or application PDB, issue an `ALTER PLUGGABLE DATABASE` statement with the `DISABLE LOST WRITE PROTECTION` clause.

Example 11-7 Disabling Shadow Lost Write Protection for a CDB Root

```
ALTER DATABASE DISABLE LOST WRITE PROTECTION;
```

Example 11-8 Disabling Shadow Lost Write Protection for a PDB

```
ALTER PLUGGABLE DATABASE DISABLE LOST WRITE PROTECTION;
```

Related Topics

- Removing or Suspending Shadow Lost Write Protection
  You can remove or suspend shadow lost write protection for a tablespace or a data file.

11.11.6 Removing or Suspending Shadow Lost Write Protection

You can remove or suspend shadow lost write protection for a tablespace or a data file.

When shadow lost write protection is no longer needed for a tablespace or data file, you can choose one of the following options:

- You can remove shadow lost write protection. This option deletes tracking information for the tablespace or data file from shadow tablespaces. This option also stops the collection of new lost write information for the tablespace or data file and stops checking for new lost writes for them.
- You can suspend shadow lost write protection. This option stops the collection of new lost write information for the tablespace or data file and stops checking for new lost writes for them. However, the old lost write information remains in the shadow tablespace. If shadow lost write protection is re-enabled for the tablespace or data file, then the old lost write information can be used for them.

When you remove or suspend shadow lost write protection for a tablespace, shadow lost write protection is removed or suspended for all of the data files of the tablespace.

To remove or suspend shadow lost write protection for a tablespace or a data file:

1. In SQL*Plus, connect to the database as a user with the required privileges:
   - If you are removing or suspending shadow lost write protection for a tablespace, then connect as a user with `ALTER TABLESPACE` privilege.
   - If you are removing or suspending shadow lost write protection for a data file used by a CDB root, then connect as a user with `ALTER DATABASE` privilege.
   - If you are removing or suspending shadow lost write protection for a data file used by an application root, PDB, or application PDB, then connect as a user with `ALTER PLUGGABLE DATABASE` privilege.
2. Perform one of the following actions:

- To remove or suspend shadow lost write protection for a tablespace, issue an `ALTER TABLESPACE` statement with the `REMOVE LOST WRITE PROTECTION` clause or the `SUSPEND LOST WRITE PROTECTION` clause, respectively.
- To remove or suspend shadow lost write protection for a data file that is used by a CDB root, issue an `ALTER DATABASE DATAFILE` *`data_file_name`* statement with the `REMOVE LOST WRITE PROTECTION` clause or `SUSPEND LOST WRITE PROTECTION` clause, respectively, and replace *`data_file_name`* with the name of the data file.
- To remove or suspend shadow lost write protection for a data file that is used by an application root, PDB, or application PDB, issue an `ALTER PLUGGABLE DATABASE DATAFILE` *`data_file_name`* statement with the `REMOVE LOST WRITE PROTECTION` clause or `SUSPEND LOST WRITE PROTECTION` clause, respectively, and replace *`data_file_name`* with the name of the data file.

Example 11-9 Removing Shadow Lost Write Protection for a Tablespace

This example removes lost write protection for the `tbsu1` tablespace.

```
ALTER TABLESPACE tbsu1 REMOVE LOST WRITE PROTECTION;
```

Example 11-10 Suspending Shadow Lost Write Protection for a Data File Used by a CDB Root

This example suspends shadow lost write protection for the `dfile1.df` data file.

```
ALTER DATABASE DATAFILE 'dfile1.df' SUSPEND LOST WRITE PROTECTION;
```

Example 11-11 Removing Shadow Lost Write Protection for a Data File Used by a PDB

This example removes shadow lost write protection for the `dfile2.df` data file, which is used by a PDB.

```
ALTER PLUGGABLE DATABASE DATAFILE 'dfile2.df' SUSPEND LOST WRITE PROTECTION;
```

11.11.7 Dropping a Shadow Tablespace

You can drop a shadow tablespace using the `DROP TABLESPACE` statement. If you use the `DROP TABLESPACE` statement with the `INCLUDING CONTENTS` clause, then the shadow tablespace is dropped along with its contents. If you use the `DROP TABLESPACE` statement without the `INCLUDING CONTENTS` clause, then before dropping the shadow tablespace, its contents are moved to another shadow tablespace, if it exists and has a sufficient free space.

11.12 Managing the SYSAUX Tablespace

The `SYSAUX` tablespace was installed as an auxiliary tablespace to the `SYSTEM` tablespace when you created your database. Some database components that formerly created and used separate tablespaces now occupy the `SYSAUX` tablespace.

If the `SYSAUX` tablespace becomes unavailable, core database functionality will remain operational. The database features that use the `SYSAUX` tablespace could fail, or function with limited capability.

- Monitoring Occupants of the SYSAUX Tablespace
  You can monitor the occupants of the `SYSAUX` tablespace.

- Moving Occupants Out Of or Into the SYSAUX Tablespace
  The `V$SYSAUX_OCCUPANTS` view provides a move procedure for each occupant of the `SYSAUX` tablespace.
- Controlling the Size of the SYSAUX Tablespace
  The `SYSAUX` tablespace is occupied by several database components, and its total size is governed by the space consumed by those components. The space consumed by the components, in turn, depends on which features or functionality are being used and on the nature of the database workload.

11.12.1 Monitoring Occupants of the SYSAUX Tablespace

You can monitor the occupants of the `SYSAUX` tablespace.

The list of registered occupants of the `SYSAUX` tablespace are discussed in "*Oracle Database SQL Language Reference*". These components can use the `SYSAUX` tablespace, and their installation provides the means of establishing their occupancy of the `SYSAUX` tablespace.

To monitor the occupants of the `SYSAUX` tablespace:

- Query the `V$SYSAUX_OCCUPANTS` view.

This view lists the following information about the occupants of the `SYSAUX` tablespace:

- Name of the occupant
- Occupant description
- Schema name
- Move procedure
- Current space usage

View information is maintained by the occupants.

See Also:

Oracle Database Reference for a detailed description of the `V$SYSAUX_OCCUPANTS` view

11.12.2 Moving Occupants Out Of or Into the SYSAUX Tablespace

The `V$SYSAUX_OCCUPANTS` view provides a move procedure for each occupant of the `SYSAUX` tablespace.

You will have an option at component install time to specify that you do not want the component to reside in `SYSAUX`. Also, if you later decide that the component should be relocated to a designated tablespace, you can use the move procedure for that component, as specified in the `V$SYSAUX_OCCUPANTS` view, to perform the move.

The move procedure also lets you move a component from another tablespace into the `SYSAUX` tablespace.

11.12.3 Controlling the Size of the SYSAUX Tablespace

The `SYSAUX` tablespace is occupied by several database components, and its total size is governed by the space consumed by those components. The space consumed by the components, in turn, depends on which features or functionality are being used and on the nature of the database workload.

The largest portion of the `SYSAUX` tablespace is occupied by the Automatic Workload Repository (AWR). The space consumed by the AWR is determined by several factors, including the number of active sessions in the system at any given time, the snapshot interval, and the historical data retention period. A typical system with an average of 10 concurrent active sessions may require approximately 200 MB to 300 MB of space for its AWR data. You can control the size of the AWR by changing the snapshot interval and historical data retention period.

Another major occupant of the `SYSAUX` tablespace is the embedded Oracle Enterprise Manager Cloud Control repository. This repository is used by Cloud Control to store its metadata. The size of this repository depends on database activity and on configuration-related information stored in the repository.

Other database components in the `SYSAUX` tablespace will grow in size only if their associated features (for example, Oracle Text and Oracle Streams) are in use. If the features are not used, then these components do not have any significant effect on the size of the `SYSAUX` tablespace.

The following table provides guidelines on sizing the `SYSAUX` tablespace based on the system configuration and expected load.

Parameter/ Recommendation	Small	Medium	Large
Number of CPUs	2	8	32
Number of concurrently active sessions	10	20	100
Number of user objects: tables and indexes	500	5,000	50,000
Estimated `SYSAUX` size at steady state with default configuration	500 MB	2 GB	5 GB

11.13 Correcting Problems with Locally Managed Tablespaces

Oracle Database includes aids for correcting problems with locally managed tablespaces.

- Diagnosing and Repairing Locally Managed Tablespace Problems
  Oracle Database includes the `DBMS_SPACE_ADMIN` package, which is a collection of aids for diagnosing and repairing problems in locally managed tablespaces.
- Scenario 1: Fixing Bitmap When Allocated Blocks are Marked Free (No Overlap)
  The `TABLESPACE_VERIFY` procedure discovers that a segment has allocated blocks that are marked free in the bitmap, but no overlap between segments is reported.
- Scenario 2: Dropping a Corrupted Segment
  You cannot drop a segment because the bitmap has segment blocks marked "free". The system has automatically marked the segment corrupted.

- Scenario 3: Fixing Bitmap Where Overlap is Reported
  The TABLESPACE_VERIFY procedure reports some overlapping. Some of the real data must be sacrificed based on previous internal errors.
- Scenario 4: Correcting Media Corruption of Bitmap Blocks
  A set of bitmap blocks has media corruption.
- Scenario 5: Migrating from a Dictionary-Managed to a Locally Managed Tablespace
  Use the TABLESPACE_MIGRATE_TO_LOCAL procedure to migrate a dictionary-managed tablespace to a locally managed tablespace.

11.13.1 Diagnosing and Repairing Locally Managed Tablespace Problems

Oracle Database includes the DBMS_SPACE_ADMIN package, which is a collection of aids for diagnosing and repairing problems in locally managed tablespaces.

DBMS_SPACE_ADMIN Package Procedures

The following table lists the DBMS_SPACE_ADMIN package procedures. See *Oracle Database PL/SQL Packages and Types Reference* for details on each procedure.

Procedure	Description
ASSM_SEGMENT_VERIFY	Verifies the integrity of segments created in tablespaces that have automatic segment space management enabled. Outputs a dump file named *sid*_ora_*process_id*.trc to the location that corresponds to the Diag Trace entry in the V$DIAG_INFO view. Use SEGMENT_VERIFY for tablespaces with manual segment space management.
ASSM_TABLESPACE_VERIFY	Verifies the integrity of tablespaces that have automatic segment space management enabled. Outputs a dump file named *sid*_ora_*process_id*.trc to the location that corresponds to the Diag Trace entry in the V$DIAG_INFO view. Use TABLESPACE_VERIFY for tablespaces with manual segment space management.
DROP_EMPTY_SEGMENTS	Drops segments from empty tables or table partitions and dependent objects
MATERIALIZE_DEFERRED_SEGMENTS	Materializes segments for tables and table partitions with deferred segment creation and their dependent objects.
SEGMENT_CORRUPT	Marks the segment corrupt or valid so that appropriate error recovery can be done
SEGMENT_DROP_CORRUPT	Drops a segment currently marked corrupt (without reclaiming space)
SEGMENT_DUMP	Dumps the segment header and bitmap blocks of a specific segment to a dump file named *sid*_ora_*process_id*.trc in the location that corresponds to the Diag Trace entry in the V$DIAG_INFO view. Provides an option to select a slightly abbreviated dump, which includes segment header and includes bitmap block summaries, without percent-free states of each block.
SEGMENT_VERIFY	Verifies the consistency of the extent map of the segment
TABLESPACE_FIX_BITMAPS	Marks the appropriate DBA range (extent) as free or used in bitmap

Procedure	Description
`TABLESPACE_FIX_SEGMENT_STATES`	Fixes the state of the segments in a tablespace in which migration was stopped
`TABLESPACE_MIGRATE_FROM_LOCAL`	Migrates a locally managed tablespace to dictionary-managed tablespace
`TABLESPACE_MIGRATE_TO_LOCAL`	Migrates a dictionary-managed tablespace to a locally managed tablespace
`TABLESPACE_REBUILD_BITMAPS`	Rebuilds the appropriate bitmaps
`TABLESPACE_REBUILD_QUOTAS`	Rebuilds quotas for a specific tablespace
`TABLESPACE_RELOCATE_BITMAPS`	Relocates the bitmaps to the specified destination
`TABLESPACE_VERIFY`	Verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized

The following scenarios describe typical situations in which you can use the `DBMS_SPACE_ADMIN` package to diagnose and resolve problems.

Note:

Some of these procedures can result in lost and unrecoverable data if not used properly. You should work with Oracle Support Services if you have doubts about these procedures.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for details about the `DBMS_SPACE_ADMIN` package
- "Viewing ADR Locations with the V$DIAG_INFO View"

11.13.2 Scenario 1: Fixing Bitmap When Allocated Blocks are Marked Free (No Overlap)

The `TABLESPACE_VERIFY` procedure discovers that a segment has allocated blocks that are marked free in the bitmap, but no overlap between segments is reported.

In this scenario, perform the following tasks:

1. Call the `SEGMENT_DUMP` procedure to dump the ranges that the administrator allocated to the segment.
2. For each range, call the `TABLESPACE_FIX_BITMAPS` procedure with the `TABLESPACE_EXTENT_MAKE_USED` option to mark the space as used.
3. Call `TABLESPACE_REBUILD_QUOTAS` to rebuild quotas.

11.13.3 Scenario 2: Dropping a Corrupted Segment

You cannot drop a segment because the bitmap has segment blocks marked "free". The system has automatically marked the segment corrupted.

In this scenario, perform the following tasks:

1. Call the `SEGMENT_VERIFY` procedure with the `SEGMENT_VERIFY_EXTENTS_GLOBAL` option. If no overlaps are reported, then proceed with steps 2 through 5.
2. Call the `SEGMENT_DUMP` procedure to dump the DBA ranges allocated to the segment.
3. For each range, call `TABLESPACE_FIX_BITMAPS` with the `TABLESPACE_EXTENT_MAKE_FREE` option to mark the space as free.
4. Call `SEGMENT_DROP_CORRUPT` to drop the `SEG$` entry.
5. Call `TABLESPACE_REBUILD_QUOTAS` to rebuild quotas.

11.13.4 Scenario 3: Fixing Bitmap Where Overlap is Reported

The `TABLESPACE_VERIFY` procedure reports some overlapping. Some of the real data must be sacrificed based on previous internal errors.

After choosing the object to be sacrificed, in this case say, table `t1`, perform the following tasks:

1. Make a list of all objects that `t1` overlaps.
2. Drop table `t1`. If necessary, follow up by calling the `SEGMENT_DROP_CORRUPT` procedure.
3. Call the `SEGMENT_VERIFY` procedure on all objects that `t1` overlapped. If necessary, call the `TABLESPACE_FIX_BITMAPS` procedure to mark appropriate bitmap blocks as used.
4. Rerun the `TABLESPACE_VERIFY` procedure to verify that the problem is resolved.

11.13.5 Scenario 4: Correcting Media Corruption of Bitmap Blocks

A set of bitmap blocks has media corruption.

In this scenario, perform the following tasks:

1. Call the `TABLESPACE_REBUILD_BITMAPS` procedure, either on all bitmap blocks, or on a single block if only one is corrupt.
2. Call the `TABLESPACE_REBUILD_QUOTAS` procedure to rebuild quotas.
3. Call the `TABLESPACE_VERIFY` procedure to verify that the bitmaps are consistent.

11.13.6 Scenario 5: Migrating from a Dictionary-Managed to a Locally Managed Tablespace

Use the `TABLESPACE_MIGRATE_TO_LOCAL` procedure to migrate a dictionary-managed tablespace to a locally managed tablespace.

This operation is done online, but space management operations are blocked until the migration has been completed. Therefore, you can read or modify data while the migration is in progress, but if you are loading a large amount of data that requires the allocation of additional extents, then the operation may be blocked.

Assume that the database block size is 2K and the existing extent sizes in tablespace `tbs_1` are 10, 50, and 10,000 blocks (used, used, and free). The `MINIMUM EXTENT` value is 20K (10 blocks). Allow the system to choose the bitmap allocation unit. The value of 10 blocks is chosen, because it is the highest common denominator and does not exceed `MINIMUM EXTENT`.

The statement to convert `tbs_1` to a locally managed tablespace is as follows:

```
EXEC DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('tbs_1');
```

If you choose to specify an allocation unit size, it must be a factor of the unit size calculated by the system.

11.14 Migrating the SYSTEM Tablespace to a Locally Managed Tablespace

Use the `DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL` procedure to migrate the `SYSTEM` tablespace from dictionary-managed to locally managed.

Before performing the migration the following conditions must be met:

- The database has a default temporary tablespace that is not `SYSTEM`.
- There are no rollback segments in the dictionary-managed tablespace.
- There is at least one online rollback segment in a locally managed tablespace, or if using automatic undo management, an undo tablespace is online.
- All tablespaces other than the tablespace containing the undo space (that is, the tablespace containing the rollback segment or the undo tablespace) are in read-only mode.
- The `SYSAUX` tablespace is offline.
- The system is in restricted mode.
- There is a cold backup of the database.

All of these conditions, except for the cold backup, are enforced by the `TABLESPACE_MIGRATE_TO_LOCAL` procedure.

The following statement performs the migration:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('SYSTEM');
```

Note:

After the `SYSTEM` tablespace is migrated to locally managed, any dictionary-managed tablespaces in the database cannot be made read/write. If you want to use the dictionary-managed tablespaces in read/write mode, then Oracle recommends that you first migrate these tablespaces to locally managed before migrating the `SYSTEM` tablespace.

11.15 Viewing Information About Tablespaces

Oracle Database includes data dictionary views that you can query for information about tablespaces.

- Tablespace Data Dictionary Views
  The following data dictionary and dynamic performance views provide useful information about the tablespaces of a database.
- Example 1: Listing Tablespaces and Default Storage Parameters
  You can query the DBA_TABLESPACES view to list the names and default storage parameters.
- Example 2: Listing the Data Files and Associated Tablespaces of a Database
  You can query the DBA_DATA_FILES view to list the names, sizes, and associated tablespaces of a database.
- Example 3: Displaying Statistics for Free Space (Extents) of Each Tablespace
  You can query the DBA_FREE_SPACE view to display statistics about free extents and coalescing activity for each tablespace in the database.

11.15.1 Tablespace Data Dictionary Views

The following data dictionary and dynamic performance views provide useful information about the tablespaces of a database.

View	Description
V$TABLESPACE	Name and number of all tablespaces from the control file.
V$ENCRYPTED_TABLESPACES	Name and encryption algorithm of all encrypted tablespaces.
DBA_TABLESPACES, USER_TABLESPACES	Descriptions of all (or user accessible) tablespaces.
DBA_TABLESPACE_GROUPS	Displays the tablespace groups and the tablespaces that belong to them.
DBA_SEGMENTS, USER_SEGMENTS	Information about segments within all (or user accessible) tablespaces.
DBA_EXTENTS, USER_EXTENTS	Information about data extents within all (or user accessible) tablespaces.
DBA_FREE_SPACE, USER_FREE_SPACE	Information about free extents within all (or user accessible) tablespaces.
DBA_TEMP_FREE_SPACE	Displays the total allocated and free space in each temporary tablespace.
V$DATAFILE	Information about all data files, including tablespace number of owning tablespace.
V$TEMPFILE	Information about all temp files, including tablespace number of owning tablespace.
DBA_DATA_FILES	Shows files (data files) belonging to tablespaces.
DBA_TEMP_FILES	Shows files (temp files) belonging to temporary tablespaces.
V$TEMP_EXTENT_MAP	Information for all extents in all locally managed temporary tablespaces.

View	Description
V$TEMP_EXTENT_POOL	For locally managed temporary tablespaces: the state of temporary space cached and used for by each instance.
V$TEMP_SPACE_HEADER	Shows space used/free for each temp file.
DBA_USERS	Default and temporary tablespaces for all users.
DBA_TS_QUOTAS	Lists tablespace quotas for all users.
V$SORT_SEGMENT	Information about every sort segment in a given instance. The view is only updated when the tablespace is of the TEMPORARY type.
V$TEMPSEG_USAGE	Describes temporary (sort) segment usage by user for temporary or permanent tablespaces.

11.15.2 Example 1: Listing Tablespaces and Default Storage Parameters

You can query the DBA_TABLESPACES view to list the names and default storage parameters.

To list the names and default storage parameters of all tablespaces in a database, use the following query on the DBA_TABLESPACES view:

```
SELECT TABLESPACE_NAME "TABLESPACE",
   INITIAL_EXTENT "INITIAL_EXT",
   NEXT_EXTENT "NEXT_EXT",
   MIN_EXTENTS "MIN_EXT",
   MAX_EXTENTS "MAX_EXT",
   PCT_INCREASE
   FROM DBA_TABLESPACES;

TABLESPACE  INITIAL_EXT  NEXT_EXT  MIN_EXT   MAX_EXT    PCT_INCREASE
----------  -----------  --------  -------   -------    ------------
RBS             1048576   1048576        2        40               0
SYSTEM           106496    106496        1        99               1
TEMP             106496    106496        1        99               0
TESTTBS           57344     16384        2        10               1
USERS             57344     57344        1        99               1
```

11.15.3 Example 2: Listing the Data Files and Associated Tablespaces of a Database

You can query the DBA_DATA_FILES view to list the names, sizes, and associated tablespaces of a database.

To list the names, sizes, and associated tablespaces of a database, enter the following query on the DBA_DATA_FILES view:

```
SELECT  FILE_NAME, BLOCKS, TABLESPACE_NAME
   FROM DBA_DATA_FILES;

FILE_NAME                                 BLOCKS  TABLESPACE_NAME
------------                          ----------  -------------------
/U02/ORACLE/IDDB3/DBF/RBS01.DBF             1536  RBS
/U02/ORACLE/IDDB3/DBF/SYSTEM01.DBF          6586  SYSTEM
/U02/ORACLE/IDDB3/DBF/TEMP01.DBF            6400  TEMP
/U02/ORACLE/IDDB3/DBF/TESTTBS01.DBF         6400  TESTTBS
/U02/ORACLE/IDDB3/DBF/USERS01.DBF            384  USERS
```

11.15.4 Example 3: Displaying Statistics for Free Space (Extents) of Each Tablespace

You can query the `DBA_FREE_SPACE` view to display statistics about free extents and coalescing activity for each tablespace in the database.

To produce statistics about free extents and coalescing activity for each tablespace in the database, enter the following query:

```
SELECT TABLESPACE_NAME "TABLESPACE", FILE_ID,
   COUNT(*)    "PIECES",
   MAX(blocks) "MAXIMUM",
   MIN(blocks) "MINIMUM",
   AVG(blocks) "AVERAGE",
   SUM(blocks) "TOTAL"
   FROM DBA_FREE_SPACE
GROUP BY TABLESPACE_NAME, FILE_ID;

TABLESPACE    FILE_ID  PIECES   MAXIMUM    MINIMUM  AVERAGE    TOTAL
----------    -------  ------   -------    -------  -------   ------
RBS                 2       1       955        955      955      955
SYSTEM              1       1       119        119      119      119
TEMP                4       1      6399       6399     6399     6399
TESTTBS             5       5      6364          3     1278     6390
USERS               3       1       363        363      363      363
```

`PIECES` shows the number of free space extents in the tablespace file, `MAXIMUM` and `MINIMUM` show the largest and smallest contiguous area of space in database blocks, `AVERAGE` shows the average size in blocks of a free space extent, and `TOTAL` shows the amount of free space in each tablespace file in blocks. This query is useful when you are going to create a new object or you know that a segment is about to extend, and you want to ensure that there is enough space in the containing tablespace.

12 Managing Data Files and Temp Files

You can manage data files and temp files by performing tasks such as creating them, altering them, and dropping them.

Note:

Temp files are a special class of data files that are associated only with temporary tablespaces. Information in this chapter applies to both data files and temp files except where differences are noted. Temp files are further described in "Creating a Locally Managed Temporary Tablespace"

- Guidelines for Managing Data Files
  You can follow guidelines for managing data files.
- Creating Data Files and Adding Data Files to a Tablespace
  You can create data files and associate them with a tablespace using several different SQL statements.
- Changing Data File Size
  You can alter the size of a data file. For example, you can increase the size of one or more data files when more space is needed in the database.
- Altering Data File Availability
  You must alter data file availability to perform certain tasks, such as performing an offline backup of a data file or relocating an offline data file.
- Renaming and Relocating Data Files
  You can rename online or offline data files to either change their names or relocate them.
- Dropping Data Files
  You use the `DROP DATAFILE` and `DROP TEMPFILE` clauses of the `ALTER TABLESPACE` statement to drop a single data file or temp file.
- Verifying Data Blocks in Data Files
  To configure the database to use checksums to verify data blocks, set the initialization parameter `DB_BLOCK_CHECKSUM` to `TYPICAL` (the default).
- Copying Files Using the Database Server
  You can use the `DBMS_FILE_TRANSFER` package to copy a file within a database or transfer a file between databases.
- Mapping Files to Physical Devices
  In an environment where data files are file system files, it is relatively straight forward to see the association between a tablespace and the underlying device. Oracle Database provides views, such as `DBA_TABLESPACES`, `DBA_DATA_FILES`, and `V$DATAFILE`, that provide a mapping of files onto devices. These mappings, along with device statistics can be used to evaluate I/O performance.

- Data Files Data Dictionary Views
  A set of data dictionary views provides useful information about the data files of a database.

See Also:

- Using Oracle Managed Files for information about creating data files and temp files that are both created and managed by the Oracle Database server
- *Oracle Database Concepts*

12.1 Guidelines for Managing Data Files

You can follow guidelines for managing data files.

- About Data Files
  Data files are physical files of the operating system that store the data of all logical structures in the database. They must be explicitly created for each tablespace.
- Determine the Number of Data Files
  You must determine the number of data files for your database.
- Determine the Size of Data Files
  When creating a tablespace, you should estimate the potential size of database objects and create sufficient data files.
- Place Data Files Appropriately
  Tablespace location is determined by the physical location of the data files that constitute that tablespace. Use the hardware resources of your computer appropriately.
- Store Data Files Separate from Redo Log Files
  Data files should not be stored on the same disk drive that stores the database redo log files. If the data files and redo log files are stored on the same disk drive and that disk drive fails, the files cannot be used in your database recovery procedures.

12.1.1 About Data Files

Data files are physical files of the operating system that store the data of all logical structures in the database. They must be explicitly created for each tablespace.

Oracle Database assigns each data file two associated file numbers, an absolute file number and a relative file number, that are used to uniquely identify it. These numbers are described in the following table:

Type of File Number	Description
Absolute	Uniquely identifies a data file *in the database*. This file number can be used in many SQL statements that reference data files in place of using the file name. The absolute file number can be found in the `FILE#` column of the `V$DATAFILE` or `V$TEMPFILE` view, or in the `FILE_ID` column of the `DBA_DATA_FILES` or `DBA_TEMP_FILES` view.

Type of File Number	Description
Relative	Uniquely identifies a data file *within a tablespace*. For small and medium size databases, relative file numbers usually have the same value as the absolute file number. However, when the number of data files in a database exceeds a threshold (typically 1023), the relative file number differs from the absolute file number. In a bigfile tablespace, the relative file number is always 1024 (4096 on OS/390 platform).

12.1.2 Determine the Number of Data Files

You must determine the number of data files for your database.

- About Determining the Number of Data Files
  At least one data file is required for the `SYSTEM` and `SYSAUX` tablespaces of a database. Your database should contain several other tablespaces with their associated data files or temp files. The number of data files that you anticipate creating for your database can affect the settings of initialization parameters and the specification of `CREATE DATABASE` statement clauses.
- Determine a Value for the DB_FILES Initialization Parameter
  When starting an Oracle Database instance, the `DB_FILES` initialization parameter indicates the amount of SGA space to reserve for data file information and thus, the maximum number of data files that can be created for the instance.
- Consider Possible Limitations When Adding Data Files to a Tablespace
  There are some limitations to consider when adding data files to a tablespace.
- Consider the Performance Impact of the Number of Data Files
  The number of data files contained in a tablespace, and ultimately the database, can have an impact upon performance.

12.1.2.1 About Determining the Number of Data Files

At least one data file is required for the `SYSTEM` and `SYSAUX` tablespaces of a database. Your database should contain several other tablespaces with their associated data files or temp files. The number of data files that you anticipate creating for your database can affect the settings of initialization parameters and the specification of `CREATE DATABASE` statement clauses.

Be aware that your operating system might impose limits on the number of data files contained in your Oracle Database. Also consider that the number of data files, and how and where they are allocated can affect the performance of your database.

Note:

One means of controlling the number of data files in your database and simplifying their management is to use bigfile tablespaces. Bigfile tablespaces comprise a single, very large data file and are especially useful in ultra large databases and where a logical volume manager is used for managing operating system files. Bigfile tablespaces are discussed in "Bigfile Tablespaces".

Consider the following guidelines when determining the number of data files for your database.

12.1.2.2 Determine a Value for the DB_FILES Initialization Parameter

When starting an Oracle Database instance, the `DB_FILES` initialization parameter indicates the amount of SGA space to reserve for data file information and thus, the maximum number of data files that can be created for the instance.

This limit applies for the life of the instance. You can change the value of `DB_FILES` (by changing the initialization parameter setting), but the new value does not take effect until you shut down and restart the instance.

When determining a value for `DB_FILES`, take the following into consideration:

- If the value of `DB_FILES` is too low, you cannot add data files beyond the `DB_FILES` limit without first shutting down the database.
- If the value of `DB_FILES` is too high, memory is unnecessarily consumed.

12.1.2.3 Consider Possible Limitations When Adding Data Files to a Tablespace

There are some limitations to consider when adding data files to a tablespace.

You can add data files to traditional smallfile tablespaces, subject to the following limitations:

- Operating systems often impose a limit on the number of files a process can open simultaneously. More data files cannot be created when the operating system limit of open files is reached.
- Operating systems impose limits on the number and size of data files.
- The database imposes a maximum limit on the number of data files for any Oracle Database opened by any instance. This limit is operating system specific.
- You cannot exceed the number of data files specified by the `DB_FILES` initialization parameter.
- When you issue `CREATE DATABASE` or `CREATE CONTROLFILE` statements, the `MAXDATAFILES` parameter specifies an initial size of the data file portion of the control file. However, if you attempt to add a new file whose number is greater than `MAXDATAFILES`, but less than or equal to `DB_FILES`, the control file will expand automatically so that the data files section can accommodate more files.

12.1.2.4 Consider the Performance Impact of the Number of Data Files

The number of data files contained in a tablespace, and ultimately the database, can have an impact upon performance.

Oracle Database allows more data files in the database than the operating system defined limit. The database DBW*n* processes can open all online data files. Oracle Database is capable of treating open file descriptors as a cache, automatically closing files when the number of open file descriptors reaches the operating system-defined limit. This can have a negative performance impact. When possible, adjust the operating system limit on open file descriptors so that it is larger than the number of online data files in the database.

See Also:

- Your operating system specific Oracle documentation for more information on operating system limits
- *Oracle Database SQL Language Reference* for more information about the `MAXDATAFILES` parameter of the `CREATE DATABASE` or `CREATE CONTROLFILE` statement

12.1.3 Determine the Size of Data Files

When creating a tablespace, you should estimate the potential size of database objects and create sufficient data files.

Later, if needed, you can create additional data files and add them to a tablespace to increase the total amount of disk space allocated to it, and consequently the database. Preferably, place data files on multiple devices to ensure that data is spread evenly across all devices.

12.1.4 Place Data Files Appropriately

Tablespace location is determined by the physical location of the data files that constitute that tablespace. Use the hardware resources of your computer appropriately.

For example, if several disk drives are available to store the database, consider placing potentially contending data files on separate disks. This way, when users query information, both disk drives can work simultaneously, retrieving data at the same time.

See Also:

Oracle Database Performance Tuning Guide for information about I/O and the placement of data files

12.1.5 Store Data Files Separate from Redo Log Files

Data files should not be stored on the same disk drive that stores the database redo log files. If the data files and redo log files are stored on the same disk drive and that disk drive fails, the files cannot be used in your database recovery procedures.

If you multiplex your redo log files, then the likelihood of losing all of your redo log files is low, so you can store data files on the same drive as some redo log files.

12.2 Creating Data Files and Adding Data Files to a Tablespace

You can create data files and associate them with a tablespace using several different SQL statements.

In all cases, you can either specify the file specifications for the data files being created, or you can use the Oracle Managed Files feature to create files that are created and managed by the database server. The table includes a brief description of the statement, as used to create data files, and references the section of this book where use of the statement is specifically described:

SQL Statement	Description	Additional Information
`CREATE TABLESPACE`	Creates a tablespace and the data files that comprise it	"Creating Tablespaces"
`CREATE TEMPORARY TABLESPACE`	Creates a locally-managed temporary tablespace and the *tempfiles* (temp files are a special kind of data file) that comprise it	"Creating a Locally Managed Temporary Tablespace"
`ALTER TABLESPACE ... ADD DATAFILE`	Creates and adds a data file to a tablespace	"Altering a Locally Managed Tablespace"
`ALTER TABLESPACE ... ADD TEMPFILE`	Creates and adds a temp file to a temporary tablespace	"Altering a Locally Managed Temporary Tablespace"
`CREATE DATABASE`	Creates a database and associated data files	*Oracle Multitenant Administrator's Guide*
`ALTER DATABASE ... CREATE DATAFILE`	Creates a new empty data file in place of an old one--useful to re-create a data file that was lost with no backup.	See *Oracle Database Backup and Recovery User's Guide*.

If you add new data files to a tablespace and do not fully specify the file names, the database creates the data files in the default database directory or the current directory, depending upon your operating system. Oracle recommends you always specify a fully qualified name for a data file. Unless you want to reuse existing files, make sure the new file names do not conflict with other files. Old files that have been previously dropped will be overwritten.

If a statement that creates a data file fails, the database removes any created operating system files. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands.

12.3 Changing Data File Size

You can alter the size of a data file. For example, you can increase the size of one or more data files when more space is needed in the database.

- Enabling and Disabling Automatic Extension for a Data File
  You can create data files or alter existing data files so that they automatically increase in size when more space is needed in the database. The file size increases in specified increments up to a specified maximum.
- Manually Resizing a Data File
  You can manually increase or decrease the size of a data file using the `ALTER DATABASE` statement.

12.3.1 Enabling and Disabling Automatic Extension for a Data File

You can create data files or alter existing data files so that they automatically increase in size when more space is needed in the database. The file size increases in specified increments up to a specified maximum.

Setting your data files to extend automatically provides these advantages:

- Reduces the need for immediate intervention when a tablespace runs out of space

- Ensures applications will not halt or be suspended because of failures to allocate extents

You can specify automatic file extension by specifying an `AUTOEXTEND ON` clause when you create data files using the following SQL statements:

- `CREATE DATABASE`
- `ALTER DATABASE`
- `CREATE TABLESPACE`
- `ALTER TABLESPACE`

To enable or disable automatic extension for a data file:

1. Determine whether a data file is auto-extensible by querying the `DBA_DATA_FILES` view and examining the `AUTOEXTENSIBLE` column.
2. Enable or disable automatic file extension for existing data files, or manually resize a data file, using the `ALTER DATABASE` statement with the `AUTOEXTEND` clause. For a bigfile tablespace, use the `ALTER TABLESPACE` statement with the `AUTOEXTEND` clause.

The following example enables automatic extension for a data file added to the `users` tablespace:

```
ALTER TABLESPACE users
    ADD DATAFILE '/u02/oracle/rbdb1/users03.dbf' SIZE 10M
      AUTOEXTEND ON
      NEXT 512K
      MAXSIZE 250M;
```

The value of `NEXT` is the minimum size of the increments added to the file when it extends. The value of `MAXSIZE` is the maximum size to which the file can automatically extend.

The next example disables the automatic extension for the data file.

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf'
    AUTOEXTEND OFF;
```

See Also:

Oracle Database SQL Language Reference for more information about the SQL statements for creating or altering data files

12.3.2 Manually Resizing a Data File

You can manually increase or decrease the size of a data file using the `ALTER DATABASE` statement.

Therefore, you can add more space to your database without adding more data files. This is beneficial if you are concerned about reaching the maximum number of data files allowed in your database.

For a bigfile tablespace, you can use the `ALTER TABLESPACE` statement to resize a data file. You are not allowed to add a data file to a bigfile tablespace.

Manually reducing the sizes of data files enables you to reclaim unused space in the database. This is useful for correcting errors in estimates of space requirements.

In the following example, assume that the data file /u02/oracle/rbdb1/stuff01.dbf has extended up to 250M. However, because its tablespace now stores smaller objects, the data file can be reduced in size.

The following statement decreases the size of data file /u02/oracle/rbdb1/stuff01.dbf:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
   RESIZE 100M;
```

Note:

It is not always possible to decrease the size of a file to a specific value. It could be that the file contains data beyond the specified decreased size, in which case the database will return an error.

12.4 Altering Data File Availability

You must alter data file availability to perform certain tasks, such as performing an offline backup of a data file or relocating an offline data file.

- About Altering Data File Availability
  You can alter the availability of individual data files or temp files by taking them offline or bringing them online. Offline data files are unavailable to the database and cannot be accessed until they are brought back online.
- Bringing Data Files Online or Taking Offline in ARCHIVELOG Mode
  To bring an individual data file online or take an individual data file offline, issue the ALTER DATABASE statement and include the DATAFILE clause.
- Taking Data Files Offline in NOARCHIVELOG Mode
  To take a data file offline when the database is in NOARCHIVELOG mode, use the ALTER DATABASE statement with both the DATAFILE and OFFLINE FOR DROP clauses.
- Altering the Availability of All Data Files or Temp Files in a Tablespace
  Clauses of the ALTER TABLESPACE statement allow you to change the online or offline status of all of the data files or temp files within a tablespace.

12.4.1 About Altering Data File Availability

You can alter the availability of individual data files or temp files by taking them offline or bringing them online. Offline data files are unavailable to the database and cannot be accessed until they are brought back online.

Reasons for altering data file availability include the following:

- You want to perform an offline backup of a data file.
- You want to rename or relocate an offline data file. You can first take the data file offline or take the tablespace offline.
- The database has problems writing to a data file and automatically takes the data file offline. Later, after resolving the problem, you can bring the data file back online manually.
- A data file becomes missing or corrupted. You must take it offline before you can open the database.

The data files of a read-only tablespace can be taken offline or brought online, but bringing a file online does not affect the read-only status of the tablespace. You cannot write to the data file until the tablespace is returned to the read/write state.

Note:

You can make all data files of a tablespace temporarily unavailable by taking the tablespace itself offline. You *must* leave these files in the tablespace to bring the tablespace back online, although you can relocate or rename them following procedures similar to those shown in "Renaming and Relocating Data Files".

For more information, see "Taking Tablespaces Offline".

To take a data file offline or bring it online, you must have the `ALTER DATABASE` system privilege. To take all data files or temp files offline using the `ALTER TABLESPACE` statement, you must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege. In an Oracle Real Application Clusters environment, the database must be open in exclusive mode.

12.4.2 Bringing Data Files Online or Taking Offline in ARCHIVELOG Mode

To bring an individual data file online or take an individual data file offline, issue the `ALTER DATABASE` statement and include the `DATAFILE` clause.

The following statement brings the specified data file online:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

To take the same file offline, issue the following statement:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

Note:

To use this form of the `ALTER DATABASE` statement, the database must be in `ARCHIVELOG` mode. This requirement prevents you from accidentally losing the data file, since taking the data file offline while in `NOARCHIVELOG` mode is likely to result in losing the file.

12.4.3 Taking Data Files Offline in NOARCHIVELOG Mode

To take a data file offline when the database is in `NOARCHIVELOG` mode, use the `ALTER DATABASE` statement with both the `DATAFILE` and `OFFLINE FOR DROP` clauses.

- The `OFFLINE` keyword causes the database to mark the data file `OFFLINE`, whether or not it is corrupted, so that you can open the database.
- The `FOR DROP` keywords mark the data file for subsequent dropping. Such a data file can no longer be brought back online.

Note:

This operation does not actually drop the data file. It remains in the data dictionary, and you must drop it yourself using one of the following methods:

– An `ALTER TABLESPACE` ... `DROP DATAFILE` statement.

  After an `OFFLINE FOR DROP`, this method works for dictionary managed tablespaces only.

– A `DROP TABLESPACE` ... `INCLUDING CONTENTS AND DATAFILES` statement

– If the preceding methods fail, an operating system command to delete the data file. This is the least desirable method, as it leaves references to the data file in the data dictionary and control files.

The following statement takes the specified data file offline and marks it to be dropped:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE FOR DROP;
```

12.4.4 Altering the Availability of All Data Files or Temp Files in a Tablespace

Clauses of the `ALTER TABLESPACE` statement allow you to change the online or offline status of all of the data files or temp files within a tablespace.

Specifically, the statements that affect online/offline status are:

- `ALTER TABLESPACE` ... `DATAFILE` {`ONLINE`|`OFFLINE`}
- `ALTER TABLESPACE` ... `TEMPFILE` {`ONLINE`|`OFFLINE`}

You are required only to enter the tablespace name, not the individual data files or temp files. All of the data files or temp files are affected, but the online/offline status of the tablespace itself is not changed.

In most cases the preceding `ALTER TABLESPACE` statements can be issued whenever the database is mounted, even if it is not open. However, the database *must not* be open if the tablespace is the `SYSTEM` tablespace, an undo tablespace, or the default temporary tablespace. The `ALTER DATABASE DATAFILE` and `ALTER DATABASE TEMPFILE` statements also have `ONLINE/OFFLINE` clauses, however in those statements you must enter all of the file names for the tablespace.

The syntax is different from the `ALTER TABLESPACE...ONLINE|OFFLINE` statement that alters tablespace availability, because that is a different operation. The `ALTER TABLESPACE` statement takes data files offline as well as the tablespace, but it cannot be used to alter the status of a temporary tablespace or its temp file(s).

12.5 Renaming and Relocating Data Files

You can rename online or offline data files to either change their names or relocate them.

- Renaming and Relocating Online Data Files
  You can use the `ALTER DATABASE MOVE DATAFILE` SQL statement to rename or relocate online data files. This statement enables you to rename or relocate a data file while the database is open and users are accessing the data file.

- Renaming and Relocating Offline Data Files
  You can rename and relocate offline data files.

12.5.1 Renaming and Relocating Online Data Files

You can use the `ALTER DATABASE MOVE DATAFILE` SQL statement to rename or relocate online data files. This statement enables you to rename or relocate a data file while the database is open and users are accessing the data file.

When you rename or relocate online data files, the pointers to the data files, as recorded in the database control file, are changed. The files are also physically renamed or relocated at the operating system level.

You might rename or relocate online data files because you want to allow users to access the data files when you perform one of the following tasks:

- Move the data files from one type of storage to another
- Move data files that are accessed infrequently to lower cost storage
- Make a tablespace read-only and move its data files to write-once storage
- Move a database into Oracle Automatic Storage Management (Oracle ASM)

When you run the `ALTER DATABASE MOVE DATAFILE` statement and a file with the same name exists in the destination location, you can specify the `REUSE` option to overwrite the existing file. When `REUSE` is not specified, and a file with the same name exists in the destination location, the existing file is not overwritten, and the statement returns an error.

By default, when you run the `ALTER DATABASE MOVE DATAFILE` statement and specify a new location for a data file, the statement moves the data file. However, you can specify the `KEEP` option to retain the data file in the old location and copy it to the new location. In this case, the database only uses the data file in the new location when the statement completes successfully.

When you rename or relocate a data file with `ALTER DATABASE MOVE DATAFILE` statement, Oracle Database creates a copy of the data file when it is performing the operation. Ensure that there is adequate disk space for the original data file and the copy during the operation.

You can view the name, location, and online status of each data file by querying the `DBA_DATA_FILES` view.

Note:

- The `ALTER DATABASE MOVE DATAFILE` statement raises an error if the specified data file is offline.
- If you are using a standby database, then you can perform an online move data file operation independently on the primary and on the standby (either physical or logical). The standby is not affected when a data file is moved on the primary, and vice versa. See *Oracle Data Guard Concepts and Administration* for more information.
- A flashback operation does not relocate a moved data file to its previous location. If you move a data file online from one location to another and later flash back the database to a point in time before the move, then the data file remains in the new location, but the contents of the data file are changed to the contents at the time specified in the flashback. See *Oracle Database Backup and Recovery User's Guide* for more information about flashback database operations.
- When you relocate a data file on the Windows platform, the original data file might be retained in the old location, even when the `KEEP` option is omitted. In this case, the database only uses the data file in the new location when the statement completes successfully. You can delete the old data file manually after the operation completes if necessary.

To rename or relocate online data files:

1. In SQL*Plus, connect to the database as a user with `ALTER DATABASE` system privilege.
2. Issue the `ALTER DATABASE MOVE DATAFILE` statement and specify the data file.

Example 12-1 Renaming an Online Data File

This example renames the data file `user1.dbf` to `user01.dbf` while keeping the data file in the same location.

```
ALTER DATABASE MOVE DATAFILE '/u01/oracle/rbdb1/user1.dbf'
  TO '/u01/oracle/rbdb1/user01.dbf';
```

Example 12-2 Relocating an Online Data File

This example moves the data file `user1.dbf` from the /u01/oracle/rbdb1/ directory to the /u02/oracle/rbdb1/ directory. After the operation, the file is no longer in the /u01/oracle/rbdb1/ directory.

```
ALTER DATABASE MOVE DATAFILE '/u01/oracle/rbdb1/user1.dbf'
  TO '/u02/oracle/rbdb1/user1.dbf';
```

Example 12-3 Copying an Online Data File

This example copies the data file `user1.dbf` from the /u01/oracle/rbdb1/ directory to the /u02/oracle/rbdb1/ directory. After the operation, the old file is retained in the /u01/oracle/rbdb1/ directory.

```
ALTER DATABASE MOVE DATAFILE '/u01/oracle/rbdb1/user1.dbf'
  TO '/u02/oracle/rbdb1/user1.dbf' KEEP;
```

Example 12-4 Relocating an Online Data File and Overwriting an Existing File

This example moves the data file `user1.dbf` from the /u01/oracle/rbdb1/ directory to the /u02/oracle/rbdb1/ directory. If a file with the same name exists in the /u02/oracle/rbdb1/ directory, then the statement overwrites the file.

```
ALTER DATABASE MOVE DATAFILE '/u01/oracle/rbdb1/user1.dbf'
  TO '/u02/oracle/rbdb1/user1.dbf' REUSE;
```

Example 12-5 Relocating an Online Data File to Oracle ASM

This example moves the data file `user1.dbf` from the /u01/oracle/rbdb1/ directory to an Oracle ASM location.

```
ALTER DATABASE MOVE DATAFILE '/u01/oracle/rbdb1/user1.dbf'
  TO '+dgroup_01/data/orcl/datafile/user1.dbf';
```

Example 12-6 Moving a File from One ASM Location to Another ASM Location

This example moves the data file from one Oracle ASM location to another Oracle ASM location.

```
ALTER DATABASE MOVE DATAFILE '+dgroup_01/data/orcl/datafile/user1.dbf'
  TO '+dgroup_02/data/orcl/datafile/user1.dbf';
```

You also can move an online data file with Oracle ASM by mirroring the data file and then removing the original file location from the mirror. The online data file move operation might be faster when you use Oracle ASM to move the file instead of the `ALTER DATABASE MOVE DATAFILE` statement.

See Also:

- *Oracle Database SQL Language Reference* for more information about the `ALTER DATABASE` statement
- *Oracle Automatic Storage Management Administrator's Guide*

12.5.2 Renaming and Relocating Offline Data Files

You can rename and relocate offline data files.

When you rename and relocate offline data files, only the pointers to the data files, as recorded in the database control file, are changed. Files are not physically renamed, and they are not copied at the operating system level.

- Procedures for Renaming and Relocating Offline Data Files in a Single Tablespace
  You can rename and relocate offline data files that can be used for a single tablespace. You must have `ALTER TABLESPACE` system privilege to perform these procedures.
- Renaming and Relocating Offline Data Files in Multiple Tablespaces
  You can rename and relocate data files in one or more tablespaces using the `ALTER DATABASE RENAME FILE` statement.

12.5.2.1 Procedures for Renaming and Relocating Offline Data Files in a Single Tablespace

You can rename and relocate offline data files that can be used for a single tablespace. You must have `ALTER TABLESPACE` system privilege to perform these procedures.

- Renaming Offline Data Files in a Single Tablespace
  You can rename offline data files in a single tablespace.
- Relocating Offline Data Files in a Single Tablespace
  You can relocate offline data files in a single tablespace.

See Also:

"Taking Tablespaces Offline" for more information about taking tablespaces offline in preparation for renaming or relocating data files

12.5.2.1.1 Renaming Offline Data Files in a Single Tablespace

You can rename offline data files in a single tablespace.

To rename offline data files in a single tablespace, complete the following steps:

1. Take the tablespace that contains the data files offline. The database must be open.

   For example:

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

2. Rename the data files using the operating system.
3. Use the `ALTER TABLESPACE` statement with the `RENAME DATAFILE` clause to change the file names within the database.

   For example, the following statement renames the data files `/u02/oracle/rbdb1/user1.dbf` and `/u02/oracle/rbdb1/user2.dbf` to`/u02/oracle/rbdb1/users01.dbf` and `/u02/oracle/rbdb1/users02.dbf`, respectively:

   ```
   ALTER TABLESPACE users
       RENAME DATAFILE '/u02/oracle/rbdb1/user1.dbf',
                       '/u02/oracle/rbdb1/user2.dbf'
                    TO '/u02/oracle/rbdb1/users01.dbf',
                       '/u02/oracle/rbdb1/users02.dbf';
   ```

   Always provide complete file names (including their paths) to properly identify the old and new data files. In particular, specify the old data file name exactly as it appears in the `DBA_DATA_FILES` view of the data dictionary.

4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.
5. Bring the tablespace back online using an `ALTER TABLESPACE` statement with the `ONLINE` clause:

   ```
   ALTER TABLESPACE users ONLINE
   ```

12.5.2.1.2 Relocating Offline Data Files in a Single Tablespace

You can relocate offline data files in a single tablespace.

Here is a sample procedure for relocating an offline data file.

Assume the following conditions:

- An open database has a tablespace named `users` that is made up of data files all located on the same disk.
- The data files of the `users` tablespace are to be relocated to different and separate disk drives.
- You are currently connected with administrator privileges to the open database.
- You have a current backup of the database.

Complete the following steps:

1. If you do not know the specific file names or sizes, you can obtain this information by issuing the following query of the data dictionary view `DBA_DATA_FILES`:

   ```
   SQL> SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
     2> WHERE TABLESPACE_NAME = 'USERS';

   FILE_NAME                                  BYTES
   ------------------------------------------ ----------------
   /u02/oracle/rbdb1/users01.dbf              102400000
   /u02/oracle/rbdb1/users02.dbf              102400000
   ```

2. Take the tablespace containing the data files offline:

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

3. Copy the data files to their new locations and rename them using the operating system. You can copy the files using the `DBMS_FILE_TRANSFER` package discussed in "Copying Files Using the Database Server".

   > **Note:**
   >
   > You can temporarily exit SQL*Plus to execute an operating system command to copy a file by using the SQL*Plus `HOST` command.

4. Rename the data files within the database.

   The data file pointers for the files that comprise the `users` tablespace, recorded in the control file of the associated database, must now be changed from the old names to the new names.

   Use the `ALTER TABLESPACE...RENAME DATAFILE` statement.

   ```
   ALTER TABLESPACE users
       RENAME DATAFILE '/u02/oracle/rbdb1/users01.dbf',
                       '/u02/oracle/rbdb1/users02.dbf'
                    TO '/u03/oracle/rbdb1/users01.dbf',
                       '/u04/oracle/rbdb1/users02.dbf';
   ```

5. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

6. Bring the tablespace back online using an `ALTER TABLESPACE` statement with the `ONLINE` clause:

   ```
   ALTER TABLESPACE users ONLINE
   ```

12.5.2.2 Renaming and Relocating Offline Data Files in Multiple Tablespaces

You can rename and relocate data files in one or more tablespaces using the `ALTER DATABASE RENAME FILE` statement.

This method is the only choice if you want to rename or relocate data files of several tablespaces in one operation. You must have the `ALTER DATABASE` system privilege.

> **Note:**
>
> To rename or relocate data files of the `SYSTEM` tablespace, the default temporary tablespace, or the active undo tablespace you must use this `ALTER DATABASE` method because you cannot take these tablespaces offline.

To rename data files in multiple tablespaces, follow these steps.

1. Ensure that the database is mounted but closed.

   > **Note:**
   >
   > Optionally, the database does not have to be closed, but the data files (or temp files) must be offline.

2. Copy the data files to be renamed to their new locations and new names, using the operating system. You can copy the files using the `DBMS_FILE_TRANSFER` package discussed in "Copying Files Using the Database Server".
3. Use `ALTER DATABASE` to rename the file pointers in the database control file.

   For example, the following statement renames the data files `/u02/oracle/rbdb1/sort01.dbf` and `/u02/oracle/rbdb1/user3.dbf` to `/u02/oracle/rbdb1/temp01.dbf` and `/u02/oracle/rbdb1/users03.dbf`, respectively:

   ```
   ALTER DATABASE
       RENAME FILE '/u02/oracle/rbdb1/sort01.dbf',
                   '/u02/oracle/rbdb1/user3.dbf'
                TO '/u02/oracle/rbdb1/temp01.dbf',
                   '/u02/oracle/rbdb1/users03.dbf';
   ```

   Always provide complete file names (including their paths) to properly identify the old and new data files. In particular, specify the old data file names exactly as they appear in the `DBA_DATA_FILES` view.
4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

12.6 Dropping Data Files

You use the `DROP DATAFILE` and `DROP TEMPFILE` clauses of the `ALTER TABLESPACE` statement to drop a single data file or temp file.

The data file must be empty. (A data file is considered to be empty when no extents remain allocated from it.) When you drop a data file or temp file, references to the data file or temp file are removed from the data dictionary and control files, and the physical file is deleted from the file system or Oracle Automatic Storage Management (Oracle ASM) disk group.

The following example drops the data file identified by the alias `example_df3.f` in the Oracle ASM disk group `DGROUP1`. The data file belongs to the `example` tablespace.

```
ALTER TABLESPACE example DROP DATAFILE '+DGROUP1/example_df3.f';
```

The next example drops the temp file `lmtemp02.dbf`, which belongs to the `lmtemp` tablespace.

```
ALTER TABLESPACE lmtemp DROP TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

This is equivalent to the following statement:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
     INCLUDING DATAFILES;
```

Note:

If there are sessions using a temp file, and you attempt to drop the temp file, then an error is returned, and the temp file is not dropped. In this case, the temp file is taken offline, and queries that attempt to use the temp file will fail while the temp file is offline.

See *Oracle Database SQL Language Reference* for `ALTER TABLESPACE` syntax details.

Restrictions for Dropping Data Files

The following are restrictions for dropping data files and temp files:

- The database must be open.
- If a data file is not empty, it cannot be dropped.

  If you must remove a data file that is not empty and that cannot be made empty by dropping schema objects, you must drop the tablespace that contains the data file.
- You cannot drop the first or only data file in a tablespace.

  Therefore, `DROP DATAFILE` cannot be used with a bigfile tablespace.
- You cannot drop data files in a read-only tablespace that was migrated from dictionary managed to locally managed. Dropping a data file from all other read-only tablespaces is supported.
- You cannot drop data files in the `SYSTEM` tablespace.
- If a data file in a locally managed tablespace is offline, it cannot be dropped.

See Also:

Dropping Tablespaces

12.7 Verifying Data Blocks in Data Files

To configure the database to use checksums to verify data blocks, set the initialization parameter `DB_BLOCK_CHECKSUM` to `TYPICAL` (the default).

This setting causes the DBW*n* process and the direct loader to calculate a checksum for each block and to store the checksum in the block header when writing the block to disk.

The checksum is verified when the block is read, but only if `DB_BLOCK_CHECKSUM` is `TRUE` and the last write of the block stored a checksum. If corruption is detected, the database returns message `ORA-01578` and writes information about the corruption to the alert log.

The value of the `DB_BLOCK_CHECKSUM` parameter can be changed dynamically using the `ALTER SYSTEM` statement. Regardless of the setting of this parameter, checksums are always used to verify data blocks in the `SYSTEM` tablespace.

See Also:

Oracle Database Reference for more information about the `DB_BLOCK_CHECKSUM` initialization parameter

12.8 Copying Files Using the Database Server

You can use the `DBMS_FILE_TRANSFER` package to copy a file within a database or transfer a file between databases.

- About Copying Files Using the Database Server
  You do not necessarily have to use the operating system to copy a file within a database, or transfer a file between databases as you would do when using the transportable tablespace feature. You can use the `DBMS_FILE_TRANSFER` package for this purpose.
- Copying a File on a Local File System
  You can use the `COPY_FILE` procedure in the `DBMS_FILE_TRANSFER` package to copy a file on a local file system.
- Third-Party File Transfer
  Although the procedures in the `DBMS_FILE_TRANSFER` package typically are invoked as local procedure calls, they can also be invoked as remote procedure calls. A remote procedure call lets you copy a file within a database even when you are connected to a different database.
- Advanced File Transfer Mechanisms
  You can create more sophisticated file transfer mechanisms using both the `DBMS_FILE_TRANSFER` package and the `DBMS_SCHEDULER` package.

- File Transfer and the DBMS_SCHEDULER Package
  You can use the `DBMS_SCHEDULER` package to transfer files automatically within a single database and between databases.

12.8.1 About Copying Files Using the Database Server

You do not necessarily have to use the operating system to copy a file within a database, or transfer a file between databases as you would do when using the transportable tablespace feature. You can use the `DBMS_FILE_TRANSFER` package for this purpose.

The `DBMS_FILE_TRANSFER` package can use a local file system or an Oracle Automatic Storage Management (Oracle ASM) disk group as the source or destination for a file transfer. Only Oracle database files (data files, temp files, control files, and so on) can be involved in transfers to and from Oracle ASM.

On UNIX systems, the owner of a file created by the `DBMS_FILE_TRANSFER` package is the owner of the shadow process running the instance. Normally, this owner is `ORACLE`. A file created using `DBMS_FILE_TRANSFER` is always writable and readable by all processes in the database, but non privileged users who need to read or write such a file directly may need access from a system administrator.

Caution:

Do not use the `DBMS_FILE_TRANSFER` package to copy or transfer a file that is being modified by a database because doing so may result in an inconsistent file.

See Also:

- "Copying a File on a Local File System" for an example of using the `DBMS_FILE_TRANSFER` package
- "Transporting Tablespaces Between Databases" for information about how to transport tablespaces between databases
- *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_FILE_TRANSFER` package.

12.8.2 Copying a File on a Local File System

You can use the `COPY_FILE` procedure in the `DBMS_FILE_TRANSFER` package to copy a file on a local file system.

The following example illustrates using the `COPY_FILE` procedure in the `DBMS_FILE_TRANSFER` package to copy a file on a local file system. The example copies a binary file named `db1.dat` from the `/usr/admin/source` directory to the `/usr/admin/destination` directory as `db1_copy.dat` on a local file system:

1. In SQL*Plus, connect as an administrative user who can grant privileges and create directory objects using SQL.

2. Use the SQL command `CREATE DIRECTORY` to create a directory object for the directory from which you want to copy the file. A directory object is similar to an alias for the directory. For example, to create a directory object called `SOURCE_DIR` for the `/usr/admin/source` directory on your computer system, execute the following statement:

   ```
   CREATE DIRECTORY SOURCE_DIR AS '/usr/admin/source';
   ```

3. Use the SQL command `CREATE DIRECTORY` to create a directory object for the directory into which you want to copy the binary file. For example, to create a directory object called `DEST_DIR` for the `/usr/admin/destination` directory on your computer system, execute the following statement:

   ```
   CREATE DIRECTORY DEST_DIR AS '/usr/admin/destination';
   ```

4. Grant the required privileges to the user who will run the `COPY_FILE` procedure. In this example, the `strmadmin` user runs the procedure.

   ```
   GRANT EXECUTE ON DBMS_FILE_TRANSFER TO strmadmin;

   GRANT READ ON DIRECTORY source_dir TO strmadmin;

   GRANT WRITE ON DIRECTORY dest_dir TO strmadmin;
   ```

5. Connect as `strmadmin` user and provide the user password when prompted:

   ```
   CONNECT strmadmin
   ```

6. Run the `COPY_FILE` procedure to copy the file:

   ```
   BEGIN
     DBMS_FILE_TRANSFER.COPY_FILE(
           source_directory_object       =>  'SOURCE_DIR',
           source_file_name              =>  'db1.dat',
           destination_directory_object  =>  'DEST_DIR',
           destination_file_name         =>  'db1_copy.dat');
   END;
   /
   ```

   The `source_file_name` parameter must specify a file that is in the directory specified by the `source_directory_object` parameter before running the procedure, and the `destination_file_name` parameter must specify the new name of the file in the new location specified in the `destination_directory_object` parameter. Relative paths and symbolic links are not allowed in the directory objects for the `source_directory_object` and `destination_directory_object` parameters.

Caution:

Do not use the `DBMS_FILE_TRANSFER` package to copy or transfer a file that is being modified by a database because doing so may result in an inconsistent file.

12.8.3 Third-Party File Transfer

Although the procedures in the `DBMS_FILE_TRANSFER` package typically are invoked as local procedure calls, they can also be invoked as remote procedure calls. A remote procedure call lets you copy a file within a database even when you are connected to a different database.

For example, you can make a copy of a file on database `DB`, even if you are connected to another database, by executing the following remote procedure call:

```
DBMS_FILE_TRANSFER.COPY_FILE@DB(...)
```

Using remote procedure calls enables you to copy a file between two databases, even if you are not connected to either database. For example, you can connect to database `A` and then transfer a file from database `B` to database `C`. In this example, database `A` is the third party because it is neither the source of nor the destination for the transferred file.

A third-party file transfer can both push and pull a file. Continuing with the previous example, you can perform a third-party file transfer if you have a database link from `A` to either `B` or `C`, and that database has a database link to the other database. Database `A` does not need a database link to both `B` and `C`.

For example, if you have a database link from `A` to `B`, and another database link from `B` to `C`, then you can run the following procedure at `A` to transfer a file from `B` to `C`:

```
DBMS_FILE_TRANSFER.PUT_FILE@B(...)
```

This configuration pushes the file.

Alternatively, if you have a database link from `A` to `C`, and another database link from `C` to `B`, then you can run the following procedure at database `A` to transfer a file from `B` to `C`:

```
DBMS_FILE_TRANSFER.GET_FILE@C(...)
```

This configuration pulls the file.

12.8.4 Advanced File Transfer Mechanisms

You can create more sophisticated file transfer mechanisms using both the `DBMS_FILE_TRANSFER` package and the `DBMS_SCHEDULER` package.

For example, when several databases have a copy of the file you want to transfer, you can consider factors such as source availability, source load, and communication bandwidth to the destination database when deciding which source database to contact first and which source databases to try if failures occur. In this case, the information about these factors must be available to you, and you must create the mechanism that considers these factors.

As another example, when early completion time is more important than load, you can submit several Scheduler jobs to transfer files in parallel. As a final example, knowing something about file layout on the source and destination databases enables you to minimize disk contention by performing or scheduling simultaneous transfers only if they use different I/O devices.

12.8.5 File Transfer and the DBMS_SCHEDULER Package

You can use the `DBMS_SCHEDULER` package to transfer files automatically within a single database and between databases.

Third-party file transfers are also supported by the `DBMS_SCHEDULER` package. You can monitor a long-running file transfer done by the Scheduler using the `V$SESSION_LONGOPS` dynamic performance view at the databases reading or writing the file. Any database links used by a Scheduler job must be fixed user database links.

You can use a restartable Scheduler job to improve the reliability of file transfers automatically, especially if there are intermittent failures. If a file transfer fails before the destination file is closed, then you can restart the file transfer from the beginning once the database has removed any partially written destination file. Hence you should consider using a restartable

Scheduler job to transfer a file if the rest of the job is restartable. See Scheduling Jobs with Oracle Scheduler for more information on Scheduler jobs.

> **Note:**
>
> If a single restartable job transfers several files, then you should consider restart scenarios in which some of the files have been transferred already and some have not been transferred yet.

12.9 Mapping Files to Physical Devices

In an environment where data files are file system files, it is relatively straight forward to see the association between a tablespace and the underlying device. Oracle Database provides views, such as `DBA_TABLESPACES`, `DBA_DATA_FILES`, and `V$DATAFILE`, that provide a mapping of files onto devices. These mappings, along with device statistics can be used to evaluate I/O performance.

However, with the introduction of host based Logical Volume Managers (LVM), and sophisticated storage subsystems that provide RAID (Redundant Array of Inexpensive Disks) features, it is not easy to determine file to device mapping. This poses a problem because it becomes difficult to determine your "hottest" files when they are hidden behind a "black box". This section presents the Oracle Database approach to resolving this problem.

> **Note:**
>
> This section presents an overview of the Oracle Database file mapping interface and explains how to use the `DBMS_STORAGE_MAP` package and dynamic performance views to expose the mapping of files onto physical devices. You can more easily access this functionality through the Oracle Enterprise Manager Cloud Control. It provides an easy to use graphical interface for mapping files to physical devices. See the Cloud Control online help for more information.

- Overview of Oracle Database File Mapping Interface
  To acquire an understanding of I/O performance, one must have detailed knowledge of the storage hierarchy in which files reside.
- How the Oracle Database File Mapping Interface Works
  Oracle Database file mapping includes the following components: the FMON is a background process, the FMPUTL process, and mapping libraries.
- Using the Oracle Database File Mapping Interface
  You can use the Oracle Database file mapping interface to enable file mapping and obtain information about file mapping in a set of views.
- File Mapping Examples
  Examples illustrates some of the powerful capabilities of the Oracle Database file mapping feature.

12.9.1 Overview of Oracle Database File Mapping Interface

To acquire an understanding of I/O performance, one must have detailed knowledge of the storage hierarchy in which files reside.

Oracle Database provides a mechanism to show a complete mapping of a file to intermediate layers of logical volumes to actual physical devices. This is accomplished though a set of dynamic performance views (`V$` views). Using these views, you can locate the exact disk on which any block of a file resides.

To build these views, storage vendors must provide mapping libraries that are responsible for mapping their particular I/O stack elements. The database communicates with these libraries through an external non-Oracle Database process that is spawned by a background process called FMON. FMON is responsible for managing the mapping information. Oracle provides a PL/SQL package, `DBMS_STORAGE_MAP`, that you use to invoke mapping operations that populate the mapping views.

Note:

If you are not using Oracle Automatic Storage Management, then the file mapping interface is not available on Windows platforms. If you are using Oracle Automatic Storage Management, then the file mapping interface is available on all platforms.

See Also:

Oracle Automatic Storage Management Administrator's Guide for information about using file mapping with Oracle ASM

12.9.2 How the Oracle Database File Mapping Interface Works

Oracle Database file mapping includes the following components: the FMON is a background process, the FMPUTL process, and mapping libraries.

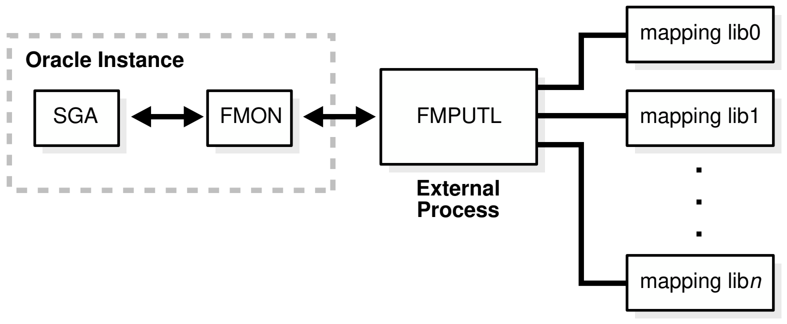
- Components of File Mapping
  The file mapping mechanism includes several components.
- Mapping Structures
  You must understand mapping structures and the Oracle Database representation of these structures to interpret the information in the mapping views.
- Example of Mapping Structures
  An example illustrates mapping structures.
- Configuration ID
  The configuration ID captures the version information associated with elements or files.

12.9.2.1 Components of File Mapping

The file mapping mechanism includes several components.

The following figure shows the components of the file mapping mechanism.

Figure 12-1 Components of File Mapping



> **Note:**
>
> Starting with Oracle Database 12*c*, the `FILE_MAPPING` initialization parameter, the `FMPUTL` process, and the mapping libraries are deprecated.

- FMON
  FMON is a background process started by the database whenever the `FILE_MAPPING` initialization parameter is set to `true`. FMON builds map information and refreshing mapping information when a change occurs.
- External Process (FMPUTL)
  FMON spawns an external non-Oracle Database process called `FMPUTL`, that communicates directly with the vendor supplied mapping libraries.
- Mapping Libraries
  Oracle Database uses mapping libraries to discover mapping information for the elements that are owned by a particular mapping library.

12.9.2.1.1 FMON

FMON is a background process started by the database whenever the `FILE_MAPPING` initialization parameter is set to `true`. FMON builds map information and refreshing mapping information when a change occurs.

FMON is responsible for:

- Building mapping information, which is stored in the SGA. This information is composed of the following structures:
  - Files
  - File system extents
  - Elements
  - Subelements

  These structures are explained in "Mapping Structures".
- Refreshing mapping information when a change occurs because of:

  - Changes to data files (size)
  - Addition or deletion of data files
  - Changes to the storage configuration (not frequent)
- Saving mapping information in the data dictionary to maintain a view of the information that is persistent across startup and shutdown operations
- Restoring mapping information into the SGA at instance startup. This avoids the need for a potentially expensive complete rebuild of the mapping information on every instance startup.

You help control this mapping using procedures that are invoked with the `DBMS_STORAGE_MAP` package.

12.9.2.1.2 External Process (FMPUTL)

FMON spawns an external non-Oracle Database process called `FMPUTL`, that communicates directly with the vendor supplied mapping libraries.

This process obtains the mapping information through all levels of the I/O stack, assuming that mapping libraries exist for all levels. On some platforms the external process requires that the `SETUID` bit is set to `ON` because root privileges are needed to map through all levels of the I/O mapping stack.

The external process is responsible for discovering the mapping libraries and dynamically loading them into its address space.

12.9.2.1.3 Mapping Libraries

Oracle Database uses mapping libraries to discover mapping information for the elements that are owned by a particular mapping library.

Through these mapping libraries information about individual I/O stack elements is communicated. This information is used to populate dynamic performance views that can be queried by users.

Mapping libraries need to exist for all levels of the stack for the mapping to be complete, and different libraries may own their own parts of the I/O mapping stack. For example, a VERITAS VxVM library would own the stack elements related to the VERITAS Volume Manager, and an EMC library would own all EMC storage specific layers of the I/O mapping stack.

Mapping libraries are vendor supplied. However, Oracle currently supplies a mapping library for EMC storage. The mapping libraries available to a database server are identified in a special file named filemap.ora.

12.9.2.2 Mapping Structures

You must understand mapping structures and the Oracle Database representation of these structures to interpret the information in the mapping views.

The following are the primary structures that compose the mapping information:

- Files

  A file mapping structure provides a set of attributes for a file, including file size, number of file system extents that the file is composed of, and the file type.
- File system extents

A file system extent mapping structure describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides.

> **Note:**
>
> File system extents are different from Oracle Database extents. File system extents are physical contiguous blocks of data written to a device as managed by the file system. Oracle Database extents are logical structures managed by the database, such as tablespace extents.

- Elements

  An element mapping structure is the abstract mapping structure that describes a storage component within the I/O stack. Elements may be mirrors, stripes, partitions, RAID5, concatenated elements, and disks. These structures are the mapping building blocks.

- Subelements

  A subelement mapping structure describes the link between an element and the next elements in the I/O mapping stack. This structure contains the subelement number, size, the element name where the subelement exists, and the element offset.

All of these mapping structures are illustrated in the following example.

12.9.2.3 Example of Mapping Structures

An example illustrates mapping structures.

Consider an Oracle Database which is composed of two data files X and Y. Both files X and Y reside on a file system mounted on volume A. File X is composed of two extents while file Y is composed of only one extent.

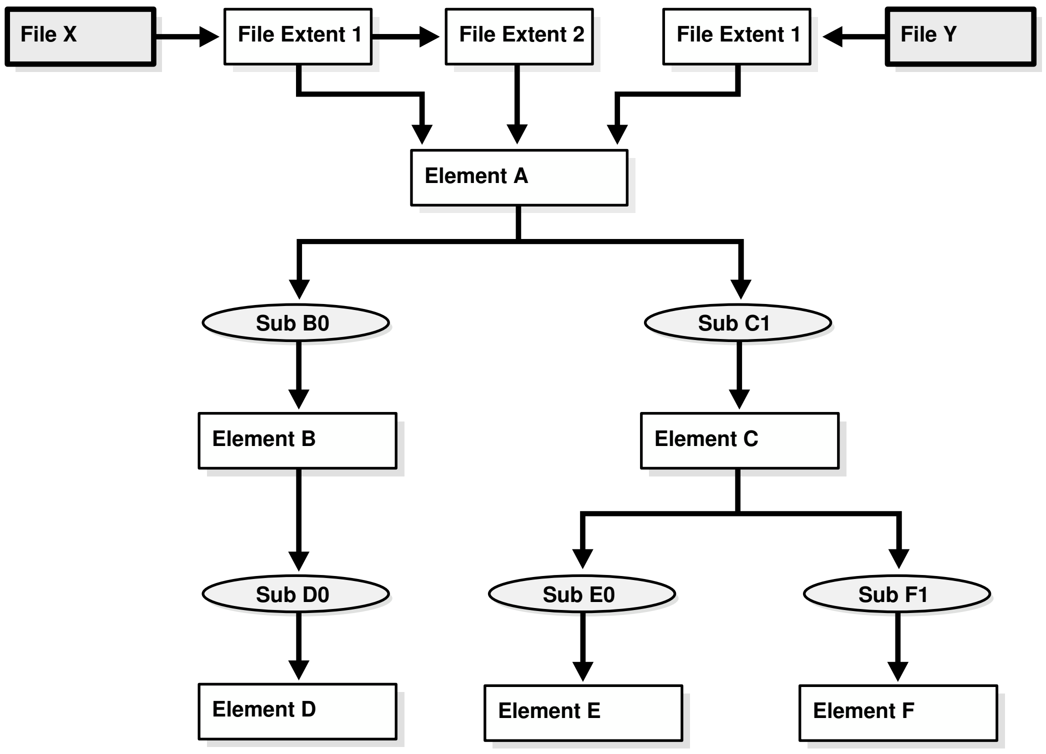
The two extents of File X and the one extent of File Y both map to Element A. Element A is striped to Elements B and C. Element A maps to Elements B and C by way of Subelements B0 and C1, respectively.

Element B is a partition of Element D (a physical disk), and is mapped to Element D by way of subelement D0.

Element C is mirrored over Elements E and F (both physical disks), and is mirrored to those physical disks by way of Subelements E0 and F1, respectively.

All of the mapping structures are illustrated in Figure 12-2.

Figure 12-2 Illustration of Mapping Structures



Note that the mapping structures represented are sufficient to describe the entire mapping information for the Oracle Database instance and consequently to map every logical block within the file into a (element name, element offset) tuple (or more in case of mirroring) at each level within the I/O stack.

12.9.2.4 Configuration ID

The configuration ID captures the version information associated with elements or files.

The vendor library provides the configuration ID and updates it whenever a change occurs. Without a configuration ID, there is no way for the database to tell whether the mapping has changed.

There are two kinds of configuration IDs:

- Persistent

  These configuration IDs are persistent across instance shutdown
- Non-persistent

  The configuration IDs are not persistent across instance shutdown. The database is only capable of refreshing the mapping information while the instance is up.

12.9.3 Using the Oracle Database File Mapping Interface

You can use the Oracle Database file mapping interface to enable file mapping and obtain information about file mapping in a set of views.

- Enabling File Mapping
  You can enable file mapping.

- Using the DBMS_STORAGE_MAP Package
  The `DBMS_STORAGE_MAP` package enables you to control the mapping operations.
- Obtaining Information from the File Mapping Views
  Mapping information generated by `DBMS_STORAGE_MAP` package is captured in dynamic performance views.

12.9.3.1 Enabling File Mapping

You can enable file mapping.

To enable file mapping:

1. Ensure that a valid filemap.ora file exists in the /opt/ORCLfmap/prot1_32/etc directory for 32-bit platforms, or in the /opt/ORCLfmap/prot1_64/etc directory for 64-bit platforms.

   Note:

   While the format and content of the filemap.ora file is discussed here, it is for informational reasons only. The filemap.ora file is created by the database when your system is installed. Until such time that vendors supply their own libraries, there will be only one entry in the filemap.ora file, and that is the Oracle-supplied EMC library. This file should be modified manually by uncommenting this entry *only* if an EMC Symmetrix array is available.

   The filemap.ora file is the configuration file that describes all of the available mapping libraries. FMON requires that a filemap.ora file exists and that it points to a valid path to mapping libraries. Otherwise, it will not start successfully.

   The following row must be included in filemap.ora for each library:

   ```
   lib=vendor_name:mapping_library_path
   ```

   where:

   - *vendor_name* should be `Oracle` for the EMC Symmetric library
   - *mapping_library_path* is the full path of the mapping library

   Note that the ordering of the libraries in this file is extremely important. The libraries are queried based on their order in the configuration file.

   The file mapping service can be started even if no mapping libraries are available. The filemap.ora file still must be present even though it is empty. In this case, the mapping service is constrained in the sense that new mapping information cannot be discovered. Only restore and drop operations are allowed in such a configuration.

2. Set the `FILE_MAPPING` initialization parameter to `TRUE`.

   The instance does not have to be shut down to set this parameter. You can set it using the following `ALTER SYSTEM` statement:

   ```
   ALTER SYSTEM SET FILE_MAPPING=TRUE;
   ```

3. Invoke the appropriate `DBMS_STORAGE_MAP` mapping procedure. You have two options:

   - In a cold startup scenario, the Oracle Database is just started and no mapping operation has been invoked yet. You execute the `DBMS_STORAGE_MAP.MAP_ALL` procedure to build the mapping information for the entire I/O subsystem associated with the database.

- In a warm start scenario where the mapping information is already built, you have the option to invoke the `DBMS_STORAGE_MAP.MAP_SAVE` procedure to save the mapping information in the data dictionary. (Note that this procedure is invoked in `DBMS_STORAGE_MAP.MAP_ALL()` by default.) This forces all of the mapping information in the SGA to be flushed to disk.

  Once you restart the database, use `DBMS_STORAGE_MAP.RESTORE()` to restore the mapping information into the SGA. If needed, `DBMS_STORAGE_MAP.MAP_ALL()` can be called to refresh the mapping information.

12.9.3.2 Using the DBMS_STORAGE_MAP Package

The `DBMS_STORAGE_MAP` package enables you to control the mapping operations.

The various procedures available to you are described in the following table.

Procedure	Use to:
`MAP_OBJECT`	Build the mapping information for the database object identified by object name, owner, and type
`MAP_ELEMENT`	Build mapping information for the specified element
`MAP_FILE`	Build mapping information for the specified file name
`MAP_ALL`	Build entire mapping information for all types of database files (excluding archive logs)
`DROP_ELEMENT`	Drop the mapping information for a specified element
`DROP_FILE`	Drop the file mapping information for the specified file name
`DROP_ALL`	Drop all mapping information in the SGA for this instance
`SAVE`	Save into the data dictionary the required information needed to regenerate the entire mapping
`RESTORE`	Load the entire mapping information from the data dictionary into the shared memory of the instance
`LOCK_MAP`	Lock the mapping information in the SGA for this instance
`UNLOCK_MAP`	Unlock the mapping information in the SGA for this instance

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_STORAGE_MAP` package
- "File Mapping Examples" for an example of using the `DBMS_STORAGE_MAP` package

12.9.3.3 Obtaining Information from the File Mapping Views

Mapping information generated by `DBMS_STORAGE_MAP` package is captured in dynamic performance views.

Brief descriptions of these views are presented here.

View	Description
V$MAP_LIBRARY	Contains a list of all mapping libraries that have been dynamically loaded by the external process
V$MAP_FILE	Contains a list of all file mapping structures in the shared memory of the instance
V$MAP_FILE_EXTENT	Contains a list of all file system extent mapping structures in the shared memory of the instance
V$MAP_ELEMENT	Contains a list of all element mapping structures in the SGA of the instance
V$MAP_EXT_ELEMENT	Contains supplementary information for all element mapping
V$MAP_SUBELEMENT	Contains a list of all subelement mapping structures in the shared memory of the instance
V$MAP_COMP_LIST	Contains supplementary information for all element mapping structures.
V$MAP_FILE_IO_STACK	The hierarchical arrangement of storage containers for the file displayed as a series of rows. Each row represents a level in the hierarchy.

However, the information generated by the DBMS_STORAGE_MAP.MAP_OBJECT procedure is captured in a global temporary table named MAP_OBJECT. This table displays the hierarchical arrangement of storage containers for objects. Each row in the table represents a level in the hierarchy. A description of the MAP_OBJECT table follows.

Column	Data Type	Description
OBJECT_NAME	VARCHAR2(2000)	Name of the object
OBJECT_OWNER	VARCHAR2(2000)	Owner of the object
OBJECT_TYPE	VARCHAR2(2000)	Object type
FILE_MAP_IDX	NUMBER	File index (corresponds to FILE_MAP_IDX in V$MAP_FILE)
DEPTH	NUMBER	Element depth within the I/O stack
ELEM_IDX	NUMBER	Index corresponding to element
CU_SIZE	NUMBER	Contiguous set of logical blocks of the file, in HKB (half KB) units, that is resident contiguously on the element
STRIDE	NUMBER	Number of HKB between contiguous units (CU) in the file that are contiguous on this element. Used in RAID5 and striped files.
NUM_CU	NUMBER	Number of contiguous units that are adjacent to each other on this element that are separated by STRIDE HKB in the file. In RAID5, the number of contiguous units also include the parity stripes.
ELEM_OFFSET	NUMBER	Element offset in HKB units
FILE_OFFSET	NUMBER	Offset in HKB units from the start of the file to the first byte of the contiguous units
DATA_TYPE	VARCHAR2(2000)	Data type (DATA, PARITY, or DATA AND PARITY)
PARITY_POS	NUMBER	Position of the parity. Only for RAID5. This field is needed to distinguish the parity from the data part.
PARITY_PERIOD	NUMBER	Parity period. Only for RAID5.

12.9.4 File Mapping Examples

Examples illustrates some of the powerful capabilities of the Oracle Database file mapping feature.

These capabilities include:

- The ability to map all the database files that span a particular device
- The ability to map a particular file into its corresponding devices
- The ability to map a particular database object, including its block distribution at all levels within the I/O stack

Consider an Oracle Database instance which is composed of two data files:

- `t_db1.f`
- `t_db2.f`

These files are created on a Solaris UFS file system mounted on a VERITAS VxVM host based striped volume, `/dev/vx/dsk/ipfdg/ipf-vol1`, that consists of the following host devices as externalized from an EMC Symmetrix array:

- `/dev/vx/rdmp/c2t1d0s2`
- `/dev/vx/rdmp/c2t1d1s2`

Note that the following examples require the execution of a `MAP_ALL()` operation.

- Example 1: Map All Database Files that Span a Device
  An example illustrates returning all Oracle Database files associated with a host device.
- Example 2: Map a File Into Its Corresponding Devices
  An example displays a topological graph of a data file.
- Example 3: Map a Database Object
  An example displays the block distribution at all levels within the I/O stack for a table.

12.9.4.1 Example 1: Map All Database Files that Span a Device

An example illustrates returning all Oracle Database files associated with a host device.

The following query returns all Oracle Database files associated with the `/dev/vx/rdmp/c2t1d1s2` host device:

```
SELECT UNIQUE me.ELEM_NAME, mf.FILE_NAME
   FROM V$MAP_FILE_IO_STACK fs, V$MAP_FILE mf, V$MAP_ELEMENT me
   WHERE mf.FILE_MAP_IDX = fs.FILE_MAP_IDX
   AND me.ELEM_IDX = fs.ELEM_IDX
   AND me.ELEM_NAME = '/dev/vx/rdmp/c2t1d1s2';
```

The query results are:

```
ELEM_NAME                  FILE_NAME
------------------------   --------------------------------
/dev/vx/rdmp/c2t1d1s2      /oracle/dbs/t_db1.f
/dev/vx/rdmp/c2t1d1s2      /oracle/dbs/t_db2.f
```

12.9.4.2 Example 2: Map a File Into Its Corresponding Devices

An example displays a topological graph of a data file.

The following query displays a topological graph of the `/oracle/dbs/t_db1.f` data file:

```
WITH fv AS
  (SELECT FILE_MAP_IDX, FILE_NAME FROM V$MAP_FILE
   WHERE FILE_NAME = '/oracle/dbs/t_db1.f')
SELECT fv.FILE_NAME, LPAD(' ', 4 * (LEVEL - 1)) || el.ELEM_NAME ELEM_NAME
   FROM V$MAP_SUBELEMENT sb, V$MAP_ELEMENT el, fv,
     (SELECT UNIQUE ELEM_IDX FROM V$MAP_FILE_IO_STACK io, fv
      WHERE io.FILE_MAP_IDX = fv.FILE_MAP_IDX) fs
   WHERE el.ELEM_IDX = sb.CHILD_IDX
   AND fs.ELEM_IDX = el.ELEM_IDX
   START WITH sb.PARENT_IDX IN
     (SELECT DISTINCT ELEM_IDX
      FROM V$MAP_FILE_EXTENT fe, fv
      WHERE fv.FILE_MAP_IDX = fe.FILE_MAP_IDX)
   CONNECT BY PRIOR sb.CHILD_IDX = sb.PARENT_IDX;
```

The resulting topological graph is:

```
FILE_NAME                    ELEM_NAME
-----------------------      -------------------------------------------------
/oracle/dbs/t_db1.f          _sym_plex_/dev/vx/rdsk/ipfdg/ipf-vol1_-1_-1
/oracle/dbs/t_db1.f              _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_0_0
/oracle/dbs/t_db1.f                  /dev/vx/rdmp/c2t1d0s2
/oracle/dbs/t_db1.f                      _sym_symdev_000183600407_00C
/oracle/dbs/t_db1.f                          _sym_hyper_000183600407_00C_0
/oracle/dbs/t_db1.f                          _sym_hyper_000183600407_00C_1
/oracle/dbs/t_db1.f              _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_1_0
/oracle/dbs/t_db1.f                  /dev/vx/rdmp/c2t1d1s2
/oracle/dbs/t_db1.f                      _sym_symdev_000183600407_00D
/oracle/dbs/t_db1.f                          _sym_hyper_000183600407_00D_0
/oracle/dbs/t_db1.f                          _sym_hyper_000183600407_00D_1
```

12.9.4.3 Example 3: Map a Database Object

An example displays the block distribution at all levels within the I/O stack for a table.

This example displays the block distribution at all levels within the I/O stack for the `scott.bonus` table.

A `MAP_OBJECT()` operation must first be executed as follows:

```
EXECUTE DBMS_STORAGE_MAP.MAP_OBJECT('BONUS','SCOTT','TABLE');
```

The query is as follows:

```
SELECT io.OBJECT_NAME o_name, io.OBJECT_OWNER o_owner, io.OBJECT_TYPE o_type,
       mf.FILE_NAME, me.ELEM_NAME, io.DEPTH,
      (SUM(io.CU_SIZE * (io.NUM_CU - DECODE(io.PARITY_PERIOD, 0, 0,
                        TRUNC(io.NUM_CU / io.PARITY_PERIOD)))) / 2) o_size
   FROM MAP_OBJECT io, V$MAP_ELEMENT me, V$MAP_FILE mf
   WHERE io.OBJECT_NAME =  'BONUS'
   AND   io.OBJECT_OWNER = 'SCOTT'
   AND   io.OBJECT_TYPE =  'TABLE'
   AND   me.ELEM_IDX = io.ELEM_IDX
   AND   mf.FILE_MAP_IDX = io.FILE_MAP_IDX
   GROUP BY io.ELEM_IDX, io.FILE_MAP_IDX, me.ELEM_NAME, mf.FILE_NAME, io.DEPTH,
         io.OBJECT_NAME, io.OBJECT_OWNER, io.OBJECT_TYPE
   ORDER BY io.DEPTH;
```

The following is the result of the query. Note that the `o_size` column is expressed in KB.

```
O_NAME O_OWNER O_TYPE  FILE_NAME           ELEM_NAME                     DEPTH   O_SIZE
------ ------- ------  ------------------- ----------------------------  ------  ------
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f /dev/vx/dsk/ipfdg/ipf-vol1         0      20
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_plex_/dev/vx/rdsk/ipf         1      20
                                           pdg/if-vol1_-1_-1
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_subdisk_/dev/vx/rdsk/         2      12
                                           ipfdg/ipf-vol1_0_1_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_subdisk_/dev/vx/rdsk/ipf      2       8
                                           dg/ipf-vol1_0_2_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f /dev/vx/rdmp/c2t1d1s2              3      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f /dev/vx/rdmp/c2t1d2s2              3       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_symdev_000183600407_00D       4      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_symdev_000183600407_00E       4       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_hyper_000183600407_00D_0      5      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_hyper_000183600407_00D_1      5      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_hyper_000183600407_00E_0      6       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f _sym_hyper_000183600407_00E_1      6       8
```

12.10 Data Files Data Dictionary Views

A set of data dictionary views provides useful information about the data files of a database.

View	Description
`DBA_DATA_FILES`	Provides descriptive information about each data file, including the tablespace to which it belongs and the file ID. The file ID can be used to join with other views for detail information.
`DBA_EXTENTS` `USER_EXTENTS`	`DBA` view describes the extents comprising all segments in the database. Contains the file ID of the data file containing the extent. `USER` view describes extents of the segments belonging to objects owned by the current user.
`DBA_FREE_SPACE` `USER_FREE_SPACE`	`DBA` view lists the free extents in all tablespaces. Includes the file ID of the data file containing the extent. `USER` view lists the free extents in the tablespaces accessible to the current user.
`V$DATAFILE`	Contains data file information from the control file
`V$DATAFILE_HEADER`	Contains information from data file headers

This example illustrates the use of one of these views, `V$DATAFILE`.

```
SELECT NAME,
    FILE#,
    STATUS,
    CHECKPOINT_CHANGE# "CHECKPOINT"
  FROM   V$DATAFILE;
```

```
NAME                                      FILE#     STATUS        CHECKPOINT
--------------------------------          -----     -------        ----------
/u01/oracle/rbdb1/system01.dbf                1     SYSTEM               3839
/u02/oracle/rbdb1/temp01.dbf                  2     ONLINE               3782
/u02/oracle/rbdb1/users03.dbf                 3     OFFLINE              3782
```

`FILE#` lists the file number of each data file; the first data file in the `SYSTEM` tablespace created with the database is always file 1. `STATUS` lists other information about a data file. If a data file is part of the `SYSTEM` tablespace, its status is `SYSTEM` (unless it requires recovery). If a data file in a non-`SYSTEM` tablespace is online, its status is `ONLINE`. If a data file in a non-`SYSTEM` tablespace is offline, its status can be either `OFFLINE` or `RECOVER`. `CHECKPOINT` lists the final SCN (system change number) written for the most recent checkpoint of a data file.

13 Transporting Data

Transporting data moves the data from one database to another.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- About Transporting Data
  You can transport data at the following levels: database, tablespaces, tables, partitions, and subpartitions.
- Transporting Databases
  You can transport a database to a new Oracle Database instance.
- Transporting Tablespaces Between Databases
  You can transport tablespaces between databases.
- Transporting Tables, Partitions, or Subpartitions Between Databases
  You can transport tables, partitions, and subpartitions between databases.
- Converting Data Between Platforms
  When you perform a transportable operation, and the source platform and the target platform are of different endianness, you must convert the data being transported to the target platform format. If the source platform and the target platform are of the same endianness, then data conversion is not necessary. You can use the `DBMS_FILE_TRANSFER` package or the RMAN `CONVERT` command to convert data.
- Guidelines for Transferring Data Files
  You should follow a set of guidelines when transferring the data files.

13.1 About Transporting Data

You can transport data at the following levels: database, tablespaces, tables, partitions, and subpartitions.

- Purpose of Transporting Data
  Transporting data is much faster than performing either an export/import or unload/load of the same data. It is faster because, for user-defined tablespaces, the data files containing all of the actual data are copied to the target location, and you use Data Pump to transfer only the metadata of the database objects to the new database.
- Transporting Data: Scenarios
  Transporting data is useful in several scenarios.
- Transporting Data Across Platforms
  You can transport data across platforms.

- General Limitations on Transporting Data
  There are general limitations on transporting data. There are also limitations that are specific to full transportable export/import, transportable tablespaces, or transportable tables.
- Compatibility Considerations for Transporting Data
  When transporting data, Oracle Database computes the lowest compatibility level at which the target database must run.

13.1.1 Purpose of Transporting Data

Transporting data is much faster than performing either an export/import or unload/load of the same data. It is faster because, for user-defined tablespaces, the data files containing all of the actual data are copied to the target location, and you use Data Pump to transfer only the metadata of the database objects to the new database.

You can transport data at any of the following levels:

- Database

  You can use the **full transportable export/import** feature to move an entire database to a different database instance.
- Tablespaces

  You can use the **transportable tablespaces** feature to move a set of tablespaces between databases.
- Tables, partitions, and subpartitions

  You can use the **transportable tables** feature to move a set of tables, partitions, and subpartitions between databases.

Transportable tablespaces and transportable tables only transports data that resides in user-defined tablespaces. However, full transportable export/import transports data that resides in both user-defined and administrative tablespaces, such as `SYSTEM` and `SYSAUX`. Full transportable export/import transports metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces. Specifically, with full transportable export/import, the export dump file includes only the metadata for objects contained within the user-defined tablespaces, but it includes both the metadata and the data for user-defined objects contained within the administrative tablespaces.

13.1.2 Transporting Data: Scenarios

Transporting data is useful in several scenarios.

- Scenarios for Full Transportable Export/import
  The full transportable export/import feature is useful in several scenarios.
- Scenarios for Transportable Tablespaces or Transportable Tables
  The transportable tablespaces or transportable tables feature is useful in several scenarios.

13.1.2.1 Scenarios for Full Transportable Export/import

The full transportable export/import feature is useful in several scenarios.

- Moving a Non-CDB Into a CDB
  The multitenant architecture enables an Oracle database to function as a multitenant container database (CDB) that includes one or many customer-created pluggable databases (PDBs). You can move a non-CDB into a CDB by transporting the database.
- Moving a Database to a New Computer System
  You can use full transportable export/import to move a database from one computer system to another. You might want to move a database to a new computer system to upgrade the hardware or to move the database to a different platform.
- Upgrading to a New Release of Oracle Database
  You can use full transportable export/import to upgrade a database from an Oracle Database 11*g* Release 2 (11.2.0.3) or later to Oracle Database 19c.

13.1.2.1.1 Moving a Non-CDB Into a CDB

The multitenant architecture enables an Oracle database to function as a multitenant container database (CDB) that includes one or many customer-created pluggable databases (PDBs). You can move a non-CDB into a CDB by transporting the database.

The transported database becomes a pluggable database (PDB) associated with the CDB. Full transportable export/import can move an Oracle Database 11*g* Release 2 (11.2.0.3) or later into an Oracle Database 19c CDB efficiently.

See Also:

- "Transporting a Database Using an Export Dump File" for instructions that describe transporting a non-CDB into a CDB using an export dump file
- "Transporting a Database Over the Network" for instructions that describe transporting a non-CDB into a CDB over the network
- *Oracle Multitenant Administrator's Guide*

13.1.2.1.2 Moving a Database to a New Computer System

You can use full transportable export/import to move a database from one computer system to another. You might want to move a database to a new computer system to upgrade the hardware or to move the database to a different platform.

See Also:

- "Transporting Databases"
- "Transporting Data Across Platforms"

13.1.2.1.3 Upgrading to a New Release of Oracle Database

You can use full transportable export/import to upgrade a database from an Oracle Database 11*g* Release 2 (11.2.0.3) or later to Oracle Database 19c.

To do so, install Oracle Database 19c and create an empty database. Next, use full transportable export/import to transport the Oracle Database 11*g* Release 2 (11.2.0.3) or later database into the Oracle Database 19c database.

See Also:

- "Transporting Databases"
- *Oracle Database Installation Guide*

13.1.2.2 Scenarios for Transportable Tablespaces or Transportable Tables

The transportable tablespaces or transportable tables feature is useful in several scenarios.

- Scenarios That Apply to Transportable Tablespaces or Transportable Tables
  For some scenarios, either transportable tablespaces or transportable tables can be useful. For other scenarios, only transportable tablespaces can be useful, or only transportable tables can be useful.
- Transporting and Attaching Partitions for Data Warehousing
  You can use transportable tables and tranportable tablespaces to attach partitions for data warehousing.
- Publishing Structured Data on CDs
  Transportable tablespaces and transportable tables both provide a way to publish structured data on CDs.
- Mounting the Same Tablespace Read-Only on Multiple Databases
  You can use transportable tablespaces to mount a tablespace read-only on multiple databases.
- Archiving Historical Data
  When you use transportable tablespaces or transportable tables, the transported data is a self-contained set of files that can be imported into any Oracle database. Therefore, you can archive old or historical data in an enterprise data warehouse using the transportable tablespaces and transportable tables procedures.
- Using Transportable Tablespaces to Perform TSPITR
  You can use transportable tablespaces to perform tablespace point-in-time recovery (TSPITR).
- Copying or Moving Individual Tables
  You can use transportable tables to move a table or a set of tables from one database to another without transporting the entire tablespaces that contain the tables. You can also copy or move individual partitions and subpartitions from one database to another using transportable tables.

13.1.2.2.1 Scenarios That Apply to Transportable Tablespaces or Transportable Tables

For some scenarios, either transportable tablespaces or transportable tables can be useful. For other scenarios, only transportable tablespaces can be useful, or only transportable tables can be useful.

Table 13-1 shows which feature can be used for each scenario.

Table 13-1 Scenarios for Transportable Tablespaces and Transportable Tables

Scenarios	Transportable Tablespaces	Transportable Tables
Transporting and Attaching Partitions for Data Warehousing	Yes	Yes
Publishing Structured Data on CDs	Yes	Yes
Archiving Historical Data	Yes	Yes
Using Transportable Tablespaces to Perform TSPITR	Yes	No
Copying or Moving Individual Tables	No	Yes

The following sections describe these scenarios in more detail.

13.1.2.2.2 Transporting and Attaching Partitions for Data Warehousing

You can use transportable tables and tranportable tablespaces to attach partitions for data warehousing.

Typical enterprise data warehouses contain one or more large fact tables. These fact tables can be partitioned by date, making the enterprise data warehouse a historical database. You can build indexes to speed up star queries. Oracle recommends that you build local indexes for such historically partitioned tables to avoid rebuilding global indexes every time you drop the oldest partition from the historical database.

Suppose every month you would like to load one month of data into the data warehouse. There is a large fact table in the data warehouse called `sales`, which has the following columns:

```
CREATE TABLE sales (invoice_no NUMBER,
   sale_year  INT NOT NULL,
   sale_month INT NOT NULL,
   sale_day   INT NOT NULL)
   PARTITION BY RANGE (sale_year, sale_month, sale_day)
     (partition jan2011 VALUES LESS THAN (2011, 2, 1),
      partition feb2011 VALUES LESS THAN (2011, 3, 1),
      partition mar2011 VALUES LESS THAN (2011, 4, 1),
      partition apr2011 VALUES LESS THAN (2011, 5, 1),
      partition may2011 VALUES LESS THAN (2011, 6, 1),
      partition jun2011 VALUES LESS THAN (2011, 7, 1));
```

You create a local non-prefixed index:

```
CREATE INDEX sales_index ON sales(invoice_no) LOCAL;
```

Initially, all partitions are empty, and are in the same default tablespace. Each month, you want to create one partition and attach it to the partitioned `sales` table.

Suppose it is July 2011, and you would like to load the July sales data into the partitioned table. In a staging database, you create a table, `jul_sales` with the same column types as the `sales` table. Optionally, you can create a new tablespace, `ts_jul`, before you create the table, and create the table in this tablespace. You can create the table `jul_sales` using the `CREATE TABLE ... AS SELECT` statement. After creating and populating `jul_sales`, you can also create an index, `jul_sale_index`, for the table, indexing the same column as the local index in the `sales` table. For detailed information about creating and populating a staging table in a data warehousing environment, see *Oracle Database Data Warehousing Guide*.

After creating the table and building the index, transport the table's data to the data warehouse in one of the following ways:

- You can use transportable tables to transport the `jul_sales` table to the data warehouse.
- If you created the `ts_jul` tablespace, then you can use transportable tablespaces to transport the tablespace `ts_jul` to the data warehouse.

In the data warehouse, add a partition to the `sales` table for the July sales data. This also creates another partition for the local non-prefixed index:

```
ALTER TABLE sales ADD PARTITION jul2011 VALUES LESS THAN (2011, 8, 1);
```

Attach the transported table `jul_sales` to the table `sales` by exchanging it with the new partition:

```
ALTER TABLE sales EXCHANGE PARTITION jul2011 WITH TABLE jul_sales
   INCLUDING INDEXES
   WITHOUT VALIDATION;
```

This statement places the July sales data into the new partition `jul2011`, attaching the new data to the partitioned table. This statement also converts the index `jul_sale_index` into a partition of the local index for the `sales` table. This statement should return immediately, because it only operates on the structural information and it simply switches database pointers. If you know that the data in the new partition does not overlap with data in previous partitions, you are advised to specify the `WITHOUT VALIDATION` clause. Otherwise, the statement goes through all the new data in the new partition in an attempt to validate the range of that partition.

If all partitions of the `sales` table came from the same staging database (the staging database is never destroyed), then the exchange statement always succeeds. In general, however, if data in a partitioned table comes from different databases, then the exchange operation might fail. For example, if the `jan2011` partition of `sales` did not come from the same staging database, then the preceding exchange operation can fail, returning the following error:

```
ORA-19728: data object number conflict between table JUL_SALES and partition JAN2011 in
table SALES
```

To resolve this conflict, move the offending partition by issuing the following statement:

```
ALTER TABLE sales MOVE PARTITION jan2011;
```

Then retry the exchange operation.

After the exchange succeeds, you can safely drop `jul_sales` and `jul_sale_index` (both are now empty). Thus you have successfully loaded the July sales data into your data warehouse.

13.1.2.2.3 Publishing Structured Data on CDs

Transportable tablespaces and transportable tables both provide a way to publish structured data on CDs.

You can copy the data to be published, including the data files and export dump file, to a CD. This CD can then be distributed. If you are using transportable tablespaces, then you must generate a transportable set before copying the data to the CD.

When customers receive this CD, they can add the CD contents to an existing database without having to copy the data files from the CD to disk storage. For example, suppose on a Microsoft Windows system D: drive is the CD drive. You can import the data in data file `catalog.f` and the export dump file `expdat.dmp` as follows:

```
impdp user_name/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_DATAFILES='D:\catalog.f'
```

You can remove the CD while the database is still up. Subsequent queries to the data return an error indicating that the database cannot open the data files on the CD. However, operations to other parts of the database are not affected. Placing the CD back into the drive makes the data readable again.

Removing the CD is the same as removing the data files of a read-only tablespace. If you shut down and restart the database, then the database indicates that it cannot find the removed data file and does not open the database (unless you set the initialization parameter `READ_ONLY_OPEN_DELAYED` to `TRUE`). When `READ_ONLY_OPEN_DELAYED` is set to `TRUE`, the database reads the file only when someone queries the data. Thus, when transporting data from a CD, set the `READ_ONLY_OPEN_DELAYED` initialization parameter to `TRUE`, unless the CD is permanently attached to the database.

13.1.2.2.4 Mounting the Same Tablespace Read-Only on Multiple Databases

You can use transportable tablespaces to mount a tablespace read-only on multiple databases.

In this way, separate databases can share the same data on disk instead of duplicating data on separate disks. The tablespace data files must be accessible by all databases. To avoid database corruption, the tablespace must remain read-only in all the databases mounting the tablespace, and the tablespace's data files must be read-only at the operating system level.

The following are two scenarios for mounting the same tablespace read-only on multiple databases:

- The tablespace originates in a database that is separate from the databases that will share the tablespace.

  You generate a transportable set in the source database, put the transportable set onto a disk that is accessible to all databases, and then import the metadata into each database on which you want to mount the tablespace.
- The tablespace already belongs to one of the databases that will share the tablespace.

  It is assumed that the data files are already on a shared disk. In the database where the tablespace already exists, you make the tablespace read-only, generate the transportable set, and then import the tablespace into the other databases, leaving the data files in the same location on the shared disk.

You can make a disk accessible by multiple computers in several ways. You can use either a cluster file system or raw disk. You can also use network file system (NFS), but be aware that if a user queries the shared tablespace while NFS is down, the database will hang until the NFS operation times out.

Later, you can drop the read-only tablespace in some of the databases. Doing so does not modify the data files for the tablespace. Thus, the drop operation does not corrupt the tablespace. Do not make the tablespace read/write unless only one database is mounting the tablespace.

13.1.2.2.5 Archiving Historical Data

When you use transportable tablespaces or transportable tables, the transported data is a self-contained set of files that can be imported into any Oracle database. Therefore, you can archive old or historical data in an enterprise data warehouse using the transportable tablespaces and transportable tables procedures.

See Also:

Oracle Database Data Warehousing Guide for more details

13.1.2.2.6 Using Transportable Tablespaces to Perform TSPITR

You can use transportable tablespaces to perform tablespace point-in-time recovery (TSPITR).

See Also:

Oracle Database Backup and Recovery User's Guide for information about how to perform TSPITR using transportable tablespaces

13.1.2.2.7 Copying or Moving Individual Tables

You can use transportable tables to move a table or a set of tables from one database to another without transporting the entire tablespaces that contain the tables. You can also copy or move individual partitions and subpartitions from one database to another using transportable tables.

See Also:

"Transporting Tables, Partitions, or Subpartitions Between Databases"

13.1.3 Transporting Data Across Platforms

You can transport data across platforms.

The functionality of transporting data across platforms can be used to:

- Enable a database to be migrated from one platform to another.
- Provide an easier and more efficient means for content providers to publish structured data and distribute it to customers running Oracle Database on different platforms.
- Simplify the distribution of data from a data warehouse environment to data marts, which are often running on smaller platforms.
- Enable the sharing of read-only tablespaces between Oracle Database installations on different operating systems or platforms, assuming that your storage system is accessible from those platforms and the platforms all have the same endianness, as described in the sections that follow.

Many, but not all, platforms are supported for cross-platform data transport. You can query the `V$TRANSPORTABLE_PLATFORM` view to see the platforms that are supported, and to determine each platform's endian format (byte ordering). The following query displays the platforms that support cross-platform data transport:

```
COLUMN PLATFORM_NAME FORMAT A40
COLUMN ENDIAN_FORMAT A14
```

```
SELECT PLATFORM_ID, PLATFORM_NAME, ENDIAN_FORMAT
  FROM V$TRANSPORTABLE_PLATFORM
  ORDER BY PLATFORM_ID;

PLATFORM_ID PLATFORM_NAME                            ENDIAN_FORMAT
----------- ---------------------------------------- --------------
          1 Solaris[tm] OE (32-bit)                  Big
          2 Solaris[tm] OE (64-bit)                  Big
          3 HP-UX (64-bit)                           Big
          4 HP-UX IA (64-bit)                        Big
          5 HP Tru64 UNIX                            Little
          6 AIX-Based Systems (64-bit)               Big
          7 Microsoft Windows IA (32-bit)            Little
          8 Microsoft Windows IA (64-bit)            Little
          9 IBM zSeries Based Linux                  Big
         10 Linux IA (32-bit)                        Little
         11 Linux IA (64-bit)                        Little
         12 Microsoft Windows x86 64-bit             Little
         13 Linux x86 64-bit                         Little
         15 HP Open VMS                              Little
         16 Apple Mac OS                             Big
         17 Solaris Operating System (x86)           Little
         18 IBM Power Based Linux                    Big
         19 HP IA Open VMS                           Little
         20 Solaris Operating System (x86-64)        Little
         21 Apple Mac OS (x86-64)                    Little
```

If the source platform and the target platform are of the same endianness, then the data is transported from the source platform to the target platform without any data conversion.

If the source platform and the target platform are of different endianness, then the data being transported must be converted to the target platform format. You can convert the data using one of the following methods:

- The `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package

  When you use one of these procedures to move data files between the source platform and the target platform, each block in each data file is converted to the target platform's endianness. The conversion occurs on the target platform.

- The RMAN `CONVERT` command

  Run the RMAN `CONVERT` command on the source or target platform. This command converts the data being transported to the target platform format.

> **Note:**
>
> Conversion of data files between different endian formats is not supported for data files having undo segments.

Before the data in a data file can be transported to a different platform, the data file header must identify the platform to which it belongs. To transport read-only tablespaces between Oracle Database installations on different platforms, make the data file read/write at least once.

> **See Also:**
>
> "Converting Data Between Platforms"

13.1.4 General Limitations on Transporting Data

There are general limitations on transporting data. There are also limitations that are specific to full transportable export/import, transportable tablespaces, or transportable tables.

Be aware of the following general limitations as you plan to transport data:

- The source and the target databases must use compatible database character sets. Specifically, one of the following must be true:
  - The database character sets of the source and the target databases are the same.
  - The source database character set is a strict (binary) subset of the target database character set, and the following three conditions are true:
    - The source database is Oracle Database 10g Release 1 (10.1.0.3) or later.
    - The tablespaces that you transport contain no table columns with character length semantics, or the maximum character width is the same in both the source and target database character sets.
    - The data that you transport contain no columns with the `CLOB` data type, or the source and the target database character sets are both single-byte or both multibyte.
  - The source database character set is a strict (binary) subset of the target database character set, and the following two conditions are true:
    - The source database is earlier than Oracle Database 10g Release 1 (10.1.0.3).
    - The maximum character width is the same in the source and target database character sets.

  > **Note:**
  >
  > The subset-superset relationship between character sets recognized by Oracle Database is documented in *Oracle Database Globalization Support Guide*.

- The source and the target databases must use compatible national character sets. Specifically, one of the following must be true:
  - The national character sets of the source and target databases are the same.
  - The source database is Oracle Database 10g Release 1 (10.1.0.3) or later, and the tablespaces to be transported contain no columns with `NCHAR`, `NVARCHAR2`, or `NCLOB` data types.
- When running a transportable export operation, the following limitations apply:
  - The default tablespace of the user performing the export must not be one of the tablespaces being transported.
  - The default tablespace of the user performing the export must be writable.

- In a CDB, you cannot transport a tablespace to a target container that contains a tablespace of the same name. However, different containers can have tablespaces with the same name.

  You can use the `REMAP_TABLESPACE` import parameter to import the database objects into a different tablespace. Alternatively, before the transport operation, you can rename either the tablespace to be transported or the target tablespace.

- Transporting data with XMLTypes has the following limitations:
  - The target database must have XML DB installed.
  - Schemas referenced by XMLType tables cannot be the XML DB standard schemas.
  - If the schema for a transported XMLType table is not present in the target database, then it is imported and registered. If the schema already exists in the target database, then a message is displayed during import.
  - You must use only Oracle Data Pump to export and import the metadata for data that contains XMLTypes.

  The following query returns a list of tablespaces that contain XMLTypes:

  ```
  select distinct p.tablespace_name from dba_tablespaces p,
    dba_xml_tables x, dba_users u, all_all_tables t where
    t.table_name=x.table_name and t.tablespace_name=p.tablespace_name
    and x.owner=u.username;
  ```

  See *Oracle XML DB Developer's Guide* for information on XMLTypes.

- Types whose interpretation is application-specific and opaque to the database (such as `RAW`, `BFILE`, and AnyType can be transported, but they are not converted as part of the cross-platform transport operation. Their actual structure is known only to the application, so the application must address any endianness issues after these types are moved to the new platform. Types and objects that use these opaque types, either directly or indirectly, are also subject to this limitation.

- When you transport a tablespace containing tables with `TIMESTAMP WITH LOCAL TIME ZONE` (TSLTZ) data between databases with different time zones, the tables with the TSLTZ data are not transported. Error messages describe the tables that were not transported. However, tables in the tablespace that do not contain TSLTZ data are transported.

  You can determine the time zone of a database with the following query:

  ```
  SELECT DBTIMEZONE FROM DUAL;
  ```

  You can alter the time zone for a database with an `ALTER DATABASE` SQL statement.

  You can use Oracle Data Pump to perform a conventional export/import of tables with TSLTZ data after the transport operation completes.

- Analytic workspaces cannot be part of cross-platform transport operations. If the source platform and target platform are different, then use Data Pump export/import to export and import analytic workspaces. See *Oracle OLAP DML Reference* for more information about analytic workspaces.

Note:

Do not run the Oracle Data Pump export utility `expdp` or import utility `impdp` as `SYSDBA`, except at the request of Oracle technical support. `SYSDBA` is used internally and has specialized functions; its behavior is not the same as for general users.

Related Topics

- Limitations on Full Transportable Export/import
  There are limitations on full transportable export/import.
- Limitations on Transportable Tablespaces
  This section lists the limitations on transportable tablespace.
- Limitations on Transportable Tables
  There are limitations on transportable tables.
- Character Sets
- XMLType Data Type

13.1.5 Compatibility Considerations for Transporting Data

When transporting data, Oracle Database computes the lowest compatibility level at which the target database must run.

You can transport a tablespace or a table from a source database to a target database having the same or higher compatibility setting using transportable tablespaces, even if the target database is on the same or a different platform. The data transport operation fails if the compatibility level of the source database is higher than the compatibility level of the target database.

The following table shows the minimum compatibility requirements of the source and target databases in various scenarios. The source and target database need not have the same compatibility setting.

Table 13-2 Minimum Compatibility Requirements

Transport Scenario	Minimum Compatibility Setting	
	Source Database	**Target Database**
Transporting a database using full transportable export/import	12.0 (`COMPATIBLE` initialization parameter setting for an Oracle Database 12*c* or later database 12 (`VERSION` Data Pump export parameter setting for an 11.2.0.3 or later database)	12.0 (`COMPATIBLE` initialization parameter setting)
Transporting a tablespace between databases on the same platform using transportable tablespaces	8.0 (`COMPATIBLE` initialization parameter setting)	8.0 (`COMPATIBLE` initialization parameter setting)
Transporting a tablespace with different database block size than the target database using transportable tablespaces	9.0 (`COMPATIBLE` initialization parameter setting)	9.0 (`COMPATIBLE` initialization parameter setting)
Transporting a tablespace between databases on different platforms using transportable tablespaces	10.0 (`COMPATIBLE` initialization parameter setting)	10.0 (`COMPATIBLE` initialization parameter setting)
Transporting tables between databases	11.2.0 (`COMPATIBLE` initialization parameter setting for an Oracle Database 12*c* or later database	11.2.0 (`COMPATIBLE` initialization parameter setting)

Note:

- When you use full transportable export/import, the source database must be an Oracle Database 11*g* Release 2 (11.2.0.3) or later database, and the target database must be an Oracle Database 12*c* or later database.
- When transporting a database from Oracle Database 11*g* Release 2 (11.2.0.3) or later database to Oracle Database 12*c* or later database, you must set the Data Pump export parameter `VERSION` to `12` or higher.
- When transporting a database from an Oracle Database 19c database to an Oracle Database 19c database, you must set the initialization parameter `COMPATIBLE` to `19.0.0` or higher.

13.2 Transporting Databases

You can transport a database to a new Oracle Database instance.

- Introduction to Full Transportable Export/Import
  You can use the full transportable export/import feature to copy an entire database from one Oracle Database instance to another.
- Limitations on Full Transportable Export/import
  There are limitations on full transportable export/import.
- Transporting a Database Using an Export Dump File
  You can transport a database using an export dump file.
- Transporting a Database Over the Network
  You can transport a database over the network.

13.2.1 Introduction to Full Transportable Export/Import

You can use the full transportable export/import feature to copy an entire database from one Oracle Database instance to another.

You can use Oracle Data Pump to produce an export dump file, transport the dump file to the target database if necessary, and then import the export dump file. Alternatively, you can use Oracle Data Pump to copy the database over the network.

The tablespaces in the database being transported can be either dictionary managed or locally managed. The tablespaces in the database are not required to be of the same block size as the target database standard block size.

Starting with Oracle Database Release 21c, you can use Oracle Data Pump to export databases to and import databases from the object store in Oracle Cloud. This simplifies migration to Oracle Cloud.

Note:

This method for transporting a database requires that you place the user-defined tablespaces in the database in read-only mode until you complete the export. If this is undesirable, then you can use the transportable tablespaces from backup feature described in *Oracle Database Backup and Recovery User's Guide*.

Related Topics

- Exporting Data from On Premises Databases to Oracle Autonomous Databases

See Also:

"About Transporting Data"

13.2.2 Limitations on Full Transportable Export/import

There are limitations on full transportable export/import.

Be aware of the following limitations on full transportable export/import:

- The general limitations described in "General Limitations on Transporting Data" apply to full transportable export/import.
- Full transportable export/import can export and import user-defined database objects in administrative tablespaces using conventional Data Pump export/import, such as direct path or external table. Administrative tablespaces are non-user tablespaces supplied with Oracle Database, such as the `SYSTEM` and `SYSAUX` tablespaces.
- Full transportable export/import cannot transport a database object that is defined in both an administrative tablespace (such as `SYSTEM` and `SYSAUX`) and a user-defined tablespace. For example, a partitioned table might be stored in both a user-defined tablespace and an administrative tablespace. If you have such database objects in your database, then you can redefine them before transporting them so that they are stored entirely in either an administrative tablespace or a user-defined tablespace. If the database objects cannot be redefined, then you can use conventional Data Pump export/import.
- When transporting a database over the network using full transportable export/import, auditing cannot be enabled for tables stored in an administrative tablespace (such as `SYSTEM` and `SYSAUX`) when the audit trail information itself is stored in a user-defined tablespace. See *Oracle Database Security Guide* for more information about auditing.
- Full transportable database import cannot be executed with Transaction Guard enabled. Disable Transaction Guard during a full database import.

Related Topics

- Introduction to Auditing

13.2.3 Transporting a Database Using an Export Dump File

You can transport a database using an export dump file.

The following list of tasks summarizes the process of transporting a database using an export dump file. Details for each task are provided in the subsequent example.

1. At the source database, configure each of the user-defined tablespaces in read-only mode and export the database.

   Ensure that the following parameters are set to the specified values:

   - `TRANSPORTABLE=ALWAYS`
   - `FULL=Y`

   If the source database is an Oracle Database 11*g* database (11.2.0.3 or later), then you must set the `VERSION` parameter to `12` or higher.

   If the source database contains any encrypted tablespaces or tablespaces containing tables with encrypted columns, then you must either specify `ENCRYPTION_PWD_PROMPT=YES`, or specify the `ENCRYPTION_PASSWORD` parameter.

   The export dump file includes the metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces, such as `SYSTEM` and `SYSAUX`.

2. Transport the export dump file.

   Copy the export dump file to a place that is accessible to the target database.

3. Transport the data files for all of the user-defined tablespaces in the database.

   Copy the data files to a place that is accessible to the target database.

   If the source platform and target platform are different, then check the endian format of each platform by running the query on the `V$TRANSPORTABLE_PLATFORM` view in "Transporting Data Across Platforms".

   If the source platform's endian format is different from the target platform's endian format, then use one of the following methods to convert the data files:

   - Use the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically.
   - Use the RMAN `CONVERT` command to convert the data files to the target platform's endian format.

   > **Note:**
   >
   > Conversion of data files between different endian formats is not supported for data files having undo segments.

   See "Converting Data Between Platforms" for more information.

4. (Optional) Restore the user-defined tablespaces to read/write mode on the source database.

5. At the target database, import the database.

When the import is complete, the user-defined tablespaces are in read/write mode.

Example

The tasks for transporting a database are illustrated in detail in this example. This example assumes that the source platform is Solaris and the target platform is Microsoft Windows.

It also assumes that the source platform has the following data files and tablespaces:

Tablespace	Type	Data File
sales	User-defined	/u01/app/oracle/oradata/mydb/sales01.dbf
customers	User-defined	/u01/app/oracle/oradata/mydb/cust01.dbf
employees	User-defined	/u01/app/oracle/oradata/mydb/emp01.dbf
SYSTEM	Administrative	/u01/app/oracle/oradata/mydb/system01.dbf
SYSAUX	Administrative	/u01/app/oracle/oradata/mydb/sysaux01.dbf

This example makes the following additional assumptions:

- The target database is a new database that is being populated with the data from the source database. The name of the source database is `mydb`.
- Both the source database and the target database are Oracle Database 19c databases.

Complete the following tasks to transport the database using an export dump file:

Task 1 Generate the Export Dump File
Generate the export dump file by completing the following steps:

1. Start SQL*Plus and connect to the database as an administrator or as a user who has either the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.
2. Make all of the user-defined tablespaces in the database read-only.

   ```
   ALTER TABLESPACE sales READ ONLY;

   ALTER TABLESPACE customers READ ONLY;

   ALTER TABLESPACE employees READ ONLY;
   ```

3. Invoke the Data Pump export utility as a user with `DATAPUMP_EXP_FULL_DATABASE` role and specify the full transportable export/import options.

   ```
   SQL> HOST

   $ expdp user_name full=y dumpfile=expdat.dmp directory=data_pump_dir
           transportable=always logfile=export.log

   Password: password
   ```

   You must always specify `TRANSPORTABLE=ALWAYS`, which determines whether the transportable option is used.

   This example specifies the following Data Pump parameters:

   - The `FULL` parameter specifies that the entire database is being exported.
   - The `DUMPFILE` parameter specifies the name of the structural information export dump file to be created, `expdat.dmp`.

- The DIRECTORY parameter specifies the directory object that points to the operating system or Oracle Automatic Storage Management location of the dump file. You must create the DIRECTORY object before invoking Data Pump, and you must grant the READ and WRITE object privileges on the directory to the user running the Export utility. See *Oracle Database SQL Language Reference* for information on the CREATE DIRECTORY command.

  In a non-CDB, the directory object DATA_PUMP_DIR is created automatically. Read and write access to this directory is automatically granted to the DBA role, and thus to users SYS and SYSTEM.

  However, the directory object DATA_PUMP_DIR is not created automatically in a PDB. Therefore, when importing into a PDB, create a directory object in the PDB and specify the directory object when you run Data Pump.

  See Also:

  – *Oracle Database Utilities* for information about the default directory when the DIRECTORY parameter is omitted

  – *Oracle Multitenant Administrator's Guide* for more information about PDBs

- The LOGFILE parameter specifies the file name of the log file to be written by the export utility. In this example, the log file is written to the same directory as the dump file, but it can be written to a different location.

To perform a full transportable export on an Oracle Database 11*g* Release 2 (11.2.0.3) or later Oracle Database 11*g* database, use the VERSION parameter, as shown in the following example:

```
expdp user_name full=y dumpfile=expdat.dmp directory=data_pump_dir
      transportable=always version=12 logfile=export.log
```

Full transportable import is supported only for Oracle Database 12*c* and later databases.

Note:

In this example, the Data Pump utility is used to export only data dictionary structural information (metadata) for the user-defined tablespaces. Actual data is unloaded only for the administrative tablespaces (SYSTEM and SYSAUX), so this operation goes relatively quickly even for large user-defined tablespaces.

4. Check the log file for errors, and take note of the dump file and data files that you must transport to the target database. expdp outputs the names and paths of these files in messages like these:

```
******************************************************************************
Dump file set for SYSTEM.SYS_EXPORT_TRANSPORTABLE_01 is:
  /u01/app/oracle/admin/mydb/dpdump/expdat.dmp
******************************************************************************
Datafiles required for transportable tablespace SALES:
  /u01/app/oracle/oradata/mydb/sales01.dbf
Datafiles required for transportable tablespace CUSTOMERS:
  /u01/app/oracle/oradata/mydb/cust01.dbf
Datafiles required for transportable tablespace EMPLOYEES:
  /u01/app/oracle/oradata/mydb/emp01.dbf
```

5. When finished, exit back to SQL*Plus:

```
$ exit
```

See Also:

Oracle Database Utilities for information about using the Data Pump utility

Task 2 Transport the Export Dump File

Transport the dump file to the directory pointed to by the `DATA_PUMP_DIR` directory object, or to any other directory of your choosing. The new location must be accessible to the target database.
At the target database, run the following query to determine the location of `DATA_PUMP_DIR`:

```
SELECT * FROM DBA_DIRECTORIES WHERE DIRECTORY_NAME = 'DATA_PUMP_DIR';

OWNER      DIRECTORY_NAME   DIRECTORY_PATH
---------- ---------------- -----------------------------------
SYS        DATA_PUMP_DIR    C:\app\orauser\admin\orawin\dpdump\
```

Task 3 Transport the Data Files for the User-Defined Tablespaces

Transport the data files of the user-defined tablespaces in the database to a place that is accessible to the target database.
In this example, transfer the following data files from the source database to the target database:

- `sales01.dbf`
- `cust01.dbf`
- `emp01.dbf`

If you are transporting the database to a platform different from the source platform, then determine if cross-platform database transport is supported for both the source and target platforms, and determine the endianness of each platform. If both platforms have the same endianness, then no conversion is necessary. Otherwise, you must do a conversion of each tablespace in the database, either at the source database or at the target database.
If you are transporting the database to a different platform, you can execute the following query on each platform. If the query returns a row, then the platform supports cross-platform tablespace transport.

```
SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
     FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
     WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
```

The following is the query result from the source platform:

```
PLATFORM_NAME                      ENDIAN_FORMAT
---------------------------------- --------------
Solaris[tm] OE (32-bit)            Big
```

The following is the query result from the target platform:

```
PLATFORM_NAME                      ENDIAN_FORMAT
---------------------------------- --------------
Microsoft Windows IA (32-bit)      Little
```

In this example, you can see that the endian formats are different. Therefore, in this case, a conversion is necessary for transporting the database. Use either the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically. Transport the data files to the location of the existing data files of the target database. On the UNIX and Linux platforms, this location is typically /u01/app/oracle/oradata/*dbname*/ or +*DISKGROUP*/*dbname*/datafile/. Alternatively, you can use the RMAN `CONVERT` command to convert the data files. See "Converting Data Between Platforms" for more information.

Note:

If no endianness conversion of the tablespaces is needed, then you can transfer the files using any file transfer method.

Task 4 (Optional) Restore Tablespaces to Read/Write Mode
Make the transported tablespaces read/write again at the source database, as follows:

```
ALTER TABLESPACE sales READ WRITE;
ALTER TABLESPACE customers READ WRITE;
ALTER TABLESPACE employees READ WRITE;
```

You can postpone this task to first ensure that the import process succeeds.

Task 5 At the Target Database, Import the Database
Invoke the Data Pump import utility as a user with `DATAPUMP_IMP_FULL_DATABASE` role and specify the full transportable export/import options.

```
impdp user_name full=Y dumpfile=expdat.dmp directory=data_pump_dir
   transport_datafiles=
      '/u01/app/oracle/oradata/mydb/sales01.dbf',
      '/u01/app/oracle/oradata/mydb/cust01.dbf',
      '/u01/app/oracle/oradata/mydb/emp01.dbf'
   logfile=import.log

Password: password
```

This example specifies the following Data Pump parameters:

- The `FULL` parameter specifies that the entire database is being imported in `FULL` mode.
- The `DUMPFILE` parameter specifies the exported file containing the metadata for the user-defined tablespaces and both the metadata and data for the administrative tablespaces to be imported.
- The `DIRECTORY` parameter specifies the directory object that identifies the location of the export dump file. You must create the `DIRECTORY` object before invoking Data Pump, and you must grant the `READ` and `WRITE` object privileges on the directory to the user running the Import utility. See *Oracle Database SQL Language Reference* for information on the `CREATE DIRECTORY` command.

In a non-CDB, the directory object `DATA_PUMP_DIR` is created automatically. Read and write access to this directory is automatically granted to the `DBA` role, and thus to users `SYS` and `SYSTEM`.

However, the directory object `DATA_PUMP_DIR` is not created automatically in a PDB. Therefore, when importing into a PDB, create a directory object in the PDB and specify the directory object when you run Data Pump.

See Also:

– *Oracle Database Utilities* for information about the default directory when the `DIRECTORY` parameter is omitted

– *Oracle Multitenant Administrator's Guide* for more information about PDBs

- The `TRANSPORT_DATAFILES` parameter identifies all of the data files to be imported.

  You can specify the `TRANSPORT_DATAFILES` parameter multiple times in a parameter file specified with the `PARFILE` parameter if there are many data files.

- The `LOGFILE` parameter specifies the file name of the log file to be written by the import utility. In this example, the log file is written to the directory from which the dump file is read, but it can be written to a different location.

After this statement executes successfully, check the import log file to ensure that no unexpected error has occurred.
When dealing with a large number of data files, specifying the list of data file names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can invoke the Data Pump import utility as follows:

```
impdp user_name parfile='par.f'
```

For example, `par.f` might contain the following lines:

```
FULL=Y
DUMPFILE=expdat.dmp
DIRECTORY=data_pump_dir
TRANSPORT_DATAFILES=
'/u01/app/oracle/oradata/mydb/sales01.dbf',
'/u01/app/oracle/oradata/mydb/cust01.dbf',
'/u01/app/oracle/oradata/mydb/emp01.dbf'
LOGFILE=import.log
```

Note:

- During the import, user-defined tablespaces might be temporarily made read/write for metadata loading. Ensure that no user changes are made to the data during the import. At the successful completion of the import, all user-defined tablespaces are made read/write.
- When performing a network database import, the `TRANSPORTABLE` parameter must be set to `always`.
- When you are importing into a PDB in a CDB, specify the connect identifier for the PDB after the user name. For example, if the connect identifier for the PDB is `hrpdb`, then enter the following when you run the Oracle Data Pump Import utility:

  ```
  impdp user_name@hrpdb ...
  ```

See Also:

- *Oracle Database Utilities* for information about using the import utility
- *Oracle Multitenant Administrator's Guide*

13.2.4 Transporting a Database Over the Network

You can transport a database over the network.

To transport a database over the network, you perform an import using the `NETWORK_LINK` parameter, the import is performed using a database link, and there is no dump file involved.

The following list of tasks summarizes the process of transporting a database over the network. Details for each task are provided in the subsequent example.

1. Create a database link from the target database to the source database.

   The import operation must be performed by a user on the target database with `DATAPUMP_IMP_FULL_DATABASE` role, and the database link must connect to a user on the source database with `DATAPUMP_EXP_FULL_DATABASE` role. The user on the source database cannot be a user with `SYSDBA` administrative privilege. If the database link is a connected user database link, then the user on the target database cannot be a user with `SYSDBA` administrative privilege. See "Users of Database Links" for information about connected user database links.
2. In the source database, make the user-defined tablespaces in the database read-only.
3. Transport the data files for the all of the user-defined tablespaces in the database.

   Copy the data files to a place that is accessible to the target database.

   If the source platform and target platform are different, then check the endian format of each platform by running the query on the `V$TRANSPORTABLE_PLATFORM` view in "Transporting Data Across Platforms".

   If the source platform's endian format is different from the target platform's endian format, then use one of the following methods to convert the data files:

- Use the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically.
- Use the RMAN `CONVERT` command to convert the data files to the target platform's endian format.

> **Note:**
>
> Conversion of data files between different endian formats is not supported for data files having undo segments.

See "Converting Data Between Platforms" for more information.

4. At the target database, import the database.

   Invoke the Data Pump utility to import the metadata for the user-defined tablespaces and both the metadata and data for the administrative tablespaces.

   Ensure that the following parameters are set to the specified values:

   - `TRANSPORTABLE=ALWAYS`
   - `TRANSPORT_DATAFILES=`*`list_of_datafiles`*
   - `FULL=Y`
   - `NETWORK_LINK=`*`source_database_link`*

     Replace *`source_database_link`* with the name of the database link to the source database.
   - `VERSION=12`

     If the source database is an Oracle Database 11*g* Release 2 (11.2.0.3) or later Oracle Database 11*g* database, then the `VERSION` parameter is required and must be set to `12`. If the source database is an Oracle Database 12*c* or later database, then the `VERSION` parameter is not required.

   If the source database contains any encrypted tablespaces or tablespaces containing tables with encrypted columns, then you must either specify `ENCRYPTION_PWD_PROMPT=YES`, or specify the `ENCRYPTION_PASSWORD` parameter.

   The Data Pump network import copies the metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces, such as `SYSTEM` and `SYSAUX`.

   When the import is complete, the user-defined tablespaces are in read/write mode.

5. (Optional) Restore the user-defined tablespaces to read/write mode on the source database.

Example

These tasks for transporting a database are illustrated more fully in the example that follows, where it is assumed the following data files and tablespaces exist:

Tablespace	Type	Data File
sales	User-defined	/u01/app/oracle/oradata/mydb/sales01.dbf
customers	User-defined	/u01/app/oracle/oradata/mydb/cust01.dbf

Tablespace	Type	Data File
employees	User-defined	/u01/app/oracle/oradata/mydb/emp01.dbf
SYSTEM	Administrative	/u01/app/oracle/oradata/mydb/system01.dbf
SYSAUX	Administrative	/u01/app/oracle/oradata/mydb/sysaux01.dbf

This example makes the following additional assumptions:

- The target database is a new database that is being populated with the data from the source database. The name of the source database is `sourcedb`.
- The source database and target database are running on the same platform with the same endianness.

  To check the endianness of a platform, run the following query:

  ```
  SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
       FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
       WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
  ```

- The `sales` tablespace is encrypted. The other tablespaces are not encrypted.
- The source database is an Oracle Database 11*g* Release 2 (11.2.0.3) database and the target database is an Oracle Database 19c database.

Note:

This example illustrates the tasks required to transport an Oracle Database 11*g* Release 2 (11.2.0.3) to a new Oracle Database 19c PDB inside a CDB.

See Also:

Oracle Multitenant Administrator's Guide

Complete the following tasks to transport the database over the network:

Task 1 Create a Database Link from the Target Database to the Source Database
Create a database link from the target database to the source database by completing the following steps:

1. Ensure that network connectivity is configured between the source database and the target database.

   See *Oracle Database Net Services Administrator's Guide* for instructions.
2. Start SQL*Plus and connect to the target database as the administrator who will transport the database with Data Pump import. This user must have `DATAPUMP_IMP_FULL_DATABASE` role to transport the database.

   See "Connecting to the Database with SQL*Plus" for instructions.
3. Create the database link:

```
CREATE PUBLIC DATABASE LINK sourcedb USING 'sourcedb';
```

Specify the service name for the source database in the using clause.

During the import operation, the database link must connect to a user on the source database with `DATAPUMP_EXP_FULL_DATABASE` role. The user on the source database cannot be a user with `SYSDBA` administrative privilege.

See Also:

- "Creating Database Links"
- *Oracle Database SQL Language Reference*

Task 2 Make the User-Defined Tablespaces Read-Only
Complete the following steps:

1. Start SQL*Plus and connect to the source database as an administrator or as a user who has either the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

   See "Connecting to the Database with SQL*Plus" for instructions.

2. Make all of the user-defined tablespaces in the database read-only.

   ```
   ALTER TABLESPACE sales READ ONLY;

   ALTER TABLESPACE customers READ ONLY;

   ALTER TABLESPACE employees READ ONLY;
   ```

Task 3 Transport the Data Files for the User-Defined Tablespaces
Transport the data files to the location of the existing data files of the target database.
On the UNIX and Linux platforms, this location is typically /u01/app/oracle/oradata/*dbname*/ or +*DISKGROUP*/*dbname*/datafile/.
In this example, transfer the following data files from the source database to the target database:

- `sales01.dbf`
- `cust01.dbf`
- `emp01.dbf`

See Also:

"Guidelines for Transferring Data Files"

Task 4 At the Target Database, Import the Database
Invoke the Data Pump import utility as a user with `DATAPUMP_IMP_FULL_DATABASE` role and specify the full transportable export/import options.

```
impdp user_name full=Y network_link=sourcedb transportable=always
   transport_datafiles=
      '/u01/app/oracle/oradata/mydb/sales01.dbf',
      '/u01/app/oracle/oradata/mydb/cust01.dbf',
```

```
      '/u01/app/oracle/oradata/mydb/emp01.dbf'
   encryption_pwd_prompt=YES version=12 logfile=import.log

Password: password
```

This example specifies the following Data Pump parameters:

- The `FULL` parameter specifies that the entire database is being imported in `FULL` mode.
- The `NETWORK_LINK` parameter specifies the database link used for the network import.
- The `TRANSPORTABLE` parameter specifies that the import uses the transportable option.
- The `TRANSPORT_DATAFILES` parameter identifies all of the data files to be imported.

  You can specify the `TRANSPORT_DATAFILES` parameter multiple times in a parameter file specified with the `PARFILE` parameter if there are many data files.
- The `ENCRYPTION_PWD_PROMPT` parameter instructs Data Pump to prompt you for the encryption password, and Data Pump encrypts data and metadata sent over the network connection. Either the `ENCRYPTION_PWD_PROMPT` parameter or the `ENCRYPTION_PASSWORD` parameter is required when encrypted tablespaces or tables with encrypted columns are part of the import operation.
- The `VERSION` parameter is set to `12` because the source database is an Oracle Database 11*g* Release 2 (11.2.0.3) or later Oracle Database 11*g* database.
- The `LOGFILE` parameter specifies the file name of the log file to be written by the import utility.

After this statement executes successfully, check the import log file to ensure that no unexpected error has occurred.
When dealing with a large number of data files, specifying the list of data file names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file.
Use of an import parameter file is also recommended when encrypted tablespaces or tables with encrypted columns are part of the import operation. In this case, specify `ENCRYPTION_PWD_PROMPT=YES` in the import parameter file.
For example, you can invoke the Data Pump import utility as follows:

```
impdp user_name parfile='par.f'
```

For example, `par.f` might contain the following lines:

```
FULL=Y
NETWORK_LINK=sourcedb
TRANSPORTABLE=always
TRANSPORT_DATAFILES=
'/u01/app/oracle/oradata/mydb/sales01.dbf',
'/u01/app/oracle/oradata/mydb/cust01.dbf',
'/u01/app/oracle/oradata/mydb/emp01.dbf'
ENCRYPTION_PWD_PROMPT=YES
VERSION=12
LOGFILE=import.log
```

Note:

- During the import, user-defined tablespaces might be temporarily made read/write for metadata loading. Ensure that no user changes are made to the data during the import. At the successful completion of the import, all user-defined tablespaces are made read/write.
- When you are importing into a PDB in a CDB, specify the connect identifier for the PDB after the user name. For example, if the connect identifier for the PDB is `hrpdb`, then enter the following when you run the Oracle Data Pump Import utility:

  ```
  impdp user_name@hrpdb ...
  ```

See Also:

Oracle Database Utilities for information about using the import utility

Task 5 (Optional) Restore User-Defined Tablespaces to Read/Write Mode
Make the user-defined tablespaces read/write again at the source database, as follows:

```
ALTER TABLESPACE sales READ WRITE;

ALTER TABLESPACE customers READ WRITE;

ALTER TABLESPACE employees READ WRITE;
```

You can postpone this task to first ensure that the import process succeeds.

13.3 Transporting Tablespaces Between Databases

You can transport tablespaces between databases.

Note:

To import a transportable tablespace set into an Oracle database on a different platform, both databases must have compatibility set to at least 10.0.0. See "Compatibility Considerations for Transporting Data" for a discussion of database compatibility for transporting tablespaces across release levels.

- Introduction to Transportable Tablespaces
  You can use the transportable tablespaces feature to copy a set of tablespaces from one Oracle Database to another.
- Limitations on Transportable Tablespaces
  This section lists the limitations on transportable tablespace.
- Transporting Tablespaces Between Databases
  You can transport a tablespace or a set of tablespaces between databases.

13.3.1 Introduction to Transportable Tablespaces

You can use the transportable tablespaces feature to copy a set of tablespaces from one Oracle Database to another.

The tablespaces being transported can be either dictionary managed or locally managed. The transported tablespaces are not required to be of the same block size as the target database standard block size. These scenarios are discussed in "Transporting Data: Scenarios".

There are two ways to transport a tablespace:

- Manually, following the steps described in this section. This involves issuing commands to SQL*Plus and Data Pump.
- Using the Transport Tablespaces Wizard in Oracle Enterprise Manager Cloud Control

  To run the Transport Tablespaces Wizard:

  1. Log in to Cloud Control with a user that has the `DATAPUMP_EXP_FULL_DATABASE` role.
  2. Access the Database Home page.
  3. From the Schema menu, select **Database Export/Import**, then **Transport Tablespaces**.

Note:

- This method for transporting tablespaces requires that you place the tablespaces to be transported in read-only mode until you complete the transporting process. If this is undesirable, you can use the transportable tablespaces from backup feature, described in *Oracle Database Backup and Recovery User's Guide*.
- You must use Data Pump for transportable tablespaces. The only circumstance under which you can use the original import and export utilities, IMP and EXP, is for a backward migration of XMLType data to an Oracle Database 10*g* Release 2 (10.2) or earlier database. See *Oracle Database Utilities* for more information on these utilities and to *Oracle XML DB Developer's Guide* for more information on XMLTypes.

See Also:

- "About Transporting Data"
- *Oracle Database Data Warehousing Guide* for information about using transportable tablespaces in a data warehousing environment

13.3.2 Limitations on Transportable Tablespaces

This section lists the limitations on transportable tablespace.

Be aware of the following limitations for transportable tablespaces:

- The general limitations described in "General Limitations on Transporting Data" apply to transportable tablespaces.
- When transporting a tablespace set, objects with underlying objects (such as materialized views) or contained objects (such as partitioned tables) are not transportable unless all of the underlying or contained objects are in the tablespace set.
- Transportable tablespaces cannot transport tables with `TIMESTAMP WITH TIMEZONE` (TSTZ) data across platforms with different time zone file versions. The transportable tablespace operation skips these tables. You can export and import these tables conventionally.

  See *Oracle Database Utilities* for more information.
- You cannot include administrative tablespaces, such as `SYSTEM` and `SYSAUX` in a transportable tablespace set.
- Transportable tablespaces cannot contain any tables with columns that are encrypted using TDE column encryption
- If a tablespace is encrypted using TDE, you can only transport this tablespace to a platform that uses the same endian format. If you need to go across endianness, you must decrypt, transport, and re-encrypt the tablespace. Starting with Oracle Database release 12.2 these operations can be performed online.
- Transportable tablespace sets include TDE policies only. Other security policies (such as redaction policies, masking policies, and so on) are not included and they must be recreated after the tablespace is imported into the new database.

13.3.3 Transporting Tablespaces Between Databases

You can transport a tablespace or a set of tablespaces between databases.

Starting with Oracle Database Release 21c, transportable jobs can be restarted at or near the point of failure.

The following list of tasks summarizes the process of transporting a tablespace. Details for each task are provided in the subsequent example.

1. Pick a self-contained set of tablespaces.
2. At the source database, configure the set of tablespaces in read-only mode and generate a transportable tablespace set.

   A transportable tablespace set (or transportable set) consists of data files for the set of tablespaces being transported and an export dump file containing structural information (metadata) for the set of tablespaces. You use Data Pump to perform the export.
3. Transport the export dump file.

   Copy the export dump file to a place that is accessible to the target database.
4. Transport the tablespace set.

   Copy the data files to a directory that is accessible to the target database.

   If the source platform and target platform are different, then check the endian format of each platform by running the query on the `V$TRANSPORTABLE_PLATFORM` view.

   If the source platform's endian format is different from the target platform's endian format, then use one of the following methods to convert the data files:

   - Use the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically.

- Use the RMAN `CONVERT` command to convert the data files to the target platform's endian format.

> **Note:**
>
> Conversion of data files between different endian formats is not supported for data files having undo segments.

5. (Optional) Restore tablespaces to read/write mode on the source database.
6. At the target database, import the tablespace set.

   Run the Data Pump utility to import the metadata for the tablespace set.

Example 13-1 Example

These tasks for transporting a tablespace are illustrated more fully in the example that follows, where it is assumed the following data files and tablespaces exist:

Tablespace	Data File
`sales_1`	`/u01/app/oracle/oradata/salesdb/sales_101.dbf`
`sales_2`	`/u01/app/oracle/oradata/salesdb/sales_201.dbf`

- Task 1: Pick a Self-Contained Set of Tablespaces
  There may be logical or physical dependencies between the database objects in the transportable set and the database objects outside of the transportable set. You can only transport a tablespace set that is self-contained, that is, none of the database objects inside a tablespace set are dependent on any of the database objects outside of that tablespace set.
- Task 2: Generate a Transportable Tablespace Set
  After ensuring that you have a self-contained set of tablespaces that you want to transport, generate a transportable tablespace set.
- Task 3: Transport the Export Dump File
  Transport the dump file to the directory pointed to by the `DATA_PUMP_DIR` directory object, or to any other directory of your choosing. The new location must be accessible to the target database.
- Task 4: Transport the Tablespace Set
  Transport the data files of the tablespaces to a directory that is accessible to the target database.
- Task 5: (Optional) Restore Tablespaces to Read/Write Mode
  Make the transported tablespaces read/write again at the source database.
- Task 6: Import the Tablespace Set
  To complete the transportable tablespaces operation, import the tablespace set.

Related Topics

- Transporting Data Across Platforms
  You can transport data across platforms.
- Converting Data Between Platforms
  When you perform a transportable operation, and the source platform and the target platform are of different endianness, you must convert the data being transported to the target platform format. If the source platform and the target platform are of the same

endianness, then data conversion is not necessary. You can use the `DBMS_FILE_TRANSFER` package or the RMAN `CONVERT` command to convert data.

13.3.3.1 Task 1: Pick a Self-Contained Set of Tablespaces

There may be logical or physical dependencies between the database objects in the transportable set and the database objects outside of the transportable set. You can only transport a tablespace set that is self-contained, that is, none of the database objects inside a tablespace set are dependent on any of the database objects outside of that tablespace set.

Some examples of self-contained tablespace violations are:

- An index inside the set of tablespaces is for a table outside of the set of tablespaces.

  Note:

  It is not a violation if a corresponding index for a table is outside of the set of tablespaces.

- A partitioned table is partially contained in the set of tablespaces.

  The tablespace set that you want to copy must contain either all partitions of a partitioned table, or none of the partitions of a partitioned table. To transport a subset of a partition table, you must exchange the partitions into tables.

  See *Oracle Database VLDB and Partitioning Guide* for information about exchanging partitions.

- A referential integrity constraint points to a table across a set boundary.

  When transporting a set of tablespaces, you can choose to include referential integrity constraints. However, doing so can affect whether a set of tablespaces is self-contained. If you decide not to transport constraints, then the constraints are not considered as pointers.

- A table inside the set of tablespaces contains a `LOB` column that points to `LOB`s outside the set of tablespaces.
- An XML DB schema (*.xsd) that was registered by user A imports a global schema that was registered by user B, and the following is true: the default tablespace for user A is tablespace A, the default tablespace for user B is tablespace B, and only tablespace A is included in the set of tablespaces.

To determine whether a set of tablespaces is self-contained, run the `TRANSPORT_SET_CHECK` procedure in the Oracle supplied package `DBMS_TTS`. You must have been granted the `EXECUTE_CATALOG_ROLE` role (initially signed to `SYS`) to run this procedure.

When you run the `DBMS_TTS.TRANSPORT_SET_CHECK` procedure, specify the list of tablespaces in the transportable set to be checked for self containment. You can optionally specify if constraints must be included. For strict or full containment, you must additionally set the `TTS_FULL_CHECK` parameter to `TRUE`.

The strict or full containment check is for cases that require capturing not only references going outside the transportable set, but also those coming into the set. Tablespace Point-in-Time Recovery (TSPITR) is one such case where dependent objects must be fully contained or fully outside the transportable set.

For example, it is a violation to perform TSPITR on a tablespace containing a table `t` but not its index `i` because the index and data will be inconsistent after the transport. A full containment check ensures that there are no dependencies going outside or coming into the transportable set. See the example for TSPITR in the *Oracle Database Backup and Recovery User's Guide*.

Note:

The default for transportable tablespaces is to check for self containment rather than full containment.

The following statement can be used to determine whether tablespaces `sales_1` and `sales_2` are self-contained, with referential integrity constraints taken into consideration (indicated by `TRUE`).

```
EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('sales_1,sales_2', TRUE);
```

After running the `DBMS_TTS.TRANSPORT_SET_CHECK` procedure, you can see all the violations by selecting from the `TRANSPORT_SET_VIOLATIONS` view. If the set of tablespaces is self-contained, then this view is empty. The following example illustrates a case where there are two violations: a foreign key constraint, `dept_fk`, across the tablespace set boundary, and a partitioned table, `jim.sales`, that is partially contained in the tablespace set.

```
SELECT * FROM TRANSPORT_SET_VIOLATIONS;

VIOLATIONS
---------------------------------------------------------------------------
Constraint DEPT_FK between table JIM.EMP in tablespace SALES_1 and table
JIM.DEPT in tablespace OTHER
Partitioned table JIM.SALES is partially contained in the transportable set
```

You must resolve these violations before `sales_1` and `sales_2` are transportable. As noted in the next task, one choice for bypassing the integrity constraint violation is to not to export the integrity constraints.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TTS` package
- *Oracle Database Backup and Recovery User's Guide* for information specific to using the `DBMS_TTS` package for TSPITR

13.3.3.2 Task 2: Generate a Transportable Tablespace Set

After ensuring that you have a self-contained set of tablespaces that you want to transport, generate a transportable tablespace set.

To generate a transportable tablespace set:

1. Start SQL*Plus and connect to the database as an administrator or as a user who has either the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.
2. Make all tablespaces in the set read-only.

```
ALTER TABLESPACE sales_1 READ ONLY;

ALTER TABLESPACE sales_2 READ ONLY;
```

3. Run the Data Pump export utility as a user with `DATAPUMP_EXP_FULL_DATABASE` role and specify the tablespaces in the transportable set.

   ```
   SQL> HOST

   $ expdp user_name dumpfile=expdat.dmp directory=data_pump_dir
           transport_tablespaces=sales_1,sales_2 logfile=tts_export.log

   Password: password
   ```

   You must always specify `TRANSPORT_TABLESPACES`, which specifies that the transportable option is used. This example specifies the following additional Data Pump parameters:

   - The `DUMPFILE` parameter specifies the name of the structural information export dump file to be created, `expdat.dmp`.
   - The `DIRECTORY` parameter specifies the directory object that points to the operating system or Oracle Automatic Storage Management location of the dump file. You must create the `DIRECTORY` object before invoking Data Pump, and you must grant the `READ` and `WRITE` object privileges on the directory to the user running the Export utility.

     However, the directory object `DATA_PUMP_DIR` is not created automatically in a PDB. Therefore, when importing into a PDB, create a directory object in the PDB and specify the directory object when you run Data Pump.
   - The `LOGFILE` parameter specifies the log file to create for the export utility. In this example, the log file is created in the same directory as the dump file, but you can specify any other directory for storing the log file.
   - Triggers and indexes are included in the export operation by default.

   To perform a transport tablespace operation with a strict containment check, use the `TRANSPORT_FULL_CHECK` parameter, as shown in the following example:

   ```
   expdp use_name dumpfile=expdat.dmp directory=data_pump_dir
         transport_tablespaces=sales_1,sales_2 transport_full_check=y
         logfile=tts_export.log
   ```

   In this case, the Data Pump export utility verifies that there are no dependencies between the objects inside the transportable set and objects outside the transportable set. If the tablespace set being transported is not self-contained, then the export fails and indicates that the transportable set is not self-contained. You must resolve these violations and then run this task again.

   Note:

   In this example, the Data Pump utility is used to export only data dictionary structural information (metadata) for the tablespaces. No actual data is unloaded, so this operation goes relatively quickly even for large tablespace sets.

4. The `expdp` utility displays the names and paths of the dump file and the data files on the command line as shown in the following example. These are the files that you need to transport to the target database. Also, check the log file for any errors.

   ```
   ******************************************************************************
   Dump file set for SYSTEM.SYS_EXPORT_TRANSPORTABLE_01 is:
   ```

```
  /u01/app/oracle/admin/salesdb/dpdump/expdat.dmp
******************************************************************************
Datafiles required for transportable tablespace SALES_1:
  /u01/app/oracle/oradata/salesdb/sales_101.dbf
Datafiles required for transportable tablespace SALES_2:
  /u01/app/oracle/oradata/salesdb/sales_201.dbf
```

5. When the Data Pump export operation is completed, exit the `expdp` utility to return to SQL*Plus:

   ```
   $ EXIT
   ```

   See Also:

   - *Oracle Database SQL Language Reference* for information on the `CREATE DIRECTORY` command
   - *Oracle Database Utilities* for information about the default directory when the `DIRECTORY` parameter is omitted
   - *Oracle Database Utilities* for information about using the Data Pump utility
   - *Oracle Multitenant Administrator's Guide* for more information about PDBs

13.3.3.3 Task 3: Transport the Export Dump File

Transport the dump file to the directory pointed to by the `DATA_PUMP_DIR` directory object, or to any other directory of your choosing. The new location must be accessible to the target database.

At the target database, run the following query to determine the location of `DATA_PUMP_DIR`:

```
SELECT * FROM DBA_DIRECTORIES WHERE DIRECTORY_NAME = 'DATA_PUMP_DIR';

OWNER      DIRECTORY_NAME   DIRECTORY_PATH
---------- ---------------- -----------------------------------
SYS        DATA_PUMP_DIR    C:\app\orauser\admin\orawin\dpdump\
```

13.3.3.4 Task 4: Transport the Tablespace Set

Transport the data files of the tablespaces to a directory that is accessible to the target database.

In this example, transfer the following files from the source database to the target database:

- `sales_101.dbf`
- `sales_201.dbf`

If you are transporting the tablespace set to a platform different from the source platform, then determine if cross-platform tablespace transport is supported for both the source and target platforms, and determine the endianness of each platform. If both platforms have the same endianness, then no conversion is necessary. Otherwise, you must do the data conversion either at the source database or at the target database.

If you are transporting `sales_1` and `sales_2` to a different platform, then you can run the following query on each platform. If the query returns a row, the platform supports cross-platform tablespace transport.

```
SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
     FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
     WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
```

The following is the query result from the source platform:

```
PLATFORM_NAME                      ENDIAN_FORMAT
---------------------------------- --------------
Solaris[tm] OE (32-bit)            Big
```

The following is the result from the target platform:

```
PLATFORM_NAME                      ENDIAN_FORMAT
---------------------------------- --------------
Microsoft Windows IA (32-bit)      Little
```

In this example, you can see that the endian formats are different. Therefore, in this case, a conversion is necessary for transporting the database. Use either the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically. Transport the data files to the location of the existing data files of the target database. On the UNIX and Linux platforms, this location is typically /u01/app/oracle/oradata/*dbname*/ or +*DISKGROUP*/*dbname*/ datafile/. Alternatively, you can use to convert the data files.

Note:

- If you use the RMAN `CONVERT` command, then conversion of data files between different endian formats is not supported for data files having undo segments.
- If no endianness conversion of the tablespaces is needed, then you can transfer the files using any file transfer method.

Related Topics

- Converting Data Between Platforms
  When you perform a transportable operation, and the source platform and the target platform are of different endianness, you must convert the data being transported to the target platform format. If the source platform and the target platform are of the same endianness, then data conversion is not necessary. You can use the `DBMS_FILE_TRANSFER` package or the RMAN `CONVERT` command to convert data.
- Guidelines for Transferring Data Files
  You should follow a set of guidelines when transferring the data files.

13.3.3.5 Task 5: (Optional) Restore Tablespaces to Read/Write Mode

Make the transported tablespaces read/write again at the source database.

The following statements make the `sales_1` and `sales_2` tablespaces read/write:

```
ALTER TABLESPACE sales_1 READ WRITE;
ALTER TABLESPACE sales_2 READ WRITE;
```

You can postpone this task to first ensure that the import process succeeds.

13.3.3.6 Task 6: Import the Tablespace Set

To complete the transportable tablespaces operation, import the tablespace set.

To import the tablespace set:

1. Run the Data Pump import utility as a user with `DATAPUMP_IMP_FULL_DATABASE` role and import the tablespace metadata.

   ```
   impdp user_name dumpfile=expdat.dmp directory=data_pump_dir
      transport_datafiles=
      'c:\app\orauser\oradata\orawin\sales_101.dbf',
      'c:\app\orauser\oradata\orawin\sales_201.dbf'
      remap_schema=sales1:crm1  remap_schema=sales2:crm2
      logfile=tts_import.log

   Password: password
   ```

   This example specifies the following Data Pump parameters:

   - The `DUMPFILE` parameter specifies the exported file containing the metadata for the tablespaces to be imported.
   - The `DIRECTORY` parameter specifies the directory object that identifies the location of the export dump file. You must create the `DIRECTORY` object before running Data Pump, and you must grant the `READ` and `WRITE` object privileges on the directory to the user running the Import utility. See *Oracle Database SQL Language Reference* for information on the `CREATE DIRECTORY` command.

     However, the database does not create the directory object `DATA_PUMP_DIR` automatically in a PDB. Therefore, when importing into a PDB, create a directory object in the PDB and specify the directory object when you run Data Pump.

     See Also:

     – *Oracle Database Utilities* for information about the default directory when the `DIRECTORY` parameter is omitted

     – *Oracle Multitenant Administrator's Guide* for more information about PDBs

   - The `TRANSPORT_DATAFILES` parameter identifies all of the data files containing the tablespaces to be imported.

     You can specify the `TRANSPORT_DATAFILES` parameter multiple times in a parameter file specified with the `PARFILE` parameter if there are many data files.

   - The `REMAP_SCHEMA` parameter changes the ownership of database objects. If you do not specify `REMAP_SCHEMA`, then all database objects (such as tables and indexes) are created in the same user schema as in the source database, and those users must already exist in the target database. If they do not exist, then the import utility returns an error. In this example, objects in the tablespace set owned by `sales1` in the source database will be owned by `crm1` in the target database after the tablespace set is imported. Similarly, objects owned by `sales2` in the source database will be owned by `crm2` in the target database. In this case, the target database is not required to have users `sales1` and `sales2`, but must have users `crm1` and `crm2`.

Starting with Oracle Database 12*c* Release 2 (12.2), the Recovery Manager (RMAN) `RECOVER` command can move tables to a different schema while remapping a table. See *Oracle Database Backup and Recovery User’s Guide* for more information.

- The `LOGFILE` parameter specifies the file name of the log file to be written by the import utility. In this example, the log file is written to the directory from which the dump file is read, but it can be written to a different location.

After this statement runs successfully, all tablespaces in the set being copied remain in read-only mode. Check the import log file to ensure that no error has occurred.

When dealing with a large number of data files, specifying the list of data file names in the statement line can be a laborious process as the data file list can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can run the Data Pump import utility as follows:

```
impdp user_name parfile='par.f'
```

The `par.f` parameter file contains the following:

```
DUMPFILE=expdat.dmp
DIRECTORY=data_pump_dir
TRANSPORT_DATAFILES=
'C:\app\orauser\oradata\orawin\sales_101.dbf',
'C:\app\orauser\oradata\orawin\sales_201.dbf'
REMAP_SCHEMA=sales1:crm1  REMAP_SCHEMA=sales2:crm2
LOGFILE=tts_import.log
```

See Also:

Oracle Database Utilities for information about using the import utility

2. If required, put the tablespaces into read/write mode on the target database.

13.4 Transporting Tables, Partitions, or Subpartitions Between Databases

You can transport tables, partitions, and subpartitions between databases.

- Introduction to Transportable Tables
  You can use the transportable tables feature to copy a set of tables, partitions, or subpartitions from one Oracle Database to another. A transportable tables operation moves metadata for the specified tables, partitions, or subpartitions to the target database.
- Limitations on Transportable Tables
  There are limitations on transportable tables.
- Transporting Tables, Partitions, or Subpartitions Using an Export Dump File
  You can transport tables, partitions, or subpartitions between databases using an export file.
- Transporting Tables, Partitions, or Subpartitions Over the Network
  To transport tables over the network, you perform an import using the `NETWORK_LINK` parameter, the import is performed using a database link, and there is no dump file involved.

13.4.1 Introduction to Transportable Tables

You can use the transportable tables feature to copy a set of tables, partitions, or subpartitions from one Oracle Database to another. A transportable tables operation moves metadata for the specified tables, partitions, or subpartitions to the target database.

A transportable tables operation automatically identifies the tablespaces used by the specified tables. To move the data, you copy the data files for these tablespaces to the target database. The Data Pump import automatically frees the blocks in the data files occupied by tables, partitions, or subpartitions that were not part of the transportable tables operation. It also frees the blocks occupied by the dependent objects of the tables that were not part of the transportable tables operation.

You can transport the tables, partitions, and subpartitions in the following ways:

- Using an export dump file

  During the export, specify the `TABLES` parameter and set the `TRANSPORTABLE` parameter to `ALWAYS`. During import, do not specify the `TRANSPORTABLE` parameter. Data Pump import recognizes the transportable tables operation automatically.
- Over the network

  During the import, specify the `TABLES` parameter, set the `TRANSPORTABLE` parameter to `ALWAYS`, and specify the `NETWORK_LINK` parameter to identify the source database.

13.4.2 Limitations on Transportable Tables

There are limitations on transportable tables.

Be aware of the following limitations for transportable tables:

- The general limitations described in "General Limitations on Transporting Data" apply to transportable tables.
- You cannot transport a table to a target database that contains a table of the same name in the same schema. However, you can use the `REMAP_TABLE` import parameter to import the data into a different table. Alternatively, before the transport operation, you can rename either the table to be transported or the target table.

  Starting with Oracle Database 12*c* Release 2 (12.2), the Recovery Manager (RMAN) `RECOVER` command can move tables to a different schema while remapping a table. See *Oracle Database Backup and Recovery User's Guide* for more information.
- You cannot transport tables with `TIMESTAMP WITH TIMEZONE` (TSTZ) data across platforms with different time zone file versions.

  See *Oracle Database Utilities* for more information.

13.4.3 Transporting Tables, Partitions, or Subpartitions Using an Export Dump File

You can transport tables, partitions, or subpartitions between databases using an export file.

The following list of tasks summarizes the process of transporting tables between databases using an export dump file. Details for each task are provided in the subsequent example.

1. Pick a set of tables, partitions, or subpartitions.

If you are transporting partitions, then you can specify partitions from only one table in a transportable tables operation, and no other tables can be transported in the same operation. Also, if only a subset of a table's partitions are exported in a transportable tables operation, then on import each partition becomes a non-partitioned table.

2. At the source database, place the tablespaces associated with the data files for the tables, partitions, or subpartitions in read-only mode.

   To view the tablespace for a table, query the `DBA_TABLES` view. To view the data file for a tablespace, query the `DBA_DATA_FILES` view.

3. Perform the Data Pump export.

4. Transport the export dump file.

   Copy the export dump file to a place that is accessible to the target database.

5. Transport the data files for the tables, partitions, or subpartitions.

   Copy the data files to a place that is accessible to the target database.

   If the source platform and target platform are different, then check the endian format of each platform by running the query on the `V$TRANSPORTABLE_PLATFORM` view in "Transporting Data Across Platforms".

   If the source platform's endian format is different from the target platform's endian format, then use one of the following methods to convert the data files:

   - Use the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically.

   - Use the RMAN `CONVERT` command to convert the data files to the target platform's endian format.

   Note:

   Conversion of data files between different endian formats is not supported for data files having undo segments.

   See "Converting Data Between Platforms" for more information.

6. (Optional) Restore tablespaces to read/write mode on the source database.

7. At the target database, perform the import.

   Invoke the Data Pump utility to import the metadata for the tables.

Example

These tasks for transporting tables, partitions, and subpartitions using a Data Pump dump file are illustrated more fully in the example that follows, where it is assumed that the following partitions exist in the `sh.sales_prt` table:

- `sales_q1_2000`
- `sales_q2_2000`
- `sales_q3_2000`
- `sales_q4_2000`

This example transports two of these partitions to the target database.

The following SQL statements create the `sales_prt` table and its and partitions in the `sh` schema and the tablespace and data file for the table. The statements also insert data into the partitions by using data in the `sh` sample schemas.

```
CREATE TABLESPACE sales_prt_tbs
   DATAFILE 'sales_prt.dbf' SIZE 20M
   ONLINE;

CREATE TABLE sh.sales_prt
    (prod_id        NUMBER(6),
     cust_id        NUMBER,
     time_id        DATE,
     channel_id     CHAR(1),
     promo_id       NUMBER(6),
     quantity_sold  NUMBER(3),
     amount_sold    NUMBER(10,2))
        PARTITION BY RANGE (time_id)
         (PARTITION SALES_Q1_2000 VALUES LESS THAN
                  (TO_DATE('01-APR-2000','DD-MON-YYYY','NLS_DATE_LANGUAGE = American')),
          PARTITION SALES_Q2_2000 VALUES LESS THAN
                  (TO_DATE('01-JUL-2000','DD-MON-YYYY','NLS_DATE_LANGUAGE = American')),
          PARTITION SALES_Q3_2000 VALUES LESS THAN
                  (TO_DATE('01-OCT-2000','DD-MON-YYYY','NLS_DATE_LANGUAGE = American')),
          PARTITION SALES_Q4_2000 VALUES LESS THAN
                  (TO_DATE('01-JAN-2001','DD-MON-YYYY','NLS_DATE_LANGUAGE = American')))
 TABLESPACE sales_prt_tbs;

INSERT INTO sh.sales_prt PARTITION(sales_q1_2000)
  SELECT * FROM sh.sales PARTITION(sales_q1_2000);

INSERT INTO sh.sales_prt PARTITION(sales_q2_2000)
  SELECT * FROM sh.sales PARTITION(sales_q2_2000);

INSERT INTO sh.sales_prt PARTITION(sales_q3_2000)
  SELECT * FROM sh.sales PARTITION(sales_q3_2000);

INSERT INTO sh.sales_prt PARTITION(sales_q4_2000)
  SELECT * FROM sh.sales PARTITION(sales_q4_2000);

COMMIT;
```

This example makes the following additional assumptions:

- The name of the source database is `sourcedb`.
- The source database and target database are running on the same platform with the same endianness. To check the endianness of a platform, run the following query:

  ```
  SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
       FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
       WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
  ```

- Only the `sales_q1_2000` and `sales_q2_2000` partitions are transported to the target database. The other two partitions are not transported.

Complete the following tasks to transport the partitions using an export dump file:

Task 1 Generate the Export Dump File
Generate the export dump file by completing the following steps:

1. Start SQL*Plus and connect to the source database as an administrator or as a user who has either the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

   See "Connecting to the Database with SQL*Plus" for instructions.

2. Make all of the tablespaces that contain the tables being transported read-only.

   ```
   ALTER TABLESPACE sales_prt_tbs READ ONLY;
   ```

3. Invoke the Data Pump export utility as a user with `DATAPUMP_EXP_FULL_DATABASE` role and specify the transportable tables options.

   ```
   SQL> HOST

   expdp user_name dumpfile=sales_prt.dmp  directory=data_pump_dir
         tables=sh.sales_prt:sales_q1_2000,sh.sales_prt:sales_q2_2000
         transportable=always logfile=exp.log

   Password: password
   ```

   You must always specify `TRANSPORTABLE=ALWAYS`, which specifies that the transportable option is used.

   This example specifies the following additional Data Pump parameters:

   - The `DUMPFILE` parameter specifies the name of the structural information export dump file to be created, `sales_prt.dmp`.
   - The `DIRECTORY` parameter specifies the directory object that points to the operating system or Oracle Automatic Storage Management location of the dump file. You must create the `DIRECTORY` object before invoking Data Pump, and you must grant the `READ` and `WRITE` object privileges on the directory to the user running the Export utility. See *Oracle Database SQL Language Reference* for information on the `CREATE DIRECTORY` command.

     However, the directory object `DATA_PUMP_DIR` is not created automatically in a PDB. Therefore, when importing into a PDB, create a directory object in the PDB and specify the directory object when you run Data Pump.

     See Also:

     – *Oracle Database Utilities* for information about the default directory when the `DIRECTORY` parameter is omitted

     – *Oracle Multitenant Administrator's Guide* for more information about PDBs

   - The `TABLES` parameter specifies the tables, partitions, or subpartitions being exported.
   - The `LOGFILE` parameter specifies the file name of the log file to be written by the export utility. In this example, the log file is written to the same directory as the dump file, but it can be written to a different location.

4. Check the log file for unexpected errors, and take note of the dump file and data files that you must transport to the target database. `expdp` outputs the names and paths of these files in messages like these:

```
Processing object type TABLE_EXPORT/TABLE/PLUGTS_BLK
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/END_PLUGTS_BLK
Master table "SYSTEM"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded
******************************************************************************
Dump file set for SYSTEM.SYS_EXPORT_TABLE_01 is:
  /u01/app/oracle/rdbms/log/sales_prt.dmp
******************************************************************************
Datafiles required for transportable tablespace SALES_PRT_TBS:
  /u01/app/oracle/oradata/sourcedb/sales_prt.dbf
Job "SYSTEM"."SYS_EXPORT_TABLE_01" successfully completed at 11:32:13
```

5. When finished, exit back to SQL*Plus:

```
$ exit
```

See Also:

Oracle Database Utilities for information about using the Data Pump utility

Task 2 Transport the Export Dump File
Transport the dump file to the directory pointed to by the `DATA_PUMP_DIR` directory object on the target database, or to any other directory of your choosing. The new location must be accessible to the target database.
In this example, transfer the `sales_prt.dmp` dump file from the source database to the target database.
At the target database, run the following query to determine the location of `DATA_PUMP_DIR`:

```
SELECT * FROM DBA_DIRECTORIES WHERE DIRECTORY_NAME = 'DATA_PUMP_DIR';

OWNER      DIRECTORY_NAME   DIRECTORY_PATH
---------- ---------------- -----------------------------------
SYS        DATA_PUMP_DIR    /u01/app/oracle/rdbms/log/
```

Task 3 Transport the Data Files for the Tables
Transport the data files of the tablespaces containing the tables being transported to a place that is accessible to the target database.
Typically, you transport the data files to the location of the existing data files of the target database. On the UNIX and Linux platforms, this location is typically /u01/app/oracle/oradata/*dbname*/ or +*DISKGROUP*/*dbname*/datafile/.
In this example, transfer the `sales_prt.dbf` data file from the source database to the target database.

See Also:

"Guidelines for Transferring Data Files"

Task 4 (Optional) Restore Tablespaces to Read/Write Mode
Make the tablespaces that contain the tables being transported read/write again at the source database, as follows:

```
ALTER TABLESPACE sales_prt_tbs READ WRITE;
```

You can postpone this task to first ensure that the import process succeeds.

Task 5 At the Target Database, Import the Partitions

At the target database, invoke the Data Pump import utility as a user with `DATAPUMP_IMP_FULL_DATABASE` role and specify the transportable tables options.

```
impdp user_name dumpfile=sales_prt.dmp directory=data_pump_dir
   transport_datafiles='/u01/app/oracle/oradata/targetdb/sales_prt.dbf'
   tables=sh.sales_prt:sales_q1_2000,sh.sales_prt:sales_q2_2000
   logfile=imp.log

Password: password
```

This example specifies the following Data Pump parameters:

- The `DUMPFILE` parameter specifies the exported file containing the metadata for the data to be imported.
- The `DIRECTORY` parameter specifies the directory object that identifies the location of the export dump file. You must create the `DIRECTORY` object before invoking Data Pump, and you must grant the `READ` and `WRITE` object privileges on the directory to the user running the Import utility. See *Oracle Database SQL Language Reference* for information on the `CREATE DIRECTORY` command.

  However, the directory object `DATA_PUMP_DIR` is not created automatically in a PDB. Therefore, when importing into a PDB, create a directory object in the PDB and specify the directory object when you run Data Pump.

  See Also:

  - *Oracle Database Utilities* for information about the default directory when the `DIRECTORY` parameter is omitted
  - *Oracle Multitenant Administrator's Guide* for more information about PDBs

- The `TRANSPORT_DATAFILES` parameter identifies all of the data files to be imported.

  You can specify the `TRANSPORT_DATAFILES` parameter multiple times in a parameter file specified with the `PARFILE` parameter if there are many data files.
- The `TABLES` parameter specifies the tables, partitions, or subpartitions being imported.
- The `LOGFILE` parameter specifies the file name of the log file to be written by the import utility. In this example, the log file is written to the directory from which the dump file is read, but it can be written to a different location.

After this statement executes successfully, check the import log file to ensure that no unexpected error has occurred.

When dealing with a large number of data files, specifying the list of data file names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can invoke the Data Pump import utility as follows:

```
impdp user_name parfile='par.f'
```

For example, `par.f` might contain the following lines:

```
DUMPFILE=sales_prt.dmp
DIRECTORY=data_pump_dir
TRANSPORT_DATAFILES='/u01/app/oracle/oradata/targetdb/sales_prt.dbf'
```

```
TABLES=sh.sales_prt:sales_q1_2000,sh.sales_prt:sales_q2_2000
LOGFILE=imp.log
```

Note:

- The partitions are imported as separate tables in the target database because this example transports a subset of partitions.
- During the import, tablespaces might be temporarily made read/write for metadata loading. Ensure that no user changes are made to the data during the import. At the successful completion of the import, all user-defined tablespaces are made read/write.
- When performing a network database import, the `TRANSPORTABLE` parameter must be set to `always`.

See Also:

Oracle Database Utilities for information about using the import utility

13.4.4 Transporting Tables, Partitions, or Subpartitions Over the Network

To transport tables over the network, you perform an import using the `NETWORK_LINK` parameter, the import is performed using a database link, and there is no dump file involved.

The following list of tasks summarizes the process of transporting tables, partitions, and subpartitions between databases over the network. Details for each task are provided in the subsequent example.

1. Pick a set of tables, partitions, or subpartitions.

   If you are transporting partitions, then you can specify partitions from only one table in a transportable tables operation, and no other tables can be transported in the same operation. Also, if only a subset of a table's partitions are exported in a transportable tables operation, then on import each partition becomes a non-partitioned table.
2. At the source database, place the tablespaces associated with the data files for the tables, partitions, or subpartitions in read-only mode.

   To view the tablespace for a table, query the `DBA_TABLES` view. To view the data file for a tablespace, query the `DBA_DATA_FILES` view.
3. Transport the data files for the tables, partitions, or subpartitions.

   Copy the data files to a place that is accessible to the target database.

   If the source platform and target platform are different, then check the endian format of each platform by running the query on the `V$TRANSPORTABLE_PLATFORM` view in "Transporting Data Across Platforms".

   If the source platform's endian format is different from the target platform's endian format, then use one of the following methods to convert the data files:

   - Use the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data files. These procedures convert the data files to the target platform's endian format automatically.

- Use the RMAN `CONVERT` command to convert the data files to the target platform's endian format.

  Note:

  Conversion of data files between different endian formats is not supported for data files having undo segments.

  See "Converting Data Between Platforms" for more information.

4. At the target database, perform the import.

   Invoke the Data Pump utility to import the metadata for the tables.

5. (Optional) Restore tablespaces to read/write mode on the source database.

Example

These tasks for transporting tables over the network are illustrated more fully in the example that follows, where it is assumed that the tables exist in the source database:

Table	Tablespace	Data File
`hr.emp_ttbs`	`emp_tsp`	/u01/app/oracle/oradata/sourcedb/emp.dbf
`oe.orders_ttbs`	`orders_tsp`	/u01/app/oracle/oradata/sourcedb/orders.dbf

This example transports these tables to the target database. To complete the example, these tables must exist on the source database.

The following SQL statements create the tables in the `hr` schema and the tablespaces and data files for the tables. The statements also insert data into the tables by using data in the `hr` and `oe` sample schemas.

```
CREATE TABLESPACE emp_tsp
   DATAFILE 'emp.dbf' SIZE 1M
   ONLINE;

CREATE TABLE hr.emp_ttbs(
   employee_id    NUMBER(6),
   first_name     VARCHAR2(20),
   last_name      VARCHAR2(25),
   email          VARCHAR2(25),
   phone_number   VARCHAR2(20),
   hire_date      DATE,
   job_id         VARCHAR2(10),
   salary         NUMBER(8,2),
   commission_pct NUMBER(2,2),
   manager_id     NUMBER(6),
   department_id  NUMBER(4))
 TABLESPACE emp_tsp;

INSERT INTO hr.emp_ttbs SELECT * FROM hr.employees;

CREATE TABLESPACE orders_tsp
   DATAFILE 'orders.dbf' SIZE 1M
   ONLINE;

CREATE TABLE oe.orders_ttbs(
   order_id      NUMBER(12),
   order_date    TIMESTAMP WITH LOCAL TIME ZONE,
   order_mode    VARCHAR2(8),
   customer_id   NUMBER(6),
   order_status  NUMBER(2),
   order_total   NUMBER(8,2),
   sales_rep_id  NUMBER(6),
   promotion_id  NUMBER(6))
 TABLESPACE orders_tsp;

INSERT INTO oe.orders_ttbs SELECT * FROM oe.orders;

COMMIT;
```

This example makes the following additional assumptions:

- The name of the source database is `sourcedb`.
- The source database and target database are running on the same platform with the same endianness. To check the endianness of a platform, run the following query:

  ```
  SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
       FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
       WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
  ```

Complete the following tasks to transport the tables over the network:

Task 1 Create a Database Link from the Target Database to the Source Database
Create a database link from the target database to the source database by completing the following steps:

1. Ensure that network connectivity is configured between the source database and the target database.

   See *Oracle Database Net Services Administrator's Guide* for instructions.

2. Start SQL*Plus and connect to the target database as the administrator who will transport the data with Data Pump import. This user must have `DATAPUMP_IMP_FULL_DATABASE` role to transport the data.

   See "Connecting to the Database with SQL*Plus" for instructions.

3. Create the database link:

   ```
   CREATE PUBLIC DATABASE LINK sourcedb USING 'sourcedb';
   ```

   Specify the service name for the source database in the using clause.

   During the import operation, the database link must connect to a user on the source database with `DATAPUMP_EXP_FULL_DATABASE` role. The user on the source database cannot be a user with `SYSDBA` administrative privilege.

See Also:

- "Creating Database Links"
- *Oracle Database SQL Language Reference*

Task 2 Make the Tablespaces Containing the Tables Read-Only
At the source database, complete the following steps:

1. Start SQL*Plus and connect to the source database as an administrator or as a user who has either the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

   See "Connecting to the Database with SQL*Plus" for instructions.

2. Make all of the tablespaces that contain data to be transported read-only.

   ```
   ALTER TABLESPACE emp_tsp READ ONLY;
   ALTER TABLESPACE orders_tsp READ ONLY;
   ```

Task 3 Transport the Data Files for the Tables
Transport the data files of the tablespaces containing the tables being transported to a place that is accessible to the target database.
Typically, you transport the data files to the location of the existing data files of the target database. On the UNIX and Linux platforms, this location is typically /u01/app/oracle/oradata/*dbname*/ or +*DISKGROUP*/*dbname*/datafile/.
In this example, transfer the `emp.dbf` and `orders.dbf` data files from the source database to the target database.

See Also:

"Guidelines for Transferring Data Files"

Task 4 At the Target Database, Import the Database
Invoke the Data Pump import utility as a user with `DATAPUMP_IMP_FULL_DATABASE` role and specify the full transportable export/import options.

```
impdp user_name network_link=sourcedb transportable=always
   transport_datafiles=
      '/u01/app/oracle/oradata/targetdb/emp.dbf'
      '/u01/app/oracle/oradata/targetdb/orders.dbf'
   tables=hr.emp_ttbs,oe.orders_ttbs
   logfile=import.log

Password: password
```

This example specifies the following Data Pump parameters:

- The `NETWORK_LINK` parameter specifies the database link to the source database used for the network import.
- The `TRANSPORTABLE` parameter specifies that the import uses the transportable option.
- The `TRANSPORT_DATAFILES` parameter identifies all of the data files to be imported.

  You can specify the `TRANSPORT_DATAFILES` parameter multiple times in a parameter file specified with the `PARFILE` parameter if there are many data files.
- The `TABLES` parameter specifies the tables to be imported.
- The `LOGFILE` parameter specifies the file name of the log file to be written by the import utility.

After this statement executes successfully, check the import log file to ensure that no unexpected error has occurred.
When dealing with a large number of data files, specifying the list of data file names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can invoke the Data Pump import utility as follows:

```
impdp user_name parfile='par.f'
```

For example, `par.f` might contain the following lines:

```
NETWORK_LINK=sourcedb
TRANSPORTABLE=always
TRANSPORT_DATAFILES=
      '/u01/app/oracle/oradata/targetdb/emp.dbf'
      '/u01/app/oracle/oradata/targetdb/orders.dbf'
TABLES=hr.emp_ttbs,oe.orders_ttbs
LOGFILE=import.log
```

Note:

During the import, user-defined tablespaces might be temporarily made read/write for metadata loading. Ensure that no user changes are made to the data during the import. At the successful completion of the import, all user-defined tablespaces are made read/write.

See Also:

Oracle Database Utilities for information about using the import utility

Task 5 (Optional) Restore Tablespaces to Read/Write Mode
Make the tables that contain the tables being transported read/write again at the source database, as follows:

```
ALTER TABLESPACE emp_tsp READ WRITE;
ALTER TABLESPACE orders_tsp READ WRITE;
```

13.5 Converting Data Between Platforms

When you perform a transportable operation, and the source platform and the target platform are of different endianness, you must convert the data being transported to the target platform format. If the source platform and the target platform are of the same endianness, then data conversion is not necessary. You can use the `DBMS_FILE_TRANSFER` package or the RMAN `CONVERT` command to convert data.

Note:

Some limitations might apply that are not described in these sections. Refer to the following documentation for more information:

- "Transporting Data Across Platforms" for information about checking the endianness of platforms
- *Oracle Database PL/SQL Packages and Types Reference* for information about limitations related to the `DBMS_FILE_TRANSFER` package
- *Oracle Database Backup and Recovery Reference* for information about limitations related to the RMAN `CONVERT` command

- Converting Data Between Platforms Using the DBMS_FILE_TRANSFER Package
  You can use the `GET_FILE` or `PUT_FILE` procedure of the `DBMS_FILE_TRANSFER` package to convert data between platforms during a data file transfer.
- Converting Data Between Platforms Using RMAN
  When you use the RMAN `CONVERT` command to convert data, you can either convert the data on the source platform after running Data Pump export, or you can convert the data on the target platform before running Data Pump import. In either case, you must transfer the data files from the source system to the target system.

13.5.1 Converting Data Between Platforms Using the DBMS_FILE_TRANSFER Package

You can use the `GET_FILE` or `PUT_FILE` procedure of the `DBMS_FILE_TRANSFER` package to convert data between platforms during a data file transfer.

When you use one of these procedures to move data files between the source platform and the target platform, each block in each data file is converted to the target platform's endianness.

This section uses an example to describe how to use the `DBMS_FILE_TRANSFER` package to convert a data file to a different platform. The example makes the following assumptions:

- The `GET_FILE` procedure will transfer the data file.
- The `mytable.342.123456789` data file is being transferred to a different platform.
- The endianness of the source platform is different from the endianness of the target platform.
- The global name of the source database is `dbsa.example.com`.
- Both the source database and the target database use Oracle Automatic Storage Management (Oracle ASM).

> **Note:**
>
> You can also use the `DBMS_FILE_TRANSFER` package to transfer data files between platforms with the same endianness.

Complete the following steps to convert the data file by transferring it with the `GET_FILE` procedure:

1. Use SQL*Plus to connect to the source database as an administrative user who can create directory objects.
2. Create a directory object to store the data files that you want to transfer to the target database.

   For example, to create a directory object named `sales_dir_source` for the `+data/dbsa/datafile` directory, execute the following SQL statement:

   ```
   CREATE OR REPLACE DIRECTORY sales_dir_source
      AS '+data/dbsa/datafile';
   ```

   The specified file system directory must exist when you create the directory object.

3. Use SQL*Plus to connect to the target database as an administrative user who can create database links, create directory objects, and run the procedures in the `DBMS_FILE_TRANSFER` package.
4. Create a database link from the target database to the source database.

   The connected user at the source database must have read privileges on the directory object that you created in Step 2.
5. Create a directory object to store the data files that you want to transfer from the source database.

   The user at the local database who will run the procedure in the `DBMS_FILE_TRANSFER` package must have write privileges on the directory object.

   For example, to create a directory object named `sales_dir_target` for the `+data/dbsb/datafile` directory, run the following SQL statement:

   ```
   CREATE OR REPLACE DIRECTORY sales_dir_target
     AS '+data/dbsb/datafile';
   ```
6. Run the `GET_FILE` procedure in the `DBMS_FILE_TRANSFER` package to transfer the data file.

   For example, run the following procedure to transfer the `mytable.342.123456789` data file from the source database to the target database using the database link you created in Step 4:

   ```
   BEGIN
     DBMS_FILE_TRANSFER.GET_FILE(
       source_directory_object      => 'sales_dir_source',
       source_file_name             => 'mytable.342.123456789',
       source_database              => 'dbsa.example.com',
       destination_directory_object => 'sales_dir_target',
       destination_file_name        => 'mytable');
   END;
   /
   ```

Note:

In this example, the destination data file name is `mytable`. Oracle ASM does not allow a fully qualified file name form in the `destination_file_name` parameter of the `GET_FILE` procedure.

Related Topics

- About Connecting to the Database with SQL*Plus
  Oracle Database includes the following components: the Oracle Database instance, which is a collection of processes and memory, and a set of disk files that contain user data and system data.
- Creating Database Links
  To support application access to the data and schema objects throughout a distributed database system, you must create all necessary database links.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about using the `DBMS_FILE_TRANSFER` package
- *Oracle Automatic Storage Management Administrator's Guide* for information about fully qualified file name forms in ASM
- *Oracle Database SQL Language Reference* for information about how to create a database link

13.5.2 Converting Data Between Platforms Using RMAN

When you use the RMAN `CONVERT` command to convert data, you can either convert the data on the source platform after running Data Pump export, or you can convert the data on the target platform before running Data Pump import. In either case, you must transfer the data files from the source system to the target system.

You can convert data with the following RMAN `CONVERT` commands:

- `CONVERT DATAFILE`
- `CONVERT TABLESPACE`
- `CONVERT DATABASE`

Note:

- Datatype restrictions apply to the RMAN `CONVERT` command.
- RMAN `CONVERT` commands do not support conversion of data files between different endian formats for data files having undo segments.

- Converting Tablespaces on the Source System After Export
An example illustrates how to use the RMAN `CONVERT TABLESPACE` command to convert tablespaces to a different platform.
- Converting Data Files on the Target System Before Import
An example illustrates how to use the RMAN `CONVERT DATAFILE` command to convert data files to a different platform.

See Also:

- *Oracle Database Backup and Recovery Reference*
- *Oracle Database Backup and Recovery User's Guide*

13.5.2.1 Converting Tablespaces on the Source System After Export

An example illustrates how to use the RMAN `CONVERT TABLESPACE` command to convert tablespaces to a different platform.

The example makes the following assumptions:

- The `sales_1` and `sales_2` tablespaces are being transported to a different platform.
- The endianness of the source platform is different from the endianness of the target platform.
- You want to convert the data on the source system, before transporting the tablespace set to the target system.
- You have completed the Data Pump export on the source database.

Complete the following steps to convert the tablespaces on the source system:

1. At a command prompt, start RMAN and connect to the source database:

   ```
   $ RMAN TARGET /

   Recovery Manager: Release 12.1.0.1.0 - Production

   Copyright (c) 1982, 2012, Oracle and/or its affiliates.  All rights reserved.

   connected to target database: salesdb (DBID=3295731590)
   ```

2. Use the RMAN `CONVERT TABLESPACE` command to convert the data files into a temporary location on the source platform.

   In this example, assume that the temporary location, directory `/tmp`, has already been created. The converted data files are assigned names by the system.

   ```
   RMAN> CONVERT TABLESPACE sales_1,sales_2
   2> TO PLATFORM 'Microsoft Windows IA (32-bit)'
   3> FORMAT '/tmp/%U';

   Starting conversion at source at 30-SEP-08
   using channel ORA_DISK_1
   channel ORA_DISK_1: starting datafile conversion
   input datafile file number=00007 name=/u01/app/oracle/oradata/salesdb/sales_101.dbf
   converted datafile=/tmp/data_D-SALESDB_I-1192614013_TS-SALES_1_FNO-7_03jru08s
   channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:45
   channel ORA_DISK_1: starting datafile conversion
   input datafile file number=00008 name=/u01/app/oracle/oradata/salesdb/sales_201.dbf
   converted datafile=/tmp/data_D-SALESDB_I-1192614013_TS-SALES_2_FNO-8_04jru0aa
   channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:25
   Finished conversion at source at 30-SEP-08
   ```

   See Also:

   Oracle Database Backup and Recovery Reference for a description of the RMAN `CONVERT` command

3. Exit Recovery Manager:

   ```
   RMAN> exit
   Recovery Manager complete.
   ```

4. Transfer the data files to the target system.

Related Topics

- Guidelines for Transferring Data Files
  You should follow a set of guidelines when transferring the data files.

13.5.2.2 Converting Data Files on the Target System Before Import

An example illustrates how to use the RMAN `CONVERT DATAFILE` command to convert data files to a different platform.

During the conversion, you identify the data files by file name, not by tablespace name. Until the tablespace metadata is imported, the target instance has no way of knowing the desired tablespace names.

The example makes the following assumptions:

- You have not yet converted the data files for the tablespaces being transported.

  If you used the `DBMS_FILE_TRANSFER` package to transfer the data files to the target system, then the data files were converted automatically during the file transfer. See "Converting Data Between Platforms Using the DBMS_FILE_TRANSFER Package".
- The following data files are being transported to a different platform:
  - C:\Temp\sales_101.dbf
  - C:\Temp\sales_201.dbf
- The endianness of the source platform is different from the endianness of the target platform.
- You want to convert the data on the target system, before performing the Data Pump import.
- The converted data files are placed in C:\app\orauser\oradata\orawin\, which is the location of the existing data files for the target system:

Complete the following steps to convert the tablespaces on the target system:

1. If you are in SQL*Plus, then return to the host system:

   ```
   SQL> HOST
   ```

2. Use the RMAN `CONVERT DATAFILE` command to convert the data files on the target platform:

   ```
   C:\>RMAN TARGET /

   Recovery Manager: Release 12.1.0.1.0 - Production

   Copyright (c) 1982, 2012, Oracle and/or its affiliates.  All rights reserved.

   connected to target database: ORAWIN (DBID=3462152886)

   RMAN> CONVERT DATAFILE
   2>'C:\Temp\sales_101.dbf',
   3>'C:\Temp\sales_201.dbf'
   4>TO PLATFORM="Microsoft Windows IA (32-bit)"
   5>FROM PLATFORM="Solaris[tm] OE (32-bit)"
   6>DB_FILE_NAME_CONVERT=
   7>'C:\Temp\', 'C:\app\orauser\oradata\orawin\'
   8> PARALLELISM=4;
   ```

If the source location, the target location, or both do not use Oracle Automatic Storage Management (Oracle ASM), then the source and target platforms are optional. RMAN determines the source platform by examining the data file, and the target platform defaults to the platform of the host running the conversion.

If both the source and target locations use Oracle ASM, then you must specify the source and target platforms in the `DB_FILE_NAME_CONVERT` clause.

See Also:

Oracle Database Backup and Recovery Reference for a description of the RMAN `CONVERT` command

3. Exit Recovery Manager:

```
RMAN> exit
Recovery Manager complete.
```

13.6 Guidelines for Transferring Data Files

You should follow a set of guidelines when transferring the data files.

If both the source and target are file systems, then you can transport using:

- Any facility for copying flat files (for example, an operating system copy utility or ftp)
- The `DBMS_FILE_TRANSFER` package
- RMAN
- Any facility for publishing on CDs

If either the source or target is an Oracle Automatic Storage Management (Oracle ASM) disk group, then you can use:

- ftp to or from the `/sys/asm` virtual folder in the XML DB repository

  See *Oracle Automatic Storage Management Administrator's Guide* for more information.
- The `DBMS_FILE_TRANSFER` package
- RMAN

Do not transport the data files for the administrative tablespaces (such as `SYSTEM` and `SYSAUX`) or any undo or temporary tablespaces.

If you are transporting data of a different block size than the standard block size of the database receiving the data, then you must first have a `DB_`*`n`*`K_CACHE_SIZE` initialization parameter entry in the receiving database parameter file.

For example, if you are transporting data with an 8K block size into a database with a 4K standard block size, then you must include a `DB_8K_CACHE_SIZE` initialization parameter entry in the parameter file. If it is not already included in the parameter file, then this parameter can be set using the `ALTER SYSTEM SET` statement.

See *Oracle Database Reference* for information about specifying values for the `DB_`*`n`*`K_CACHE_SIZE` initialization parameter.

Starting with Oracle Database 12*c*, the `GET_FILE` or `PUT_FILE` procedure in the `DBMS_FILE_TRANSFER` package can convert data between platforms during the data file transfer. See "Converting Data Between Platforms".

Starting with Oracle Database 12*c*, RMAN can transfer files using network-enabled restore. RMAN restores database files, over the network, from a remote database instance by using the `FROM SERVICE` clause of the `RESTORE` command. The primary advantage of network-enabled restore is that it eliminates the requirement for a restore of the backup to a staging area on disk and the need to transfer the copy. Therefore, network-enabled restore saves disk space and time. This technique can also provide the following advantages during file transfer: compression, encryption, and transfer of used data blocks only. See *Oracle Database Backup and Recovery User's Guide* for more information.

Note:

Exercise caution when using the UNIX `dd` utility to copy raw-device files between databases, and note that Oracle Database 12*c* and later do not support raw devices for database files. The `dd` utility can be used to copy an entire source raw-device file, or it can be invoked with options that instruct it to copy only a specific range of blocks from the source raw-device file.

It is difficult to ascertain actual data file size for a raw-device file because of hidden control information that is stored as part of the data file. If you must use the `dd` utility to operate on raw devices, then specify the entire source raw-device file contents. If you move database file content from a raw device to either ASM or a file system to adhere to the desupport of raw devices with Oracle Database 12*c* and later, then use an Oracle-provided tool such as RMAN.

See Also:

"Copying Files Using the Database Server" for information about using the `DBMS_FILE_TRANSFER` package to copy the files that are being transported and their metadata

14

Managing Undo

For a default installation, Oracle Database automatically manages undo. There is typically no need for DBA intervention. However, if your installation uses Oracle Flashback operations, you may need to perform some undo management tasks to ensure the success of these operations.

- What Is Undo?
  Oracle Database creates and manages information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. These records are collectively referred to as **undo**.
- Introduction to Automatic Undo Management
  Oracle Database can manage undo information and space automatically.
- Setting the Minimum Undo Retention Period
  You specify the minimum undo retention period (in seconds) by setting the `UNDO_RETENTION` initialization parameter.
- Sizing a Fixed-Size Undo Tablespace
  Automatic tuning of undo retention typically achieves better results with a fixed-size undo tablespace. If you decide to use a fixed-size undo tablespace, then the Undo Advisor can help you estimate needed capacity.
- Managing Undo Tablespaces
  You manage undo tablespaces by completing tasks such as creating, altering, and dropping them. You can also switch undo tablespaces and establish user quotas for undo space.
- Migrating to Automatic Undo Management
  If you are currently using rollback segments to manage undo space, Oracle strongly recommends that you migrate your database to automatic undo management.
- Managing Temporary Undo
  By default, undo records for temporary tables are stored in the undo tablespace and are logged in the redo, which is the same way undo is managed for persistent tables. However, you can use the `TEMP_UNDO_ENABLED` initialization parameter to separate undo for temporary tables from undo for persistent tables. When this parameter is set to `TRUE`, the undo for temporary tables is called **temporary undo**.
- Undo Space Data Dictionary Views
  You can query a set of views for information about undo space in the automatic undo management mode.

See Also:

Using Oracle Managed Files for information about creating an undo tablespace whose data files are both created and managed by Oracle Database.

14.1 What Is Undo?

Oracle Database creates and manages information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. These records are collectively referred to as **undo**.

Undo records are used to:

- Roll back transactions when a `ROLLBACK` statement is issued
- Recover the database
- Provide read consistency
- Analyze data as of an earlier point in time by using Oracle Flashback Query
- Recover from logical corruptions using Oracle Flashback features

When a `ROLLBACK` statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the data files. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

See Also:

Oracle Database Concepts

14.2 Introduction to Automatic Undo Management

Oracle Database can manage undo information and space automatically.

- Overview of Automatic Undo Management
  Oracle provides a fully automated mechanism, referred to as automatic undo management, for managing undo information and space. With automatic undo management, the database manages undo segments in an undo tablespace.
- The Undo Retention Period
  The **undo retention period** is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it.

14.2.1 Overview of Automatic Undo Management

Oracle provides a fully automated mechanism, referred to as automatic undo management, for managing undo information and space. With automatic undo management, the database manages undo segments in an undo tablespace.

Automatic undo management is the default mode for a newly installed database. An auto-extending undo tablespace named `UNDOTBS1` is automatically created when you create the database with Database Configuration Assistant (DBCA).

You can also create an undo tablespace explicitly. The methods of creating an undo tablespace are explained in "Creating an Undo Tablespace".

When the database instance starts, the database automatically selects the first available undo tablespace. If no undo tablespace is available, then the instance starts without an undo tablespace and stores undo records in the `SYSTEM` tablespace. This is not recommended, and an alert message is written to the alert log file to warn that the system is running without an undo tablespace.

If the database contains multiple undo tablespaces, then you can optionally specify at startup that you want to use a specific undo tablespace. This is done by setting the `UNDO_TABLESPACE` initialization parameter, as shown in this example:

```
UNDO_TABLESPACE = undotbs_01
```

If the tablespace specified in the initialization parameter does not exist, the `STARTUP` command fails. The `UNDO_TABLESPACE` parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment.

The database can also run in *manual undo management mode*. In this mode, undo space is managed through rollback segments, and no undo tablespace is used.

Note:

Space management for rollback segments is complex. Oracle strongly recommends leaving the database in automatic undo management mode.

The following is a summary of the initialization parameters for undo management:

Initialization Parameter	Description
`UNDO_MANAGEMENT`	If `AUTO` or null, enables automatic undo management. If `MANUAL`, sets manual undo management mode. The default is `AUTO`.
`UNDO_TABLESPACE`	Optional, and valid only in automatic undo management mode. Specifies the name of an undo tablespace. Use only when the database has multiple undo tablespaces and you want to direct the database instance to use a particular undo tablespace.

When automatic undo management is enabled, if the initialization parameter file contains parameters relating to manual undo management, they are ignored.

Note:

Earlier releases of Oracle Database default to manual undo management mode. To change to automatic undo management, you must first create an undo tablespace and then change the `UNDO_MANAGEMENT` initialization parameter to `AUTO`. If your Oracle Database is Oracle9*i* or later and you want to change to automatic undo management, see *Oracle Database Upgrade Guide* for instructions.

A null `UNDO_MANAGEMENT` initialization parameter defaults to automatic undo management mode in Oracle Database 11*g* and later, but defaults to manual undo management mode in earlier releases. You must therefore use caution when upgrading a previous release to the current release. *Oracle Database Upgrade Guide* describes the correct method of migrating to automatic undo management mode, including information on how to size the undo tablespace.

14.2.2 The Undo Retention Period

The **undo retention period** is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it.

- About the Undo Retention Period
  When automatic undo management is enabled, there is always a current **undo retention period**, which is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it.
- Automatic Tuning of Undo Retention
  Oracle Database automatically tunes the undo retention period based on how the undo tablespace is configured.
- Retention Guarantee
  To guarantee the success of long-running queries or Oracle Flashback operations, you can enable retention guarantee.
- Undo Retention Tuning and Alert Thresholds
  For a fixed-size undo tablespace, the database calculates the best possible retention based on database statistics and on the size of the undo tablespace.
- Tracking the Tuned Undo Retention Period
  You can determine the current retention period by querying the `TUNED_UNDORETENTION` column of the `V$UNDOSTAT` view.

14.2.2.1 About the Undo Retention Period

When automatic undo management is enabled, there is always a current **undo retention period**, which is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it.

After a transaction is committed, undo data is no longer needed for rollback or transaction recovery purposes. However, for consistent read purposes, long-running queries may require this old undo information for producing older images of data blocks. Furthermore, the success of several Oracle Flashback features can also depend upon the availability of older undo information. For these reasons, it is desirable to retain the old undo information for as long as possible.

Old (committed) undo information that is older than the current undo retention period is said to be *expired* and its space is available to be overwritten by new transactions. Old undo information with an age that is less than the current undo retention period is said to be *unexpired* and is retained for consistent read and Oracle Flashback operations.

Oracle Database automatically tunes the undo retention period based on undo tablespace size and system activity. You can optionally specify a minimum undo retention period (in seconds) by setting the `UNDO_RETENTION` initialization parameter. The exact impact this parameter on undo retention is as follows:

- The `UNDO_RETENTION` parameter is ignored for a fixed size undo tablespace. The database always tunes the undo retention period for the best possible retention, based on system activity and undo tablespace size. See "Automatic Tuning of Undo Retention" for more information.
- For an undo tablespace with the `AUTOEXTEND` option enabled, the database attempts to honor the minimum retention period specified by `UNDO_RETENTION`. When space is low, instead of overwriting unexpired undo information, the tablespace auto-extends. If the `MAXSIZE` clause is specified for an auto-extending undo tablespace, when the maximum

size is reached, the database may begin to overwrite unexpired undo information. The `UNDOTBS1` tablespace that is automatically created by DBCA is auto-extending.

The `UNDO_RETENTION` parameter is not inhertiable in a CDB database.

If you want to set the parameter `UNDO_RETENTION` in all PDBs with the same value, use the CONTAINER=ALL clause from the CDB$ROOT. For example,

```
alter session set container = cdb$root;
alter system set undo_retention = 2000 container=all scope=both;
```

This modified the parameter value only in memory for all PDBs, and in both the SPFILE and memory for the CDB$ROOT. The parameter is not persisted across PDB restarts.

You can modify the `UNDO_RETENTION` parameter so that it is persistent across restarts in each PDB. For example,

```
alter session set container = pdb1;
alter system set undo_retention = 2000 scope=both;
```

Note:

Modifying `UNDO_RETENTION` is only useful if you are using the Oracle Flashback feature, like Flashback Query or Active Dataguard, that require undo to be retained for longer than the longest running query in the system. The purpose of `UNDO_RETENTION` is to enforce, on a best-effort basis, a minimum value of undo retention by the autotuning algorithm, but it is not guaranteed.

14.2.2.2 Automatic Tuning of Undo Retention

Oracle Database automatically tunes the undo retention period based on how the undo tablespace is configured.

Oracle Database automatically tunes the undo retention period based on various parameters such as undo tablespace size, undo generation rate, maximum query length, and the `RETENTION GUARANTEE` setting. A bigger undo tablespace size provides the ability to retain undo for a longer duration. Therefore, it's important to choose the size of a fixed-size undo tablespace or `MAXSIZE` of an `AUTOEXTEND` undo tablespace that is sufficiently large to accommodate your workload. If you choose an undo tablespace size that is too small, the following two errors could occur:

- DML could fail because there is not enough space to accommodate undo for new transactions.
- Long-running queries could fail with a snapshot too old error, which means that there was insufficient undo data for read consistency.

See Sizing a Fixed-Size Undo Tablespace for more information.

If the rate of undo consumption is high, undo cannot be retained for long by the database. Therefore, the retention will be automatically tuned to a lower value. Alternately, a low undo consumption rate will provide a higher undo retention.

See V$UNDOSTAT `UNDOBLKS` column to get the undo blocks consumed over 10-minute intervals.

Whether the undo tablespace is fixed size or configured with the `AUTOEXTEND` option, the database dynamically tunes the undo retention period to be somewhat longer than the longest-running active query on the system. If queries run for a very long duration and the database cannot provide a high retention, it dynamically tunes the undo retention period to a lower best possible retention for that tablespace size and the current system load.

However, the retention period based on the longest-running active query may be insufficient to accommodate Oracle Flashback operations. Oracle Flashback operations resulting in snapshot too old errors are the indicator for which you must intervene to ensure that sufficient undo data is retained to support these operations. To better accommodate Oracle Flashback features, you should set the `UNDO_RETENTION` parameter to a value equal to the longest expected Oracle Flashback operation.

See V$UNDOSTAT `MAXQUERYLEN` column to get the longest-running active query duration.

Setting the `UNDO_RETENTION` parameter for Active Data Guard does not have any effect since the physical standby server does not run any DMLs. To accommodate the queries on Active Data Guard, specify the minimum retention on the primary database instance using the `UNDO_RETENTION` parameter and monitor V$UNDOSTAT's `TUNED_UNDORETENTION` on the primary server.

Note:

Automatic tuning of undo retention is not supported for LOBs. This is because undo information for LOBs is stored in the segment itself and not in the undo tablespace. For LOBs, the database attempts to honor the minimum undo retention period specified by `UNDO_RETENTION`. However, if space becomes low, unexpired LOB undo information may be overwritten.

See Also:

Setting the Minimum Undo Retention Period

Retention Guarantee

14.2.2.3 Retention Guarantee

To guarantee the success of long-running queries or Oracle Flashback operations, you can enable retention guarantee.

If retention guarantee is enabled, then the specified minimum undo retention is guaranteed; the database never overwrites unexpired undo data even if it means that transactions fail due to lack of space in the undo tablespace. If retention guarantee is not enabled, then the database can overwrite unexpired undo when space is low, thus lowering the undo retention for the system. This option is disabled by default.

WARNING:

Enabling retention guarantee can cause multiple DML operations to fail. Use with caution.

You enable retention guarantee by specifying the `RETENTION GUARANTEE` clause for the undo tablespace when you create it with either the `CREATE DATABASE` or `CREATE UNDO TABLESPACE` statement. Or, you can later specify this clause in an `ALTER TABLESPACE` statement. You disable retention guarantee with the `RETENTION NOGUARANTEE` clause.

You can use the `DBA_TABLESPACES` view to determine the retention guarantee setting for the undo tablespace. A column named `RETENTION` contains a value of `GUARANTEE`, `NOGUARANTEE`, or `NOT APPLY`, where `NOT APPLY` is used for tablespaces other than the undo tablespace.

14.2.2.4 Undo Retention Tuning and Alert Thresholds

For a fixed-size undo tablespace, the database calculates the best possible retention based on database statistics and on the size of the undo tablespace.

For optimal undo management, rather than tuning based on 100% of the tablespace size, the database tunes the undo retention period based on 70% of the tablespace size, or on the warning alert threshold percentage for space used, whichever is lower. (The warning alert threshold defaults to 70%, but can be changed.) Therefore, if you set the warning alert threshold of the undo tablespace below 70%, this may reduce the tuned size of the undo retention period. For more information on tablespace alert thresholds, see "Managing Tablespace Alerts".

14.2.2.5 Tracking the Tuned Undo Retention Period

You can determine the current retention period by querying the `TUNED_UNDORETENTION` column of the `V$UNDOSTAT` view.

This view contains one row for each 10-minute statistics collection interval over the last 4 days. (Beyond 4 days, the data is available in the `DBA_HIST_UNDOSTAT` view.) `TUNED_UNDORETENTION` is given in seconds.

```
select to_char(begin_time, 'DD-MON-RR HH24:MI') begin_time,
to_char(end_time, 'DD-MON-RR HH24:MI') end_time, tuned_undoretention
from v$undostat order by end_time;

BEGIN_TIME      END_TIME        TUNED_UNDORETENTION
--------------- --------------- -------------------
04-FEB-05 00:01 04-FEB-05 00:11               12100
      ...
07-FEB-05 23:21 07-FEB-05 23:31               86700
07-FEB-05 23:31 07-FEB-05 23:41               86700
07-FEB-05 23:41 07-FEB-05 23:51               86700
07-FEB-05 23:51 07-FEB-05 23:52               86700

576 rows selected.
```

See *Oracle Database Reference* for more information about `V$UNDOSTAT`.

14.3 Setting the Minimum Undo Retention Period

You specify the minimum undo retention period (in seconds) by setting the `UNDO_RETENTION` initialization parameter.

As described in "About the Undo Retention Period", the current undo retention period may be automatically tuned to be greater than `UNDO_RETENTION`, or, unless retention guarantee is enabled, less than `UNDO_RETENTION` if space in the undo tablespace is low.

To set the minimum undo retention period:

- Do one of the following:
  - Set `UNDO_RETENTION` in the initialization parameter file.

    ```
    UNDO_RETENTION = 1800
    ```

  - Change `UNDO_RETENTION` at any time using the `ALTER SYSTEM` statement:

    ```
    ALTER SYSTEM SET UNDO_RETENTION = 2400;
    ```

The effect of an `UNDO_RETENTION` parameter change is immediate, but it can only be honored if the current undo tablespace has enough space. The `UNDO_RETENTION` parameter is not inheritable in a CDB configuration and should only be changed from its default value to accommodate Oracle Flashback operations or Active Data Guard.

14.4 Sizing a Fixed-Size Undo Tablespace

Automatic tuning of undo retention typically achieves better results with a fixed-size undo tablespace. If you decide to use a fixed-size undo tablespace, then the Undo Advisor can help you estimate needed capacity.

You can access the Undo Advisor through Oracle Enterprise Manager Database Express (EM Express) or through the `DBMS_ADVISOR` PL/SQL package. EM Express is the preferred method of accessing the advisor.

The Undo Advisor relies for its analysis on data collected in the Automatic Workload Repository (AWR). It is therefore important that the AWR have adequate workload statistics available so that the Undo Advisor can make accurate recommendations. For newly created databases, adequate statistics may not be available immediately. In such cases, continue to use the default auto-extending undo tablespace until at least one workload cycle completes.

An adjustment to the collection interval and retention period for AWR statistics can affect the precision and the type of recommendations that the advisor produces. See *Oracle Database Performance Tuning Guide* for more information.

To use the Undo Advisor, you first estimate these two values:

- The length of your expected longest running query

  After the database has completed a workload cycle, you can view the Longest Running Query field on the System Activity subpage of the Automatic Undo Management page.

- The longest interval that you will require for Oracle Flashback operations

  For example, if you expect to run Oracle Flashback queries for up to 48 hours in the past, your Oracle Flashback requirement is 48 hours.

You then take the maximum of these two values and use that value as input to the Undo Advisor.

Running the Undo Advisor does not alter the size of the undo tablespace. The advisor just returns a recommendation. You must use `ALTER DATABASE` statements to change the tablespace data files to fixed sizes.

The following example assumes that the undo tablespace has one auto-extending data file named `undotbs.dbf`. The example changes the tablespace to a fixed size of 300MB.

```
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' RESIZE 300M;
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' AUTOEXTEND OFF;
```

Note:

To make the undo tablespace fixed-size, Oracle suggests that you first allow enough time after database creation to run a full workload, thus allowing the undo tablespace to grow to its minimum required size to handle the workload. Then, you can use the Undo Advisor to determine, if desired, how much larger to set the size of the undo tablespace to allow for long-running queries and Oracle Flashback operations.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

- Activating the Undo Advisor PL/SQL Interface
  You can activate the Undo Advisor by creating an undo advisor task through the advisor framework.

14.4.1 Activating the Undo Advisor PL/SQL Interface

You can activate the Undo Advisor by creating an undo advisor task through the advisor framework.

The following example creates an undo advisor task to evaluate the undo tablespace. The name of the advisor is 'Undo Advisor'. The analysis is based on Automatic Workload Repository snapshots, which you must specify by setting parameters `START_SNAPSHOT` and `END_SNAPSHOT`. In the following example, the `START_SNAPSHOT` is "1" and `END_SNAPSHOT` is "2".

```
DECLARE
   tid    NUMBER;
   tname  VARCHAR2(30);
   oid    NUMBER;
BEGIN
   DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
   DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'INSTANCE', 1);
   DBMS_ADVISOR.execute_task(tname);
END;
/
```

After you have created the advisor task, you can view the output and recommendations in the Automatic Database Diagnostic Monitor in EM Express. This information is also available in the

`DBA_ADVISOR_*` data dictionary views (`DBA_ADVISOR_TASKS`, `DBA_ADVISOR_OBJECTS`, `DBA_ADVISOR_FINDINGS`, `DBA_ADVISOR_RECOMMENDATIONS`, and so on).

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

See Also:

- "Using the Segment Advisor" for an example of creating an advisor task for a different advisor
- *Oracle Database Reference* for information about the `DBA_ADVISOR_*` data dictionary views

14.5 Managing Undo Tablespaces

You manage undo tablespaces by completing tasks such as creating, altering, and dropping them. You can also switch undo tablespaces and establish user quotas for undo space.

- Creating an Undo Tablespace
  Although Database Configuration Assistant (DBCA) automatically creates an undo tablespace for new installations, there may be occasions when you want to manually create an undo tablespace.
- Altering an Undo Tablespace
  You can alter an undo tablespaces using the `ALTER TABLESPACE` statement.
- Dropping an Undo Tablespace
  Use the `DROP TABLESPACE` statement to drop an undo tablespace.
- Switching Undo Tablespaces
  You can switch from using one undo tablespace to another. Because the `UNDO_TABLESPACE` initialization parameter is a dynamic parameter, the `ALTER SYSTEM SET` statement can be used to assign a new undo tablespace.
- Establishing User Quotas for Undo Space
  You can use the Oracle Database Resource Manager to establish user quotas for undo space. The Database Resource Manager directive `UNDO_POOL` allows DBAs to limit the amount of undo space consumed by a group of users (resource consumer group).
- Managing Space Threshold Alerts for the Undo Tablespace
  Oracle Database provides proactive help in managing tablespace disk space use by alerting you when tablespaces run low on available space.

14.5.1 Creating an Undo Tablespace

Although Database Configuration Assistant (DBCA) automatically creates an undo tablespace for new installations, there may be occasions when you want to manually create an undo tablespace.

- About Creating an Undo Tablespace
  When you are creating a database, you can create an undo tablespace with the `CREATE DATABASE` statement. In an existing database, you can create an undo tablespace with the `CREATE UNDO TABLESPACE` statement.
- Using CREATE DATABASE to Create an Undo Tablespace
  You can create a specific undo tablespace using the `UNDO TABLESPACE` clause of the `CREATE DATABASE` statement.
- Using the CREATE UNDO TABLESPACE Statement
  The `CREATE UNDO TABLESPACE` statement is the same as the `CREATE TABLESPACE` statement, but the `UNDO` keyword is specified. The database determines most of the attributes of the undo tablespace, but you can specify the `DATAFILE` clause.

14.5.1.1 About Creating an Undo Tablespace

When you are creating a database, you can create an undo tablespace with the `CREATE DATABASE` statement. In an existing database, you can create an undo tablespace with the `CREATE UNDO TABLESPACE` statement.

There are two methods of creating an undo tablespace. The first method creates the undo tablespace when the `CREATE DATABASE` statement is issued. This occurs when you are creating a new database, and the instance is started in automatic undo management mode (`UNDO_MANAGEMENT = AUTO`). The second method is used with an existing database. It uses the `CREATE UNDO TABLESPACE` statement.

You cannot create database objects in an undo tablespace. It is reserved for system-managed undo data.

Oracle Database enables you to create a single-file undo tablespace. Single-file, or bigfile, tablespaces are discussed in "Bigfile Tablespaces".

14.5.1.2 Using CREATE DATABASE to Create an Undo Tablespace

You can create a specific undo tablespace using the `UNDO TABLESPACE` clause of the `CREATE DATABASE` statement.

The following statement illustrates using the `UNDO TABLESPACE` clause in a `CREATE DATABASE` statement. The undo tablespace is named `undotbs_01` and one data file, `/u01/oracle/rbdb1/undo0101.dbf`, is allocated for it.

```
CREATE DATABASE rbdb1
     CONTROLFILE REUSE
     .
     .
     .
     UNDO TABLESPACE undotbs_01 DATAFILE '/u01/oracle/rbdb1/undo0101.dbf';
```

If the undo tablespace cannot be created successfully during `CREATE DATABASE`, the entire `CREATE DATABASE` operation fails. You must clean up the database files, correct the error and retry the `CREATE DATABASE` operation.

The `CREATE DATABASE` statement also lets you create a single-file undo tablespace at database creation.

Related Topics

- *Oracle Multitenant Administrator's Guide*

- *Oracle Database SQL Language Reference*

14.5.1.3 Using the CREATE UNDO TABLESPACE Statement

The `CREATE UNDO TABLESPACE` statement is the same as the `CREATE TABLESPACE` statement, but the `UNDO` keyword is specified. The database determines most of the attributes of the undo tablespace, but you can specify the `DATAFILE` clause.

This example creates the `undotbs_02` undo tablespace with the `AUTOEXTEND` option:

```
CREATE UNDO TABLESPACE undotbs_02
     DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
```

You can create multiple undo tablespaces, but only one of them can be active at any one time.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntax for using the `CREATE UNDO TABLESPACE` statement to create an undo tablespace

14.5.2 Altering an Undo Tablespace

You can alter an undo tablespaces using the `ALTER TABLESPACE` statement.

However, since most aspects of undo tablespaces are system managed, you need only be concerned with the following actions:

- Adding a data file
- Renaming a data file
- Bringing a data file online or taking it offline
- Beginning or ending an open backup on a data file
- Enabling and disabling undo retention guarantee

These are also the only attributes you are permitted to alter.

If an undo tablespace runs out of space, or you want to prevent it from doing so, you can add more files to it or resize existing data files.

The following example adds another data file to undo tablespace undotbs_01:

```
ALTER TABLESPACE undotbs_01
     ADD DATAFILE '/u01/oracle/rbdb1/undo0102.dbf' AUTOEXTEND ON NEXT 1M
         MAXSIZE UNLIMITED;
```

You can use the `ALTER DATABASE...DATAFILE` statement to resize or extend a data file.

See Also:

- "Changing Data File Size"
- *Oracle Database SQL Language Reference* for `ALTER TABLESPACE` syntax

14.5.3 Dropping an Undo Tablespace

Use the `DROP TABLESPACE` statement to drop an undo tablespace.

The following example drops the undo tablespace `undotbs_01`:

```
DROP TABLESPACE undotbs_01;
```

An undo tablespace can only be dropped if it is not currently used by any instance. If the undo tablespace contains any outstanding transactions (for example, a transaction died but has not yet been recovered), the `DROP TABLESPACE` statement fails. However, since `DROP TABLESPACE` drops an undo tablespace even if it contains unexpired undo information (within retention period), you must be careful not to drop an undo tablespace if undo information is needed by some existing queries.

`DROP TABLESPACE` for undo tablespaces behaves like `DROP TABLESPACE...INCLUDING CONTENTS`. All contents of the undo tablespace are removed.

See Also:

Oracle Database SQL Language Reference for `DROP TABLESPACE` syntax

14.5.4 Switching Undo Tablespaces

You can switch from using one undo tablespace to another. Because the `UNDO_TABLESPACE` initialization parameter is a dynamic parameter, the `ALTER SYSTEM SET` statement can be used to assign a new undo tablespace.

The following statement switches to a new undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs_02;
```

Assuming `undotbs_01` is the current undo tablespace, after this command successfully executes, the instance uses `undotbs_02` in place of `undotbs_01` as its undo tablespace.

If any of the following conditions exist for the tablespace being switched to, an error is reported and no switching occurs:

- The tablespace does not exist
- The tablespace is not an undo tablespace
- The tablespace is already being used by another instance (in an Oracle RAC environment only)

The database is online while the switch operation is performed, and user transactions can be executed while this command is being executed. When the switch operation completes successfully, all transactions started after the switch operation began are assigned to transaction tables in the new undo tablespace.

The switch operation does not wait for transactions in the old undo tablespace to commit. If there are any pending transactions in the old undo tablespace, the old undo tablespace enters into a `PENDING OFFLINE` mode (status). In this mode, existing transactions can continue to execute, but undo records for new user transactions cannot be stored in this undo tablespace.

An undo tablespace can exist in this `PENDING OFFLINE` mode, even after the switch operation completes successfully. A `PENDING OFFLINE` undo tablespace cannot be used by another instance, nor can it be dropped. Eventually, after all active transactions have committed, the undo tablespace automatically goes from the `PENDING OFFLINE` mode to the `OFFLINE` mode. From then on, the undo tablespace is available for other instances (in an Oracle Real Application Cluster environment).

If the parameter value for `UNDO TABLESPACE` is set to '' (two single quotes), then the current undo tablespace is switched out and the next available undo tablespace is switched in. Use this statement with care because there may be no undo tablespace available.

The following example unassigns the current undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

14.5.5 Establishing User Quotas for Undo Space

You can use the Oracle Database Resource Manager to establish user quotas for undo space. The Database Resource Manager directive `UNDO_POOL` allows DBAs to limit the amount of undo space consumed by a group of users (resource consumer group).

You can specify an undo pool for each consumer group. An undo pool controls the amount of total undo that can be generated by a consumer group. When the total undo generated by a consumer group exceeds its undo limit, the current `UPDATE` transaction generating the undo is terminated. No other members of the consumer group can perform further updates until undo space is freed from the pool.

When no `UNDO_POOL` directive is explicitly defined, users are allowed unlimited undo space.

See Also:

Managing Resources with Oracle Database Resource Manager

14.5.6 Managing Space Threshold Alerts for the Undo Tablespace

Oracle Database provides proactive help in managing tablespace disk space use by alerting you when tablespaces run low on available space.

See "Managing Tablespace Alerts" for information on how to set alert thresholds for the undo tablespace.

In addition to the proactive undo space alerts, Oracle Database also provides alerts if your system has long-running queries that cause `SNAPSHOT TOO OLD` errors. To prevent excessive alerts, the long query alert is issued at most once every 24 hours. When the alert is generated, you can check the Undo Advisor Page of EM Express to get more information about the undo tablespace.

Note:

Oracle Enterprise Manager Database Express (EM Express) is deprecated, and will be removed in a future Oracle Database release.

14.6 Migrating to Automatic Undo Management

If you are currently using rollback segments to manage undo space, Oracle strongly recommends that you migrate your database to automatic undo management.

For instructions, see *Oracle Database Upgrade Guide*.

14.7 Managing Temporary Undo

By default, undo records for temporary tables are stored in the undo tablespace and are logged in the redo, which is the same way undo is managed for persistent tables. However, you can use the `TEMP_UNDO_ENABLED` initialization parameter to separate undo for temporary tables from undo for persistent tables. When this parameter is set to `TRUE`, the undo for temporary tables is called **temporary undo**.

- About Managing Temporary Undo
  Temporary undo records are stored in the database's temporary tablespaces and thus are not logged in the redo log. When temporary undo is enabled, some of the segments used by the temporary tablespaces store the temporary undo, and these segments are called **temporary undo segments**.
- Enabling and Disabling Temporary Undo
  You can enable or disable temporary undo for a session or for the system. To do so, set the `TEMP_UNDO_ENABLED` initialization parameter.

14.7.1 About Managing Temporary Undo

Temporary undo records are stored in the database's temporary tablespaces and thus are not logged in the redo log. When temporary undo is enabled, some of the segments used by the temporary tablespaces store the temporary undo, and these segments are called **temporary undo segments**.

When temporary undo is enabled, it might be necessary to increase the size of the temporary tablespaces to account for the undo records.

Enabling temporary undo provides the following benefits:

- Temporary undo reduces the amount of undo stored in the undo tablespaces.

  Less undo in the undo tablespaces can result in more realistic undo retention period requirements for undo records.
- Temporary undo reduces the size of the redo log.

  Performance is improved because less data is written to the redo log, and components that parse redo log records, such as LogMiner, perform better because there is less redo data to parse.
- Temporary undo enables data manipulation language (DML) operations on temporary tables in a physical standby database with the Oracle Active Data Guard option. However,

data definition language (DDL) operations that create temporary tables must be issued on the primary database.

You can enable temporary undo for a specific session or for the whole system. When you enable temporary undo for a session using an `ALTER SESSION` statement, the session creates temporary undo without affecting other sessions. When you enable temporary undo for the system using an `ALTER SYSTEM` statement, all existing sessions and new sessions create temporary undo.

When a session uses temporary objects for the first time, the current value of the `TEMP_UNDO_ENABLED` initialization parameter is set for the rest of the session. Therefore, if temporary undo is enabled for a session and the session uses temporary objects, then temporary undo cannot be disabled for the session. Similarly, if temporary undo is disabled for a session and the session uses temporary objects, then temporary undo cannot be enabled for the session.

Temporary undo is enabled by default for a physical standby database with the Oracle Active Data Guard option. The `TEMP_UNDO_ENABLED` initialization parameter has no effect on a physical standby database with Active Data Guard option because of the default setting.

Note:

Temporary undo can be enabled only if the compatibility level of the database is 12.0.0 or higher.

See Also:

- "Creating a Temporary Table"
- "About the Undo Retention Period"
- *Oracle Database Reference* for more information about the `TEMP_UNDO_ENABLED` initialization parameter
- *Oracle Data Guard Concepts and Administration*
- *Oracle Database Concepts* for more information about temporary undo segments

14.7.2 Enabling and Disabling Temporary Undo

You can enable or disable temporary undo for a session or for the system. To do so, set the `TEMP_UNDO_ENABLED` initialization parameter.

To enable or disable temporary undo:

1. In SQL*Plus, connect to the database.

   If you are enabling or disabling temporary undo for a session, then start the session in SQL*Plus.

   If you are enabling or disabling temporary undo for the system, then connect as an administrative user with the `ALTER SYSTEM` system privilege in SQL*Plus.

   See "Connecting to the Database with SQL*Plus".

2. Set the `TEMP_UNDO_ENABLED` initialization parameter:
   - To enable temporary undo for a session, run the following SQL statement:

     ```
     ALTER SESSION SET TEMP_UNDO_ENABLED = TRUE;
     ```

   - To disable temporary undo for a session, run the following SQL statement:

     ```
     ALTER SESSION SET TEMP_UNDO_ENABLED = FALSE;
     ```

   - To enable temporary undo for the system, run the following SQL statement:

     ```
     ALTER SYSTEM SET TEMP_UNDO_ENABLED = TRUE;
     ```

     After temporary undo is enabled for the system, a session can disable temporary undo using the `ALTER SESSION` statement.
   - To disable temporary undo for the system, run the following SQL statement:

     ```
     ALTER SYSTEM SET TEMP_UNDO_ENABLED = FALSE;
     ```

     After temporary undo is disabled for the system, a session can enable temporary undo using the `ALTER SESSION` statement.

You can also enable temporary undo for the system by setting `TEMP_UNDO_ENABLED` to `TRUE` in a server parameter file or a text initialization parameter file. In this case, all new sessions create temporary undo unless temporary undo is disabled for the system by an `ALTER SYSTEM` statement or for a session by an `ALTER SESSION` statement.

See Also:

- *Oracle Database Reference* for more information about the `TEMP_UNDO_ENABLED` initialization parameter
- *Oracle Data Guard Concepts and Administration* for information about enabling and disabling temporary undo in an Oracle Data Guard environment

14.8 Undo Space Data Dictionary Views

You can query a set of views for information about undo space in the automatic undo management mode.

In addition to views listed here, you can obtain information from the views available for viewing tablespace and data file information. See "Data Files Data Dictionary Views" for information on getting information about those views.

The following dynamic performance views are useful for obtaining space information about the undo tablespace:

View	Description
`V$UNDOSTAT`	Contains statistics for monitoring and tuning undo space. Use this view to help estimate the amount of undo space required for the current workload. The database also uses this information to help tune undo usage in the system. This view is meaningful only in automatic undo management mode.

View	Description
`V$TEMPUNDOSTAT`	Contains statistics for monitoring and tuning temporary undo space. Use this view to help estimate the amount of temporary undo space required in the temporary tablespaces for the current workload. The database also uses this information to help tune temporary undo usage in the system. This view is meaningful only when temporary undo is enabled.
`V$ROLLSTAT`	For automatic undo management mode, information reflects behavior of the undo segments in the undo tablespace
`V$TRANSACTION`	Contains undo segment information
`DBA_UNDO_EXTENTS`	Shows the status and size of each extent in the undo tablespace.
`DBA_HIST_UNDOSTAT`	Contains statistical snapshots of `V$UNDOSTAT` information.

The `V$UNDOSTAT` view is useful for monitoring the effects of transaction execution on undo space in the current instance. Statistics are available for undo space consumption, transaction concurrency, the tuning of undo retention, and the length and SQL ID of long-running queries in the instance.

Each row in the view contains statistics collected in the instance for a ten-minute interval. The rows are in descending order by the `BEGIN_TIME` column value. Each row belongs to the time interval marked by (`BEGIN_TIME`, `END_TIME`). Each column represents the data collected for the particular statistic in that time interval. The first row of the view contains statistics for the (partial) current time period. The view contains a total of 576 rows, spanning a 4 day cycle.

The following example shows the results of a query on the `V$UNDOSTAT` view.

```
SELECT TO_CHAR(BEGIN_TIME, 'MM/DD/YYYY HH24:MI:SS') BEGIN_TIME,
       TO_CHAR(END_TIME, 'MM/DD/YYYY HH24:MI:SS') END_TIME,
       UNDOTSN, UNDOBLKS, TXNCOUNT, MAXCONCURRENCY AS "MAXCON"
       FROM v$UNDOSTAT WHERE rownum <= 144;

BEGIN_TIME          END_TIME               UNDOTSN   UNDOBLKS   TXNCOUNT     MAXCON
------------------- ------------------- ---------- ---------- ---------- ----------
10/28/2004 14:25:12 10/28/2004 14:32:17          8         74   12071108          3
10/28/2004 14:15:12 10/28/2004 14:25:12          8         49   12070698          2
10/28/2004 14:05:12 10/28/2004 14:15:12          8        125   12070220          1
10/28/2004 13:55:12 10/28/2004 14:05:12          8         99   12066511          3
...
10/27/2004 14:45:12 10/27/2004 14:55:12          8         15   11831676          1
10/27/2004 14:35:12 10/27/2004 14:45:12          8        154   11831165          2

144 rows selected.
```

The preceding example shows how undo space is consumed in the system for the previous 24 hours from the time 14:35:12 on 10/27/2004.

15

Using Oracle Managed Files

Oracle Database can manage the files that comprise the database.

- About Oracle Managed Files
  Oracle Managed Files eases database administration, reduces errors, and reduces wasted disk space.
- Enabling the Creation and Use of Oracle Managed Files
  You set certain initialization parameters to enable and use Oracle Managed Files.
- Creating Oracle Managed Files
  You can use Oracle Managed Files to create data files, temp files, control files, redo log files, and archived log.
- Operation of Oracle Managed Files
  The file names of Oracle Managed Files are accepted in SQL statements wherever a file name is used to identify an existing file.
- Scenarios for Using Oracle Managed Files
  Scenarios illustrate how to use Oracle Managed Files.

15.1 About Oracle Managed Files

Oracle Managed Files eases database administration, reduces errors, and reduces wasted disk space.

- What Is Oracle Managed Files?
  Using Oracle Managed Files simplifies the administration of an Oracle Database. Oracle Managed Files eliminates the need for you, the DBA, to directly manage the operating system files that comprise an Oracle Database.
- Who Can Use Oracle Managed Files?
  Oracle Managed Files is most useful for certain types of databases.
- What Is a Logical Volume Manager?
  A logical volume manager (LVM) is a software package available with most operating systems. Sometimes it is called a logical disk manager (LDM). It allows pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software.
- What Is a File System?
  A file system is a data structure built inside a contiguous disk address space. A file manager (FM) is a software package that manipulates file systems, but it is sometimes called the file system.
- Benefits of Using Oracle Managed Files
  Oracle Managed Files provides several benefits.
- Oracle Managed Files and Existing Functionality
  Using Oracle Managed Files does not eliminate any existing functionality.

15.1.1 What Is Oracle Managed Files?

Using Oracle Managed Files simplifies the administration of an Oracle Database. Oracle Managed Files eliminates the need for you, the DBA, to directly manage the operating system files that comprise an Oracle Database.

With Oracle Managed Files, you specify file system directories in which the database automatically creates, names, and manages files at the database object level. For example, you need only specify that you want to create a tablespace; you do not need to specify the name and path of the tablespace's data file with the `DATAFILE` clause. This feature works well with a logical volume manager (LVM).

The database internally uses standard file system interfaces to create and delete files as needed for the following database structures:

- Tablespaces
- Redo log files
- Control files
- Archived logs
- Block change tracking files
- Flashback logs
- RMAN backups

Through initialization parameters, you specify the file system directory to be used for a particular type of file. The database then ensures that a unique file, an Oracle managed file, is created and deleted when no longer needed.

This feature does not affect the creation or naming of administrative files such as trace files, audit files, alert logs, and core files.

See Also:

Oracle Automatic Storage Management Administrator's Guide for information about Oracle Automatic Storage Management (Oracle ASM), the Oracle Database integrated file system and volume manager that extends the power of Oracle Managed Files. With Oracle Managed Files, files are created and managed automatically for you, but with Oracle ASM, you get the additional benefits of features such as striping, software mirroring, and dynamic storage configuration, without the need to purchase a third-party logical volume manager.

15.1.2 Who Can Use Oracle Managed Files?

Oracle Managed Files is most useful for certain types of databases.

Oracle Managed Files are most useful for the following types of databases:

- Databases that are supported by the following:
  - A logical volume manager that supports striping/RAID and dynamically extensible logical volumes

  - A file system that provides large, extensible files
- Low end or test databases

Because Oracle Managed Files require that you use the operating system file system, you lose control over how files are laid out on the disks, and thus, you lose some I/O tuning ability.

15.1.3 What Is a Logical Volume Manager?

A logical volume manager (LVM) is a software package available with most operating systems. Sometimes it is called a logical disk manager (LDM). It allows pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software.

An LVM can make the logical volume have better capacity, performance, reliability, and availability characteristics than any of the underlying physical disks. It uses techniques such as mirroring, striping, concatenation, and RAID 5 to implement these characteristics.

Some LVMs allow the characteristics of a logical volume to be changed after it is created, even while it is in use. The volume may be resized or mirrored, or it may be relocated to different physical disks.

15.1.4 What Is a File System?

A file system is a data structure built inside a contiguous disk address space. A file manager (FM) is a software package that manipulates file systems, but it is sometimes called the file system.

All operating systems have file managers. The primary task of a file manager is to allocate and deallocate disk space into files within a file system.

A file system allows the disk space to be allocated to a large number of files. Each file is made to appear as a contiguous address space to applications such as Oracle Database. The files may not actually be contiguous within the disk space of the file system. Files can be created, read, written, resized, and deleted. Each file has a name associated with it that is used to refer to the file.

A file system is commonly built on top of a logical volume constructed by an LVM. Thus all the files in a particular file system have the same performance, reliability, and availability characteristics inherited from the underlying logical volume. A file system is a single pool of storage that is shared by all the files in the file system. If a file system is out of space, then none of the files in that file system can grow. Space available in one file system does not affect space in another file system. However some LVM/FM combinations allow space to be added or removed from a file system.

An operating system can support multiple file systems. Multiple file systems are constructed to give different storage characteristics to different files as well as to divide the available disk space into pools that do not affect each other.

15.1.5 Benefits of Using Oracle Managed Files

Oracle Managed Files provides several benefits.

Consider the following benefits of using Oracle Managed Files:

- They make the administration of the database easier.

There is no need to invent file names and define specific storage requirements. A consistent set of rules is used to name all relevant files. The file system defines the characteristics of the storage and the pool where it is allocated.

- They reduce corruption caused by administrators specifying the wrong file.

  Each Oracle managed file and file name is unique. Using the same file in two different databases is a common mistake that can cause very large down times and loss of committed transactions. Using two different names that refer to the same file is another mistake that causes major corruptions.

- They reduce wasted disk space consumed by obsolete files.

  Oracle Database automatically removes old Oracle Managed Files when they are no longer needed. Much disk space is wasted in large systems simply because no one is sure if a particular file is still required. This also simplifies the administrative task of removing files that are no longer required on disk and prevents the mistake of deleting the wrong file.

- They simplify creation of test and development databases.

  You can minimize the time spent making decisions regarding file structure and naming, and you have fewer file management tasks. You can focus better on meeting the actual requirements of your test or development database.

- Oracle Managed Files make development of portable third-party tools easier.

  Oracle Managed Files eliminate the need to put operating system specific file names in SQL scripts.

15.1.6 Oracle Managed Files and Existing Functionality

Using Oracle Managed Files does not eliminate any existing functionality.

Existing databases are able to operate as they always have. New files can be created as managed files while old ones are administered in the old way. Thus, a database can have a mixture of Oracle managed and unmanaged files.

15.2 Enabling the Creation and Use of Oracle Managed Files

You set certain initialization parameters to enable and use Oracle Managed Files.

- Initialization Parameters That Enable Oracle Managed Files
  The following table lists the initialization parameters that enable the use of Oracle Managed Files.
- Setting the DB_CREATE_FILE_DEST Initialization Parameter
  The `DB_CREATE_FILE_DEST` initialization parameter specifies the location of important database files.
- Setting the DB_RECOVERY_FILE_DEST Parameter
  Include the `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE` parameters in your initialization parameter file to identify the default location for the Fast Recovery Area.
- Setting the DB_CREATE_ONLINE_LOG_DEST_n Initialization Parameters
  The `DB_CREATE_ONLINE_LOG_DEST_`*n* initialization parameters specify the locations of the redo log files and the control files.

15.2.1 Initialization Parameters That Enable Oracle Managed Files

The following table lists the initialization parameters that enable the use of Oracle Managed Files.

Initialization Parameter	Description
`DB_CREATE_FILE_DEST`	Defines the location of the default file system directory or Oracle ASM disk group where the database creates data files or temp files when no file specification is given in the create operation. Also used as the default location for redo log and control files if `DB_CREATE_ONLINE_LOG_DEST_n` are not specified.
`DB_CREATE_ONLINE_LOG_DEST_n`	Defines the location of the default file system directory or Oracle ASM disk group for redo log files and control file creation when no file specification is given in the create operation. By changing *n*, you can use this initialization parameter multiple times, where *n* specifies a multiplexed copy of the redo log or control file. You can specify up to five multiplexed copies.
`DB_RECOVERY_FILE_DEST`	Defines the location of the Fast Recovery Area, which is the default file system directory or Oracle ASM disk group where the database creates RMAN backups when no format option is used, archived logs when no other local destination is configured, and flashback logs. Also used as the default location for redo log and control files or multiplexed copies of redo log and control files if `DB_CREATE_ONLINE_LOG_DEST_n` are not specified. When this parameter is specified, the `DB_RECOVERY_FILE_DEST_SIZE` initialization parameter must also be specified.

The file system directories specified by these parameters must already exist; the database does not create them. The directory must also have permissions to allow the database to create the files in it.

The default location is used whenever a location is not explicitly specified for the operation creating the file. The database creates the file name, and a file thus created is an Oracle managed file.

Both of these initialization parameters are dynamic, and can be set using the `ALTER SYSTEM` or `ALTER SESSION` statement.

See Also:

- *Oracle Database Reference* for additional information about initialization parameters
- "How Oracle Managed Files Are Named"

15.2.2 Setting the DB_CREATE_FILE_DEST Initialization Parameter

The `DB_CREATE_FILE_DEST` initialization parameter specifies the location of important database files.

Include the `DB_CREATE_FILE_DEST` initialization parameter in your initialization parameter file to identify the default location for the database server to create:

- Data files
- Temp files
- Redo log files
- Control files
- Block change tracking files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. The following example sets `/u01/app/oracle/oradata` as the default directory to use when creating Oracle Managed Files:

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
```

15.2.3 Setting the DB_RECOVERY_FILE_DEST Parameter

Include the `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE` parameters in your initialization parameter file to identify the default location for the Fast Recovery Area.

The Fast Recovery Area contains:

- Redo log files or multiplexed copies of redo log files
- Control files or multiplexed copies of control files
- RMAN backups (data file copies, control file copies, backup pieces, control file autobackups)
- Archived logs
- Flashback logs

You specify the name of file system directory that becomes the default location for creation of the operating system files for these entities. For example:

```
DB_RECOVERY_FILE_DEST = '/u01/app/oracle/fast_recovery_area'
DB_RECOVERY_FILE_DEST_SIZE = 20G
```

15.2.4 Setting the DB_CREATE_ONLINE_LOG_DEST_n Initialization Parameters

The `DB_CREATE_ONLINE_LOG_DEST_`*`n`* initialization parameters specify the locations of the redo log files and the control files.

Include the `DB_CREATE_ONLINE_LOG_DEST_`*`n`* initialization parameters in your initialization parameter file to identify the default locations for the database server to create:

- Redo log files
- Control files

You specify the name of a file system directory or Oracle ASM disk group that becomes the default location for the creation of the files for these entities. You can specify up to five multiplexed locations.

For the creation of redo log files and control files only, this parameter overrides any default location specified in the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` initialization parameters. If you do not specify a `DB_CREATE_FILE_DEST` parameter, but you do specify the `DB_CREATE_ONLINE_LOG_DEST_`*n* parameter, then only redo log files and control files can be created as Oracle Managed Files.

It is recommended that you specify at least two parameters. For example:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

This allows multiplexing, which provides greater fault-tolerance for the redo log and control file if one of the destinations fails.

15.3 Creating Oracle Managed Files

You can use Oracle Managed Files to create data files, temp files, control files, redo log files, and archived log.

- When Oracle Database Creates Oracle Managed Files
  Oracle Database creates Oracle Managed Files when certain conditions are met.
- How Oracle Managed Files Are Named
  The file names of Oracle Managed Files comply with the Optimal Flexible Architecture (OFA) standard for file naming.
- Creating Oracle Managed Files at Database Creation
  The `CREATE DATABASE` statement can perform actions related to Oracle Managed Files.
- Creating Data Files for Tablespaces Using Oracle Managed Files
  Oracle Database can create data files for tablespaces using Oracle Managed Files when certain conditions are met.
- Creating Temp Files for Temporary Tablespaces Using Oracle Managed Files
  Oracle Database can create temp files for temporary tablespaces using Oracle Managed Files when certain conditions are met.
- Creating Control Files Using Oracle Managed Files
  Oracle Database can create control files using Oracle Managed Files when certain conditions are met.
- Creating Redo Log Files Using Oracle Managed Files
  Redo log files are created at database creation time. They can also be created when you issue either of the following statements: `ALTER DATABASE ADD LOGFILE` and `ALTER DATABASE OPEN RESETLOGS`.
- Creating Archived Logs Using Oracle Managed Files
  Archived logs are created by a background process or by a SQL statement.

15.3.1 When Oracle Database Creates Oracle Managed Files

Oracle Database creates Oracle Managed Files when certain conditions are met.

If you have met any of the following conditions, then Oracle Database creates Oracle Managed Files for you, as appropriate, when no file specification is given in the create operation:

- You have included any of the `DB_CREATE_FILE_DEST`, `DB_RECOVERY_FILE_DEST`, or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters in your initialization parameter file.
- You have issued the `ALTER SYSTEM` statement to dynamically set any of `DB_RECOVERY_FILE_DEST`, `DB_CREATE_FILE_DEST`, or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters
- You have issued the `ALTER SESSION` statement to dynamically set any of the `DB_CREATE_FILE_DEST`, `DB_RECOVERY_FILE_DEST`, or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters.

If a statement that creates an Oracle managed file finds an error or does not complete due to some failure, then any Oracle Managed Files created by the statement are automatically deleted as part of the recovery of the error or failure. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands.

15.3.2 How Oracle Managed Files Are Named

The file names of Oracle Managed Files comply with the Optimal Flexible Architecture (OFA) standard for file naming.

Note:

The naming scheme described in this section applies only to files created in operating system file systems. The naming scheme for files created in Oracle Automatic Storage Management (Oracle ASM) disk groups is described in *Oracle Automatic Storage Management Administrator's Guide*.

The assigned names are intended to meet the following requirements:

- Database files are easily distinguishable from all other files.
- Files of one database type are easily distinguishable from other database types.
- Files are clearly associated with important attributes specific to the file type. For example, a data file name may include the tablespace name to allow for easy association of data file to tablespace, or an archived log name may include the thread, sequence, and creation date.

No two Oracle Managed Files are given the same name. The name that is used for creation of an Oracle managed file is constructed from three sources:

- The default creation location
- A file name template that is chosen based on the type of the file. The template also depends on the operating system platform and whether or not Oracle Automatic Storage Management is used.
- A unique string created by Oracle Database or the operating system. This ensures that file creation does not damage an existing file and that the file cannot be mistaken for some other file.

As a specific example, file names for Oracle Managed Files have the following format on a Solaris file system:

```
destination_prefix/o1_mf_%t_%u_.dbf
```

where:

- `destination_prefix` is `destination_location`/`db_unique_name`/`datafile`

  where:

  – `destination_location` is the location specified in `DB_CREATE_FILE_DEST`

  – `db_unique_name` is the globally unique name (`DB_UNIQUE_NAME` initialization parameter) of the target database. If there is no `DB_UNIQUE_NAME` parameter, then the `DB_NAME` initialization parameter value is used.
- %t is the tablespace name.
- %u is an eight-character string that guarantees uniqueness

For example, assume the following parameter settings:

```
DB_CREATE_FILE_DEST   = /u01/app/oracle/oradata
DB_UNIQUE_NAME = PAYROLL
```

Then an example data file name would be:

```
/u01/app/oracle/oradata/PAYROLL/datafile/o1_mf_tbs1_2ixh90q_.dbf
```

Names for other file types are similar. Names on other platforms are also similar, subject to the constraints of the naming rules of the platform.

The examples on the following pages use Oracle managed file names as they might appear with a Solaris file system as an OMF destination.

Note:

The database identifies an Oracle managed file based on its name. If you rename the file, the database is no longer able to recognize it as an Oracle managed file and will not manage the file accordingly.

15.3.3 Creating Oracle Managed Files at Database Creation

The `CREATE DATABASE` statement can perform actions related to Oracle Managed Files.

Note:

The rules and defaults in this section also apply to creating a database with Database Configuration Assistant (DBCA). With DBCA, you use a graphical interface to enable Oracle Managed Files and to specify file locations that correspond to the initialization parameters described in this section.

- Specifying Control Files at Database Creation
  At database creation, the control file is created in the files specified by the `CONTROL_FILES` initialization parameter.

- Specifying Redo Log Files at Database Creation
  The LOGFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle managed redo log files.
- Specifying the SYSTEM and SYSAUX Tablespace Data Files at Database Creation
  The DATAFILE or SYSAUX DATAFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle managed data files for the SYSTEM and SYSAUX tablespaces.
- Specifying the Undo Tablespace Data File at Database Creation
  The DATAFILE subclause of the UNDO TABLESPACE clause is optional and a file name is not required in the file specification.
- Specifying the Default Temporary Tablespace Temp File at Database Creation
  The TEMPFILE subclause is optional for the DEFAULT TEMPORARY TABLESPACE clause and a file name is not required in the file specification.
- CREATE DATABASE Statement Using Oracle Managed Files: Examples
  Examples illustrate creating a database with the CREATE DATABASE statement when using the Oracle Managed Files feature.

See Also:

Oracle Database SQL Language Reference for a description of the CREATE DATABASE statement

15.3.3.1 Specifying Control Files at Database Creation

At database creation, the control file is created in the files specified by the CONTROL_FILES initialization parameter.

If the CONTROL_FILES parameter is not set and at least one of the initialization parameters required for the creation of Oracle Managed Files is set, then an Oracle managed control file is created in the default control file destinations. In order of precedence, the default destination is defined as follows:

- One or more control files as specified in the DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameter. The file in the first directory is the primary control file. When DB_CREATE_ONLINE_LOG_DEST_*n* is specified, the database does not create a control file in DB_CREATE_FILE_DEST or in DB_RECOVERY_FILE_DEST (the Fast Recovery Area).
- If no value is specified for DB_CREATE_ONLINE_LOG_DEST_*n*, but values are set for both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST, then the database creates one control file in each location. The location specified in DB_CREATE_FILE_DEST is the primary control file.
- If a value is specified only for DB_CREATE_FILE_DEST, then the database creates one control file in that location.
- If a value is specified only for DB_RECOVERY_FILE_DEST, then the database creates one control file in that location.

If the CONTROL_FILES parameter is not set and none of these initialization parameters are set, then the Oracle Database default action is operating system dependent. At least one copy of a control file is created in an operating system dependent default location. Any copies of control files created in this fashion are not Oracle Managed Files, and you must add a CONTROL_FILES initialization parameter to any initialization parameter file.

If the database creates an Oracle managed control file, and if there is a server parameter file, then the database creates a CONTROL_FILES initialization parameter entry in the server parameter file. If there is no server parameter file, then you must manually include a CONTROL_FILES initialization parameter entry in the text initialization parameter file.

See Also:

Managing Control Files

15.3.3.2 Specifying Redo Log Files at Database Creation

The LOGFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle managed redo log files.

If the LOGFILE clause is omitted, then redo log files are created in the default redo log file destinations. In order of precedence, the default destination is defined as follows:

- If either the DB_CREATE_ONLINE_LOG_DEST_*n* is set, then the database creates a log file member in each directory specified, up to the value of the MAXLOGMEMBERS initialization parameter.
- If the DB_CREATE_ONLINE_LOG_DEST_*n* parameter is not set, but both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters are set, then the database creates one Oracle managed log file member in each of those locations. The log file in the DB_CREATE_FILE_DEST destination is the first member.
- If only the DB_CREATE_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.
- If only the DB_RECOVERY_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.

The default size of an Oracle managed redo log file is 100 MB.

Optionally, you can create Oracle managed redo log files, and override default attributes, by including the LOGFILE clause but omitting a file name. Redo log files are created the same way, except for the following: If no file name is provided in the LOGFILE clause of CREATE DATABASE, and none of the initialization parameters required for creating Oracle Managed Files are provided, then the CREATE DATABASE statement fails.

See Also:

" Managing the Redo Log"

15.3.3.3 Specifying the SYSTEM and SYSAUX Tablespace Data Files at Database Creation

The `DATAFILE` or `SYSAUX DATAFILE` clause is not required in the `CREATE DATABASE` statement, and omitting it provides a simple means of creating Oracle managed data files for the `SYSTEM` and `SYSAUX` tablespaces.

If the `DATAFILE` clause is omitted, then one of the following actions occurs:

- If `DB_CREATE_FILE_DEST` is set, then one Oracle managed data file for the `SYSTEM` tablespace and another for the `SYSAUX` tablespace are created in the `DB_CREATE_FILE_DEST` directory.
- If `DB_CREATE_FILE_DEST` is not set, then the database creates one `SYSTEM` and one `SYSAUX` tablespace data file whose names and sizes are operating system dependent. Any `SYSTEM` or `SYSAUX` tablespace data file created in this manner is not an Oracle managed file.

By default, Oracle managed data files, including those for the `SYSTEM` and `SYSAUX` tablespaces, are 100MB and autoextensible. When autoextension is required, the database extends the data file by its existing size or 100 MB, whichever is smaller. You can also explicitly specify the autoextensible unit using the `NEXT` parameter of the `STORAGE` clause when you specify the data file (in a `CREATE` or `ALTER TABLESPACE` operation).

Optionally, you can create an Oracle managed data file for the `SYSTEM` or `SYSAUX` tablespace and override default attributes. This is done by including the `DATAFILE` clause, omitting a file name, but specifying overriding attributes. When a file name is not supplied and the `DB_CREATE_FILE_DEST` parameter is set, an Oracle managed data file for the `SYSTEM` or `SYSAUX` tablespace is created in the `DB_CREATE_FILE_DEST` directory with the specified attributes being overridden. However, if a file name is not supplied and the `DB_CREATE_FILE_DEST` parameter is not set, then the `CREATE DATABASE` statement fails.

When overriding the default attributes of an Oracle managed file, if a `SIZE` value is specified but no `AUTOEXTEND` clause is specified, then the data file is *not* autoextensible.

15.3.3.4 Specifying the Undo Tablespace Data File at Database Creation

The `DATAFILE` subclause of the `UNDO TABLESPACE` clause is optional and a file name is not required in the file specification.

If a file name is not supplied and the `DB_CREATE_FILE_DEST` parameter is set, then an Oracle managed data file is created in the `DB_CREATE_FILE_DEST` directory. If `DB_CREATE_FILE_DEST` is not set, then the statement fails with a syntax error.

The `UNDO TABLESPACE` clause itself is optional in the `CREATE DATABASE` statement. If it is not supplied, and automatic undo management mode is enabled (the default), then a default undo tablespace named `SYS_UNDOTS` is created and a 20 MB data file that is autoextensible is allocated as follows:

- If `DB_CREATE_FILE_DEST` is set, then an Oracle managed data file is created in the indicated directory.
- If `DB_CREATE_FILE_DEST` is not set, then the data file location is operating system specific.

See Also:

" Managing Undo "

15.3.3.5 Specifying the Default Temporary Tablespace Temp File at Database Creation

The `TEMPFILE` subclause is optional for the `DEFAULT TEMPORARY TABLESPACE` clause and a file name is not required in the file specification.

If a file name is not supplied and the `DB_CREATE_FILE_DEST` parameter set, then an Oracle managed temp file is created in the `DB_CREATE_FILE_DEST` directory. If `DB_CREATE_FILE_DEST` is not set, then the `CREATE DATABASE` statement fails with a syntax error.

The `DEFAULT TEMPORARY TABLESPACE` clause itself is optional. If it is not specified, then no default temporary tablespace is created.

The default size for an Oracle managed temp file is 100 MB and the file is autoextensible with an unlimited maximum size.

15.3.3.6 CREATE DATABASE Statement Using Oracle Managed Files: Examples

Examples illustrate creating a database with the `CREATE DATABASE` statement when using the Oracle Managed Files feature.

CREATE DATABASE: Example 1

This example creates a database with the following Oracle Managed Files:

- A `SYSTEM` tablespace data file in directory `/u01/app/oracle/oradata` that is autoextensible up to an unlimited size.
- A `SYSAUX` tablespace data file in directory `/u01/app/oracle/oradata` that is autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two online log groups with two members of 100 MB each, one each in `/u02/oradata` and `/u03/oradata`.
- If automatic undo management mode is enabled (the default), then an undo tablespace data file in directory `/u01/app/oracle/oradata` that is 20 MB and autoextensible up to an unlimited size. An undo tablespace named `SYS_UNDOTS` is created.
- If no `CONTROL_FILES` initialization parameter is specified, then two control files, one each in `/u02/oradata` and `/u03/oradata`. The control file in `/u02/oradata` is the primary control file.

The following parameter settings relating to Oracle Managed Files, are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

The following statement is issued at the SQL prompt:

```
CREATE DATABASE sample;
```

To create the database with a locally managed `SYSTEM` tablespace, add the `EXTENT MANAGEMENT LOCAL` clause:

```
CREATE DATABASE sample EXTENT MANAGEMENT LOCAL;
```

Without this clause, the `SYSTEM` tablespace is dictionary managed. Oracle recommends that you create a locally managed `SYSTEM` tablespace.

CREATE DATABASE: Example 2

This example creates a database with the following Oracle Managed Files:

- A `SYSTEM` tablespace data file in directory `/u01/app/oracle/oradata` that is autoextensible up to an unlimited size.
- A `SYSAUX` tablespace data file in directory `/u01/app/oracle/oradata` that is autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two redo log files of 100 MB each in directory `/u01/app/oracle/oradata.` They are not multiplexed.
- An undo tablespace data file in directory `/u01/app/oracle/oradata` that is 20 MB and autoextensible up to an unlimited size. An undo tablespace named `SYS_UNDOTS` is created.
- A control file in `/u01/app/oracle/oradata`.

In this example, it is assumed that:

- No `DB_CREATE_ONLINE_LOG_DEST_`*n* initialization parameters are specified in the initialization parameter file.
- No `CONTROL_FILES` initialization parameter was specified in the initialization parameter file.
- Automatic undo management mode is enabled.

The following statements are issued at the SQL prompt:

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata';
CREATE DATABASE sample2 EXTENT MANAGEMENT LOCAL;
```

This database configuration is not recommended for a production database. The example illustrates how a very low-end database or simple test database can easily be created. To better protect this database from failures, at least one more control file should be created and the redo log should be multiplexed.

CREATE DATABASE: Example 3

In this example, the file size for the Oracle Managed Files for the default temporary tablespace and undo tablespace are specified. A database with the following Oracle Managed Files is created:

- A 400 MB `SYSTEM` tablespace data file in directory `/u01/app/oracle/oradata`. Because `SIZE` is specified, the file in not autoextensible.
- A 200 MB `SYSAUX` tablespace data file in directory `/u01/app/oracle/oradata.` Because `SIZE` is specified, the file in not autoextensible. The tablespace is locally managed with automatic segment-space management.
- Two redo log groups with two members of 100 MB each, one each in directories `/u02/oradata` and `/u03/oradata.`

- For the default temporary tablespace `dflt_ts`, a 10 MB temp file in directory `/u01/app/oracle/oradata`. Because `SIZE` is specified, the file in not autoextensible.
- For the undo tablespace `undo_ts`, a 100 MB data file in directory `/u01/app/oracle/oradata`. Because `SIZE` is specified, the file is not autoextensible.
- If no `CONTROL_FILES` initialization parameter was specified, then two control files, one each in directories `/u02/oradata` and `/u03/oradata`. The control file in `/u02/oradata` is the primary control file.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

The following statement is issued at the SQL prompt:

```
CREATE DATABASE sample3
EXTENT MANAGEMENT LOCAL
DATAFILE SIZE 400M
SYSAUX DATAFILE SIZE 200M
DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
UNDO TABLESPACE undo_ts DATAFILE SIZE 100M;
```

See Also:

Oracle Multitenant Administrator's Guide

15.3.4 Creating Data Files for Tablespaces Using Oracle Managed Files

Oracle Database can create data files for tablespaces using Oracle Managed Files when certain conditions are met.

- About Creating Data Files for Tablespaces Using Oracle Managed Files
  When certain conditions are met, the following SQL statements can create data files for tablespaces using Oracle Managed Files: `CREATE TABLESPACE`, `CREATE UNDO TABLESPACE`, and `ALTER TABLESPACE ... ADD DATAFILE`.
- CREATE TABLESPACE: Examples
  Examples illustrate creating tablespaces with Oracle Managed Files.
- CREATE UNDO TABLESPACE: Example
  An example illustrates creating an undo tablespace.
- ALTER TABLESPACE: Example
  An example illustrates adding an Oracle managed autoextensible data file to a tablespace.

15.3.4.1 About Creating Data Files for Tablespaces Using Oracle Managed Files

When certain conditions are met, the following SQL statements can create data files for tablespaces using Oracle Managed Files: `CREATE TABLESPACE`, `CREATE UNDO TABLESPACE`, and `ALTER TABLESPACE ... ADD DATAFILE`.

The following statements can create data files:

- `CREATE TABLESPACE`

- `CREATE UNDO TABLESPACE`
- `ALTER TABLESPACE` ... `ADD DATAFILE`

When creating a tablespace, either a permanent tablespace or an undo tablespace, the `DATAFILE` clause is optional. When you include the `DATAFILE` clause, the file name is optional. If the `DATAFILE` clause or file name is not provided, then the following rules apply:

- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, then an Oracle managed data file is created in the location specified by the parameter.
- If the `DB_CREATE_FILE_DEST` initialization parameter is not specified, then the statement creating the data file fails.

When you add a data file to a tablespace with the `ALTER TABLESPACE...ADD DATAFILE` statement the file name is optional. If the file name is not specified, then the same rules apply as discussed in the previous paragraph.

By default, an Oracle managed data file for a permanent tablespace is 100 MB and is autoextensible with an unlimited maximum size. However, if in your `DATAFILE` clause you override these defaults by specifying a `SIZE` value (and no `AUTOEXTEND` clause), then the data file is *not* autoextensible.

See Also:

- "Specifying the SYSTEM and SYSAUX Tablespace Data Files at Database Creation"
- "Specifying the Undo Tablespace Data File at Database Creation"
- " Managing Tablespaces"

15.3.4.2 CREATE TABLESPACE: Examples

Examples illustrate creating tablespaces with Oracle Managed Files.

See Also:

Oracle Database SQL Language Reference for a description of the `CREATE TABLESPACE` statement

CREATE TABLESPACE: Example 1

The following example sets the default location for data file creations to `/u01/oradata` and then creates a tablespace `tbs_1` with a data file in that location. The data file is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_1;
```

CREATE TABLESPACE: Example 2

This example creates a tablespace named `tbs_2` with a data file in the directory `/u01/oradata`. The data file initial size is 400 MB, and because the SIZE clause is specified, the data file is not autoextensible.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M;
```

CREATE TABLESPACE: Example 3

This example creates a tablespace named `tbs_3` with an autoextensible data file in the directory `/u01/oradata` with a maximum size of 800 MB and an initial size of 100 MB:

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_3 DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

CREATE TABLESPACE: Example 4

The following example sets the default location for data file creations to `/u01/oradata` and then creates a tablespace named `tbs_4` in that directory with two data files. Both data files have an initial size of 200 MB, and because a `SIZE` value is specified, they are not autoextensible

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_4 DATAFILE SIZE 200M, SIZE 200M;
```

15.3.4.3 CREATE UNDO TABLESPACE: Example

An example illustrates creating an undo tablespace.

The following example creates an undo tablespace named `undotbs_1` with a data file in the directory `/u01/oradata`. The data file for the undo tablespace is 100 MB and is autoextensible with an unlimited maximum size.

1. Set the following initialization parameter:

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   ```

2. Issue the following SQL statement:

   ```
   SQL> CREATE UNDO TABLESPACE undotbs_1;
   ```

See Also:

Oracle Database SQL Language Reference for a description of the `CREATE UNDO TABLESPACE` statement

15.3.4.4 ALTER TABLESPACE: Example

An example illustrates adding an Oracle managed autoextensible data file to a tablespace.

This example adds an Oracle managed autoextensible data file to the `tbs_1` tablespace. The data file has an initial size of 100 MB and a maximum size of 800 MB.

1. Set the following initialization parameter:

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   ```

2. Issue the following SQL statement:

   ```
   SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;
   ```

See Also:

Oracle Database SQL Language Reference for a description of the `ALTER TABLESPACE` statement

15.3.5 Creating Temp Files for Temporary Tablespaces Using Oracle Managed Files

Oracle Database can create temp files for temporary tablespaces using Oracle Managed Files when certain conditions are met.

- About Creating Temp Files for Temporary Tablespaces Using Oracle Managed Files
  When certain conditions are met, the following SQL statements can create temp files for tablespaces using Oracle Managed Files: `CREATE TEMPORARY TABLESPACE` and `ALTER TABLESPACE ... ADD TEMPFILE`.
- CREATE TEMPORARY TABLESPACE: Example
  An example illustrates creating a temporary tablespace.
- ALTER TABLESPACE... ADD TEMPFILE: Example
  An example illustrates adding a temp file to a temporary tablespace.

15.3.5.1 About Creating Temp Files for Temporary Tablespaces Using Oracle Managed Files

When certain conditions are met, the following SQL statements can create temp files for tablespaces using Oracle Managed Files: `CREATE TEMPORARY TABLESPACE` and `ALTER TABLESPACE ... ADD TEMPFILE`.

The following statements that create temp files are relevant to the discussion in this section:

- `CREATE TEMPORARY TABLESPACE`
- `ALTER TABLESPACE` ... `ADD TEMPFILE`

When creating a temporary tablespace the `TEMPFILE` clause is optional. If you include the `TEMPFILE` clause, then the file name is optional. If the `TEMPFILE` clause or file name is not provided, then the following rules apply:

- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, then an Oracle managed temp file is created in the location specified by the parameter.
- If the `DB_CREATE_FILE_DEST` initialization parameter is not specified, then the statement creating the temp file fails.

When you add a temp file to a tablespace with the `ALTER TABLESPACE...ADD TEMPFILE` statement the file name is optional. If the file name is not specified, then the same rules apply as discussed in the previous paragraph.

When overriding the default attributes of an Oracle managed file, if a `SIZE` value is specified but no `AUTOEXTEND` clause is specified, then the data file is *not* autoextensible.

See Also:

"Specifying the Default Temporary Tablespace Temp File at Database Creation"

15.3.5.2 CREATE TEMPORARY TABLESPACE: Example

An example illustrates creating a temporary tablespace.

The following example sets the default location for data file creations to `/u01/oradata` and then creates a tablespace named `temptbs_1` with a temp file in that location. The temp file is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

See Also:

Oracle Database SQL Language Reference for a description of the `CREATE TABLESPACE` statement

15.3.5.3 ALTER TABLESPACE... ADD TEMPFILE: Example

An example illustrates adding a temp file to a temporary tablespace.

The following example sets the default location for data file creations to `/u03/oradata` and then adds a temp file in the default location to a tablespace named `temptbs_1`. The temp file initial size is 100 MB. It is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

See Also:

Oracle Database SQL Language Reference for a description of the `ALTER TABLESPACE` statement

15.3.6 Creating Control Files Using Oracle Managed Files

Oracle Database can create control files using Oracle Managed Files when certain conditions are met.

- About Creating Control Files Using Oracle Managed Files
  When certain conditions are met, the CREATE CONTROLFILE SQL statements can create control files using Oracle Managed Files.
- CREATE CONTROLFILE Using NORESETLOGS Keyword: Example
  An example illustrates creating a control file using the CREATE CONTROLFILE statement with the NORESETLOGS keyword.
- CREATE CONTROLFILE Using RESETLOGS Keyword: Example
  An example illustrates creating a control file using the CREATE CONTROLFILE statement with the RESETLOGS keyword.

15.3.6.1 About Creating Control Files Using Oracle Managed Files

When certain conditions are met, the CREATE CONTROLFILE SQL statements can create control files using Oracle Managed Files.

When you issue the CREATE CONTROLFILE statement, a control file is created (or reused, if REUSE is specified) in the files specified by the CONTROL_FILES initialization parameter. If the CONTROL_FILES parameter is not set, then the control file is created in the default control file destinations. The default destination is determined according to the precedence documented in "Specifying Control Files at Database Creation".

If Oracle Database creates an Oracle managed control file, and there is a server parameter file, then the database creates a CONTROL_FILES initialization parameter for the server parameter file. If there is no server parameter file, then you must create a CONTROL_FILES initialization parameter manually and include it in the initialization parameter file.

If the data files in the database are Oracle Managed Files, then the database-generated file names for the files must be supplied in the DATAFILE clause of the statement.

If the redo log files are Oracle Managed Files, then the NORESETLOGS or RESETLOGS keyword determines what can be supplied in the LOGFILE clause:

- If the NORESETLOGS keyword is used, then the database-generated file names for the Oracle managed redo log files must be supplied in the LOGFILE clause.
- If the RESETLOGS keyword is used, then the redo log file names can be supplied as with the CREATE DATABASE statement. See "Specifying Redo Log Files at Database Creation".

The sections that follow contain examples of using the CREATE CONTROLFILE statement with Oracle Managed Files.

See Also:

- *Oracle Database SQL Language Reference* for a description of the CREATE CONTROLFILE statement
- "Specifying Control Files at Database Creation"

15.3.6.2 CREATE CONTROLFILE Using NORESETLOGS Keyword: Example

An example illustrates creating a control file using the `CREATE CONTROLFILE` statement with the `NORESETLOGS` keyword.

The following `CREATE CONTROLFILE` statement is generated by an `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` statement for a database with Oracle managed data files and redo log files:

```
CREATE CONTROLFILE
     DATABASE sample
     LOGFILE
       GROUP 1 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_1_o220rtt9_.log',
                '/u02/oradata/SAMPLE/onlinelog/o1_mf_1_v2o0b2i3_.log')
                 SIZE 100M,
       GROUP 2 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_2_p22056iw_.log',
                '/u02/oradata/SAMPLE/onlinelog/o1_mf_2_p02rcyg3_.log')
                 SIZE 100M
     NORESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_xu34ybm2_.dbf'
              SIZE 100M,
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_aawbmz51_.dbf'
              SIZE 100M,
              '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_apqbmz51_.dbf'
              SIZE 100M
     MAXLOGFILES 5
     MAXLOGHISTORY 100
     MAXDATAFILES 10
     MAXINSTANCES 2
     ARCHIVELOG;
```

15.3.6.3 CREATE CONTROLFILE Using RESETLOGS Keyword: Example

An example illustrates creating a control file using the `CREATE CONTROLFILE` statement with the `RESETLOGS` keyword.

The following is an example of a `CREATE CONTROLFILE` statement with the `RESETLOGS` option. Some combination of `DB_CREATE_FILE_DEST`, `DB_RECOVERY_FILE_DEST`, and `DB_CREATE_ONLINE_LOG_DEST_`*n* or must be set.

```
CREATE CONTROLFILE
     DATABASE sample
     RESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_aawbmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_axybmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_azzbmz51_.dbf'
     SIZE 100M
     MAXLOGFILES 5
     MAXLOGHISTORY 100
     MAXDATAFILES 10
     MAXINSTANCES 2
     ARCHIVELOG;
```

Later, you must issue the `ALTER DATABASE OPEN RESETLOGS` statement to re-create the redo log files. This is discussed in "Using the ALTER DATABASE OPEN RESETLOGS Statement". If the previous log files are Oracle Managed Files, then they are not deleted.

15.3.7 Creating Redo Log Files Using Oracle Managed Files

Redo log files are created at database creation time. They can also be created when you issue either of the following statements: `ALTER DATABASE ADD LOGFILE` and `ALTER DATABASE OPEN RESETLOGS`.

- Using the ALTER DATABASE ADD LOGFILE Statement
  The `ALTER DATABASE ADD LOGFILE` statement lets you later add a new group to your current redo log.
- Using the ALTER DATABASE OPEN RESETLOGS Statement
  If you previously created a control file specifying `RESETLOGS` and either did not specify file names or specified nonexistent file names, then the database creates redo log files for you when you issue the `ALTER DATABASE OPEN RESETLOGS` statement.

See Also:

Oracle Database SQL Language Reference for a description of the `ALTER DATABASE` statement

15.3.7.1 Using the ALTER DATABASE ADD LOGFILE Statement

The `ALTER DATABASE ADD LOGFILE` statement lets you later add a new group to your current redo log.

The file name in the `ADD LOGFILE` clause is optional if you are using Oracle Managed Files. If a file name is not provided, then a redo log file is created in the default log file destination. The default destination is determined according to the precedence documented in "Specifying Redo Log Files at Database Creation".

If a file name is not provided and you have not provided one of the initialization parameters required for creating Oracle Managed Files, then the statement returns an error.

The default size for an Oracle managed log file is 100 MB.

You continue to add and drop redo log file members by specifying complete file names.

See Also:

- "Specifying Redo Log Files at Database Creation"
- "About Creating Control Files Using Oracle Managed Files"

Adding New Redo Log Files: Example

The following example creates a log group with a member in `/u01/oradata` and another member in `/u02/oradata`. The size of each log file is 100 MB.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> ALTER DATABASE ADD LOGFILE;
```

15.3.7.2 Using the ALTER DATABASE OPEN RESETLOGS Statement

If you previously created a control file specifying `RESETLOGS` and either did not specify file names or specified nonexistent file names, then the database creates redo log files for you when you issue the `ALTER DATABASE OPEN RESETLOGS` statement.

The rules for determining the directories in which to store redo log files, when none are specified in the control file, are the same as those discussed in "Specifying Redo Log Files at Database Creation".

15.3.8 Creating Archived Logs Using Oracle Managed Files

Archived logs are created by a background process or by a SQL statement.

Archived logs are created in the `DB_RECOVERY_FILE_DEST` location when:

- The `ARC` or `LGWR` background process archives an online redo log or
- An `ALTER SYSTEM ARCHIVE LOG CURRENT` statement is issued.

For example, assume that the following parameter settings are included in the initialization parameter file:

```
DB_RECOVERY_FILE_DEST_SIZE = 20G
DB_RECOVERY_FILE_DEST      = '/u01/oradata'
LOG_ARCHIVE_DEST_1         = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

15.4 Operation of Oracle Managed Files

The file names of Oracle Managed Files are accepted in SQL statements wherever a file name is used to identify an existing file.

These file names, like other file names, are stored in the control file and, if using Recovery Manager (RMAN) for backup and recovery, in the RMAN catalog. They are visible in all of the usual fixed and dynamic performance views that are available for monitoring data files and temp files (for example, `V$DATAFILE` or `DBA_DATA_FILES`).

The following are some examples of statements using database-generated file names:

```
SQL> ALTER DATABASE
  2> RENAME FILE '/u01/oradata/mydb/datafile/o1_mf_tbs01_ziw3bopb_.dbf'
  3> TO '/u01/oradata/mydb/tbs0101.dbf';

SQL> ALTER DATABASE
  2> DROP LOGFILE '/u01/oradata/mydb/onlinelog/o1_mf_1_wo94n2xi_.log';

SQL> ALTER TABLE emp
  2> ALLOCATE EXTENT
  3> (DATAFILE '/u01/oradata/mydb/datafile/o1_mf_tbs1_2ixfh90q_.dbf');
```

You can backup and restore Oracle managed data files, temp files, and control files as you would corresponding non Oracle Managed Files. Using database-generated file names does not impact the use of logical backup files such as export files. This is particularly important for tablespace point-in-time recovery (TSPITR) and transportable tablespace export files.

There are some cases where Oracle Managed Files behave differently, including operations that drop files or rename file, and operations involving standby databases.

- Dropping Data Files and Temp Files
  Unlike files that are not managed by the database, when an Oracle managed data file or temp file is dropped, the file name is removed from the control file and the file is automatically deleted from the file system.
- Dropping Redo Log Files
  When an Oracle managed redo log file is dropped, its Oracle Managed Files are deleted. You specify the group or members to be dropped.
- Renaming Files
  With Oracle Managed Files, SQL statements that rename files do not actually rename the files on the operating system, but rather, the names in the control file are changed.
- Managing Standby Databases
  The data files, control files, and redo log files in a standby database can be managed by the database. This is independent of whether Oracle Managed Files are used on the primary database.

15.4.1 Dropping Data Files and Temp Files

Unlike files that are not managed by the database, when an Oracle managed data file or temp file is dropped, the file name is removed from the control file and the file is automatically deleted from the file system.

The statements that delete Oracle Managed Files when they are dropped are:

- `DROP TABLESPACE`
- `ALTER DATABASE TEMPFILE ... DROP`

You can also use these statements, which always delete files, Oracle managed or not:

- `ALTER TABLESPACE ... DROP DATAFILE`
- `ALTER TABLESPACE ... DROP TEMPFILE`

15.4.2 Dropping Redo Log Files

When an Oracle managed redo log file is dropped, its Oracle Managed Files are deleted. You specify the group or members to be dropped.

The following statements drop and delete redo log files:

- `ALTER DATABASE DROP LOGFILE`
- `ALTER DATABASE DROP LOGFILE MEMBER`

15.4.3 Renaming Files

With Oracle Managed Files, SQL statements that rename files do not actually rename the files on the operating system, but rather, the names in the control file are changed.

The following statements are used to rename files:

- `ALTER DATABASE RENAME FILE`

- `ALTER TABLESPACE ... RENAME DATAFILE`

You must specify each file name using the conventions for file names on your operating system when you issue this statement.

> **Note:**
>
> If the old file is an Oracle managed file and it exists, then it is deleted.

15.4.4 Managing Standby Databases

The data files, control files, and redo log files in a standby database can be managed by the database. This is independent of whether Oracle Managed Files are used on the primary database.

When recovery of a standby database encounters redo for the creation of a data file, if the data file is an Oracle managed file, then the recovery process creates an empty file in the local default file system location. This allows the redo for the new file to be applied immediately without any human intervention.

When recovery of a standby database encounters redo for the deletion of a tablespace, it deletes any Oracle managed data files in the local file system. Note that this is independent of the `INCLUDING DATAFILES` option issued at the primary database.

15.5 Scenarios for Using Oracle Managed Files

Scenarios illustrate how to use Oracle Managed Files.

- Scenario 1: Create and Manage a Database with Multiplexed Redo Logs
  An example illustrates creating and managing a database with multiplexed redo logs.
- Scenario 2: Create and Manage a Database with Database and Fast Recovery Areas
  An example illustrates creating and managing a database with both database and fast recovery areas.
- Scenario 3: Adding Oracle Managed Files to an Existing Database
  An example illustrates adding Oracle Managed Files to an existing database.

15.5.1 Scenario 1: Create and Manage a Database with Multiplexed Redo Logs

An example illustrates creating and managing a database with multiplexed redo logs.

In this scenario, a DBA creates a database where the data files and redo log files are created in separate directories. The redo log files and control files are multiplexed. The database uses an undo tablespace, and has a default temporary tablespace. The following are tasks involved with creating and maintaining this database.

1. Setting the initialization parameters

   The DBA includes three generic file creation defaults in the initialization parameter file before creating the database. Automatic undo management mode (the default) is also specified.

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
   ```

```
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
UNDO_MANAGEMENT = AUTO
```

The `DB_CREATE_FILE_DEST` parameter sets the default file system directory for the data files and temp files.

The `DB_CREATE_ONLINE_LOG_DEST_1` and `DB_CREATE_ONLINE_LOG_DEST_2` parameters set the default file system directories for redo log file and control file creation. Each redo log file and control file is multiplexed across the two directories.

2. Creating a database

   Once the initialization parameters are set, the database can be created by using this statement:

   ```
   SQL> CREATE DATABASE sample
   2>   DEFAULT TEMPORARY TABLESPACE dflttmp;
   ```

   Because a `DATAFILE` clause is not present and the `DB_CREATE_FILE_DEST` initialization parameter is set, the `SYSTEM` tablespace data file is created in the default file system (`/u01/oradata` in this scenario). The file name is uniquely generated by the database. The file is autoextensible with an initial size of 100 MB and an unlimited maximum size. The file is an Oracle managed file. A similar data file is created for the `SYSAUX` tablespace.

   Because a `LOGFILE` clause is not present, two redo log groups are created. Each log group has two members, with one member in the `DB_CREATE_ONLINE_LOG_DEST_1` location and the other member in the `DB_CREATE_ONLINE_LOG_DEST_2` location. The file names are uniquely generated by the database. The log files are created with a size of 100 MB. The log file members are Oracle Managed Files.

   Similarly, because the `CONTROL_FILES` initialization parameter is not present, and two `DB_CREATE_ONLINE_LOG_DEST_`*n* initialization parameters are specified, two control files are created. The control file located in the `DB_CREATE_ONLINE_LOG_DEST_1` location is the primary control file; the control file located in the `DB_CREATE_ONLINE_LOG_DEST_2` location is a multiplexed copy. The file names are uniquely generated by the database. They are Oracle Managed Files. Assuming there is a server parameter file, a `CONTROL_FILES` initialization parameter is generated.

   Automatic undo management mode is specified, but because an undo tablespace is not specified and the `DB_CREATE_FILE_DEST` initialization parameter is set, a default undo tablespace named `UNDOTBS` is created in the directory specified by `DB_CREATE_FILE_DEST`. The data file is a 20 MB data file that is autoextensible. It is an Oracle managed file.

   Lastly, a default temporary tablespace named `dflttmp` is specified. Because `DB_CREATE_FILE_DEST` is included in the parameter file, the temp file for `dflttmp` is created in the directory specified by that parameter. The temp file is 100 MB and is autoextensible with an unlimited maximum size. It is an Oracle managed file.

   The resultant file tree, with generated file names, is as follows:

   ```
   /u01
        /oradata
             /SAMPLE
                  /datafile
                       /o1_mf_system_cmr7t30p_.dbf
                       /o1_mf_sysaux_cmr7t88p_.dbf
                       /o1_mf_sys_undo_2ixfh90q_.dbf
                       /o1_mf_dflttmp_157se6ff_.tmp
   /u02
        /oradata
             /SAMPLE
                  /onlinelog
   ```

```
                    /o1_mf_1_0orrm31z_.log
                    /o1_mf_2_2xyz16am_.log
               /controlfile
                    /o1_mf_cmr7t30p_.ctl
/u03
     /oradata
          /SAMPLE
               /onlinelog
                    /o1_mf_1_ixfvm8w9_.log
                    /o1_mf_2_q89tmp28_.log
               /controlfile
                    /o1_mf_x1sr8t36_.ctl
```

The internally generated file names can be seen when selecting from the usual views. For example:

```
SQL> SELECT NAME FROM V$DATAFILE;

NAME
----------------------------------------------------
/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_2ixfh90q_.dbf

3 rows selected
```

3. Managing control files

   The control file was created when generating the database, and a `CONTROL_FILES` initialization parameter was added to the parameter file. If needed, then the DBA can re-create the control file or build a new one for the database using the `CREATE CONTROLFILE` statement.

   The correct Oracle managed file names must be used in the `DATAFILE` and `LOGFILE` clauses. The `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` statement generates a script with the correct file names. Alternatively, the file names can be found by selecting from the `V$DATAFILE`, `V$TEMPFILE`, and `V$LOGFILE` views. The following example re-creates the control file for the sample database:

```
CREATE CONTROLFILE REUSE
  DATABASE sample
  LOGFILE
    GROUP 1('/u02/oradata/SAMPLE/onlinelog/o1_mf_1_0orrm31z_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_1_ixfvm8w9_.log'),
    GROUP 2('/u02/oradata/SAMPLE/onlinelog/o1_mf_2_2xyz16am_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_2_q89tmp28_.log')
  NORESETLOGS
  DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_2ixfh90q_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_dflttmp_157se6ff_.tmp'
  MAXLOGFILES 5
  MAXLOGHISTORY 100
  MAXDATAFILES 10
  MAXINSTANCES 2
  ARCHIVELOG;
```

   The control file created by this statement is located as specified by the `CONTROL_FILES` initialization parameter that was generated when the database was created. The `REUSE` clause causes any existing files to be overwritten.

4. Managing the redo log

To create a new group of redo log files, the DBA can use the `ALTER DATABASE ADD LOGFILE` statement. The following statement adds a log file with a member in the `DB_CREATE_ONLINE_LOG_DEST_1` location and a member in the `DB_CREATE_ONLINE_LOG_DEST_2` location. These files are Oracle Managed Files.

```
SQL> ALTER DATABASE ADD LOGFILE;
```

Log file members continue to be added and dropped by specifying complete file names.

The `GROUP` clause can be used to drop a log group. In the following example the operating system file associated with each Oracle managed log file member is automatically deleted.

```
SQL> ALTER DATABASE DROP LOGFILE GROUP 3;
```

5. Managing tablespaces

   The default storage for all data files for future tablespace creations in the `sample` database is the location specified by the `DB_CREATE_FILE_DEST` initialization parameter (`/u01/oradata` in this scenario). Any data files for which no file name is specified, are created in the file system specified by the initialization parameter `DB_CREATE_FILE_DEST`. For example:

   ```
   SQL> CREATE TABLESPACE tbs_1;
   ```

   The preceding statement creates a tablespace whose storage is in `/u01/oradata`. A data file is created with an initial of 100 MB and it is autoextensible with an unlimited maximum size. The data file is an Oracle managed file.

   When the tablespace is dropped, the Oracle Managed Files for the tablespace are automatically removed. The following statement drops the tablespace and all the Oracle Managed Files used for its storage:

   ```
   SQL> DROP TABLESPACE tbs_1;
   ```

   Once the first data file is full, the database does not automatically create a new data file. More space can be added to the tablespace by adding another Oracle managed data file. The following statement adds another data file in the location specified by `DB_CREATE_FILE_DEST`:

   ```
   SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE;
   ```

   The default file system can be changed by changing the initialization parameter. This does not change any existing data files. It only affects future creations. This can be done dynamically using the following statement:

   ```
   SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u04/oradata';
   ```

6. Archiving redo information

   Archiving of redo log files is no different for Oracle Managed Files, than it is for unmanaged files. A file system location for the archived redo log files can be specified using the `LOG_ARCHIVE_DEST_`*`n`* initialization parameters. The file names are formed based on the `LOG_ARCHIVE_FORMAT` parameter or its default. The archived logs are not Oracle Managed Files.

7. Backup, restore, and recover

   Since an Oracle managed file is compatible with standard operating system files, you can use operating system utilities to backup or restore Oracle Managed Files. All existing methods for backing up, restoring, and recovering the database work for Oracle Managed Files.

15.5.2 Scenario 2: Create and Manage a Database with Database and Fast Recovery Areas

An example illustrates creating and managing a database with both database and fast recovery areas.

In this scenario, a DBA creates a database where the control files and redo log files are multiplexed. Archived logs and RMAN backups are created in the Fast Recovery Area. The following tasks are involved in creating and maintaining this database:

1. Setting the initialization parameters

   The DBA includes the following generic file creation defaults:

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_RECOVERY_FILE_DEST_SIZE = 10G
   DB_RECOVERY_FILE_DEST = '/u02/oradata'
   LOG_ARCHIVE_DEST_1 = 'LOCATION = USE_DB_RECOVERY_FILE_DEST'
   ```

   The `DB_CREATE_FILE_DEST` parameter sets the default file system directory for data files, temp files, control files, and redo logs.

   The `DB_RECOVERY_FILE_DEST` parameter sets the default file system directory for control files, redo logs, and RMAN backups.

   The `LOG_ARCHIVE_DEST_1` configuration '`LOCATION=USE_DB_RECOVERY_FILE_DEST`' redirects archived logs to the `DB_RECOVERY_FILE_DEST` location.

   The `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` parameters set the default directory for log file and control file creation. Each redo log and control file is multiplexed across the two directories.

2. Creating a database
3. Managing control files
4. Managing the redo log
5. Managing tablespaces

   Tasks 2, 3, 4, and 5 are the same as in Scenario 1, except that the control files and redo logs are multiplexed across the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` locations.

6. Archiving redo log information

   Archiving online logs is no different for Oracle Managed Files than it is for unmanaged files. The archived logs are created in `DB_RECOVERY_FILE_DEST` and are Oracle Managed Files.

7. Backup, restore, and recover

   An Oracle managed file is compatible with standard operating system files, so you can use operating system utilities to backup or restore Oracle Managed Files. All existing methods for backing up, restoring, and recovering the database work for Oracle Managed Files. When no format option is specified, all disk backups by RMAN are created in the `DB_RECOVERY_FILE_DEST` location. The backups are Oracle Managed Files.

15.5.3 Scenario 3: Adding Oracle Managed Files to an Existing Database

An example illustrates adding Oracle Managed Files to an existing database.

Assume in this case that an existing database does not have any Oracle Managed Files, but the DBA would like to create new tablespaces with Oracle Managed Files and locate them in directory `/u03/oradata`.

1. Setting the initialization parameters

   To allow automatic data file creation, set the `DB_CREATE_FILE_DEST` initialization parameter to the file system directory in which to create the data files. This can be done dynamically as follows:

   ```
   SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
   ```

2. Creating tablespaces

   Once `DB_CREATE_FILE_DEST` is set, the `DATAFILE` clause can be omitted from a `CREATE TABLESPACE` statement. The data file is created in the location specified by `DB_CREATE_FILE_DEST` by default. For example:

   ```
   SQL> CREATE TABLESPACE tbs_2;
   ```

   When the `tbs_2` tablespace is dropped, its data files are automatically deleted.

16
Using Persistent Memory Database

Mapping the database directly into persistent memory (PMEM) provides significant performance enhancements.

16.1 About Persistent Memory Database

The Persistent Memory Database feature includes directly mapped buffer cache and Persistent Memory Filestore (PMEM Filestore).

16.1.1 What Is Persistent Memory Database?

The Persistent Memory Database feature enables you to place database files in non-volatile memory. This feature supports a single-instance Oracle Database instance on PMEM Filestore.

16.1.2 What Is Oracle Persistent Memory Filestore?

PMEM Filestore is a pointer-switching PMEM file system that supports atomic updates of Oracle Database data blocks. PMEM Filestore is the underlying file store used for a Persistent Memory database. PMEM Filestore provides the external interface for mapping and accessing an Oracle database directly in persistent memory.

Managing an Oracle database on PMEM Filestore is similar to managing an Oracle database on a native file system. PMEM Filestore implements the Filesystem in Userspace (FUSE) protocol, enabling Oracle DBAs to perform normal file-level maintenance. FUSE allows non-privileged (non-root) users, such as the Oracle Database software owner, to create and manage filesystems as well as the directories and files contained within them.

A typical file system uses raw storage as its backing store, while PMEM Filestore gets storage from a native operating system file in a PMEM DAX file system. This file is called the backing file and is visible as a file in the operating system. PMEM Filestore subdivides the storage within the backing file and presents it as a local file system.

After a PMEM Filestore is created and mounted, you will see a local file system under the user-specified mount point. This local file system supports directories and common operating system commands such as `ls` and `cp`. This local file system is the PMEM Filestore and can be used to store Oracle Database files. Note that the PMEM Filestore is only visible when the Oracle Database instance is started.

You can use PMEM Filestore for database datafiles and control files. For performance reasons, Oracle recommends that you store redo log files as independent files in a DAX-aware filesystem such as EXT4/XFS. Administrative files such as trace files and audit files cannot be stored in PMEM Filestore. The server parameter file (SPFILE) cannot be stored in PMEM Filestore because PMEM Filestore configuration parameters can be specified in the SPFILE.

16.1.3 What Is Directly Mapped Buffer Cache?

Directly mapped buffer cache is a mechanism in Oracle Database to directly read data on persistent memory, bypassing the traditional DRAM buffer cache. This mechanism also tracks data access and automatically brings frequently read data, and data for updating, from PMEM to DRAM buffer cache for faster access. The directly mapped buffer cache mechanism is automatically invoked when a datafile is placed in a PMEM filestore.

16.1.4 Benefits of Using Persistent Memory Database

With Oracle Persistent Memory Database, database files can be placed in persistent memory which enables Oracle to take advantage of performance enhancements inherent to this technology. Consider the following benefits of using Persistent Memory Database:

- The PMEM Filestore provides atomic writes to full Oracle database blocks. This eliminates the need for media recovery due to partially written blocks after a power outage.
- Persistent Memory Database performs I/O to PMEM storage via memory copy. This is much faster than performing I/O via traditional operating system calls.
- Database queries save the traditional read from storage and memory copy into buffer cache because the Oracle Database server accesses data directly from persistent memory.

16.2 Setting Initialization Parameters for Persistent Memory Database

You can set the `PMEM_FILESTORE` initialization parameter to specify a PMEM Filestore that the Oracle Database instance will mount automatically when it is started.

- Persistent Memory Database Initialization Parameters

16.2.1 Persistent Memory Database Initialization Parameters

The `PMEM_FILESTORE` initialization parameter specifies a PMEM Filestore that the Oracle Database instance will automatically mount when it is started. The parameter is set to an ordered pair of strings. The first string in the parameter value list is the directory where PMEM Filestore is mounted. The second string is the backing file.

A PMEM Filestore backing file is visible as a file in the operating system file system hierarchy. While a typical file system uses raw storage as its backing store, PMEM Filestore gets storage from a native operating system file in a PMEM DAX file system. On Linux, the PMEM Filestore backing file should be in an XFS or ext4 file system mounted using the `-o dax` option.

16.3 Creating a PMEM Filestore for an Oracle Database

To use the Persistent Memory Database feature, you create a PMEM filestore for Oracle Database files.

- Creating a PMEM Filestore Before Creating the Database
- Creating an Oracle Database in the PMEM Filestore
- Migrating an Oracle Database to a PMEM Filestore

16.3.1 Creating a PMEM Filestore Before Creating the Database

Perform the following steps to create a PMEM filestore for an Oracle database:

1. Start the Oracle Database instance in `NOMOUNT` mode.
2. Execute the `CREATE PMEM FILESTORE` command to create the PMEM filestore and provide:
   - A mount point for the file store. The final subdirectory name must be the same as the PMEM filestore name.
   - A backing file from a native XFS or ext4 file system mounted in DAX mode.The backing file is used by the PMEM filestore to keep all the files created in the filestore.
   - A block size, which will typically be the same as the default block size of the database datafiles.
     As an example:

     ```
     CREATE PMEM FILESTORE db1_pmemfs
     MOUNTPOINT '/u1/db/db1_pmemfs'
     BACKINGFILE '/u1/db_storage/db1'
     SIZE 2T
     BLOCKSIZE 8K
     AUTOEXTEND ON NEXT 10G
     MAXSIZE 3T;
     ```

     The PMEM filestore is automatically mounted after it is created. The PMEM filestore will appear under the specified mount point as if it is a native file system.
3. Configure the PMEM filestore to be mounted during instance startup.
   If you used an SPFILE to start the database instance, the server adds the `PMEM_FILESTORE` initialization parameter to the SPFILE. This parameter causes the PMEM filestore to be automatically mounted when the database instance is started. The SPFILE will contain this entry following the execution of the `CREATE PMEM FILESTORE` command: `PMEM_FILESTORE=('/u1/db/db1_pmemfs', '/u1/db_storage/db1')`. The first string in the parameter value list is the directory where the PMEM filestore is mounted. The second string is the backing file.

   If you did not use an SPFILE to start the database instance, you must manually add the `PMEM_FILESTORE` initialization parameter so that the PMEM filestore is mounted during instance startup. Or you must manually mount the PMEM filestore using the `ALTER PMEM FILESTORE … MOUNT` command.

16.3.2 Creating an Oracle Database in the PMEM Filestore

After creating the PMEM filestore, you can use it as if it is a native file system to create an Oracle Database. You can create the database under the mount point you specified when you created the PMEM filestore. See *Oracle Multitenant Administrator's Guide* for detailed information.

16.3.3 Migrating an Oracle Database to a PMEM Filestore

After creating the PMEM filestore, perform the following steps to migrate an existing Oracle Database or select tablespaces to the PMEM filestore:

1. Copy the database files by using the RMAN `RESTORE` command or operating system commands.
2. Optionally, change the block size of the redo log files by creating new online redo log files and then dropping the existing redo log files.

You can also use the `ALTER DATABASE` command with the `MOVE DATAFILE` clause to move an online datafile to the PMEM filestore.

16.4 Managing a PMEM Filestore

You can view information about a PMEM filestore and perform various operations on the PMEM filestore, including mounting and dismounting, changing the attributes, and dropping the filestore.

- Viewing Information About a PMEM Filestore
  `V$PMEM_FILESTORE` provides information about the PMEM filestore.
- Mounting a PMEM Filestore
  If you did not set the `PMEM_FILESTORE` initialization parameter to automatically mount the PMEM filestore when the database instance is started, you must mount the PMEM filestore manually.
- Dismounting a PMEM Filestore
  If you want to alter the attributes of the PMEM filestore, you must first dismount the filestore.
- Changing the Attributes of a PMEM Filestore
  You can change the attributes of a PMEM filestore, including the mount point, the backing file, and the size.
- Dropping a PMEM Filestore
  You can drop a PMEM filestore whether it is empty or not.

16.4.1 Viewing Information About a PMEM Filestore

`V$PMEM_FILESTORE` provides information about the PMEM filestore.

You can query `V$PMEM_FILESTORE` to view information about the PMEM filestore including:

- Directory path for the mount point of the PMEM filestore
- File path for the backing file of the PMEM filestore
- Block size of the PMEM filestore (in bytes)

- Current size of the PMEM filestore (in bytes)
- Whether it is autoextensible and details about the autoextensible configuration
- Space usage (free space and used space)

16.4.2 Mounting a PMEM Filestore

If you did not set the `PMEM_FILESTORE` initialization parameter to automatically mount the PMEM filestore when the database instance is started, you must mount the PMEM filestore manually.

The following statement is used to mount the PMEM filestore:

- `ALTER PMEM FILESTORE` *`filestore_name`* `MOUNT`

If the initialization parameter file does not include the `PMEM_FILESTORE` parameter, you must include the `MOUNTPOINT` and `BACKINGFILE` clauses, along with the `FORCE` keyword when you execute this command.

You can also include the `MOUNTPOINT` and `BACKINGFILE` clauses with the `FORCE` keyword to specify different file paths than were specified in the intialization parameter file. If you used a server parameter file (SPFILE) to start the database instance, it will be updated with the file paths specified in the `ALTER PMEM FILESTORE MOUNT` command.

You must be connected to the root container (`CDB$ROOT`) as a user with the `SYSDBA` privilege to execute this command.

16.4.3 Dismounting a PMEM Filestore

If you want to alter the attributes of the PMEM filestore, you must first dismount the filestore.

The following statement is used to dismount the PMEM filestore:

- `ALTER PMEM FILESTORE` *`filestore_name`* `DISMOUNT`

You must be connected to the root container (`CDB$ROOT`) as a user with the `SYSDBA` privilege to execute this command.

16.4.4 Changing the Attributes of a PMEM Filestore

You can change the attributes of a PMEM filestore, including the mount point, the backing file, and the size.

The following statement is used to change the attributes of the PMEM filestore:

- `ALTER PMEM FILESTORE`

If you want to change the mount point or backing file, you must first dismount the filestore. See Dismounting a PMEM Filestore.

You cannot change the block size of the filestore.

You must be connected to the root container (`CDB$ROOT`) as a user with the `SYSDBA` privilege to execute this command.

16.4.5 Dropping a PMEM Filestore

You can drop a PMEM filestore whether it is empty or not.

The following statement is used to drop the PMEM filestore:

- `DROP PMEM FILESTORE`

Specify `INCLUDING CONTENTS` to remove all of the files in the PMEM filestore. To drop the filestore immediately, use the `FORCE` keyword with `INCLUDING CONTENTS`. Specify `EXCLUDING CONTENTS` so that the PMEM filestore will be dropped only when it is empty.

If the database instance was started with an SPFILE, the SPFILE will be updated when `DROP PMEM FILESTORE` is executed.

Part III

Schema Objects

You can create and manage schema objects in Oracle Database.

17

Managing Schema Objects

You can create and manage several types of schema objects with Oracle Database.

- About Common and Local Objects
  A **common object** is defined in either the CDB root or an application root, and can be referenced using metadata links or object links. A local object is every object that is not a common object.
- About the Container for Schema Objects
  Schema objects are created in the current container.
- Creating Multiple Tables and Views in a Single Operation
  You can create several tables and views and grant privileges in one operation using the `CREATE SCHEMA` statement. If an individual table, view or grant fails, the entire statement is rolled back. None of the objects are created, nor are the privileges granted.
- Analyzing Tables, Indexes, and Clusters
  You can collecting statistics on schema objects, analyze the statistics, and validate the schema objects.
- Truncating Tables and Clusters
  You can delete all rows of a table or all rows in a group of clustered tables so that the table (or cluster) still exists, but is completely empty. For example, consider a table that contains monthly data, and at the end of each month, you must empty it (delete all rows) after archiving its data.
- Enabling and Disabling Triggers
  Database triggers are procedures that are stored in the database and activated ("fired") when specific conditions occur, such as adding a row to a table.
- Managing Integrity Constraints
  Integrity constraints are rules that restrict the values for one or more columns in a table. Constraint clauses can appear in either `CREATE TABLE` or `ALTER TABLE` statements, and identify the column or columns affected by the constraint and identify the conditions of the constraint.
- Renaming Schema Objects
  There are several ways to rename an object.
- Managing Object Dependencies
  Oracle Database provides an automatic mechanism to ensure that a dependent object is always up to date with respect to its referenced objects. You can also manually recompile invalid object.
- Managing Object Name Resolution
  Object names referenced in SQL statements can consist of several pieces, separated by periods. Oracle Database performs specific actions to resolve an object name.
- Switching to a Different Schema
  Use an `ALTER SESSION` SQL statement to switch to a different schema.
- Managing Editions
  Application developers who are upgrading their applications using edition-based redefinition may ask you to perform edition-related tasks that require DBA privileges.

- Displaying Information About Schema Objects
  Oracle Database provides a PL/SQL package that enables you to determine the DDL that created an object and data dictionary views that you can use to display information about schema objects.

17.1 About Common and Local Objects

A **common object** is defined in either the CDB root or an application root, and can be referenced using metadata links or object links. A local object is every object that is not a common object.

Database-supplied common objects are defined in `CDB$ROOT` and cannot be changed. Oracle Database does not support creation of common objects in `CDB$ROOT`.

You can create most schema objects—such as tables, views, PL/SQL and Java program units, sequences, and so on—as common objects in an application root. If the object exists in an application root, then it is called an **application common object**.

A local user can own a common object. Also, a common user can own a local object, but only when the object is not data-linked or metadata-linked, and is also neither a metadata link nor a data link.

See Also:

Oracle Database Security Guide to learn more about privilege management for common objects

17.2 About the Container for Schema Objects

Schema objects are created in the current container.

Before you create schema objects, ensure that you are in the container that store these schema objects.

To create a schema object in a pluggable database (PDB), connect to the PDB as a common user or local user with the required privileges. Then, run the required SQL*Plus command.

17.3 Creating Multiple Tables and Views in a Single Operation

You can create several tables and views and grant privileges in one operation using the `CREATE SCHEMA` statement. If an individual table, view or grant fails, the entire statement is rolled back. None of the objects are created, nor are the privileges granted.

Specifically, the `CREATE SCHEMA` statement can include *only* `CREATE TABLE`, `CREATE VIEW`, and `GRANT` statements. You must have the privileges necessary to issue the included statements. You are not actually creating a schema, that is done when the user is created with a `CREATE USER` statement. Rather, you are populating the schema.

The following statement creates two tables and a view that joins data from the two tables:

```
CREATE SCHEMA AUTHORIZATION scott
    CREATE TABLE dept (
        deptno NUMBER(3,0) PRIMARY KEY,
        dname VARCHAR2(15),
        loc VARCHAR2(25))
    CREATE TABLE emp (
        empno NUMBER(5,0) PRIMARY KEY,
        ename VARCHAR2(15) NOT NULL,
        job VARCHAR2(10),
        mgr NUMBER(5,0),
        hiredate DATE DEFAULT (sysdate),
        sal NUMBER(7,2),
        comm NUMBER(7,2),
        deptno NUMBER(3,0) NOT NULL
        CONSTRAINT dept_fkey REFERENCES dept)
   CREATE VIEW sales_staff AS
        SELECT empno, ename, sal, comm
        FROM emp
        WHERE deptno = 30
        WITH CHECK OPTION CONSTRAINT sales_staff_cnst
        GRANT SELECT ON sales_staff TO human_resources;
```

The `CREATE SCHEMA` statement does not support Oracle Database extensions to the ANSI `CREATE TABLE` and `CREATE VIEW` statements, including the `STORAGE` clause.

See Also:

Oracle Database SQL Language Reference for syntax and other information about the `CREATE SCHEMA` statement

17.4 Analyzing Tables, Indexes, and Clusters

You can collecting statistics on schema objects, analyze the statistics, and validate the schema objects.

- About Analyzing Tables, Indexes, and Clusters
  You can collect information about schema objects and analyze that information.
- Using DBMS_STATS to Collect Table and Index Statistics
  You can use the `DBMS_STATS` package or the `ANALYZE` statement to gather statistics about the physical storage characteristics of a table, index, or cluster. These statistics are stored in the data dictionary and can be used by the optimizer to choose the most efficient execution plan for SQL statements accessing analyzed objects.
- Validating Tables, Indexes, Clusters, and Materialized Views
  To verify the integrity of the structure of a table, index, cluster, or materialized view, use the `ANALYZE` statement with the `VALIDATE STRUCTURE` option.
- Cross Validation of a Table and an Index with a Query
  In some cases, an `ANALYZE` statement takes an inordinate amount of time to complete. In these cases, you can use a SQL query to validate an index.
- Listing Chained Rows of Tables and Clusters
  You can look at the chained and migrated rows of a table or cluster using the `ANALYZE` statement with the `LIST CHAINED ROWS` clause. The results of this statement are stored in a specified table created explicitly to accept the information returned by the `LIST CHAINED ROWS` clause. These results are useful in determining whether you have enough room for updates to rows.

17.4.1 About Analyzing Tables, Indexes, and Clusters

You can collect information about schema objects and analyze that information.

You analyze a schema object (table, index, or cluster) to:

- Collect and manage statistics for it
- Verify the validity of its storage format
- Identify migrated and chained rows of a table or cluster

Note:

Do not use the `COMPUTE` and `ESTIMATE` clauses of `ANALYZE` to collect optimizer statistics. These clauses have been deprecated. Instead, use the `DBMS_STATS` package, which lets you collect statistics in parallel, collect global statistics for partitioned objects, and fine tune your statistics collection in other ways. The cost-based optimizer, which depends upon statistics, will eventually use only statistics that have been collected by `DBMS_STATS`. See *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_STATS` package.

You must use the `ANALYZE` statement (rather than `DBMS_STATS`) for statistics collection not related to the cost-based optimizer, such as:

- To use the `VALIDATE` or `LIST CHAINED ROWS` clauses
- To collect information on freelist blocks

17.4.2 Using DBMS_STATS to Collect Table and Index Statistics

You can use the `DBMS_STATS` package or the `ANALYZE` statement to gather statistics about the physical storage characteristics of a table, index, or cluster. These statistics are stored in the data dictionary and can be used by the optimizer to choose the most efficient execution plan for SQL statements accessing analyzed objects.

Oracle recommends using the more versatile `DBMS_STATS` package for gathering optimizer statistics, but you must use the `ANALYZE` statement to collect statistics unrelated to the optimizer, such as empty blocks, average space, and so forth.

The `DBMS_STATS` package allows both the gathering of statistics, including utilizing parallel execution, and the external manipulation of statistics. Statistics can be stored in tables outside of the data dictionary, where they can be manipulated without affecting the optimizer. Statistics can be copied between databases or backup copies can be made.

The following `DBMS_STATS` procedures enable the gathering of optimizer statistics:

- `GATHER_INDEX_STATS`
- `GATHER_TABLE_STATS`
- `GATHER_SCHEMA_STATS`
- `GATHER_DATABASE_STATS`

See Also:

- *Oracle Database SQL Tuning Guide* for information about using `DBMS_STATS` to gather statistics for the optimizer
- *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_STATS` package

17.4.3 Validating Tables, Indexes, Clusters, and Materialized Views

To verify the integrity of the structure of a table, index, cluster, or materialized view, use the `ANALYZE` statement with the `VALIDATE STRUCTURE` option.

If the structure is valid, then no error is returned. However, if the structure is corrupt, then you receive an error message.

For example, in rare cases such as hardware or other system failures, an index can become corrupted and not perform correctly. When validating the index, you can confirm that every entry in the index points to the correct row of the associated table. If the index is corrupt, then you can drop and re-create it.

If a table, index, or cluster is corrupt, then drop it and re-create it. If a materialized view is corrupt, then perform a complete refresh and ensure that you have remedied the problem. If the problem is not corrected, then drop and re-create the materialized view.

The following statement analyzes the `emp` table:

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

You can validate an object and all dependent objects (for example, indexes) by including the `CASCADE` option. The following statement validates the `emp` table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

By default the `CASCADE` option performs a complete validation. Because this operation can be resource intensive, you can perform a faster version of the validation by using the `FAST` clause. This version checks for the existence of corruptions using an optimized check algorithm, but does not report details about the corruption. If the `FAST` check finds a corruption, then you can then use the `CASCADE` option without the `FAST` clause to locate it. The following statement performs a fast validation on the `emp` table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE FAST;
```

If fast validation takes an inordinate amount of time, then you have the option of validating individual indexes with a SQL query. See "Cross Validation of a Table and an Index with a Query".

You can specify that you want to perform structure validation online while DML is occurring against the object being validated. Validation is less comprehensive with ongoing DML affecting the object, but this is offset by the flexibility of being able to perform `ANALYZE` online. The following statement validates the `emp` table and all associated indexes online:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE ONLINE;
```

See Also:

Oracle Database SQL Language Reference for more information on the ANALYZE statement

17.4.4 Cross Validation of a Table and an Index with a Query

In some cases, an ANALYZE statement takes an inordinate amount of time to complete. In these cases, you can use a SQL query to validate an index.

If the query determines that there is an inconsistency between a table and an index, then you can use an ANALYZE statement for a thorough analysis of the index. Since typically most objects in a database are not corrupt, you can use this quick query to eliminate a number of tables as candidates for corruption and only use the ANALYZE statement on tables that might be corrupt.

To validate an index, run the following query:

```
SELECT /*+ FULL(ALIAS) PARALLEL(ALIAS, DOP) */ SUM(ORA_HASH(ROWID))
   FROM table_name ALIAS
   WHERE ALIAS.index_column IS NOT NULL
      MINUS SELECT /*+ INDEX_FFS(ALIAS index_name)
      PARALLEL_INDEX(ALIAS, index_name, DOP) */ SUM(ORA_HASH(ROWID))
   FROM table_name ALIAS WHERE ALIAS.index_column IS NOT NULL;
```

When you run the query, make the following substitutions:

- Enter the table name for the *table_name* placeholder.
- Enter the index column for the *index_column* placeholder.
- Enter the index name for the *index_name* placeholder.

If the query returns any rows, then there is a possible inconsistency, and you can use an ANALYZE statement for further diagnosis.

See Also:

Oracle Database SQL Language Reference for more information about the ANALYZE statement

17.4.5 Listing Chained Rows of Tables and Clusters

You can look at the chained and migrated rows of a table or cluster using the ANALYZE statement with the LIST CHAINED ROWS clause. The results of this statement are stored in a specified table created explicitly to accept the information returned by the LIST CHAINED ROWS clause. These results are useful in determining whether you have enough room for updates to rows.

- Creating a CHAINED_ROWS Table
  To create the table to accept data returned by an ANALYZE...LIST CHAINED ROWS statement, execute the UTLCHAIN.SQL or UTLCHN1.SQL script.

- Eliminating Migrated or Chained Rows in a Table
  You can use the information in the `CHAINED_ROWS` table to reduce or eliminate migrated and chained rows in an existing table.

17.4.5.1 Creating a CHAINED_ROWS Table

To create the table to accept data returned by an `ANALYZE...LIST CHAINED ROWS` statement, execute the `UTLCHAIN.SQL` or `UTLCHN1.SQL` script.

These scripts are provided by the database. They create a table named `CHAINED_ROWS` in the schema of the user submitting the script.

> **Note:**
>
> Your choice of script to execute for creating the `CHAINED_ROWS` table depends on the compatibility level of your database and the type of table you are analyzing. See the *Oracle Database SQL Language Reference* for more information.

After a `CHAINED_ROWS` table is created, you specify it in the `INTO` clause of the `ANALYZE` statement. For example, the following statement inserts rows containing information about the chained rows in the `emp_dept` cluster into the `CHAINED_ROWS` table:

```
ANALYZE CLUSTER emp_dept LIST CHAINED ROWS INTO CHAINED_ROWS;
```

> **See Also:**
>
> - *Oracle Database Reference* for a description of the `CHAINED_ROWS` table
> - "Using the Segment Advisor" for information on how the Segment Advisor reports tables with excess row chaining.

17.4.5.2 Eliminating Migrated or Chained Rows in a Table

You can use the information in the `CHAINED_ROWS` table to reduce or eliminate migrated and chained rows in an existing table.

Use the following procedure:

1. Use the `ANALYZE` statement to collect information about migrated and chained rows.

   ```
   ANALYZE TABLE order_hist LIST CHAINED ROWS;
   ```

2. Query the output table:

   ```
   SELECT *
   FROM CHAINED_ROWS
   WHERE TABLE_NAME = 'ORDER_HIST';

   OWNER_NAME  TABLE_NAME  CLUST... HEAD_ROWID          TIMESTAMP
   ----------  ----------  -----... ------------------  ---------
   SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAA  04-MAR-96
   ```

```
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAB  04-MAR-96
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAC  04-MAR-96
```

The output lists all rows that are either migrated or chained.

3. If the output table shows that you have many migrated or chained rows, then you can eliminate migrated rows by continuing through the following steps:
4. Create an intermediate table with the same columns as the existing table to hold the migrated and chained rows:

```
CREATE TABLE int_order_hist
   AS SELECT *
      FROM order_hist
      WHERE ROWID IN
         (SELECT HEAD_ROWID
            FROM CHAINED_ROWS
            WHERE TABLE_NAME = 'ORDER_HIST');
```

5. Delete the migrated and chained rows from the existing table:

```
DELETE FROM order_hist
   WHERE ROWID IN
      (SELECT HEAD_ROWID
         FROM CHAINED_ROWS
         WHERE TABLE_NAME = 'ORDER_HIST');
```

6. Insert the rows of the intermediate table into the existing table:

```
INSERT INTO order_hist
   SELECT *
   FROM int_order_hist;
```

7. Drop the intermediate table:

```
DROP TABLE int_order_history;
```

8. Delete the information collected in step 1 from the output table:

```
DELETE FROM CHAINED_ROWS
   WHERE TABLE_NAME = 'ORDER_HIST';
```

9. Use the `ANALYZE` statement again, and query the output table.

Any rows that appear in the output table are chained. You can eliminate chained rows only by increasing your data block size. It might not be possible to avoid chaining in all situations. Chaining is often unavoidable with tables that have a `LONG` column or large `CHAR` or `VARCHAR2` columns.

17.5 Truncating Tables and Clusters

You can delete all rows of a table or all rows in a group of clustered tables so that the table (or cluster) still exists, but is completely empty. For example, consider a table that contains monthly data, and at the end of each month, you must empty it (delete all rows) after archiving its data.

- Using DELETE to Truncate a Table
  You can delete the rows of a table using the `DELETE` SQL statement.
- Using DROP and CREATE to Truncate a Table
  You can drop a table and then re-create the table to truncate it.
- Using TRUNCATE
  You can delete all rows of the table using the `TRUNCATE` statement.

17.5.1 Using DELETE to Truncate a Table

You can delete the rows of a table using the `DELETE` SQL statement.

For example, the following statement deletes all rows from the `emp` table:

```
DELETE FROM emp;
```

If there are many rows present in a table or cluster when using the `DELETE` statement, significant system resources are consumed as the rows are deleted. For example, CPU time, redo log space, and undo segment space from the table and any associated indexes require resources. Also, as each row is deleted, triggers can be fired. The space previously allocated to the resulting empty table or cluster remains associated with that object. With `DELETE` you can choose which rows to delete, whereas `TRUNCATE` and `DROP` affect the entire object.

See Also:

Oracle Database SQL Language Reference for syntax and other information about the `DELETE` statement

17.5.2 Using DROP and CREATE to Truncate a Table

You can drop a table and then re-create the table to truncate it.

For example, the following statements drop and then re-create the `emp` table:

```
DROP TABLE emp;
CREATE TABLE emp ( ... );
```

When dropping and re-creating a table or cluster, all associated indexes, integrity constraints, and triggers are also dropped, and all objects that depend on the dropped table or clustered table are invalidated. Also, all grants for the dropped table or clustered table are dropped.

17.5.3 Using TRUNCATE

You can delete all rows of the table using the `TRUNCATE` statement.

For example, the following statement truncates the `emp` table:

```
TRUNCATE TABLE emp;
```

Using the `TRUNCATE` statement provides a fast, efficient method for deleting all rows from a table or cluster. A `TRUNCATE` statement does not generate any undo information and it commits immediately. It is a DDL statement and cannot be rolled back. A `TRUNCATE` statement does not affect any structures associated with the table being truncated (constraints and triggers) or authorizations. A `TRUNCATE` statement also specifies whether space currently allocated for the table is returned to the containing tablespace after truncation.

You can truncate any table or cluster in your own schema. Any user who has the `DROP ANY TABLE` system privilege can truncate a table or cluster in any schema.

Before truncating a table or clustered table containing a parent key, all referencing foreign keys in different tables must be disabled. A self-referential constraint does not have to be disabled.

As a `TRUNCATE` statement deletes rows from a table, triggers associated with the table are not fired. Also, a `TRUNCATE` statement does not generate any audit information corresponding to `DELETE` statements if auditing is enabled. Instead, a single audit record is generated for the `TRUNCATE` statement being issued.

A hash cluster cannot be truncated, nor can tables within a hash or index cluster be individually truncated. Truncation of an index cluster deletes all rows from all tables in the cluster. If all the rows must be deleted from an individual clustered table, use the `DELETE` statement or drop and re-create the table.

The `TRUNCATE` statement has several options that control whether space currently allocated for a table or cluster is returned to the containing tablespace after truncation.

These options also apply to any associated indexes. When a table or cluster is truncated, all associated indexes are also truncated. The storage parameters for a truncated table, cluster, or associated indexes are not changed as a result of the truncation.

These `TRUNCATE` options are:

- `DROP STORAGE`, the default option, reduces the number of extents allocated to the resulting table to the original setting for `MINEXTENTS`. Freed extents are then returned to the system and can be used by other objects.
- `DROP ALL STORAGE` drops the segment. In addition to the `TRUNCATE TABLE` statement, `DROP ALL STORAGE` also applies to the `ALTER TABLE TRUNCATE (SUB) PARTITION` statement. This option also drops any dependent object segments associated with the partition being truncated.

  `DROP ALL STORAGE` is not supported for clusters.

  ```
  TRUNCATE TABLE emp DROP ALL STORAGE;
  ```
- `REUSE STORAGE` specifies that all space currently allocated for the table or cluster remains allocated to it. For example, the following statement truncates the `emp_dept` cluster, leaving all extents previously allocated for the cluster available for subsequent inserts and deletes:

  ```
  TRUNCATE CLUSTER emp_dept REUSE STORAGE;
  ```

See Also:

- *Oracle Database SQL Language Reference* for syntax and other information about the `TRUNCATE TABLE` statement
- *Oracle Database SQL Language Reference* for syntax and other information about the `TRUNCATE CLUSTER` statement
- *Oracle Database Security Guide* for information about auditing

17.6 Enabling and Disabling Triggers

Database triggers are procedures that are stored in the database and activated ("fired") when specific conditions occur, such as adding a row to a table.

You can use triggers to supplement the standard capabilities of the database to provide a highly customized database management system. For example, you can create a trigger to restrict DML operations against a table, allowing only statements issued during regular business hours.

- About Enabling and Disabling Triggers
  An enabled trigger executes its trigger body if a triggering statement is issued and the trigger restriction, if any, evaluates to true. By default, triggers are enabled when first created. A disabled trigger does not execute its trigger body, even if a triggering statement is issued and the trigger restriction (if any) evaluates to true.
- Enabling Triggers
  You enable a disabled trigger using the `ALTER TRIGGER` statement with the `ENABLE` option.
- Disabling Triggers
  You disable a trigger using the `ALTER TRIGGER` statement with the `DISABLE` option.

17.6.1 About Enabling and Disabling Triggers

An enabled trigger executes its trigger body if a triggering statement is issued and the trigger restriction, if any, evaluates to true. By default, triggers are enabled when first created. A disabled trigger does not execute its trigger body, even if a triggering statement is issued and the trigger restriction (if any) evaluates to true.

Database triggers can be associated with a table, schema, or database. They are implicitly fired when:

- DML statements are executed (`INSERT`, `UPDATE`, `DELETE`) against an associated table
- Certain DDL statements are executed (for example: `ALTER`, `CREATE`, `DROP`) on objects within a database or schema
- A specified database event occurs (for example: `STARTUP`, `SHUTDOWN`, `SERVERERROR`)

This is not a complete list. See the *Oracle Database SQL Language Reference* for a full list of statements and database events that cause triggers to fire.

Create triggers with the `CREATE TRIGGER` statement. They can be defined as firing `BEFORE` or `AFTER` the triggering event, or `INSTEAD OF` it. The following statement creates a trigger `scott.emp_permit_changes` on table `scott.emp`. The trigger fires before any of the specified statements are executed.

```
CREATE TRIGGER scott.emp_permit_changes
     BEFORE
     DELETE OR INSERT OR UPDATE
     ON scott.emp
     .
     .
     .
pl/sql block
     .
     .
     .
```

You can later remove a trigger from the database by issuing the `DROP TRIGGER` statement.

To enable or disable triggers using the `ALTER TABLE` statement, you must own the table, have the `ALTER` object privilege for the table, or have the `ALTER ANY TABLE` system privilege. To enable or disable an individual trigger using the `ALTER TRIGGER` statement, you must own the trigger or have the `ALTER ANY TRIGGER` system privilege.

See Also:

- *Oracle Database Concepts* for a more detailed description of triggers
- *Oracle Database SQL Language Reference* for syntax of the `CREATE TRIGGER` statement
- *Oracle Database PL/SQL Language Reference* for information about creating and using triggers

17.6.2 Enabling Triggers

You enable a disabled trigger using the `ALTER TRIGGER` statement with the `ENABLE` option.

To enable the disabled trigger named `reorder` on the `inventory` table, enter the following statement:

```
ALTER TRIGGER reorder ENABLE;
```

To enable all triggers defined for a specific table, use the `ALTER TABLE` statement with the `ENABLE ALL TRIGGERS` option. To enable all triggers defined for the `INVENTORY` table, enter the following statement:

```
ALTER TABLE inventory
    ENABLE ALL TRIGGERS;
```

See Also:

Oracle Database SQL Language Reference for syntax and other information about the `ALTER TRIGGER` statement

17.6.3 Disabling Triggers

You disable a trigger using the `ALTER TRIGGER` statement with the `DISABLE` option.

Consider temporarily disabling a trigger if one of the following conditions is true:

- An object that the trigger references is not available.
- You must perform a large data load and want it to proceed quickly without firing triggers.
- You are loading data into the table to which the trigger applies.

To disable the trigger `reorder` on the `inventory` table, enter the following statement:

```
ALTER TRIGGER reorder DISABLE;
```

You can disable all triggers associated with a table at the same time using the `ALTER TABLE` statement with the `DISABLE ALL TRIGGERS` option. For example, to disable all triggers defined for the `inventory` table, enter the following statement:

```
ALTER TABLE inventory
    DISABLE ALL TRIGGERS;
```

17.7 Managing Integrity Constraints

Integrity constraints are rules that restrict the values for one or more columns in a table. Constraint clauses can appear in either `CREATE TABLE` or `ALTER TABLE` statements, and identify the column or columns affected by the constraint and identify the conditions of the constraint.

- Integrity Constraint States
  Integrity constraints enforce business rules and prevent the entry of invalid information into tables.
- Setting Integrity Constraints Upon Definition
  When an integrity constraint is defined in a `CREATE TABLE` or `ALTER TABLE` statement, it can be enabled, disabled, or validated or not validated as determined by your specification of the `ENABLE`/`DISABLE` clause. If the `ENABLE`/`DISABLE` clause is not specified in a constraint definition, the database automatically enables and validates the constraint.
- Modifying, Renaming, or Dropping Existing Integrity Constraints
  You can use the `ALTER TABLE` statement to enable, disable, modify, or drop a constraint. When the database is using a `UNIQUE` or `PRIMARY KEY` index to enforce a constraint, and constraints associated with that index are dropped or disabled, the index is dropped, unless you specify otherwise.
- Deferring Constraint Checks
  When the database checks a constraint, it signals an error if the constraint is not satisfied. You can defer checking the validity of constraints until the end of a transaction. When you issue the `SET CONSTRAINTS` statement, the `SET CONSTRAINTS` mode lasts for the duration of the transaction, or until another `SET CONSTRAINTS` statement resets the mode.
- Reporting Constraint Exceptions
  If exceptions exist when a constraint is validated, then an error is returned and the integrity constraint remains novalidated. When a statement is not successfully executed because integrity constraint exceptions exist, the statement is rolled back. If exceptions exist, then you cannot validate the constraint until all exceptions to the constraint are either updated or deleted.
- Viewing Constraint Information
  Oracle Database provides a set of views that enable you to see constraint definitions on tables and to identify columns that are specified in constraints.

See Also:

- *Oracle Database Concepts* for a more thorough discussion of integrity constraints
- *Oracle Database Development Guide* for detailed information and examples of using integrity constraints in applications

17.7.1 Integrity Constraint States

Integrity constraints enforce business rules and prevent the entry of invalid information into tables.

- About Integrity Constraint States
  You can specify that a constraint is enabled (`ENABLE`) or disabled (`DISABLE`). If a constraint is enabled, data is checked as it is entered or updated in the database, and data that does not conform to the constraint is prevented from being entered. If a constraint is disabled, then data that does not conform can be allowed to enter the database.
- About Disabling Constraints
  To enforce the rules defined by integrity constraints, the constraints should always be enabled, but you can consider disabling them in certain situations.
- About Enabling Constraints
  While a constraint is enabled, no row violating the constraint can be inserted into the table.
- About the Enable Novalidate Constraint State
  When a constraint is in the enable novalidate state, all subsequent statements are checked for conformity to the constraint. However, any existing data in the table is not checked.
- Efficient Use of Integrity Constraints: A Procedure
  It is important to use integrity constraint states in a particular order.

17.7.1.1 About Integrity Constraint States

You can specify that a constraint is enabled (`ENABLE`) or disabled (`DISABLE`). If a constraint is enabled, data is checked as it is entered or updated in the database, and data that does not conform to the constraint is prevented from being entered. If a constraint is disabled, then data that does not conform can be allowed to enter the database.

Additionally, you can specify that existing data in the table must conform to the constraint (`VALIDATE`). Conversely, if you specify `NOVALIDATE`, you are not ensured that existing data conforms.

An integrity constraint defined on a table can be in one of the following states:

- `ENABLE`, `VALIDATE`
- `ENABLE`, `NOVALIDATE`
- `DISABLE`, `VALIDATE`
- `DISABLE`, `NOVALIDATE`

For details about the meaning of these states and an understanding of their consequences, see the *Oracle Database SQL Language Reference.* Some of these consequences are discussed here.

17.7.1.2 About Disabling Constraints

To enforce the rules defined by integrity constraints, the constraints should always be enabled, but you can consider disabling them in certain situations.

However, consider temporarily disabling the integrity constraints of a table for the following performance reasons:

- When loading large amounts of data into a table
- When performing batch operations that make massive changes to a table (for example, changing every employee's number by adding 1000 to the existing number)
- When importing or exporting one table at a time

In all three cases, temporarily disabling integrity constraints can improve the performance of the operation, especially in data warehouse configurations.

It is possible to enter data that violates a constraint while that constraint is disabled. Thus, you should always enable the constraint after completing any of the operations listed in the preceding bullet list.

17.7.1.3 About Enabling Constraints

While a constraint is enabled, no row violating the constraint can be inserted into the table.

However, while the constraint is disabled such a row can be inserted. This row is known as an exception to the constraint. If the constraint is in the enable novalidated state, violations resulting from data entered while the constraint was disabled remain. The rows that violate the constraint must be either updated or deleted in order for the constraint to be put in the validated state.

You can identify exceptions to a specific integrity constraint while attempting to enable the constraint. See "Reporting Constraint Exceptions". All rows violating constraints are noted in an `EXCEPTIONS` table, which you can examine.

17.7.1.4 About the Enable Novalidate Constraint State

When a constraint is in the enable novalidate state, all subsequent statements are checked for conformity to the constraint. However, any existing data in the table is not checked.

A table with enable novalidated constraints can contain invalid data, but it is not possible to add new invalid data to it. Enabling constraints in the novalidated state is most useful in data warehouse configurations that are uploading valid OLTP data.

Enabling a constraint does not require validation. Enabling a constraint novalidate is much faster than enabling and validating a constraint. Also, validating a constraint that is already enabled does not require any DML locks during validation (unlike validating a previously disabled constraint). Enforcement guarantees that no violations are introduced during the validation. Hence, enabling without validating enables you to reduce the downtime typically associated with enabling a constraint.

17.7.1.5 Efficient Use of Integrity Constraints: A Procedure

It is important to use integrity constraint states in a particular order.

Using integrity constraint states in the following order can ensure the best benefits:

1. Set state of constraint to DISABLE.
2. Perform the operation (load, export, import).
3. Set state of constraint to ENABLE NOVALIDATE.
4. Set state of constraint to ENABLE.

For example:

```
SQL> CREATE TABLE eg(n NUMBER NOT NULL CONSTRAINT n_gt_0 CHECK (n > 0));

Table created.

SQL> SELECT status AS enabled, validated
     FROM   user_constraints
     WHERE  table_name = 'EG' and constraint_name = 'N_GT_0';

ENABLED  VALIDATED
```

```
-------- -------------
ENABLED  VALIDATED

SQL> ALTER TABLE eg MODIFY CONSTRAINT n_gt_0 DISABLE;

Table altered.

SQL> SELECT status AS enabled, validated
     FROM   user_constraints
     WHERE  table_name = 'EG' and constraint_name = 'N_GT_0';

ENABLED  VALIDATED
-------- -------------
DISABLED NOT VALIDATED

SQL> ALTER TABLE eg MODIFY CONSTRAINT n_gt_0 enable NOVALIDATE;

Table altered.

SQL> SELECT status AS enabled, validated
     FROM   user_constraints
     WHERE  table_name = 'EG' and constraint_name = 'N_GT_0';

ENABLED  VALIDATED
-------- -------------
ENABLED  NOT VALIDATED

SQL> ALTER TABLE eg MODIFY CONSTRAINT n_gt_0 ENABLE;

Table altered.

SQL> SELECT status AS enabled, validated
     FROM   user_constraints
     WHERE  table_name = 'EG' and constraint_name = 'N_GT_0';

ENABLED  VALIDATED
-------- -------------
ENABLED  VALIDATED
```

Some benefits of using constraints in this order are:

- No locks are held.
- All constraints can go to enable state concurrently.
- Constraint enabling is done in parallel.
- Concurrent activity on table is permitted.

The PRIMARY KEY and FOREIGN KEY constraints may not permit concurrent activity due to waits for library cache locks.

17.7.2 Setting Integrity Constraints Upon Definition

When an integrity constraint is defined in a `CREATE TABLE` or `ALTER TABLE` statement, it can be enabled, disabled, or validated or not validated as determined by your specification of the

ENABLE/DISABLE clause. If the ENABLE/DISABLE clause is not specified in a constraint definition, the database automatically enables and validates the constraint.

- Disabling Constraints Upon Definition
  You can disable an integrity constraint when you define it.
- Enabling Constraints Upon Definition
  You can enable an integrity constraint when you define it.

17.7.2.1 Disabling Constraints Upon Definition

You can disable an integrity constraint when you define it.

The following CREATE TABLE and ALTER TABLE statements both define and disable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY DISABLE,   . . . ;

ALTER TABLE emp
   ADD PRIMARY KEY (empno) DISABLE;
```

An ALTER TABLE statement that defines and disables an integrity constraint never fails because of rows in the table that violate the integrity constraint. The definition of the constraint is allowed because its rule is not enforced.

17.7.2.2 Enabling Constraints Upon Definition

You can enable an integrity constraint when you define it.

The following CREATE TABLE and ALTER TABLE statements both define and enable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) CONSTRAINT emp.pk PRIMARY KEY,   . . . ;

ALTER TABLE emp
    ADD CONSTRAINT emp.pk PRIMARY KEY (empno);
```

An ALTER TABLE statement that defines and attempts to enable an integrity constraint can fail because rows of the table violate the integrity constraint. If this case, the statement is rolled back and the constraint definition is not stored and not enabled.

When you enable a UNIQUE or PRIMARY KEY constraint an associated index is created.

Note:

An efficient procedure for enabling a constraint that can make use of parallelism is described in "Efficient Use of Integrity Constraints: A Procedure".

See Also:

"Creating an Index Associated with a Constraint"

17.7.3 Modifying, Renaming, or Dropping Existing Integrity Constraints

You can use the ALTER TABLE statement to enable, disable, modify, or drop a constraint. When the database is using a UNIQUE or PRIMARY KEY index to enforce a constraint, and constraints associated with that index are dropped or disabled, the index is dropped, unless you specify otherwise.

While enabled foreign keys reference a PRIMARY or UNIQUE key, you cannot disable or drop the PRIMARY or UNIQUE key constraint or the index.

- Disabling and Enabling Constraints
  You can disable enabled integrity constraints and enable disabled integrity constraints.
- Renaming Constraints
  The ALTER TABLE...RENAME CONSTRAINT statement enables you to rename any currently existing constraint for a table. The new constraint name must not conflict with any existing constraint names for a user.
- Dropping Constraints
  You can drop an integrity constraint if the rule that it enforces is no longer true, or if the constraint is no longer needed.

17.7.3.1 Disabling and Enabling Constraints

You can disable enabled integrity constraints and enable disabled integrity constraints.

The following statements disable integrity constraints. The second statement specifies that the associated indexes are to be kept.

```
ALTER TABLE dept
    DISABLE CONSTRAINT dname_ukey;

ALTER TABLE dept
    DISABLE PRIMARY KEY KEEP INDEX,
    DISABLE UNIQUE (dname, loc) KEEP INDEX;
```

The following statements enable novalidate disabled integrity constraints:

```
ALTER TABLE dept
    ENABLE NOVALIDATE CONSTRAINT dname_ukey;

ALTER TABLE dept
    ENABLE NOVALIDATE PRIMARY KEY,
    ENABLE NOVALIDATE UNIQUE (dname, loc);
```

The following statements enable or validate disabled integrity constraints:

```
ALTER TABLE dept
    MODIFY CONSTRAINT dname_key VALIDATE;

ALTER TABLE dept
    MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

The following statements enable disabled integrity constraints:

```
ALTER TABLE dept
    ENABLE CONSTRAINT dname_ukey;

ALTER TABLE dept
```

```
    ENABLE PRIMARY KEY,
    ENABLE UNIQUE (dname, loc);
```

To disable or drop a UNIQUE key or PRIMARY KEY constraint and all dependent FOREIGN KEY constraints in a single step, use the CASCADE option of the DISABLE or DROP clauses. For example, the following statement disables a PRIMARY KEY constraint and any FOREIGN KEY constraints that depend on it:

```
ALTER TABLE dept
    DISABLE PRIMARY KEY CASCADE;
```

17.7.3.2 Renaming Constraints

The ALTER TABLE...RENAME CONSTRAINT statement enables you to rename any currently existing constraint for a table. The new constraint name must not conflict with any existing constraint names for a user.

The following statement renames the dname_ukey constraint for table dept:

```
ALTER TABLE dept
    RENAME CONSTRAINT dname_ukey TO dname_unikey;
```

When you rename a constraint, all dependencies on the base table remain valid.

The RENAME CONSTRAINT clause provides a means of renaming system generated constraint names.

17.7.3.3 Dropping Constraints

You can drop an integrity constraint if the rule that it enforces is no longer true, or if the constraint is no longer needed.

You can drop the constraint using the ALTER TABLE statement with one of the following clauses:

- DROP PRIMARY KEY
- DROP UNIQUE
- DROP CONSTRAINT

The following two statements drop integrity constraints. The second statement keeps the index associated with the PRIMARY KEY constraint:

```
ALTER TABLE dept
    DROP UNIQUE (dname, loc);

ALTER TABLE emp
    DROP PRIMARY KEY KEEP INDEX
    DROP CONSTRAINT dept_fkey;
```

If FOREIGN KEYs reference a UNIQUE or PRIMARY KEY, you must include the CASCADE CONSTRAINTS clause in the DROP statement, or you cannot drop the constraint.

17.7.4 Deferring Constraint Checks

When the database checks a constraint, it signals an error if the constraint is not satisfied. You can defer checking the validity of constraints until the end of a transaction. When you issue the SET CONSTRAINTS statement, the SET CONSTRAINTS mode lasts for the duration of the transaction, or until another SET CONSTRAINTS statement resets the mode.

Note:

- You cannot issue a `SET CONSTRAINT` statement inside a trigger.
- Deferrable unique and primary keys must use nonunique indexes.

- Set All Constraints Deferred
  When constraints must be deferred for a transaction, you must set all constraints deferred before you actually begin processing any data within the application being used to manipulate the data.
- Check the Commit (Optional)
  You can check for constraint violations before committing by issuing the `SET CONSTRAINTS ALL IMMEDIATE` statement just before issuing the `COMMIT`.

17.7.4.1 Set All Constraints Deferred

When constraints must be deferred for a transaction, you must set all constraints deferred before you actually begin processing any data within the application being used to manipulate the data.

Use the following DML statement to set all deferrable constraints deferred:

```
SET CONSTRAINTS ALL DEFERRED;
```

Note:

The `SET CONSTRAINTS` statement applies only to the current transaction. The defaults specified when you create a constraint remain as long as the constraint exists. The `ALTER SESSION SET CONSTRAINTS` statement applies for the current session only.

17.7.4.2 Check the Commit (Optional)

You can check for constraint violations before committing by issuing the `SET CONSTRAINTS ALL IMMEDIATE` statement just before issuing the `COMMIT`.

If there are any problems with a constraint, then this statement fails and the constraint causing the error is identified. If you commit while constraints are violated, then the transaction is rolled back and you receive an error message.

17.7.5 Reporting Constraint Exceptions

If exceptions exist when a constraint is validated, then an error is returned and the integrity constraint remains novalidated. When a statement is not successfully executed because integrity constraint exceptions exist, the statement is rolled back. If exceptions exist, then you cannot validate the constraint until all exceptions to the constraint are either updated or deleted.

To determine which rows violate the integrity constraint, issue the `ALTER TABLE` statement with the `EXCEPTIONS` option in the `ENABLE` clause. The `EXCEPTIONS` option places the rowid, table owner, table name, and constraint name of all exception rows into a specified table.

You must create an appropriate exceptions report table to accept information from the EXCEPTIONS option of the ENABLE clause before enabling the constraint. You can create an exception table by executing the UTLEXCPT.SQL script or the UTLEXPT1.SQL script.

Note:

Your choice of script to execute for creating the EXCEPTIONS table depends on the type of table you are analyzing. See the *Oracle Database SQL Language Reference* for more information.

Both of these scripts create a table named EXCEPTIONS. You can create additional exceptions tables with different names by modifying and resubmitting the script.

The following statement attempts to validate the PRIMARY KEY of the dept table, and if exceptions exist, information is inserted into a table named EXCEPTIONS:

```
ALTER TABLE dept ENABLE PRIMARY KEY EXCEPTIONS INTO EXCEPTIONS;
```

If duplicate primary key values exist in the dept table and the name of the PRIMARY KEY constraint on dept is sys_c00610, then the following query will display those exceptions:

```
SELECT * FROM EXCEPTIONS;
```

The following exceptions are shown:

```
fROWID               OWNER      TABLE_NAME      CONSTRAINT
------------------  ---------  --------------  -----------
AAAAZ9AABAAABvqAAB  SCOTT      DEPT            SYS_C00610
AAAAZ9AABAAABvqAAG  SCOTT      DEPT            SYS_C00610
```

A more informative query would be to join the rows in an exception report table and the master table to list the actual rows that violate a specific constraint, as shown in the following statement and results:

```
SELECT deptno, dname, loc FROM dept, EXCEPTIONS
    WHERE EXCEPTIONS.constraint = 'SYS_C00610'
    AND dept.rowid = EXCEPTIONS.row_id;

DEPTNO     DNAME              LOC
---------- --------------     -----------
10         ACCOUNTING         NEW YORK
10         RESEARCH           DALLAS
```

All rows that violate a constraint must be either updated or deleted from the table containing the constraint. When updating exceptions, you must change the value violating the constraint to a value consistent with the constraint or to a null. After the row in the master table is updated or deleted, the corresponding rows for the exception in the exception report table should be deleted to avoid confusion with later exception reports. The statements that update the master table and the exception report table should be in the same transaction to ensure transaction consistency.

To correct the exceptions in the previous examples, you might issue the following transaction:

```
UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
DELETE FROM EXCEPTIONS WHERE constraint = 'SYS_C00610';
COMMIT;
```

When managing exceptions, the goal is to eliminate all exceptions in your exception report table.

Note:

While you are correcting current exceptions for a table with the constraint disabled, it is possible for other users to issue statements creating new exceptions. You can avoid this by marking the constraint `ENABLE NOVALIDATE` before you start eliminating exceptions.

See Also:

Oracle Database Reference for a description of the `EXCEPTIONS` table

17.7.6 Viewing Constraint Information

Oracle Database provides a set of views that enable you to see constraint definitions on tables and to identify columns that are specified in constraints.

View	Description

`DBA_CONSTRAINTS`
`ALL_CONSTRAINTS`
`USER_CONSTRAINTS`
	`DBA` view describes all constraint definitions in the database. `ALL` view describes constraint definitions accessible to current user. `USER` view describes constraint definitions owned by the current user.

`DBA_CONS_COLUMNS`
`ALL_CONS_COLUMNS`
`USER_CONS_COLUMNS`
	`DBA` view describes all columns in the database that are specified in constraints. `ALL` view describes only those columns accessible to current user that are specified in constraints. `USER` view describes only those columns owned by the current user that are specified in constraints.

See Also:

- *Oracle Database Reference* for information about the `*_CONSTRAINTS` views
- *Oracle Database Reference* for information about the `*_CONS_COLUMNS` views

17.8 Renaming Schema Objects

There are several ways to rename an object.

To rename an object, it must be in your schema. You can rename schema objects in either of the following ways:

- Drop and re-create the object

- Rename the object using the `RENAME` statement
- Rename the object using the `ALTER` ... `RENAME` statement (for indexes and triggers)

If you drop and re-create an object, all privileges granted for that object are lost. Privileges must be regranted when the object is re-created.

A table, view, sequence, or a private synonym of a table, view, or sequence can be renamed using the `RENAME` statement. When using the `RENAME` statement, integrity constraints, indexes, and grants made for the object are carried forward for the new name. For example, the following statement renames the `sales_staff` view:

```
RENAME sales_staff TO dept_30;
```

Note:

You cannot use `RENAME` for a stored PL/SQL program unit, public synonym, or cluster. To rename such an object, you must drop and re-create it.

Before renaming a schema object, consider the following effects:

- All views and PL/SQL program units dependent on a renamed object become invalid, and must be recompiled before next use.
- All synonyms for a renamed object return an error when used.

See Also:

Oracle Database SQL Language Reference for syntax of the `RENAME` statement

17.9 Managing Object Dependencies

Oracle Database provides an automatic mechanism to ensure that a dependent object is always up to date with respect to its referenced objects. You can also manually recompile invalid object.

- About Object Dependencies and Object Invalidation
  Some types of schema objects reference other objects. An object that references another object is called a **dependent object**, and an object being referenced is a **referenced object**. These references are established at compile time, and if the compiler cannot resolve them, the dependent object being compiled is marked *invalid*.
- Manually Recompiling Invalid Objects with DDL
  You can use an `ALTER` statement to manually recompile a single schema object.
- Manually Recompiling Invalid Objects with PL/SQL Package Procedures
  The `RECOMP_SERIAL` procedure recompiles all invalid objects in a specified schema, or all invalid objects in the database if you do not supply the schema name argument. The `RECOMP_PARALLEL` procedure does the same, but in parallel, employing multiple CPUs.

17.9.1 About Object Dependencies and Object Invalidation

Some types of schema objects reference other objects. An object that references another object is called a **dependent object**, and an object being referenced is a **referenced object**. These references are established at compile time, and if the compiler cannot resolve them, the dependent object being compiled is marked *invalid*.

For example, a view contains a query that references tables or other views, and a PL/SQL subprogram might invoke other subprograms and might use static SQL to reference tables or views.

Oracle Database provides an automatic mechanism to ensure that a dependent object is always up to date with respect to its referenced objects. When a dependent object is created, the database tracks dependencies between the dependent object and its referenced objects. When a referenced object is changed in a way that might affect a dependent object, the dependent object is marked invalid. An invalid dependent object must be recompiled against the new definition of a referenced object before the dependent object can be used. Recompilation occurs automatically when the invalid dependent object is referenced.

It is important to be aware of changes that can invalidate schema objects, because invalidation affects applications running on the database. This section describes how objects become invalid, how you can identify invalid objects, and how you can validate invalid objects.

Object Invalidation

In a typical running application, you would not expect to see views or stored procedures become invalid, because applications typically do not change table structures or change view or stored procedure definitions during normal execution. Changes to tables, views, or PL/SQL units typically occur when an application is patched or upgraded using a patch script or ad-hoc DDL statements. Dependent objects might be left invalid after a patch has been applied to change a set of referenced objects.

Use the following query to display the set of invalid objects in the database:

```
SELECT object_name, object_type FROM dba_objects
WHERE status = 'INVALID';
```

The Database Home page in Oracle Enterprise Manager Cloud Control displays an alert when schema objects become invalid.

Object invalidation affects applications in two ways. First, an invalid object must be revalidated before it can be used by an application. Revalidation adds latency to application execution. If the number of invalid objects is large, the added latency on the first execution can be significant. Second, invalidation of a procedure, function or package can cause exceptions in other sessions concurrently executing the procedure, function or package. If a patch is applied when the application is in use in a different session, the session executing the application notices that an object in use has been invalidated and raises one of the following 4 exceptions: ORA-04061, ORA-04064, ORA-04065 or ORA-04068. These exceptions must be remedied by restarting application sessions following a patch.

You can force the database to recompile a schema object using the appropriate SQL statement with the `COMPILE` clause. See "Manually Recompiling Invalid Objects with DDL" for more information.

If you know that there are a large number of invalid objects, use the `UTL_RECOMP` PL/SQL package to perform a mass recompilation. See "Manually Recompiling Invalid Objects with PL/SQL Package Procedures" for details.

The following are some general rules for the invalidation of schema objects:

- Between a referenced object and each of its dependent objects, the database tracks the elements of the referenced object that are involved in the dependency. For example, if a single-table view selects only a subset of columns in a table, only those columns are involved in the dependency. For each dependent of an object, if a change is made to the definition of any element involved in the dependency (including dropping the element), the dependent object is invalidated. Conversely, if changes are made only to definitions of elements that are not involved in the dependency, the dependent object remains valid.

  In many cases, therefore, developers can avoid invalidation of dependent objects and unnecessary extra work for the database if they exercise care when changing schema objects.
- Dependent objects are *cascade invalidated*. If any object becomes invalid for any reason, all of that object's dependent objects are immediately invalidated.
- If you revoke any object privileges on a schema object, dependent objects are cascade invalidated.

> **See Also:**
>
> *Oracle Database Concepts* for more detailed information about schema object dependencies

17.9.2 Manually Recompiling Invalid Objects with DDL

You can use an `ALTER` statement to manually recompile a single schema object.

For example, to recompile package body `Pkg1`, you would execute the following DDL statement:

```
ALTER PACKAGE pkg1 COMPILE REUSE SETTINGS;
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for syntax and other information about the various `ALTER` statements

17.9.3 Manually Recompiling Invalid Objects with PL/SQL Package Procedures

The `RECOMP_SERIAL` procedure recompiles all invalid objects in a specified schema, or all invalid objects in the database if you do not supply the schema name argument. The `RECOMP_PARALLEL` procedure does the same, but in parallel, employing multiple CPUs.

Following an application upgrade or patch, it is good practice to revalidate invalid objects to avoid application latencies that result from on-demand object revalidation. Oracle provides the `UTL_RECOMP` package to assist in object revalidation.

Examples

Execute the following PL/SQL block to revalidate all invalid objects in the database, in parallel and in dependency order:

```
begin
   utl_recomp.recomp_parallel();
end;
/
```

You can also revalidate individual invalid objects using the package `DBMS_UTILITY`. The following PL/SQL block revalidates the procedure `UPDATE_SALARY` in schema `HR`:

```
begin
   dbms_utility.validate('HR', 'UPDATE_SALARY', namespace=>1);
end;
/
```

The following PL/SQL block revalidates the package body `HR.ACCT_MGMT`:

```
begin
   dbms_utility.validate('HR', 'ACCT_MGMT', namespace=>2);
end;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `UTL_RECOMP` package
- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_UTILITY` package

17.10 Managing Object Name Resolution

Object names referenced in SQL statements can consist of several pieces, separated by periods. Oracle Database performs specific actions to resolve an object name.

The following describes how the database resolves an object name.

1. Oracle Database attempts to qualify the first piece of the name referenced in the SQL statement. For example, in `scott.emp`, `scott` is the first piece. If there is only one piece, the one piece is considered the first piece.
   a. In the current schema, the database searches for an object whose name matches the first piece of the object name. If it does not find such an object, it continues with step 1.b.
   b. The database searches for a public synonym that matches the first piece of the name. If it does not find one, it continues with step 1.c.
   c. The database searches for a schema whose name matches the first piece of the object name. If it finds one, then the schema is the qualified schema, and it continues with step 1.d.

If no schema is found in step 1.c, the object cannot be qualified and the database returns an error.

d. In the qualified schema, the database searches for an object whose name matches the second piece of the object name.

   If the second piece does not correspond to an object in the previously qualified schema or there is not a second piece, then the database returns an error.

2. A schema object has been qualified. Any remaining pieces of the name must match a valid part of the found object. For example, if `scott.emp.deptno` is the name, `scott` is qualified as a schema, `emp` is qualified as a table, and `deptno` must correspond to a column (because `emp` is a table). If `emp` is qualified as a package, `deptno` must correspond to a public constant, variable, procedure, or function of that package.

When global object names are used in a distributed database, either explicitly or indirectly within a synonym, the local database resolves the reference locally. For example, it resolves a synonym to global object name of a remote table. The partially resolved statement is shipped to the remote database, and the remote database completes the resolution of the object as described here.

Because of how the database resolves references, it is possible for an object to depend on the nonexistence of other objects. This situation occurs when the dependent object uses a reference that would be interpreted differently were another object present. For example, assume the following:

- At the current point in time, the `company` schema contains a table named `emp`.
- A `PUBLIC` synonym named `emp` is created for `company.emp` and the `SELECT` privilege for `company.emp` is granted to the `PUBLIC` role.
- The `jward` schema does not contain a table or private synonym named `emp`.
- The user `jward` creates a view in his schema with the following statement:

```
CREATE VIEW dept_salaries AS
     SELECT deptno, MIN(sal), AVG(sal), MAX(sal) FROM emp
     GROUP BY deptno
     ORDER BY deptno;
```

When `jward` creates the `dept_salaries` view, the reference to `emp` is resolved by first looking for `jward.emp` as a table, view, or private synonym, none of which is found, and then as a public synonym named `emp`, which is found. As a result, the database notes that `jward.dept_salaries` depends on the nonexistence of `jward.emp` and on the existence of `public.emp`.

Now assume that `jward` decides to create a new view named `emp` in his schema using the following statement:

```
CREATE VIEW emp AS
     SELECT empno, ename, mgr, deptno
     FROM company.emp;
```

Notice that `jward.emp` does not have the same structure as `company.emp`.

As it attempts to resolve references in object definitions, the database internally makes note of dependencies that the new dependent object has on "nonexistent" objects--schema objects that, if they existed, would change the interpretation of the object's definition. Such dependencies must be noted in case a nonexistent object is later created. If a nonexistent object is created, all dependent objects must be invalidated so that dependent objects can be recompiled and verified and all dependent function-based indexes must be marked unusable.

Therefore, in the previous example, as `jward.emp` is created, `jward.dept_salaries` is invalidated because it depends on `jward.emp`. Then when `jward.dept_salaries` is used, the database attempts to recompile the view. As the database resolves the reference to `emp`, it finds `jward.emp` (`public.emp` is no longer the referenced object). Because `jward.emp` does not have a `sal` column, the database finds errors when replacing the view, leaving it invalid.

In summary, you must manage dependencies on nonexistent objects checked during object resolution in case the nonexistent object is later created.

See Also:

"Schema Objects and Database Links" for information about name resolution in a distributed database

17.11 Switching to a Different Schema

Use an `ALTER SESSION` SQL statement to switch to a different schema.

The following statement sets the schema of the current session to the schema name specified in the statement.

```
ALTER SESSION SET CURRENT_SCHEMA = <schema name>
```

In subsequent SQL statements, Oracle Database uses this schema name as the schema qualifier when the qualifier is omitted. In addition, the database uses the temporary tablespace of the specified schema for sorts, joins, and storage of temporary database objects. The session retains its original privileges and does not acquire any extra privileges by the preceding `ALTER SESSION` statement.

In the following example, provide the password when prompted:

```
CONNECT scott
ALTER SESSION SET CURRENT_SCHEMA = joe;
SELECT * FROM emp;
```

Because `emp` is not schema-qualified, the table name is resolved under schema `joe`. But if `scott` does not have select privilege on table `joe.emp`, then `scott` cannot execute the `SELECT` statement.

17.12 Managing Editions

Application developers who are upgrading their applications using edition-based redefinition may ask you to perform edition-related tasks that require DBA privileges.

- About Editions and Edition-Based Redefinition
  Edition-based redefinition enables you to upgrade an application's database objects while the application is in use, thus minimizing or eliminating down time. This is accomplished by changing (redefining) database objects in a private environment known as an **edition**.
- DBA Tasks for Edition-Based Redefinition
  A user must have the required privileges to perform tasks related to edition-based redefinition.

- Setting the Database Default Edition
  There is always a default edition for the database. This is the edition that a database session initially uses if it does not explicitly indicate an edition when connecting.
- Querying the Database Default Edition
  The database default edition is stored as a database property.
- Setting the Edition Attribute of a Database Service
  You can set the edition attribute of a database service when you create the service, or you can modify an existing database service to set its edition attribute.
- Using an Edition
  To view or modify objects in a particular edition, you must *use* the edition first. You can specify an edition to use when you connect to the database. If you do not specify an edition, then your session starts in the database default edition.
- Editions Data Dictionary Views
  There are several data dictionary views that aid with managing editions.

17.12.1 About Editions and Edition-Based Redefinition

Edition-based redefinition enables you to upgrade an application's database objects while the application is in use, thus minimizing or eliminating down time. This is accomplished by changing (redefining) database objects in a private environment known as an **edition**.

Only when all changes have been made and tested do you make the new version of the application available to users.

See Also:

Oracle Database Development Guide for a complete discussion of edition-based redefinition

17.12.2 DBA Tasks for Edition-Based Redefinition

A user must have the required privileges to perform tasks related to edition-based redefinition.

Table 17-1 summarizes the edition-related tasks that require privileges typically granted only to DBAs. Any user that is granted the `DBA` role can perform these tasks.

Table 17-1 DBA Tasks for Edition-Based Redefinition

Task	See
Grant or revoke privileges to create, alter, and drop editions	The `CREATE EDITION` and `DROP EDITION` SQL statements
Enable editions for a schema	*Oracle Database Development Guide*
Set the database default edition	"Setting the Database Default Edition"
Set the edition attribute of a database service	"Setting the Edition Attribute of a Database Service"

17.12.3 Setting the Database Default Edition

There is always a default edition for the database. This is the edition that a database session initially uses if it does not explicitly indicate an edition when connecting.

To set the database default edition:

1. Connect to the database as a user with the `ALTER DATABASE` privilege and `USE` privilege `WITH GRANT OPTION` on the edition.
2. Enter the following statement:

   ```
   ALTER DATABASE DEFAULT EDITION = edition_name;
   ```

> **See Also:**
>
> "Connecting to the Database with SQL*Plus"

17.12.4 Querying the Database Default Edition

The database default edition is stored as a database property.

To query the database default edition:

1. Connect to the database as any user.
2. Enter the following statement:

   ```
   SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
      PROPERTY_NAME = 'DEFAULT_EDITION';

   PROPERTY_VALUE
   ------------------------------
   ORA$BASE
   ```

> **Note:**
>
> The property name `DEFAULT_EDITION` is case sensitive and must be supplied as upper case.

17.12.5 Setting the Edition Attribute of a Database Service

You can set the edition attribute of a database service when you create the service, or you can modify an existing database service to set its edition attribute.

> **Note:**
>
> The number of database services for an instance has an upper limit. See *Oracle Database Reference* for more information about this limit.

- About Setting the Edition Attribute of a Database Service
  When you set the edition attribute of a service, all subsequent connections that specify the service, such as client connections and `DBMS_SCHEDULER` jobs, use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the session edition.
- Setting the Edition Attribute During Database Service Creation
  You can use the `SRVCTL` utility or the `DBMS_SERVICE` package to set the edition attribute of a database service when you create the service.
- Setting the Edition Attribute of an Existing Database Service
  You can use the `SRVCTL` utility or the `DBMS_SERVICE` package to set the edition attribute of an existing database service.

17.12.5.1 About Setting the Edition Attribute of a Database Service

When you set the edition attribute of a service, all subsequent connections that specify the service, such as client connections and `DBMS_SCHEDULER` jobs, use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the session edition.

To check the edition attribute of a database service, query the `EDITION` column in the `ALL_SERVICES` view or the `DBA_SERVICES` view.

17.12.5.2 Setting the Edition Attribute During Database Service Creation

You can use the `SRVCTL` utility or the `DBMS_SERVICE` package to set the edition attribute of a database service when you create the service.

Follow the instructions in "*Oracle Database SQL Language Reference*" and use the appropriate option for setting the edition attribute for the database service:

- If your single-instance database is being managed by Oracle Restart, use the `SRVCTL` utility to create the database service and specify the `-edition` option to set its edition attribute.

  For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example creates a new database service named `crmbatch` and sets the edition attribute of the database service to `e2`:

  ```
  srvctl add service -db dbcrm -service crmbatch -edition e2
  ```

- If your single-instance database is not being managed by Oracle Restart, use the `DBMS_SERVICE.CREATE_SERVICE` procedure, and specify the `edition` parameter to set the edition attribute of the database service.

17.12.5.3 Setting the Edition Attribute of an Existing Database Service

You can use the `SRVCTL` utility or the `DBMS_SERVICE` package to set the edition attribute of an existing database service.

To set the edition attribute of an existing database service:

1. Stop the database service.
2. Set the edition attribute of the database service using the appropriate option:
   - If your single-instance database is being managed by Oracle Restart, use the `SRVCTL` utility to modify the database service and specify the `-edition` option to set its edition attribute.

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example modifies a database service named `crmbatch` and sets the edition attribute of the service to `e3`:

```
srvctl modify service -db dbcrm -service crmbatch -edition e3
```

- If your single-instance database is not being managed by Oracle Restart, use the `DBMS_SERVICE.MODIFY_SERVICE` procedure, and specify the `edition` parameter to set the edition attribute of the database service. Ensure that the `modify_edition` parameter is set to `TRUE` when you run the `MODIFY_SERVICE` procedure.

3. Start the database service.

See Also:

- Configuring Automatic Restart of an Oracle Database for information managing database services using Oracle Restart
- *Oracle Database PL/SQL Packages and Types Reference* for information about managing database services using the `DBMS_SERVICE` package

17.12.6 Using an Edition

To view or modify objects in a particular edition, you must *use* the edition first. You can specify an edition to use when you connect to the database. If you do not specify an edition, then your session starts in the database default edition.

To use a different edition, submit the following statement:

```
ALTER SESSION SET EDITION=edition_name;
```

The following statements first set the current edition to `e2` and then to `ora$base`:

```
ALTER SESSION SET EDITION=e2;
...
ALTER SESSION SET EDITION=ora$base;
```

See Also:

- *Oracle Database Development Guide* for more information about using editions, and for instructions for determining the current edition
- "Connecting to the Database with SQL*Plus"

17.12.7 Editions Data Dictionary Views

There are several data dictionary views that aid with managing editions.

The following table lists three of them. For a complete list, see *Oracle Database Development Guide*.

View	Description
`*_EDITIONS`	Lists all editions in the database. (Note: `USER_EDITIONS` does not exist.)
`*_OBJECTS`	Describes every object in the database that is visible (actual or inherited) in the current edition.
`*_OBJECTS_AE`	Describes every actual object in the database, across all editions.

17.13 Displaying Information About Schema Objects

Oracle Database provides a PL/SQL package that enables you to determine the DDL that created an object and data dictionary views that you can use to display information about schema objects.

- Using a PL/SQL Package to Display Information About Schema Objects
  The Oracle-supplied PL/SQL package procedure `DBMS_METADATA.GET_DDL` lets you obtain metadata (in the form of DDL used to create the object) about a schema object.
- Schema Objects Data Dictionary Views
  These views display general information about schema objects.

17.13.1 Using a PL/SQL Package to Display Information About Schema Objects

The Oracle-supplied PL/SQL package procedure `DBMS_METADATA.GET_DDL` lets you obtain metadata (in the form of DDL used to create the object) about a schema object.

See Also:

Oracle Database PL/SQL Packages and Types Reference for a description of the `DBMS_METADATA` package

Example: Using the DBMS_METADATA Package

The `DBMS_METADATA` package is a powerful tool for obtaining the complete definition of a schema object. It enables you to obtain all of the attributes of an object in one pass. The object is described as DDL that can be used to (re)create it.

In the following statements the `GET_DDL` function is used to fetch the DDL for all tables in the current schema, filtering out nested tables and overflow segments. The `SET_TRANSFORM_PARAM` (with the handle value equal to `DBMS_METADATA.SESSION_TRANSFORM` meaning "for the current session") is used to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the session-level transform parameters are reset to their defaults. Once set, transform parameter values remain in effect until specifically reset to their defaults.

```
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
     DBMS_METADATA.SESSION_TRANSFORM,'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
     FROM USER_ALL_TABLES u
     WHERE u.nested='NO'
     AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
     DBMS_METADATA.SESSION_TRANSFORM,'DEFAULT');
```

The output from `DBMS_METADATA.GET_DDL` is a `LONG` data type. When using SQL*Plus, your output may be truncated by default. Issue the following SQL*Plus command before issuing the `DBMS_METADATA.GET_DDL` statement to ensure that your output is not truncated:

```
SQL> SET LONG 9999
```

17.13.2 Schema Objects Data Dictionary Views

These views display general information about schema objects.

View	Description

`DBA_OBJECTS`
`ALL_OBJECTS`
`USER_OBJECTS`
	`DBA` view describes all schema objects in the database. `ALL` view describes objects accessible to current user. `USER` view describes objects owned by the current user.

`DBA_CATALOG`
`ALL_CATALOG`
`USER_CATALOG`
	List the name, type, and owner (`USER` view does not display owner) for all tables, views, synonyms, and sequences in the database.

`DBA_DEPENDENCIES`
`ALL_DEPENDENCIES`
`USER_DEPENDENCIES`
	List all dependencies between procedures, packages, functions, package bodies, and triggers, including dependencies on views without any database links.

- Example 1: Displaying Schema Objects By Type
  You can query the `USER_OBJECTS` view to list all of the objects owned by the user issuing the query.
- Example 2: Displaying Dependencies of Views and Synonyms
  When you create a view or a synonym, the view or synonym is based on its underlying base object. The `ALL_DEPENDENCIES`, `USER_DEPENDENCIES`, and `DBA_DEPENDENCIES` data dictionary views can be used to reveal the dependencies for a view.

17.13.2.1 Example 1: Displaying Schema Objects By Type

You can query the `USER_OBJECTS` view to list all of the objects owned by the user issuing the query.

The following query lists all of the objects owned by the user issuing the query:

```
SELECT OBJECT_NAME, OBJECT_TYPE
    FROM USER_OBJECTS;
```

The following is the query output:

```
OBJECT_NAME                  OBJECT_TYPE
-------------------------   -------------------
EMP_DEPT                     CLUSTER
EMP                          TABLE
DEPT                         TABLE
EMP_DEPT_INDEX               INDEX
PUBLIC_EMP                   SYNONYM
EMP_MGR                      VIEW
```

17.13.2.2 Example 2: Displaying Dependencies of Views and Synonyms

When you create a view or a synonym, the view or synonym is based on its underlying base object. The `ALL_DEPENDENCIES`, `USER_DEPENDENCIES`, and `DBA_DEPENDENCIES` data dictionary views can be used to reveal the dependencies for a view.

The `ALL_SYNONYMS`, `USER_SYNONYMS`, and `DBA_SYNONYMS` data dictionary views can be used to list the base object of a synonym. For example, the following query lists the base objects for the synonyms created by user `jward`:

```
SELECT TABLE_OWNER, TABLE_NAME, SYNONYM_NAME
    FROM DBA_SYNONYMS
    WHERE OWNER = 'JWARD';
```

The following is the query output:

```
TABLE_OWNER            TABLE_NAME  SYNONYM_NAME
---------------------- ----------- -----------------
SCOTT                  DEPT        DEPT
SCOTT                  EMP         EMP
```

18

Managing Space for Schema Objects

Managing space for schema objects involves tasks such as managing tablespace alerts and space allocation, reclaiming unused space, dropping unused object storage, monitoring space usage, and capacity planning.

- Managing Tablespace Alerts
  Oracle Database provides proactive help in managing disk space for tablespaces by alerting you when available space is running low.
- Managing Resumable Space Allocation
  You can suspend, and later resume, the execution of large database operations.
- Reclaiming Unused Space
  You can reclaim unused space. Segment Advisor, is an Oracle Database component that identifies segments that have space available for reclamation.
- Dropping Unused Object Storage
  The `DBMS_SPACE_ADMIN` package includes the `DROP_EMPTY_SEGMENTS` procedure, which enables you to drop segments for empty tables and partitions that have been migrated from previous releases. This includes segments of dependent objects of the table, such as index segments, where possible.
- Understanding Space Usage of Data Types
  When creating tables and other data structures, you must know how much space they will require. Each data type has different space requirements.
- Displaying Information About Space Usage for Schema Objects
  Oracle Database provides data dictionary views and PL/SQL packages that allow you to display information about the space usage of schema objects.
- Capacity Planning for Database Objects
  Oracle Database provides two ways to plan capacity for database objects: with Cloud Control or with the `DBMS_SPACE` PL/SQL package. Three procedures in the `DBMS_SPACE` package enable you to predict the size of new objects and monitor the size of existing database objects.

18.1 Managing Tablespace Alerts

Oracle Database provides proactive help in managing disk space for tablespaces by alerting you when available space is running low.

- About Managing Tablespace Alerts
  Two alert thresholds are defined by default: **warning** and **critical**. The warning threshold is the limit at which space is beginning to run low. The critical threshold is a serious limit that warrants your immediate attention. The database issues alerts at both thresholds.
- Setting Alert Thresholds
  For each tablespace, you can set just percent-full thresholds, just free-space-remaining thresholds, or both types of thresholds simultaneously. Setting either type of threshold to zero disables it.

- Viewing Alerts
  You view alerts by accessing a Database Home page in Cloud Control and viewing the Incidents and Problems section.
- Limitations
  Threshold-based alerts have the some limitations.

18.1.1 About Managing Tablespace Alerts

Two alert thresholds are defined by default: **warning** and **critical**. The warning threshold is the limit at which space is beginning to run low. The critical threshold is a serious limit that warrants your immediate attention. The database issues alerts at both thresholds.

There are two ways to specify alert thresholds for both locally managed and dictionary managed tablespaces:

- By percent full

  For both warning and critical thresholds, when space used becomes greater than or equal to a percent of total space, an alert is issued.
- By free space remaining (in kilobytes (KB))

  For both warning and critical thresholds, when remaining space falls below an amount in KB, an alert is issued. Free-space-remaining thresholds are more useful for very large tablespaces.

Alerts for locally managed tablespaces are server-generated. For dictionary managed tablespaces, Oracle Enterprise Manager Cloud Control (Cloud Control) provides this functionality. See "Monitoring a Database with Server-Generated Alerts" for more information.

New tablespaces are assigned alert thresholds as follows:

- **Locally managed tablespace**—When you create a new locally managed tablespace, it is assigned the default threshold values defined for the database. A newly created database has a default of 85% full for the warning threshold and 97% full for the critical threshold. Defaults for free space remaining thresholds for a new database are both zero (disabled). You can change these database defaults, as described later in this section.
- **Dictionary managed tablespace**—When you create a new dictionary managed tablespace, it is assigned the threshold values that Cloud Control lists for "All others" in the metrics categories "Tablespace Free Space (MB) (dictionary managed)" and "Tablespace Space Used (%) (dictionary managed)." You change these values on the Metric and Policy Settings page.

> **Note:**
>
> In a database that is upgraded from Oracle 9*i* or earlier to Oracle Database 10*g* or later, database defaults for all locally managed tablespace alert thresholds are set to zero. This setting effectively disables the alert mechanism to avoid excessive alerts in a newly migrated database.

18.1.2 Setting Alert Thresholds

For each tablespace, you can set just percent-full thresholds, just free-space-remaining thresholds, or both types of thresholds simultaneously. Setting either type of threshold to zero disables it.

The ideal setting for the warning threshold is one that issues an alert early enough for you to resolve the problem before it becomes critical. The critical threshold should be one that issues an alert still early enough so that you can take immediate action to avoid loss of service.

To set alert threshold values for locally managed tablespaces:

- Do one of the following:
  - Use the Tablespaces page of Cloud Control.

    See the Cloud Control online help for information about changing the space usage alert thresholds for a tablespace.
  - Use the `DBMS_SERVER_ALERT.SET_THRESHOLD` package procedure.

    See *Oracle Database PL/SQL Packages and Types Reference* for details.

To set alert threshold values for dictionary managed tablespaces:

- Use the Tablespaces page of Cloud Control.

  See the Cloud Control online help for information about changing the space usage alert thresholds for a tablespace.

Example - Setting an Alert Threshold with Cloud Control

You receive an alert in Cloud Control when a space usage threshold for a tablespace is reached. There are two types of space usage alerts that you can enable: **warning**, for when tablespace space is somewhat low, and **critical**, for when the tablespace is almost completely full and action must be taken immediately.

For both warning and critical alerts, you can specify alert thresholds in the following ways:

- By space used (%)

  When space used becomes greater than or equal to a percentage of total space, an alert is issued.
- By free space (MB)

  When remaining space falls below an amount (in MB), an alert is issued.

  Free-space thresholds are more useful for large tablespaces. For example, for a 10 TB tablespace, setting the percentage full critical alert to as high as 99 percent means that the database would issue an alert when there is still 100 GB of free space remaining. Usually, 100 GB remaining would not be a critical situation, and the alert would not be useful. For this tablespace, it might be better to use a free-space threshold, which you could set to issue a critical alert when 5 GB of free space remains.

For both warning and critical alerts for a tablespace, you can enable either the space used threshold or the free-space threshold, or you can enable both thresholds.

To change space usage alert thresholds for tablespaces:

1. Go to the Database Home page.

2. From the **Administration** menu, select **Storage**, then **Tablespaces**.

   The Tablespaces page appears.
3. Select the tablespace whose threshold you want to change, and then click **Edit**.

   The Edit Tablespace page appears, showing the General subpage.
4. Click the **Thresholds** tab at the top of the page to display the Thresholds subpage.
5. In the Space Used (%) section, do one of the following:
   - Accept the default thresholds.
   - Select **Specify Thresholds**, and then enter a **Warning (%)** threshold and a **Critical (%)** threshold.
   - Select **Disable Thresholds** to disable the percentage full thresholds.
6. In the Free Space (MB) section, do one of the following:
   - Accept the default thresholds.
   - Select **Specify Thresholds**, and then enter a **Warning (MB)** threshold and a **Critical (MB)** threshold.
   - Select **Disable Thresholds** to disable the threshold for free space remaining.
7. Click **Apply**.

   A confirmation message appears.

Example—Setting an Alert Threshold Value with a Package Procedure

The following example sets the free-space-remaining thresholds in the USERS tablespace to 10 MB (warning) and 2 MB (critical), and disables the percent-full thresholds. The USERS tablespace is a locally managed tablespace.

```
BEGIN
DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id              => DBMS_SERVER_ALERT.TABLESPACE_BYT_FREE,
   warning_operator        => DBMS_SERVER_ALERT.OPERATOR_LE,
   warning_value           => '10240',
   critical_operator       => DBMS_SERVER_ALERT.OPERATOR_LE,
   critical_value          => '2048',
   observation_period      => 1,
   consecutive_occurrences => 1,
   instance_name           => NULL,
   object_type             => DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
   object_name             => 'USERS');

DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id              => DBMS_SERVER_ALERT.TABLESPACE_PCT_FULL,
   warning_operator        => DBMS_SERVER_ALERT.OPERATOR_GT,
   warning_value           => '0',
   critical_operator       => DBMS_SERVER_ALERT.OPERATOR_GT,
   critical_value          => '0',
   observation_period      => 1,
   consecutive_occurrences => 1,
   instance_name           => NULL,
   object_type             => DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
   object_name             => 'USERS');
END;
/
```

Note:

When setting nonzero values for percent-full thresholds, use the greater-than-or-equal-to operator, `OPERATOR_GE`.

Restoring a Tablespace to Database Default Thresholds

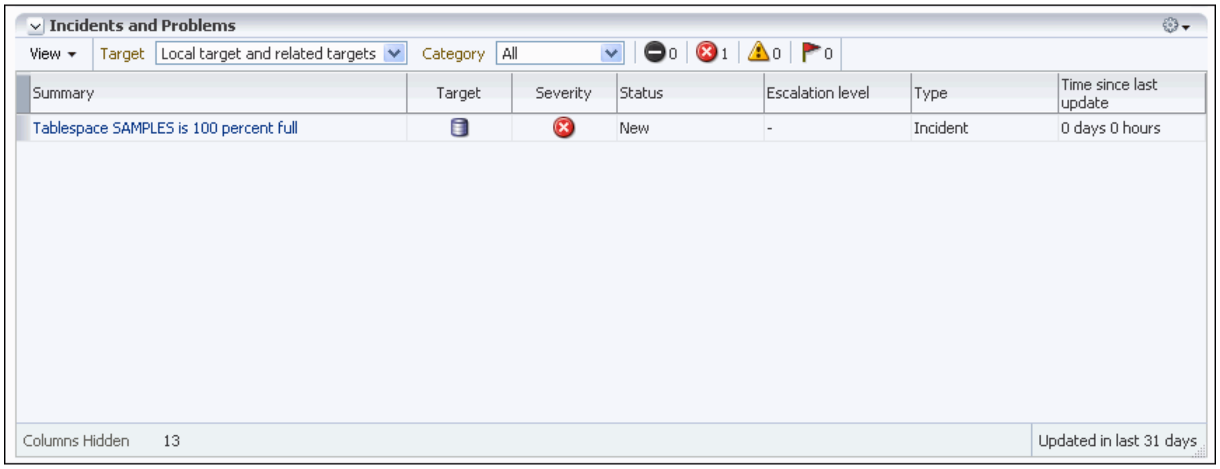
After explicitly setting values for locally managed tablespace alert thresholds, you can cause the values to revert to the database defaults by setting them to `NULL` with `DBMS_SERVER_ALERT.SET_THRESHOLD`.

Modifying Database Default Thresholds

To modify database default thresholds for locally managed tablespaces, invoke `DBMS_SERVER_ALERT.SET_THRESHOLD` as shown in the previous example, but set `object_name` to `NULL`. All tablespaces that use the database default are then switched to the new default.

18.1.3 Viewing Alerts

You view alerts by accessing a Database Home page in Cloud Control and viewing the Incidents and Problems section.



Summary	Target	Severity	Status	Escalation level	Type	Time since last update
Tablespace SAMPLES is 100 percent full			New	-	Incident	0 days 0 hours

You can also view alerts for locally managed tablespaces with the `DBA_OUTSTANDING_ALERTS` view. See "Server-Generated Alerts Data Dictionary Views" for more information.

18.1.4 Limitations

Threshold-based alerts have the some limitations.

These limitations include the following:

- Alerts are not issued for locally managed tablespaces that are offline or in read-only mode. However, the database reactivates the alert system for such tablespaces after they become read/write or available.
- When you take a tablespace offline or put it in read-only mode, you should disable the alerts for the tablespace by setting the thresholds to zero. You can then reenable the alerts

by resetting the thresholds when the tablespace is once again online and in read/write mode.

See Also:

- "Monitoring a Database with Server-Generated Alerts" for additional information on server-generated alerts in general
- *Oracle Database PL/SQL Packages and Types Reference* for information on the procedures of the `DBMS_SERVER_ALERT` package and how to use them
- "Reclaiming Unused Space" for various ways to reclaim space that is no longer being used in the tablespace
- "Purging Objects in the Recycle Bin" for information on reclaiming recycle bin space

18.2 Managing Resumable Space Allocation

You can suspend, and later resume, the execution of large database operations.

- Resumable Space Allocation Overview
  Oracle Database provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. Therefore, you can take corrective action instead of the Oracle Database server returning an error to the user. After the error condition is corrected, the suspended operation automatically resumes. This feature is called **resumable space allocation**. The statements that are affected are called resumable statements.
- Enabling and Disabling Resumable Space Allocation
  You enable and disable resumable space allocation by running SQL statements and setting certain initialization parameters.
- Using a LOGON Trigger to Set Default Resumable Mode
  Another method of setting default resumable mode, other than setting the `RESUMABLE_TIMEOUT` initialization parameter, is that you can register a database level `LOGON` trigger to alter a user's session to enable resumable and set a timeout interval.
- Detecting Suspended Statements
  When a resumable statement is suspended, the error is not raised to the client. In order for corrective action to be taken, Oracle Database provides alternative methods for notifying users of the error and for providing information about the circumstances.
- Operation-Suspended Alert
  When a resumable session is suspended, an operation-suspended alert is issued on the object that needs allocation of resource for the operation to complete.
- Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger
  An example illustrates how to create a system wide `AFTER SUSPEND` trigger and register it as user `SYS` at the database level.

18.2.1 Resumable Space Allocation Overview

Oracle Database provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. Therefore, you can take corrective action instead of the Oracle Database server returning an error to the user. After the error condition is corrected, the suspended operation automatically resumes. This feature is called **resumable space allocation**. The statements that are affected are called resumable statements.

- How Resumable Space Allocation Works
  An overview shows how resumable space allocation works.
- What Operations are Resumable?
  Some operations are resumable.
- What Errors are Correctable?
  Some errors are correctable.
- Resumable Space Allocation and Distributed Operations
  In a distributed environment, if a user enables or disables resumable space allocation, or a DBA alters the `RESUMABLE_TIMEOUT` initialization parameter, then the local instance is affected. `RESUMABLE` cannot be enabled remotely.
- Parallel Execution and Resumable Space Allocation
  In parallel execution, if one of the parallel execution server processes encounters a correctable error, then that server process suspends its execution.

18.2.1.1 How Resumable Space Allocation Works

An overview shows how resumable space allocation works.

1. A statement executes in resumable mode only if its session has been enabled for resumable space allocation by one of the following actions:
   - The `ALTER SESSION ENABLE RESUMABLE` statement is issued in the session before the statement executes when the `RESUMABLE_TIMEOUT` initialization parameter is set to a nonzero value.
   - The `ALTER SESSION ENABLE RESUMABLE TIMEOUT` *`timeout_value`* statement is issued in the session before the statement executes, and the *`timeout_value`* is a nonzero value.
2. A resumable statement is suspended when one of the following conditions occur (these conditions result in corresponding errors being signalled for non-resumable statements):
   - Out of space condition
   - Maximum extents reached condition
   - Space quota exceeded condition.
3. When the execution of a resumable statement is suspended, there are mechanisms to perform user supplied operations, log errors, and query the status of the statement execution. When a resumable statement is suspended the following actions are taken:
   - The error is reported in the alert log.
   - The system issues the Resumable Session Suspended alert.
   - If the user registered a trigger on the `AFTER SUSPEND` system event, the user trigger is executed. A user supplied PL/SQL procedure can access the error message data using the `DBMS_RESUMABLE` package and the `DBA_` or `USER_RESUMABLE` view.
4. Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held through a statement suspend and resume.

5. When the error condition is resolved (for example, as a result of user intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution and the Resumable Session Suspended alert is cleared.
6. A suspended statement can be forced to throw the exception using the `DBMS_RESUMABLE.ABORT()` procedure. This procedure can be called by a DBA, or by the user who issued the statement.
7. A suspension time out interval, specified by the `RESUMABLE_TIMEOUT` initialization parameter or by the timeout value in the `ALTER SESSION ENABLE RESUMABLE TIMEOUT` statement, is associated with resumable statements. A resumable statement that is suspended for the timeout interval wakes up and returns the exception to the user if the error condition is not resolved within the timeout interval.
8. A resumable statement can be suspended and resumed multiple times during execution.

18.2.1.2 What Operations are Resumable?

Some operations are resumable.

The following operations are resumable:

- Queries

  `SELECT` statements that run out of temporary space (for sort areas) are candidates for resumable execution. When using OCI, the calls `OCIStmtExecute()` and `OCIStmtFetch()` are candidates.
- DML

  `INSERT`, `UPDATE`, and `DELETE` statements are candidates. The interface used to execute them does not matter; it can be OCI, PL/SQL, or another interface. Also, `INSERT INTO...SELECT` from external tables can be resumable.
- Import/Export

  As for SQL*Loader, a command line parameter controls whether statements are resumable after recoverable errors.
- DDL

  The following statements are candidates for resumable execution:
  - `CREATE TABLE` ... `AS SELECT`
  - `CREATE INDEX`
  - `ALTER INDEX` ... `REBUILD`
  - `ALTER TABLE` ... `MOVE PARTITION`
  - `ALTER TABLE` ... `SPLIT PARTITION`
  - `ALTER INDEX` ... `REBUILD PARTITION`
  - `ALTER INDEX` ... `SPLIT PARTITION`
  - `CREATE MATERIALIZED VIEW`
  - `CREATE MATERIALIZED VIEW LOG`

18.2.1.3 What Errors are Correctable?

Some errors are correctable.

There are three classes of correctable errors:

- Out of space condition

  The operation cannot acquire any more extents for a table/index/temporary segment/undo segment/cluster/LOB/table partition/index partition in a tablespace. For example, the following errors fall in this category:

  ```
  ORA-01653 unable to extend table ... in tablespace ...
  ORA-01654 unable to extend index ... in tablespace ...
  ```

- Maximum extents reached condition

  The number of extents in a table/index/temporary segment/undo segment/cluster/LOB/table partition/index partition equals the maximum extents defined on the object. For example, the following errors fall in this category:

  ```
  ORA-01631 max # extents ... reached in table ...
  ORA-01632 max # extents ... reached in index ...
  ```

- Space quota exceeded condition

  The user has exceeded his assigned space quota in the tablespace. Specifically, this is noted by the following error:

  ```
  ORA-01536 space quote exceeded for tablespace string
  ```

18.2.1.4 Resumable Space Allocation and Distributed Operations

In a distributed environment, if a user enables or disables resumable space allocation, or a DBA alters the `RESUMABLE_TIMEOUT` initialization parameter, then the local instance is affected. `RESUMABLE` cannot be enabled remotely.

In a distributed transaction, sessions on remote instances are suspended only if the remote instance has already enabled `RESUMABLE` on the instance or sessions at its site.

18.2.1.5 Parallel Execution and Resumable Space Allocation

In parallel execution, if one of the parallel execution server processes encounters a correctable error, then that server process suspends its execution.

Other parallel execution server processes will continue executing their respective tasks, until either they encounter an error or are blocked (directly or indirectly) by the suspended server process. When the correctable error is resolved, the suspended process resumes execution and the parallel operation continues execution. If the suspended operation is terminated, then the parallel operation terminates, throwing the error to the user.

Different parallel execution server processes may encounter one or more correctable errors. This may result in firing an `AFTER SUSPEND` trigger multiple times, in parallel. Also, if a parallel execution server process encounters a non-correctable error while another parallel execution server process is suspended, the suspended statement is immediately terminated.

For parallel execution, every parallel execution coordinator and server process has its own entry in the `DBA_` or `USER_RESUMABLE` view.

18.2.2 Enabling and Disabling Resumable Space Allocation

You enable and disable resumable space allocation by running SQL statements and setting certain initialization parameters.

- About Enabling and Disabling Resumable Space Allocation
  Resumable space allocation is only possible when statements are executed within a session that has resumable mode enabled.
- Setting the RESUMABLE_TIMEOUT Initialization Parameter
  You can specify a default system wide timeout interval by setting the `RESUMABLE_TIMEOUT` initialization parameter.
- Using ALTER SESSION to Enable and Disable Resumable Space Allocation
  Within a session, a user can issue the `ALTER SESSION SET` statement to set the `RESUMABLE_TIMEOUT` initialization parameter and enable resumable space allocation, change a timeout value, or to disable resumable mode.

18.2.2.1 About Enabling and Disabling Resumable Space Allocation

Resumable space allocation is only possible when statements are executed within a session that has resumable mode enabled.

Resumable space allocation is enabled for a session when the `ALTER SESSION ENABLE RESUMABLE` statement is executed, and the `RESUMABLE_TIMEOUT` initialization parameter is set to a non-zero value for the session. When the `RESUMABLE_TIMEOUT` initialization parameter is set at the system level, it is the default for an `ALTER SESSION ENABLE RESUMABLE` statement that does not specify a timeout value. When an `ALTER SESSION ENABLE RESUMABLE` statement specifies a timeout value, it overrides the system default.

Resumable space allocation is disabled for a session in all of the following cases when the `ALTER SESSION ENABLE RESUMABLE` statement is executed:

- The session does not execute an `ALTER SESSION ENABLE RESUMABLE` statement.
- The session executes an `ALTER SESSION DISABLE RESUMABLE` statement.
- The session executes an `ALTER SESSION ENABLE RESUMABLE` statement, and the timeout value is zero.

Note:

Because suspended statements can hold up some system resources, users must be granted the `RESUMABLE` system privilege before they are allowed to enable resumable space allocation and execute resumable statements.

18.2.2.2 Setting the RESUMABLE_TIMEOUT Initialization Parameter

You can specify a default system wide timeout interval by setting the `RESUMABLE_TIMEOUT` initialization parameter.

For example, the following setting of the `RESUMABLE_TIMEOUT` parameter in the initialization parameter file sets the timeout period to 1 hour:

```
RESUMABLE_TIMEOUT  = 3600
```

If this parameter is set to 0, then resumable space allocation is disabled even for sessions that run an `ALTER SESSION ENABLE RESUMABLE` statement without a timeout value.

You can also use the `ALTER SYSTEM SET` statement to change the value of this parameter at the system level. For example, the following statement disables resumable space allocation for all sessions that run an `ALTER SESSION ENABLE RESUMABLE` statement without a timeout value:

```
ALTER SYSTEM SET RESUMABLE_TIMEOUT=0;
```

18.2.2.3 Using ALTER SESSION to Enable and Disable Resumable Space Allocation

Within a session, a user can issue the `ALTER SESSION SET` statement to set the `RESUMABLE_TIMEOUT` initialization parameter and enable resumable space allocation, change a timeout value, or to disable resumable mode.

A user can enable resumable mode for a session with the default system `RESUMABLE_TIMEOUT` value using the following SQL statement:

```
ALTER SESSION ENABLE RESUMABLE;
```

To disable resumable mode, a user issues the following statement:

```
ALTER SESSION DISABLE RESUMABLE;
```

The default for a new session is resumable mode disabled.

The user can also specify a timeout interval, and can provide a name used to identify a resumable statement. These are discussed separately in following sections.

- Specifying a Timeout Interval
  When you enable resumable mode, you can set a timeout period, after which a suspended statement will error if no intervention has taken place.
- Naming Resumable Statements
  Resumable statements can be identified by name.

See Also:

"Using a LOGON Trigger to Set Default Resumable Mode"

18.2.2.3.1 Specifying a Timeout Interval

When you enable resumable mode, you can set a timeout period, after which a suspended statement will error if no intervention has taken place.

The following statement specifies that resumable transactions will time out and error after 3600 seconds:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;
```

The value of `TIMEOUT` remains in effect until it is changed by another `ALTER SESSION ENABLE RESUMABLE` statement, it is changed by another means, or the session ends. If the `RESUMABLE_TIMEOUT` initialization parameter is not set, then the default timeout interval when using the `ENABLE RESUMABLE TIMEOUT` clause to enable resumable mode is 7200 seconds.

See Also:

"Setting the RESUMABLE_TIMEOUT Initialization Parameter " for other methods of changing the timeout interval for resumable space allocation

18.2.2.3.2 Naming Resumable Statements

Resumable statements can be identified by name.

The following statement assigns a name to resumable statements:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600 NAME 'insert into table';
```

The `NAME` value remains in effect until it is changed by another `ALTER SESSION ENABLE RESUMABLE` statement, or the session ends. The default value for `NAME` is '`User` *`username`*(*`userid`*), `Session` *`sessionid`*, `Instance` *`instanceid`*'.

The name of the statement is used to identify the resumable statement in the `DBA_RESUMABLE` and `USER_RESUMABLE` views.

18.2.3 Using a LOGON Trigger to Set Default Resumable Mode

Another method of setting default resumable mode, other than setting the `RESUMABLE_TIMEOUT` initialization parameter, is that you can register a database level `LOGON` trigger to alter a user's session to enable resumable and set a timeout interval.

Note:

If there are multiple triggers registered that change default mode and timeout for resumable statements, the result will be unspecified because Oracle Database does not guarantee the order of trigger invocation.

18.2.4 Detecting Suspended Statements

When a resumable statement is suspended, the error is not raised to the client. In order for corrective action to be taken, Oracle Database provides alternative methods for notifying users of the error and for providing information about the circumstances.

- Notifying Users: The AFTER SUSPEND System Event and Trigger
  When a resumable statement encounters a correctable error, the system internally generates the `AFTER SUSPEND` system event. Users can register triggers for this event at both the database and schema level. If a user registers a trigger to handle this system event, the trigger is executed after a SQL statement has been suspended.
- Using Views to Obtain Information About Suspended Statements
  You can query a set of views for information about the status of resumable statements.
- Using the DBMS_RESUMABLE Package
  The `DBMS_RESUMABLE` package helps control resumable space allocation.

18.2.4.1 Notifying Users: The AFTER SUSPEND System Event and Trigger

When a resumable statement encounters a correctable error, the system internally generates the `AFTER SUSPEND` system event. Users can register triggers for this event at both the database and schema level. If a user registers a trigger to handle this system event, the trigger is executed after a SQL statement has been suspended.

SQL statements executed within a `AFTER SUSPEND` trigger are always non-resumable and are always autonomous. Transactions started within the trigger use the `SYSTEM` rollback segment. These conditions are imposed to overcome deadlocks and reduce the chance of the trigger experiencing the same error condition as the statement.

Users can use the `USER_RESUMABLE` or `DBA_RESUMABLE` views, or the `DBMS_RESUMABLE.SPACE_ERROR_INFO` function, within triggers to get information about the resumable statements.

Triggers can also call the `DBMS_RESUMABLE` package to terminate suspended statements and modify resumable timeout values. In the following example, the default system timeout is changed by creating a system wide `AFTER SUSPEND` trigger that calls `DBMS_RESUMABLE` to set the timeout to 3 hours:

```
CREATE OR REPLACE TRIGGER resumable_default_timeout
AFTER SUSPEND
ON DATABASE
BEGIN
   DBMS_RESUMABLE.SET_TIMEOUT(10800);
END;
/
```

See Also:

Oracle Database PL/SQL Language Reference for information about triggers and system events

18.2.4.2 Using Views to Obtain Information About Suspended Statements

You can query a set of views for information about the status of resumable statements.

View	Description
`DBA_RESUMABLE` `USER_RESUMABLE`	These views contain rows for all currently executing or suspended resumable statements. They can be used by a DBA, `AFTER SUSPEND` trigger, or another session to monitor the progress of, or obtain specific information about, resumable statements.
`V$SESSION_WAIT`	When a statement is suspended the session invoking the statement is put into a wait state. A row is inserted into this view for the session with the `EVENT` column containing "statement suspended, wait error to be cleared".

18.2.4.3 Using the DBMS_RESUMABLE Package

The `DBMS_RESUMABLE` package helps control resumable space allocation.

You can invoke the following procedures:

Procedure	Description
`ABORT(sessionID)`	This procedure terminates a suspended resumable statement. The parameter `sessionID` is the session ID in which the statement is executing. For parallel DML/DDL, `sessionID` is any session ID which participates in the parallel DML/DDL. Oracle Database guarantees that the `ABORT` operation always succeeds. It may be called either inside or outside of the `AFTER SUSPEND` trigger. The caller of `ABORT` must be the owner of the session with `sessionID`, have `ALTER SYSTEM` privilege, or have DBA privileges.
`GET_SESSION_TIMEOUT(sessionID)`	This function returns the current timeout value of resumable space allocation for the session with `sessionID`. This returned timeout is in seconds. If the session does not exist, this function returns -1.
`SET_SESSION_TIMEOUT(sessionID, timeout)`	This procedure sets the timeout interval of resumable space allocation for the session with `sessionID`. The parameter `timeout` is in seconds. The new `timeout` setting will applies to the session immediately. If the session does not exist, no action is taken.
`GET_TIMEOUT()`	This function returns the current `timeout` value of resumable space allocation for the current session. The returned value is in seconds.
`SET_TIMEOUT(timeout)`	This procedure sets a `timeout` value for resumable space allocation for the current session. The parameter `timeout` is in seconds. The new timeout setting applies to the session immediately.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details about the `DBMS_RESUMABLE` package.

18.2.5 Operation-Suspended Alert

When a resumable session is suspended, an operation-suspended alert is issued on the object that needs allocation of resource for the operation to complete.

Once the resource is allocated and the operation completes, the operation-suspended alert is cleared. See "Managing Tablespace Alerts" for more information on system-generated alerts.

18.2.6 Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger

An example illustrates how to create a system wide `AFTER SUSPEND` trigger and register it as user `SYS` at the database level.

Whenever a resumable statement is suspended in any session, this trigger can have either of two effects:

- If an undo segment has reached its space limit, then a message is sent to the DBA and the statement is terminated.
- If any other recoverable error has occurred, the timeout interval is reset to 8 hours.

Here are the statements for this example:

```
CREATE OR REPLACE TRIGGER resumable_default
AFTER SUSPEND
ON DATABASE
DECLARE
   /* declare transaction in this trigger is autonomous */
   /* this is not required because transactions within a trigger
      are always autonomous */
   PRAGMA AUTONOMOUS_TRANSACTION;
   cur_sid           NUMBER;
   cur_inst          NUMBER;
   errno             NUMBER;
   err_type          VARCHAR2;
   object_owner      VARCHAR2;
   object_type       VARCHAR2;
   table_space_name  VARCHAR2;
   object_name       VARCHAR2;
   sub_object_name   VARCHAR2;
   error_txt         VARCHAR2;
   msg_body          VARCHAR2;
   ret_value         BOOLEAN;
   mail_conn         UTL_SMTP.CONNECTION;
BEGIN
   -- Get session ID
   SELECT DISTINCT(SID) INTO cur_SID FROM V$MYSTAT;

   -- Get instance number
   cur_inst := userenv('instance');

   -- Get space error information
   ret_value :=
   DBMS_RESUMABLE.SPACE_ERROR_INFO(err_type,object_type,object_owner,
        table_space_name,object_name, sub_object_name);
   /*
   -- If the error is related to undo segments, log error, send email
   -- to DBA, and terminate the statement. Otherwise, set timeout to 8 hours.
   --
   -- sys.rbs_error is a table which is to be
   -- created by a DBA manually and defined as
   -- (sql_text VARCHAR2(1000), error_msg VARCHAR2(4000),
   -- suspend_time DATE)
   */

   IF OBJECT_TYPE = 'UNDO SEGMENT' THEN
       /* LOG ERROR */
```

```
            INSERT INTO sys.rbs_error (
                SELECT SQL_TEXT, ERROR_MSG, SUSPEND_TIME
                FROM DBMS_RESUMABLE
                WHERE SESSION_ID = cur_sid AND INSTANCE_ID = cur_inst
             );
            SELECT ERROR_MSG INTO error_txt FROM DBMS_RESUMABLE
                WHERE SESSION_ID = cur_sid and INSTANCE_ID = cur_inst;

             -- Send email to receipient through UTL_SMTP package
             msg_body:='Subject: Space Error Occurred

                        Space limit reached for undo segment ' || object_name ||
                        on ' || TO_CHAR(SYSDATE, 'Month dd, YYYY, HH:MIam') ||
                        '. Error message was ' || error_txt;

             mail_conn := UTL_SMTP.OPEN_CONNECTION('localhost', 25);
             UTL_SMTP.HELO(mail_conn, 'localhost');
             UTL_SMTP.MAIL(mail_conn, 'sender@localhost');
             UTL_SMTP.RCPT(mail_conn, 'recipient@localhost');
             UTL_SMTP.DATA(mail_conn, msg_body);
             UTL_SMTP.QUIT(mail_conn);

             -- Terminate the statement
             DBMS_RESUMABLE.ABORT(cur_sid);
         ELSE
             -- Set timeout to 8 hours
             DBMS_RESUMABLE.SET_TIMEOUT(28800);
         END IF;

         /* commit autonomous transaction */
         COMMIT;
     END;
     /
```

18.3 Reclaiming Unused Space

You can reclaim unused space. Segment Advisor, is an Oracle Database component that identifies segments that have space available for reclamation.

- About Reclaimable Unused Space
  Over time, updates and deletes on objects within a tablespace can create pockets of empty space that individually are not large enough to be reused for new data. This type of empty space is referred to as fragmented free space.
- The Segment Advisor
  The Segment Advisor identifies segments that have space available for reclamation.
- Shrinking Database Segments Online
  You use online segment shrink to reclaim fragmented free space below the high water mark in an Oracle Database segment.
- Deallocating Unused Space
  When you deallocate unused space, the database frees the unused space at the unused (high water mark) end of the database segment and makes the space available for other segments in the tablespace.

18.3.1 About Reclaimable Unused Space

Over time, updates and deletes on objects within a tablespace can create pockets of empty space that individually are not large enough to be reused for new data. This type of empty space is referred to as fragmented free space.

Objects with fragmented free space can result in much wasted space, and can impact database performance. The preferred way to defragment and reclaim this space is to perform an **online segment shrink**. This process consolidates fragmented free space below the high water mark and compacts the segment. After compaction, the high water mark is moved, resulting in new free space above the high water mark. That space above the high water mark is then deallocated. The segment remains available for queries and DML during most of the operation, and no extra disk space need be allocated.

You use the **Segment Advisor** to identify segments that would benefit from online segment shrink. Only segments in locally managed tablespaces with automatic segment space management (ASSM) are eligible. Other restrictions on segment type exist. For more information, see "Shrinking Database Segments Online".

If a table with reclaimable space is not eligible for online segment shrink, or if you want to make changes to logical or physical attributes of the table while reclaiming space, then you can use **online table redefinition** as an alternative to segment shrink. Online redefinition is also referred to as **reorganization**. Unlike online segment shrink, it requires extra disk space to be allocated. See "Redefining Tables Online" for more information.

18.3.2 The Segment Advisor

The Segment Advisor identifies segments that have space available for reclamation.

- About the Segment Advisor
  The Segment Advisor performs its analysis by examining usage and growth statistics in the Automatic Workload Repository (AWR), and by sampling the data in the segment.
- Using the Segment Advisor
  To use the Segment Advisor, check the results of Automatic Segment Advisor, and, optionally, run the Segment Advisor manually.
- Automatic Segment Advisor
  The Automatic Segment Advisor is an automated maintenance task that is configured to run during all maintenance windows.
- Running the Segment Advisor Manually
  You can manually run the Segment Advisor at any time with Cloud Control or with PL/SQL package procedure calls.
- Viewing Segment Advisor Results
  The Segment Advisor creates several types of results: recommendations, findings, actions, and objects.
- Configuring the Automatic Segment Advisor
  The Automatic Segment Advisor is an automated maintenance task. As such, you can use Cloud Control or PL/SQL package procedure calls to modify when (and if) this task runs. You can also control the resources allotted to it by modifying the appropriate resource plans.
- Viewing Automatic Segment Advisor Information
  You can query views to display information specific to the Automatic Segment Advisor.

18.3.2.1 About the Segment Advisor

The Segment Advisor performs its analysis by examining usage and growth statistics in the Automatic Workload Repository (AWR), and by sampling the data in the segment.

It is configured to run during maintenance windows as an automated maintenance task, and you can also run it on demand (manually). The Segment Advisor automated maintenance task is known as the Automatic Segment Advisor. You can use this information for capacity planning and for arriving at an informed decision about which segments to shrink.

The Segment Advisor generates the following types of advice:

- If the Segment Advisor determines that an object has a significant amount of free space, it recommends online segment shrink. If the object is a table that is not eligible for shrinking, as in the case of a table in a tablespace without automatic segment space management, the Segment Advisor recommends online table redefinition.
- If the Segment Advisor determines that a table could benefit from compression with the advanced row compression method, it makes a recommendation to that effect. (Automatic Segment Advisor only. See "Automatic Segment Advisor".)
- If the Segment Advisor encounters a table with row chaining above a certain threshold, it records that fact that the table has an excess of chained rows.

> **Note:**
>
> The Segment Advisor flags only the type of row chaining that results from updates that increase row length.

If you receive a space management alert, or if you decide that you want to reclaim space, you should start with the Segment Advisor.

18.3.2.2 Using the Segment Advisor

To use the Segment Advisor, check the results of Automatic Segment Advisor, and, optionally, run the Segment Advisor manually.

To use the Segment Advisor:

1. Check the results of the Automatic Segment Advisor.

   To understand the Automatic Segment Advisor, see "Automatic Segment Advisor", later in this section. For details on how to view results, see "Viewing Segment Advisor Results".
2. (Optional) Obtain updated results on individual segments by rerunning the Segment Advisor manually.

   See "Running the Segment Advisor Manually", later in this section.

18.3.2.3 Automatic Segment Advisor

The Automatic Segment Advisor is an automated maintenance task that is configured to run during all maintenance windows.

The Automatic Segment Advisor does not analyze every database object. Instead, it examines database statistics, samples segment data, and then selects the following objects to analyze:

- Tablespaces that have exceeded a critical or warning space threshold
- Segments that have the most activity
- Segments that have the highest growth rate

In addition, the Automatic Segment Advisor evaluates tables that are at least 10MB and that have at least three indexes to determine the amount of space saved if the tables are compressed with the advanced row compression method.

If an object is selected for analysis but the maintenance window expires before the Segment Advisor can process the object, the object is included in the next Automatic Segment Advisor run.

You cannot change the set of tablespaces and segments that the Automatic Segment Advisor selects for analysis. You can, however, enable or disable the Automatic Segment Advisor task, change the times during which the Automatic Segment Advisor is scheduled to run, or adjust automated maintenance task system resource utilization. See "Configuring the Automatic Segment Advisor" for more information.

See Also:

- "Viewing Segment Advisor Results"
- Managing Automated Database Maintenance Tasks
- "Consider Using Table Compression" for more information on advanced row compression

18.3.2.4 Running the Segment Advisor Manually

You can manually run the Segment Advisor at any time with Cloud Control or with PL/SQL package procedure calls.

Reasons to manually run the Segment Advisor include the following:

- You want to analyze a tablespace or segment that was not selected by the Automatic Segment Advisor.
- You want to repeat the analysis of an individual tablespace or segment to get more up-to-date recommendations.

You can request advice from the Segment Advisor at three levels:

- **Segment level**—Advice is generated for a single segment, such as an unpartitioned table, a partition or subpartition of a partitioned table, an index, or a LOB column.
- **Object level**—Advice is generated for an entire object, such as a table or index. If the object is partitioned, advice is generated on all the partitions of the object. In addition, if you run Segment Advisor manually from Cloud Control, you can request advice on the object's dependent objects, such as indexes and LOB segments for a table.
- **Tablespace level**—Advice is generated for every segment in a tablespace.

The `OBJECT_TYPE` column of Table 18-2 shows the types of objects for which you can request advice.

- Running the Segment Advisor Manually with Cloud Control
  You can run the Segment Advisor manually with Cloud Control

- Running the Segment Advisor Manually with PL/SQL
  You can run the Segment Advisor with the `DBMS_ADVISOR` package.

18.3.2.4.1 Running the Segment Advisor Manually with Cloud Control

You can run the Segment Advisor manually with Cloud Control

You must have the `OEM_ADVISOR` role to run the Segment Advisor manually with Cloud Control. There are two ways to run the Segment Advisor:

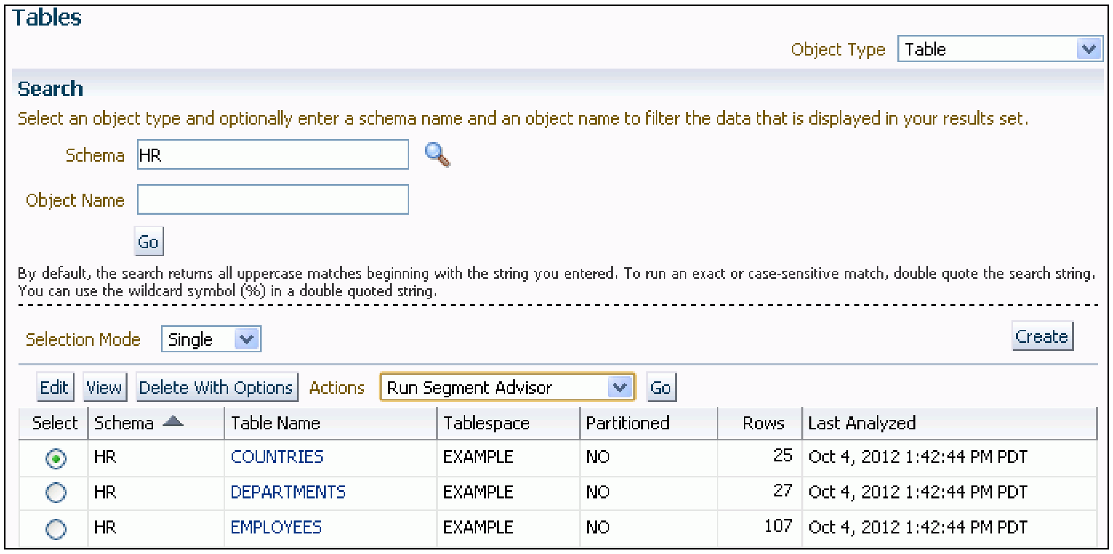
- Using the Segment Advisor Wizard

  This method enables you to request advice at the tablespace level or object level. At the object level, you can request advice on tables, indexes, table partitions, and index partitions.

- Using the Run Segment Advisor command on a schema object page.

  For example, if you display a list of tables on the Tables page (accessible from the Schema menu), you can select a table and then select **Run Segment Advisor** from the Actions menu.

Figure 18-1 Tables page



Tables

Object Type: Table

Search

Select an object type and optionally enter a schema name and an object name to filter the data that is displayed in your results set.

Schema: HR

Object Name:

By default, the search returns all uppercase matches beginning with the string you entered. To run an exact or case-sensitive match, double quote the search string. You can use the wildcard symbol (%) in a double quoted string.

Selection Mode: Single

Actions: Run Segment Advisor

Select	Schema	Table Name	Tablespace	Partitioned	Rows	Last Analyzed
●	HR	COUNTRIES	EXAMPLE	NO	25	Oct 4, 2012 1:42:44 PM PDT
○	HR	DEPARTMENTS	EXAMPLE	NO	27	Oct 4, 2012 1:42:44 PM PDT
○	HR	EMPLOYEES	EXAMPLE	NO	107	Oct 4, 2012 1:42:44 PM PDT

This method enables you to include the schema object's dependent objects in the Segment Advisor run. For example, if you select a table and select **Run Segment Advisor**, Cloud Control displays the table's dependent objects, such as partitions, index segments, LOB segments, and so on. You can then select dependent objects to include in the run.

In both cases, Cloud Control creates the Segment Advisor task as an Oracle Database Scheduler job. You can schedule the job to run immediately, or can take advantage of advanced scheduling features offered by the Scheduler.

To run the Segment Advisor manually with the Segment Advisor Wizard:

1. Access the Database Home page.
2. From the Performance menu, select **Advisors Home**.

   The Advisor Central page appears. (See Figure 18-2.)

3. Under Advisors, click **Segment Advisor**.

   The first page of the Segment Advisor wizard appears.

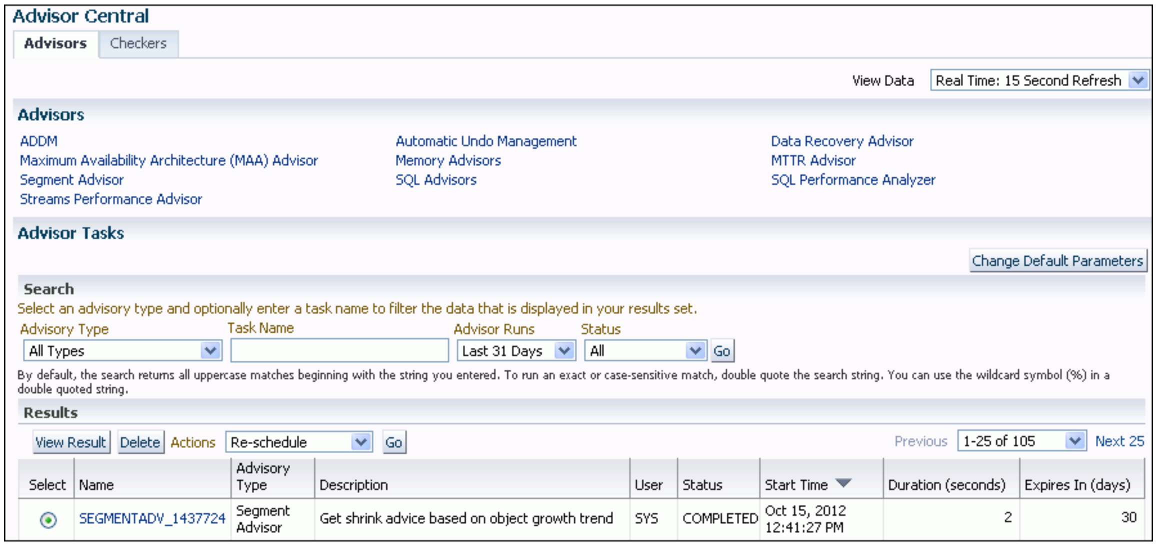
4. Follow the wizard steps to schedule the Segment Advisor job, and then click **Submit** on the final wizard page.

   The Advisor Central page reappears, with the new Segment Advisor job at the top of the list under the Results heading. The job status is `SCHEDULED` or `RUNNING`. (If you do not see your job, then use the search fields above the list to display it.)

5. Check the status of the job. If it is not `COMPLETED`, then use the **Refresh** control at the top of the page to refresh the page. (Do not use your browser's Refresh icon.)

   When the job status changes to `COMPLETED`, select the job by clicking in the **Select** column, and then click **View Result**.

Figure 18-2 Advisor Central page



Advisor Central

Advisors | Checkers

View Data: Real Time: 15 Second Refresh

Advisors

ADDM
Maximum Availability Architecture (MAA) Advisor
Segment Advisor
Streams Performance Advisor
Automatic Undo Management
Memory Advisors
SQL Advisors
Data Recovery Advisor
MTTR Advisor
SQL Performance Analyzer

Advisor Tasks

Change Default Parameters

Search

Select an advisory type and optionally enter a task name to filter the data that is displayed in your results set.

Advisory Type	Task Name	Advisor Runs	Status
All Types		Last 31 Days	All

Go

By default, the search returns all uppercase matches beginning with the string you entered. To run an exact or case-sensitive match, double quote the search string. You can use the wildcard symbol (%) in a double quoted string.

Results

View Result | Delete | Actions: Re-schedule | Go | Previous | 1-25 of 105 | Next 25

Select	Name	Advisory Type	Description	User	Status	Start Time	Duration (seconds)	Expires In (days)
●	SEGMENTADV_1437724	Segment Advisor	Get shrink advice based on object growth trend	SYS	COMPLETED	Oct 15, 2012 12:41:27 PM	2	30

See Also:

Scheduling Jobs with Oracle Scheduler for more information about the advanced scheduling features of the Scheduler.

18.3.2.4.2 Running the Segment Advisor Manually with PL/SQL

You can run the Segment Advisor with the `DBMS_ADVISOR` package.

You use package procedures to create a Segment Advisor task, set task arguments, and then execute the task. You must have the `ADVISOR` privilege. Table 18-1 shows the procedures that are relevant for the Segment Advisor. See *Oracle Database PL/SQL Packages and Types Reference* for more details on these procedures.

Table 18-1 DBMS_ADVISOR package procedures relevant to the Segment Advisor

Package Procedure Name	Description
`CREATE_TASK`	Use this procedure to create the Segment Advisor task. Specify 'Segment Advisor' as the value of the `ADVISOR_NAME` parameter.
`CREATE_OBJECT`	Use this procedure to identify the target object for segment space advice. The parameter values of this procedure depend upon the object type. Table 18-2 lists the parameter values for each type of object. Note: To request advice on an IOT overflow segment, use an object type of `TABLE`, `TABLE PARTITION`, or `TABLE SUBPARTITION`. Use the following query to find the overflow segment for an IOT and to determine the overflow segment table name to use with `CREATE_OBJECT`: `select table_name, iot_name, iot_type from dba_tables;`
`SET_TASK_PARAMETER`	Use this procedure to describe the segment advice that you need. Table 18-3 shows the relevant input parameters of this procedure. Parameters not listed here are not used by the Segment Advisor.
`EXECUTE_TASK`	Use this procedure to execute the Segment Advisor task.

Table 18-2 Input Parameters for DBMS_ADVISOR.CREATE_OBJECT

OBJECT_TYPE	ATTR1	ATTR2	ATTR3	ATTR4
`TABLESPACE`	*tablespace name*	`NULL`	`NULL`	Unused. Specify `NULL`.
`TABLE`	*schema name*	*table name*	`NULL`	Unused. Specify `NULL`.
`INDEX`	*schema name*	*index name*	`NULL`	Unused. Specify `NULL`.
`TABLE PARTITION`	*schema name*	*table name*	*table partition name*	Unused. Specify `NULL`.
`INDEX PARTITION`	*schema name*	*index name*	*index partition name*	Unused. Specify `NULL`.
`TABLE SUBPARTITION`	*schema name*	*table name*	*table subpartition name*	Unused. Specify `NULL`.
`INDEX SUBPARTITION`	*schema name*	*index name*	*index subpartition name*	Unused. Specify `NULL`.
`LOB`	*schema name*	*segment name*	`NULL`	Unused. Specify `NULL`.
`LOB PARTITION`	*schema name*	*segment name*	*lob partition name*	Unused. Specify `NULL`.
`LOB SUBPARTITION`	*schema name*	*segment name*	*lob subpartition name*	Unused. Specify `NULL`.

Table 18-3   Input for DBMS_ADVISOR.SET_TASK_PARAMETER

Input Parameter	Description	Possible Values	Default Value
`time_limit`	The time limit for the Segment Advisor run, specified in seconds.	Any number of seconds	`UNLIMITED`
`recommend_all`	Whether the Segment Advisor should generate findings for all segments.	`TRUE`: Findings are generated on all segments specified, whether or not space reclamation is recommended. `FALSE`: Findings are generated only for those objects that generate recommendations for space reclamation.	`TRUE`

Example

The example that follows shows how to use the `DBMS_ADVISOR` procedures to run the Segment Advisor for the sample table `hr.employees`. The user executing these package procedures must have the `EXECUTE` object privilege on the package or the `ADVISOR` system privilege.

Note that passing an object type of `TABLE` to `DBMS_ADVISOR.CREATE_OBJECT` amounts to an object level request. If the table is not partitioned, the table segment is analyzed (without any dependent segments like index or LOB segments). If the table is partitioned, the Segment Advisor analyzes all table partitions and generates separate findings and recommendations for each.

```
variable id number;
begin
  declare
  name varchar2(100);
  descr varchar2(500);
  obj_id number;
  begin
  name:='Manual_Employees';
  descr:='Segment Advisor Example';

  dbms_advisor.create_task (
    advisor_name      => 'Segment Advisor',
    task_id           => :id,
    task_name         => name,
    task_desc         => descr);

  dbms_advisor.create_object (
    task_name         => name,
    object_type       => 'TABLE',
    attr1             => 'HR',
    attr2             => 'EMPLOYEES',
    attr3             => NULL,
    attr4             => NULL,
    attr5             => NULL,
    object_id         => obj_id);

  dbms_advisor.set_task_parameter(
    task_name         => name,
    parameter         => 'recommend_all',
    value             => 'TRUE');
  dbms_advisor.execute_task(name);
  end;
end;
/
```

18.3.2.5 Viewing Segment Advisor Results

The Segment Advisor creates several types of results: recommendations, findings, actions, and objects.

You can view results in the following ways:

- With Cloud Control
- By querying the `DBA_ADVISOR_*` views
- By calling the `DBMS_SPACE.ASA_RECOMMENDATIONS` function

Table 18-4 describes the various result types and their associated `DBA_ADVISOR_*` views.

Table 18-4 Segment Advisor Result Types

Result Type	Associated View	Description
Recommendations	`DBA_ADVISOR_RECOMMENDATIONS`	If a segment would benefit from a segment shrink, reorganization, or compression, the Segment Advisor generates a recommendation for the segment. Table 18-5 shows examples of generated findings and recommendations.
Findings	`DBA_ADVISOR_FINDINGS`	Findings are a report of what the Segment Advisor observed in analyzed segments. Findings include space used and free space statistics for each analyzed segment. Not all findings result in a recommendation. (There may be only a few recommendations, but there could be many findings.) When running the Segment Advisor manually with PL/SQL, if you specify `'TRUE'` for `recommend_all` in the `SET_TASK_PARAMETER` procedure, then the Segment Advisor generates a finding for each segment that qualifies for analysis, whether or not a recommendation is made for that segment. For row chaining advice, the Automatic Segment Advisor generates findings only, and not recommendations. If the Automatic Segment Advisor has no space reclamation recommendations to make, it does not generate findings. However, the Automatic Segment Advisor may generate findings for tables that could benefit from advanced row compression.
Actions	`DBA_ADVISOR_ACTIONS`	Every recommendation is associated with a suggested action to perform: either segment shrink, online redefinition (reorganization), or compression. The `DBA_ADVISOR_ACTIONS` view provides either the SQL that you can use to perform a segment shrink or table compression, or a suggestion to reorganize the object.
Objects	`DBA_ADVISOR_OBJECTS`	All findings, recommendations, and actions are associated with an object. If the Segment Advisor analyzes multiple segments, as with a tablespace or partitioned table, then one entry is created in the `DBA_ADVISOR_OBJECTS` view for each analyzed segment. Table 18-2 defines the columns in this view to query for information on the analyzed segments. You can correlate the objects in this view with the objects in the findings, recommendations, and actions views.

- Viewing Segment Advisor Results with Cloud Control
  With Cloud Control, you can view Segment Advisor results for both Automatic Segment Advisor runs and manual Segment Advisor runs.
- Viewing Segment Advisor Results by Querying the DBA_ADVISOR_* Views
  You can view Segment Advisor results by querying the `DBA_ADVISOR_*` views.
- Viewing Segment Advisor Results with DBMS_SPACE.ASA_RECOMMENDATIONS
  The `ASA_RECOMMENDATIONS` procedure in the `DBMS_SPACE` package returns a nested table object that contains findings or recommendations for Automatic Segment Advisor runs and, optionally, manual Segment Advisor runs.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details on the `DBMS_SPACE.ASA_RECOMMENDATIONS` function

18.3.2.5.1 Viewing Segment Advisor Results with Cloud Control

With Cloud Control, you can view Segment Advisor results for both Automatic Segment Advisor runs and manual Segment Advisor runs.

You can view the following types of results:

- All recommendations (multiple automatic and manual Segment Advisor runs)
- Recommendations from the last Automatic Segment Advisor run
- Recommendations from a specific run
- Row chaining findings

You can also view a list of the segments that were analyzed by the last Automatic Segment Advisor run.

To view Segment Advisor results with Cloud Control—All runs:

1. Access the Database Home page.
2. From the Administration menu, select **Storage**, then **Segment Advisor**.

   The Segment Advisor Recommendations page appears. Recommendations are organized by tablespace.
3. If any recommendations are present, select a tablespace, and then click **Recommendation Details**.

   The Recommendation Details page appears. You can initiate the recommended activity from this page (shrink or reorganize).

   Tip:

   The list entries are sorted in descending order by reclaimable space. You can click column headings to change the sort order or to change from ascending to descending order.

To view Segment Advisor results with Cloud Control—Last Automatic Segment Advisor run:

1. Access the Database Home page.
2. From the Administration menu, select **Storage**, then **Segment Advisor**.

   The Segment Advisor Recommendations page appears. Recommendations are organized by tablespace.

   The Segment Advisor Recommendations page appears.
3. In the View list, select **Recommendations from Last Automatic Run**.
4. If any recommendations are present, select a tablespace and click **Recommendation Details**.

   The Recommendation Details page appears. You can initiate the recommended activity from this page (shrink or reorganize).

To view Segment Advisor results with Cloud Control—Specific run:

1. Access the Database Home page.
2. From the Performance menu, select **Advisors Home**.

   The Advisor Central page appears. (See Figure 18-2.)
3. Check that your task appears in the list under the Results heading. If it does not, complete these steps:
   a. In the Search section of the page, under Advisor Type, select **Segment Advisor**.
   b. In the Advisor Runs list, select **All** or the desired time period.
   c. (Optional) Enter a task name.
   d. Click **Go**.

      Your Segment Advisor task appears in the Results section.
4. Check the status of the job. If it is not `COMPLETED`, use the **Refresh** control at the top of the page to refresh the page. (Do not use your browser's Refresh icon.)
5. Click the task name.

   The Segment Advisor Task page appears, with recommendations organized by tablespace.
6. Select a tablespace in the list, and then click **Recommendation Details**.

   The Recommendation Details page appears. You can initiate the recommended activity from this page (shrink or reorganize).

To view row chaining findings:

1. Access the Database Home page.
2. From the Administration menu, select **Storage**, then **Segment Advisor**.

   The Segment Advisor Recommendations page appears. Recommendations are organized by tablespace.

   The Segment Advisor Recommendations page appears.
3. Under the Related Links heading, click **Chained Row Analysis**.

   The Chained Row Analysis page appears, showing all segments that have chained rows, with a chained rows percentage for each.

18.3.2.5.2 Viewing Segment Advisor Results by Querying the DBA_ADVISOR_* Views

You can view Segment Advisor results by querying the `DBA_ADVISOR_*` views.

The headings of Table 18-5 show the columns in the `DBA_ADVISOR_*` views that contain output from the Segment Advisor. See *Oracle Database Reference* for a description of these views. The table contents summarize the possible outcomes. In addition, Table 18-2 defines the columns in the `DBA_ADVISOR_OBJECTS` view that contain information on the analyzed segments.

Before querying the `DBA_ADVISOR_*` views, you can check that the Segment Advisor task is complete by querying the `STATUS` column in `DBA_ADVISOR_TASKS`.

```
select task_name, status from dba_advisor_tasks
   where owner = 'STEVE' and advisor_name = 'Segment Advisor';

TASK_NAME                      STATUS
------------------------------ -----------
Manual Employees               COMPLETED
```

The following example shows how to query the `DBA_ADVISOR_*` views to retrieve findings from all Segment Advisor runs submitted by user `STEVE`:

```
select af.task_name, ao.attr2 segname, ao.attr3 partition, ao.type, af.message
  from dba_advisor_findings af, dba_advisor_objects ao
  where ao.task_id = af.task_id
  and ao.object_id = af.object_id
  and ao.owner = 'STEVE';
```

```
TASK_NAME          SEGNAME      PARTITION       TYPE             MESSAGE
------------------ ------------ --------------- ---------------- --------------------------
Manual_Employees   EMPLOYEES                    TABLE            The free space in the obje
                                                                 ct is less than 10MB.

Manual_Salestable4 SALESTABLE4  SALESTABLE4_P1  TABLE PARTITION  Perform shrink, estimated
                                                                 savings is 74444154 bytes.

Manual_Salestable4 SALESTABLE4  SALESTABLE4_P2  TABLE PARTITION  The free space in the obje
                                                                 ct is less than 10MB.
```

Table 18-5 Segment Advisor Outcomes: Summary

MESSAGE column of DBA_ADVISOR_FINDINGS	MORE_INFO column of DBA_ADVISOR_FINDINGS	BENEFIT_TYPE column of DBA_ADVISOR_RECOMMENDATIONS	ATTR1 column of DBA_ADVISOR_ACTIONS
Insufficient information to make a recommendation.	-	-	-
The free space in the object is less than 10MB.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	-	-
The object has some free space but cannot be shrunk because...	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	-	-

Table 18-5 (Cont.) Segment Advisor Outcomes: Summary

MESSAGE column of DBA_ADVISOR_FINDINGS	MORE_INFO column of DBA_ADVISOR_FINDINGS	BENEFIT_TYPE column of DBA_ADVISOR_RECOMMENDATIONS	ATTR1 column of DBA_ADVISOR_ACTIONS
The free space in the object is less than the size of the last extent.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	-	-
Perform shrink, estimated savings is *xxx* bytes.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	Perform shrink, estimated savings is *xxx* bytes.	The command to execute. For example: `ALTER` *`object`* `SHRINK SPACE`;)
Enable row movement of the table *schema*.*table* and perform shrink, estimated savings is *xxx* bytes.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	Enable row movement of the table *schema*.*table* and perform shrink, estimated savings is *xxx* bytes	The command to execute. For example: `ALTER` *`object`* `SHRINK SPACE`;)
Perform re-org on the object *object*, estimated savings is *xxx* bytes. (Note: This finding is for objects with reclaimable space that are not eligible for online segment shrink.)	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	Perform re-org on the object *object*, estimated savings is *xxx* bytes.	Perform re-org
The object has chained rows that can be removed by re-org.	*xx* percent chained rows can be removed by re-org.	-	-
Compress object *object_name*, estimated savings is *xxx* bytes. (This outcome is generated by the Automatic Segment Advisor only)	Compress object *object_name*, estimated savings is *xxx* bytes.	-	The command to execute. For example: `ALTER TABLE T1 ROW STORE COMPRESS ADVANCED` For this finding, see also the `ATTR2` column of `DBA_ADVISOR_ACTIONS`.

18.3.2.5.3 Viewing Segment Advisor Results with DBMS_SPACE.ASA_RECOMMENDATIONS

The `ASA_RECOMMENDATIONS` procedure in the `DBMS_SPACE` package returns a nested table object that contains findings or recommendations for Automatic Segment Advisor runs and, optionally, manual Segment Advisor runs.

Calling this procedure may be easier than working with the `DBA_ADVISOR_*` views, because the procedure performs all the required joins for you and returns information in an easily consumable format.

The following query returns recommendations by the most recent run of the Auto Segment Advisor, with the suggested command to run to follow the recommendations:

```
select tablespace_name, segment_name, segment_type, partition_name,
recommendations, c1 from
table(dbms_space.asa_recommendations('FALSE', 'FALSE', 'FALSE'));


TABLESPACE_NAME                SEGMENT_NAME                   SEGMENT_TYPE
------------------------------ ------------------------------ --------------
PARTITION_NAME
------------------------------
```

```
RECOMMENDATIONS
-----------------------------------------------------------------------------
C1
-----------------------------------------------------------------------------
TVMDS_ASSM                     ORDERS1                        TABLE PARTITION
ORDERS1_P2
Perform shrink, estimated savings is 57666422 bytes.
alter table "STEVE"."ORDERS1" modify partition "ORDERS1_P2" shrink space

TVMDS_ASSM                     ORDERS1                        TABLE PARTITION
ORDERS1_P1
Perform shrink, estimated savings is 45083514 bytes.
alter table "STEVE"."ORDERS1" modify partition "ORDERS1_P1" shrink space

TVMDS_ASSM_NEW                 ORDERS_NEW                     TABLE

Perform shrink, estimated savings is 155398992 bytes.
alter table "STEVE"."ORDERS_NEW" shrink space

TVMDS_ASSM_NEW                 ORDERS_NEW_INDEX               INDEX

Perform shrink, estimated savings is 102759445 bytes.
alter index "STEVE"."ORDERS_NEW_INDEX" shrink space
```

See *Oracle Database PL/SQL Packages and Types Reference* for details on `DBMS_SPACE.ASA_RECOMMENDATIONS`.

18.3.2.6 Configuring the Automatic Segment Advisor

The Automatic Segment Advisor is an automated maintenance task. As such, you can use Cloud Control or PL/SQL package procedure calls to modify when (and if) this task runs. You can also control the resources allotted to it by modifying the appropriate resource plans.

You can call PL/SQL package procedures to make these changes, but the easier way to is to use Cloud Control.

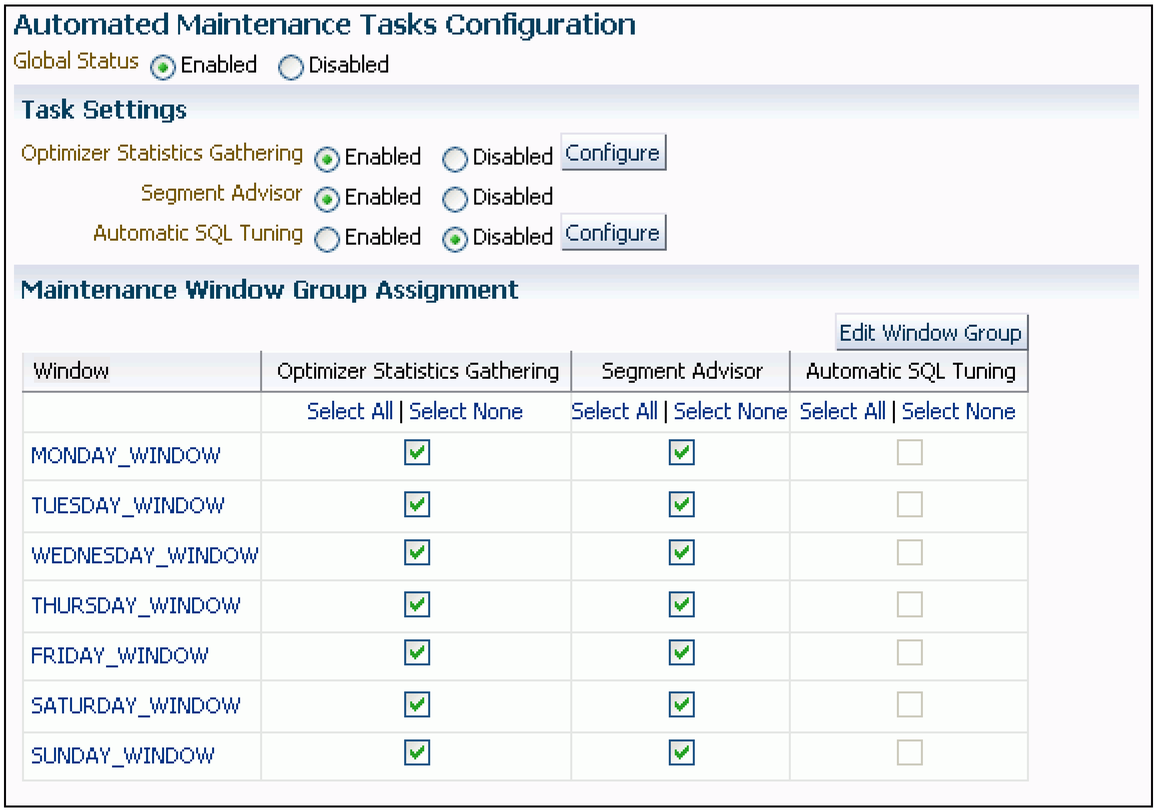
To configure the Automatic Segment Advisor task with Cloud Control:

1. Log in to Cloud Control as user `SYSTEM`.
2. Access the Database Home page.
3. From the Administration menu, select **Storage**, then **Segment Advisor**.

   The Segment Advisor Recommendations page appears.
4. Under the Related Links heading, click the link entitled **Automated Maintenance Tasks**.

   The Automated Maintenance Tasks page appears.
5. Click **Configure**.

   The Automated Maintenance Tasks Configuration page appears.



Automated Maintenance Tasks Configuration

Global Status [✓ Enabled] Disabled

Task Settings

Optimizer Statistics Gathering [✓ Enabled] Disabled Configure

Segment Advisor [✓ Enabled] Disabled

Automatic SQL Tuning Enabled [✓ Disabled] Configure

Maintenance Window Group Assignment

Edit Window Group

Window	Optimizer Statistics Gathering	Segment Advisor	Automatic SQL Tuning
	Select All | Select None	Select All | Select None	Select All | Select None
MONDAY_WINDOW	✓	✓	☐
TUESDAY_WINDOW	✓	✓	☐
WEDNESDAY_WINDOW	✓	✓	☐
THURSDAY_WINDOW	✓	✓	☐
FRIDAY_WINDOW	✓	✓	☐
SATURDAY_WINDOW	✓	✓	☐
SUNDAY_WINDOW	✓	✓	☐

6. To completely disable the Automatic Segment Advisor, under Task Settings, select **Disabled** next to the Segment Advisor label, and then click **Apply**.
7. To disable the Automatic Segment Advisor for specific maintenance windows, clear the desired check boxes under the Segment Advisor column, and then click **Apply**.
8. To modify the start and end times and durations of maintenance windows, click **Edit Window Group**.

   The Edit Window Group page appears. Click the name of a maintenance window, and then click **Edit** to change the window's schedule.

See Also:

- Managing Automated Database Maintenance Tasks

18.3.2.7 Viewing Automatic Segment Advisor Information

You can query views to display information specific to the Automatic Segment Advisor.

View	Description
`DBA_AUTO_SEGADV_SUMMARY`	Each row of this view summarizes one Automatic Segment Advisor run. Fields include number of tablespaces and segments processed, and number of recommendations made.

View	Description
`DBA_AUTO_SEGADV_CTL`	Contains control information that the Automatic Segment Advisor uses to select and process segments. Each row contains information on a single object (tablespace or segment), including whether the object has been processed, and if so, the task ID under which it was processed and the reason for selecting it.

18.3.3 Shrinking Database Segments Online

You use online segment shrink to reclaim fragmented free space below the high water mark in an Oracle Database segment.

The benefits of segment shrink are these:

- Compaction of data leads to better cache utilization, which in turn leads to better online transaction processing (OLTP) performance.
- The compacted data requires fewer blocks to be scanned in full table scans, which in turns leads to better decision support system (DSS) performance.

Segment shrink is an online, in-place operation. DML operations and queries can be issued during the data movement phase of segment shrink. Concurrent DML operations are blocked for a short time at the end of the shrink operation, when the space is deallocated. Indexes are maintained during the shrink operation and remain usable after the operation is complete. Segment shrink does not require extra disk space to be allocated.

Segment shrink reclaims unused space both above and below the high water mark. In contrast, space deallocation reclaims unused space only above the high water mark. In shrink operations, by default, the database compacts the segment, adjusts the high water mark, and releases the reclaimed space.

Segment shrink requires that rows be moved to new locations. Therefore, you must first enable row movement in the object you want to shrink and disable any rowid-based triggers defined on the object. You enable row movement in a table with the `ALTER TABLE` ... `ENABLE ROW MOVEMENT` command.

Shrink operations can be performed only on segments in locally managed tablespaces with automatic segment space management (ASSM). Within an ASSM tablespace, all segment types are eligible for online segment shrink except these:

- IOT mapping tables
- Tables with rowid based materialized views
- Tables with function-based indexes
- `SECUREFILE` LOBs
- Tables compressed with the following compression methods:
  - Basic table compression using `ROW STORE COMPRESS BASIC`
  - Warehouse compression using `COLUMN STORE COMPRESS FOR QUERY`
  - Archive compression using `COLUMN STORE COMPRESS FOR ARCHIVE`

  However, tables compressed with advanced row compression using `ROW STORE COMPRESS ADVANCED` are eligible for online segment shrink. See "Consider Using Table Compression" for information about table compression methods.

Note:

Shrinking database segments online might cause dependent database objects to become invalid. See "About Object Dependencies and Object Invalidation".

See Also:

Oracle Database SQL Language Reference for more information on the `ALTER TABLE` command.

Invoking Online Segment Shrink

Before invoking online segment shrink, view the findings and recommendations of the Segment Advisor. For more information, see "Using the Segment Advisor".

You invoke online segment shrink with Cloud Control or with SQL commands in SQL*Plus. The remainder of this section discusses the command line method.

Note:

You can invoke segment shrink directly from the Recommendation Details page in Cloud Control. Or, to invoke segment shrink for an individual table in Cloud Control, display the table on the Tables page, select the table, and then click **Shrink Segment** in the Actions list. (See Figure 18-1.) Perform a similar operation in Cloud Control to shrink indexes, materialized views, and so on.

You can shrink space in a table, index-organized table, index, partition, subpartition, materialized view, or materialized view log. You do this using `ALTER TABLE`, `ALTER INDEX`, `ALTER MATERIALIZED VIEW`, or `ALTER MATERIALIZED VIEW LOG` statement with the `SHRINK SPACE` clause.

Two optional clauses let you control how the shrink operation proceeds:

- The `COMPACT` clause lets you divide the shrink segment operation into two phases. When you specify `COMPACT`, Oracle Database defragments the segment space and compacts the table rows but postpones the resetting of the high water mark and the deallocation of the space until a future time. This option is useful if you have long-running queries that might span the operation and attempt to read from blocks that have been reclaimed. The defragmentation and compaction results are saved to disk, so the data movement does not have to be redone during the second phase. You can reissue the `SHRINK SPACE` clause without the `COMPACT` clause during off-peak hours to complete the second phase.
- The `CASCADE` clause extends the segment shrink operation to all dependent segments of the object. For example, if you specify `CASCADE` when shrinking a table segment, all indexes of the table will also be shrunk. (You need not specify `CASCADE` to shrink the partitions of a partitioned table.) To see a list of dependent segments of a given object, you can run the `OBJECT_DEPENDENT_SEGMENTS` procedure of the `DBMS_SPACE` package.

As with other DDL operations, segment shrink causes subsequent SQL statements to be reparsed because of invalidation of cursors unless you specify the `COMPACT` clause.

Examples

Shrink a table and all of its dependent segments (including `BASICFILE` and `SECUREFILE` LOB segments):

```
ALTER TABLE employees SHRINK SPACE CASCADE;
```

Shrink a `BASICFILE` LOB segment only:

```
ALTER TABLE employees MODIFY LOB (perf_review) (SHRINK SPACE);
```

Shrink a single partition of a partitioned table:

```
ALTER TABLE customers MODIFY PARTITION cust_P1 SHRINK SPACE;
```

Shrink an IOT index segment and the overflow segment:

```
ALTER TABLE cities SHRINK SPACE CASCADE;
```

Shrink an IOT overflow segment only:

```
ALTER TABLE cities OVERFLOW SHRINK SPACE;
```

Shrink a `SECUREFILE` LOB segment and its partitions:

```
ALTER TABLE employees MODIFY LOB (sperf_review) (SHRINK SPACE);
```

See Also:

- *Oracle Database SQL Language Reference* for the syntax and restrictions of the `ALTER TABLE`, `ALTER INDEX`, `ALTER MATERIALIZED VIEW`, and `ALTER MATERIALIZED VIEW LOG` statements with the `SHRINK SPACE` clause
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOB segments

18.3.4 Deallocating Unused Space

When you deallocate unused space, the database frees the unused space at the unused (high water mark) end of the database segment and makes the space available for other segments in the tablespace.

Before deallocation, you can run the `UNUSED_SPACE` procedure of the `DBMS_SPACE` package, which returns information about the position of the high water mark and the amount of unused space in a segment. For segments in locally managed tablespaces with automatic segment space management, use the `SPACE_USAGE` procedure for more accurate information on unused space.

See Also:

Oracle Database PL/SQL Packages and Types Reference contains the description of the `DBMS_SPACE` package

The following statements deallocate unused space in a segment (table, index or cluster):

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

The `KEEP` clause is optional and lets you specify the amount of space retained in the segment. You can verify that the deallocated space is freed by examining the `DBA_FREE_SPACE` view.

See Also:

- *Oracle Database SQL Language Reference* for details on the syntax and semantics of deallocating unused space
- *Oracle Database Reference* for more information about the `DBA_FREE_SPACE` view

18.4 Dropping Unused Object Storage

The `DBMS_SPACE_ADMIN` package includes the `DROP_EMPTY_SEGMENTS` procedure, which enables you to drop segments for empty tables and partitions that have been migrated from previous releases. This includes segments of dependent objects of the table, such as index segments, where possible.

The following example drops empty segments from every table in the database.

```
BEGIN
  DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS();
END;
```

The following drops empty segments from the `HR.EMPLOYEES` table, including dependent objects.

```
BEGIN
  DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS(
    schema_name  => 'HR',
    table_name   => 'EMPLOYEES');
END;
```

This procedure requires 11.2.0 or higher compatibility level.

See Also:

See *Oracle Database PL/SQL Packages and Types Reference* for details about this procedure

18.5 Understanding Space Usage of Data Types

When creating tables and other data structures, you must know how much space they will require. Each data type has different space requirements.

The *Oracle Database PL/SQL Language Reference* and *Oracle Database SQL Language Reference* contain extensive descriptions of data types and their space requirements.

18.6 Displaying Information About Space Usage for Schema Objects

Oracle Database provides data dictionary views and PL/SQL packages that allow you to display information about the space usage of schema objects.

- Using PL/SQL Packages to Display Information About Schema Object Space Usage
  A set of `DBMS_SPACE` subprograms provide information about schema objects.
- Schema Objects Space Usage Data Dictionary Views
  A set of data dictionary views display information about space usage in schema objects.

18.6.1 Using PL/SQL Packages to Display Information About Schema Object Space Usage

A set of `DBMS_SPACE` subprograms provide information about schema objects.

Package and Procedure/Function	Description
`DBMS_SPACE.UNUSED_SPACE`	Returns information about unused space in an object (table, index, or cluster).
`DBMS_SPACE.FREE_BLOCKS`	Returns information about free data blocks in an object (table, index, or cluster) whose segment free space is managed by free lists (segment space management is `MANUAL`).
`DBMS_SPACE.SPACE_USAGE`	Returns information about free data blocks in an object (table, index, or cluster) whose segment space management is `AUTO`.

See Also:

Oracle Database PL/SQL Packages and Types Reference for a description of the `DBMS_SPACE` package

Example: Using DBMS_SPACE.UNUSED_SPACE

The following SQL*Plus example uses the `DBMS_SPACE` package to obtain unused space information.

```
SQL> VARIABLE total_blocks NUMBER
SQL> VARIABLE total_bytes NUMBER
SQL> VARIABLE unused_blocks NUMBER
SQL> VARIABLE unused_bytes NUMBER
SQL> VARIABLE lastextf NUMBER
SQL> VARIABLE last_extb NUMBER
SQL> VARIABLE lastusedblock NUMBER
SQL> exec DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks, -
>    :total_bytes,:unused_blocks, :unused_bytes, :lastextf, -
>    :last_extb, :lastusedblock);

PL/SQL procedure successfully completed.

SQL> PRINT

TOTAL_BLOCKS
------------
           5

TOTAL_BYTES
-----------
      10240

...

LASTUSEDBLOCK
-------------
            3
```

18.6.2 Schema Objects Space Usage Data Dictionary Views

A set of data dictionary views display information about space usage in schema objects.

These views display information about space usage in schema objects:

View	Description
`DBA_SEGMENTS` `USER_SEGMENTS`	DBA view describes storage allocated for all database segments. User view describes storage allocated for segments for the current user.
`DBA_EXTENTS` `USER_EXTENTS`	DBA view describes extents comprising all segments in the database. User view describes extents comprising segments for the current user.
`DBA_FREE_SPACE` `USER_FREE_SPACE`	DBA view lists free extents in all tablespaces. User view shows free space information for tablespaces for which the user has quota.

- Example 1: Displaying Segment Information
  You can query the `DBA_SEGMENTS` view to display segment information.
- Example 2: Displaying Extent Information
  You can query the `DBA_EXTENTS` data dictionary view for information about the currently allocated extents in a database.

- Example 3: Displaying the Free Space (Extents) in a Tablespace
  You can query the DBA_FREE_SPACE data dictionary view for information about the free extents (extents not allocated to any segment) in a database.

18.6.2.1 Example 1: Displaying Segment Information

You can query the DBA_SEGMENTS view to display segment information.

The following query returns the name and size of each index segment in schema hr:

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
    FROM DBA_SEGMENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

The query output is:

```
SEGMENT_NAME              TABLESPACE_NAME    BYTES BLOCKS EXTENTS
------------------------- --------------- -------- ------ -------
COUNTRY_C_ID_PK           EXAMPLE            65536     32       1
DEPT_ID_PK                EXAMPLE            65536     32       1
DEPT_LOCATION_IX          EXAMPLE            65536     32       1
EMP_DEPARTMENT_IX         EXAMPLE            65536     32       1
EMP_EMAIL_UK              EXAMPLE            65536     32       1
EMP_EMP_ID_PK             EXAMPLE            65536     32       1
EMP_JOB_IX                EXAMPLE            65536     32       1
EMP_MANAGER_IX            EXAMPLE            65536     32       1
EMP_NAME_IX               EXAMPLE            65536     32       1
JHIST_DEPARTMENT_IX       EXAMPLE            65536     32       1
JHIST_EMPLOYEE_IX         EXAMPLE            65536     32       1
JHIST_EMP_ID_ST_DATE_PK   EXAMPLE            65536     32       1
JHIST_JOB_IX              EXAMPLE            65536     32       1
JOB_ID_PK                 EXAMPLE            65536     32       1
LOC_CITY_IX               EXAMPLE            65536     32       1
LOC_COUNTRY_IX            EXAMPLE            65536     32       1
LOC_ID_PK                 EXAMPLE            65536     32       1
LOC_STATE_PROVINCE_IX     EXAMPLE            65536     32       1
REG_ID_PK                 EXAMPLE            65536     32       1

19 rows selected.
```

18.6.2.2 Example 2: Displaying Extent Information

You can query the DBA_EXTENTS data dictionary view for information about the currently allocated extents in a database.

For example, the following query identifies the extents allocated to each index segment in the hr schema and the size of each of those extents:

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, TABLESPACE_NAME, EXTENT_ID, BYTES, BLOCKS
    FROM DBA_EXTENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

The query output is:

```
SEGMENT_NAME              SEGMENT_TYPE TABLESPACE_NAME EXTENT_ID    BYTES BLOCKS
------------------------- ------------ --------------- --------- -------- ------
COUNTRY_C_ID_PK           INDEX        EXAMPLE                 0    65536     32
```

```
DEPT_ID_PK                INDEX        EXAMPLE                0    65536     32
DEPT_LOCATION_IX          INDEX        EXAMPLE                0    65536     32
EMP_DEPARTMENT_IX         INDEX        EXAMPLE                0    65536     32
EMP_EMAIL_UK              INDEX        EXAMPLE                0    65536     32
EMP_EMP_ID_PK             INDEX        EXAMPLE                0    65536     32
EMP_JOB_IX                INDEX        EXAMPLE                0    65536     32
EMP_MANAGER_IX            INDEX        EXAMPLE                0    65536     32
EMP_NAME_IX               INDEX        EXAMPLE                0    65536     32
JHIST_DEPARTMENT_IX       INDEX        EXAMPLE                0    65536     32
JHIST_EMPLOYEE_IX         INDEX        EXAMPLE                0    65536     32
JHIST_EMP_ID_ST_DATE_PK   INDEX        EXAMPLE                0    65536     32
JHIST_JOB_IX              INDEX        EXAMPLE                0    65536     32
JOB_ID_PK                 INDEX        EXAMPLE                0    65536     32
LOC_CITY_IX               INDEX        EXAMPLE                0    65536     32
LOC_COUNTRY_IX            INDEX        EXAMPLE                0    65536     32
LOC_ID_PK                 INDEX        EXAMPLE                0    65536     32
LOC_STATE_PROVINCE_IX     INDEX        EXAMPLE                0    65536     32
REG_ID_PK                 INDEX        EXAMPLE                0    65536     32

19 rows selected.
```

For the hr schema, no segment has multiple extents allocated to it.

18.6.2.3 Example 3: Displaying the Free Space (Extents) in a Tablespace

You can query the DBA_FREE_SPACE data dictionary view for information about the free extents (extents not allocated to any segment) in a database.

For example, the following query reveals the amount of free space available as free extents in the SMUNDO tablespace:

```
SELECT TABLESPACE_NAME, FILE_ID, BYTES, BLOCKS
    FROM DBA_FREE_SPACE
    WHERE TABLESPACE_NAME='SMUNDO';
```

The query output is:

```
TABLESPACE_NAME  FILE_ID    BYTES  BLOCKS
--------------- -------- -------- ------
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3   131072     64
SMUNDO                 3   131072     64
SMUNDO                 3    65536     32
SMUNDO                 3  3407872   1664

10 rows selected.
```

18.7 Capacity Planning for Database Objects

Oracle Database provides two ways to plan capacity for database objects: with Cloud Control or with the DBMS_SPACE PL/SQL package. Three procedures in the DBMS_SPACE package enable you to predict the size of new objects and monitor the size of existing database objects.

This documentation discusses the PL/SQL method. See Cloud Control online help and "Using the Segment Advisor" for details on capacity planning with Cloud Control.

- Estimating the Space Use of a Table
  The size of a database table can vary greatly depending on tablespace storage attributes, tablespace block size, and many other factors. The `CREATE_TABLE_COST` procedure of the `DBMS_SPACE` package lets you estimate the space use cost of creating a table.
- Estimating the Space Use of an Index
  The `CREATE_INDEX_COST` procedure of the `DBMS_SPACE` package lets you estimate the space use cost of creating an index on an existing table.
- Obtaining Object Growth Trends
  The `OBJECT_GROWTH_TREND` function of the `DBMS_SPACE` package produces a table of one or more rows, where each row describes the space use of the object at a specific time.

18.7.1 Estimating the Space Use of a Table

The size of a database table can vary greatly depending on tablespace storage attributes, tablespace block size, and many other factors. The `CREATE_TABLE_COST` procedure of the `DBMS_SPACE` package lets you estimate the space use cost of creating a table.

See *Oracle Database PL/SQL Packages and Types Reference* for details on the parameters of this procedure.

The procedure has two variants. The first variant uses average row size to estimate size. The second variant uses column information to estimate table size. Both variants require as input the following values:

- `TABLESPACE_NAME`: The tablespace in which the object will be created. The default is the `SYSTEM` tablespace.
- `ROW_COUNT`: The anticipated number of rows in the table.
- `PCT_FREE`: The percentage of free space you want to reserve in each block for future expansion of existing rows due to updates.

In addition, the first variant also requires as input a value for `AVG_ROW_SIZE`, which is the anticipated average row size in bytes.

The second variant also requires for each anticipated column values for `COLINFOS`, which is an object type comprising the attributes `COL_TYPE` (the data type of the column) and `COL_SIZE` (the number of characters or bytes in the column).

The procedure returns two values:

- `USED_BYTES`: The actual bytes used by the data, including overhead for block metadata, `PCT_FREE` space, and so forth.
- `ALLOC_BYTES`: The amount of space anticipated to be allocated for the object taking into account the tablespace extent characteristics.

Note:

The default size of the first extent of any new segment for a partitioned table is 8 MB instead of 64 KB. This helps improve performance of inserts and queries on partitioned tables. Although partitioned tables will start with a larger initial size, once sufficient data is inserted, the space consumption will be the same as in previous releases. You can override this default by setting the `INITIAL` size in the storage clause for the table. This new default only applies to table partitions and LOB partitions.

18.7.2 Estimating the Space Use of an Index

The `CREATE_INDEX_COST` procedure of the `DBMS_SPACE` package lets you estimate the space use cost of creating an index on an existing table.

The procedure requires as input the following values:

- `DDL`: The `CREATE INDEX` statement that would create the index. The table specified in this DDL statement must be an existing table.
- [Optional] `PLAN_TABLE`: The name of the plan table to use. The default is `NULL`.

The results returned by this procedure depend on statistics gathered on the segment. Therefore, be sure to obtain statistics shortly before executing this procedure. In the absence of recent statistics, the procedure does not issue an error, but it may return inappropriate results. The procedure returns the following values:

- `USED_BYTES`: The number of bytes representing the actual index data.
- `ALLOC_BYTES`: The amount of space allocated for the index in the tablespace.

18.7.3 Obtaining Object Growth Trends

The `OBJECT_GROWTH_TREND` function of the `DBMS_SPACE` package produces a table of one or more rows, where each row describes the space use of the object at a specific time.

The function retrieves the space use totals from the Automatic Workload Repository or computes current space use and combines it with historic space use changes retrieved from Automatic Workload Repository. See *Oracle Database PL/SQL Packages and Types Reference* for detailed information on the parameters of this function.

The function requires as input the following values:

- `OBJECT_OWNER`: The owner of the object.
- `OBJECT_NAME`: The name of the object.
- `PARTITION_NAME`: The name of the table or index partition, is relevant. Specify `NULL` otherwise.
- `OBJECT_TYPE`: The type of the object.
- `START_TIME`: A `TIMESTAMP` value indicating the beginning of the growth trend analysis.
- `END_TIME`: A `TIMESTAMP` value indicating the end of the growth trend analysis. The default is "`NOW`".

- `INTERVAL`: The length in minutes of the reporting interval during which the function should retrieve space use information.
- `SKIP_INTERPOLATED`: Determines whether the function should omit values based on recorded statistics before and after the `INTERVAL` ('`YES`') or not ('`NO`'). This setting is useful when the result table will be displayed as a table rather than a chart, because you can see more clearly how the actual recording interval relates to the requested reporting interval.

The function returns a table, each of row of which provides space use information on the object for one interval. If the return table is very large, the results are pipelined so that another application can consume the information as it is being produced. The output table has the following columns:

- `TIMEPOINT`: A `TIMESTAMP` value indicating the time of the reporting interval.

  Records are not produced for values of `TIME` that precede the oldest recorded statistics for the object.
- `SPACE_USAGE`: The number of bytes actually being used by the object data.
- `SPACE_ALLOC`: The number of bytes allocated to the object in the tablespace at that time.
- `QUALITY`: A value indicating how well the requested reporting interval matches the actual recording of statistics. This information is useful because there is no guaranteed reporting interval for object size use statistics, and the actual reporting interval varies over time and from object to object.

  The values of the `QUALITY` column are:
-
  - `GOOD`: The value whenever the value of `TIME` is based on recorded statistics with a recorded timestamp within 10% of the `INTERVAL` specified in the input parameters.
  - `INTERPOLATED`: The value did not meet the criteria for `GOOD`, but was based on recorded statistics before and after the value of `TIME`. Current in-memory statistics can be collected across all instances in a cluster and treated as the "recorded" value for the present time.
  - `PROJECTION`: The value of `TIME` is in the future as of the time the table was produced. In an Oracle Real Application Clusters environment, the rules for recording statistics allow each instance to choose independently which objects will be selected.

  The output returned by this function is an aggregation of values recorded across all instances in an Oracle RAC environment. Each value can be computed from a combination of `GOOD` and `INTERPOLATED` values. The aggregate value returned is marked `GOOD` if at least 80% of that value was derived from `GOOD` instance values.

19

Managing Tables

Managing tables includes tasks such as creating tables, loading tables, altering tables, and dropping tables.

Live SQL:

To view and run examples related to the ones in this chapter on Oracle Live SQL, go to *Oracle Live SQL: Creating and Modifying Tables*.

- About Tables
  Tables are the basic unit of data storage in an Oracle Database. Data is stored in rows and columns.
- Guidelines for Managing Tables
  Following guidelines can make the management of your tables easier and can improve performance when creating the table, as well as when loading, updating, and querying the table data.
- Creating Tables
  Create tables using the SQL statement `CREATE TABLE`.
- Loading Tables
  There are several techniques for loading data into tables.
- Optimizing the Performance of Bulk Updates
  The `EXECUTE_UPDATE` procedure in the `DBMS_REDEFINITION` package can optimize the performance of bulk updates to a table. Performance is optimized because the updates are not logged in the redo log.
- Automatically Collecting Statistics on Tables
  The PL/SQL package `DBMS_STATS` lets you generate and manage statistics for cost-based optimization. You can use this package to gather, modify, view, export, import, and delete statistics. You can also use this package to identify or name statistics that have been gathered.
- Altering Tables
  You alter a table using the `ALTER TABLE` statement. To alter a table, the table must be contained in your schema, or you must have either the `ALTER` object privilege for the table or the `ALTER ANY TABLE` system privilege.
- Redefining Tables Online
  You can modify the logical or physical structure of a table.
- Researching and Reversing Erroneous Table Changes
  To enable you to research and reverse erroneous changes to tables, Oracle Database provides a group of features that you can use to view past states of database objects or to return database objects to a previous state without using point-in-time media recovery. These features are known as **Oracle Flashback features**.

- Recovering Tables Using Oracle Flashback Table
  Oracle Flashback Table enables you to restore a table to its state as of a previous point in time.
- Dropping Tables
  To drop a table that you no longer need, use the `DROP TABLE` statement.
- Using Flashback Drop and Managing the Recycle Bin
  When you drop a table, the database does not immediately remove the space associated with the table. The database renames the table and places it and any associated objects in a recycle bin, where, in case the table was dropped in error, it can be recovered at a later time. This feature is called Flashback Drop, and the `FLASHBACK TABLE` statement is used to restore the table.
- Managing Index-Organized Tables
  An index-organized table's storage organization is a variant of a primary B-tree index. Unlike a heap-organized table, data is stored in primary key order.
- Managing Partitioned Tables
  Partitioned tables enable your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Each partition can have separate physical attributes, such as compression enabled or disabled, type of compression, physical storage settings, and tablespace, thus providing a structure that can be better tuned for availability and performance. In addition, each partition can be managed individually, which can simplify and reduce the time required for backup and administration.
- Managing External Tables
  External tables are the tables that do not reside in the database. They reside outside the database, in Object storage or external files, such as operating system files or Hadoop Distributed File System (HDFS) files.
- Managing Hybrid Partitioned Tables
  A hybrid partitioned table is a partitioned table in which some partitions reside in the database and some partitions reside outside the database in external files, such as operating system files or Hadoop Distributed File System (HDFS) files.
- Managing Immutable Tables
  Immutable tables provide protection against unauthorized data modification.
- Managing Blockchain Tables
  Blockchain tables protect data that records important actions, assets, entities, and documents from unauthorized modification or deletion by criminals, hackers, and fraud. Blockchain tables prevent unauthorized changes made using the database and detect unauthorized changes that bypass the database.
- Tables Data Dictionary Views
  You can query a set of data dictionary views for information about tables.

19.1 About Tables

Tables are the basic unit of data storage in an Oracle Database. Data is stored in rows and columns.

You define a table with a table name, such as `employees`, and a set of columns. You give each column a column name, such as `employee_id`, `last_name`, and `job_id`; a data type, such as `VARCHAR2`, `DATE`, or `NUMBER`; and a width. The width can be predetermined by the data type, as in `DATE`. If columns are of the `NUMBER` data type, define precision and scale instead of width. A row is a collection of column information corresponding to a single record.

You can specify rules for each column of a table. These rules are called integrity constraints. One example is a `NOT NULL` integrity constraint. This constraint forces the column to contain a value in every row.

You can invoke Transparent Data Encryption to encrypt data before storing it. If users attempt to circumvent the database access control mechanisms by looking inside Oracle data files directly with operating system tools, encryption prevents these users from viewing sensitive data.

Tables can also include virtual columns. A **virtual column** is like any other table column, except that its value is derived by evaluating an expression. The expression can include columns from the same table, constants, SQL functions, and user-defined PL/SQL functions. You cannot explicitly write to a virtual column.

Some column types, such as `LOB`s, varrays, and nested tables, are stored in their own segments. `LOB`s and varrays are stored in `LOB` segments, while nested tables are stored in storage tables. You can specify a `STORAGE` clause for these segments that will override storage parameters specified at the table level.

After you create a table, you insert rows of data using SQL statements or using an Oracle bulk load utility. Table data can then be queried, deleted, or updated using SQL.

See Also:

- *Oracle Database Concepts* for an overview of tables
- *Oracle Database SQL Language Reference* for descriptions of Oracle Database data types
- Managing Space for Schema Objects for guidelines for managing space for tables
- Managing Schema Objects for information on additional aspects of managing tables, such as specifying integrity constraints and analyzing tables
- *Oracle Database Advanced Security Guide* for a discussion of Transparent Data Encryption

19.2 Guidelines for Managing Tables

Following guidelines can make the management of your tables easier and can improve performance when creating the table, as well as when loading, updating, and querying the table data.

- Design Tables Before Creating Them
  Usually, the application developer is responsible for designing the elements of an application, including the tables. Database administrators are responsible for establishing the attributes of the underlying tablespace that will hold the application tables.
- Specify the Type of Table to Create
  You can create different types of tables with Oracle Database.
- Specify the Location of Each Table
  It is advisable to specify the `TABLESPACE` clause in a `CREATE TABLE` statement to identify the tablespace that is to store the new table. For partitioned tables, you can optionally identify the tablespace that is to store each partition.

- Consider Parallelizing Table Creation
  You can use parallel execution when creating tables using a subquery (`AS SELECT`) in the `CREATE TABLE` statement. Because multiple processes work together to create the table, performance of the table creation operation is improved.
- Consider Using NOLOGGING When Creating Tables
  To create a table most efficiently use the `NOLOGGING` clause in the `CREATE TABLE...AS SELECT` statement. The `NOLOGGING` clause causes minimal redo information to be generated during the table creation.
- Consider Using Table Compression
  As your database grows in size, consider using table compression to save space and improve performance.
- Managing Table Compression Using Enterprise Manager Cloud Control
  You can manage table compression with Oracle Enterprise Manager Cloud Control.
- Consider Using Segment-Level and Row-Level Compression Tiering
  Segment-level compression tiering enables you to specify compression at the segment level within a table. Row-level compression tiering enables you to specify compression at the row level within a table. You can use a combination of these on the same table for fine-grained control over how the data in the table is stored and managed.
- Consider Using Attribute-Clustered Tables
  An attribute-clustered table is a heap-organized table that stores data in close proximity on disk based on user-specified clustering directives.
- Consider Using Zone Maps
  A zone is a set of contiguous data blocks on disk. A zone map tracks the minimum and maximum of specified columns for all individual zones.
- Consider Storing Tables in the In-Memory Column Store
  The In-Memory Column Store is an optional portion of the system global area (SGA) that stores copies of tables, table partitions, and other database objects that is optimized for rapid scans. In the In-Memory Column Store, table data is stored by column rather than row in the SGA.
- Consider Using Invisible Columns
  You can use invisible column to make changes to a table without disrupting applications that use the table.
- Consider Encrypting Columns That Contain Sensitive Data
  You can encrypt individual table columns that contain sensitive data. Examples of sensitive data include social security numbers, credit card numbers, and medical records. Column encryption is transparent to your applications, with some restrictions.
- Understand Deferred Segment Creation
  When you create heap-organized tables in a locally managed tablespace, the database defers table segment creation until the first row is inserted.
- Materializing Segments
  The `DBMS_SPACE_ADMIN` package includes the `MATERIALIZE_DEFERRED_SEGMENTS()` procedure, which enables you to materialize segments for tables, table partitions, and dependent objects created with deferred segment creation enabled.
- Estimate Table Size and Plan Accordingly
  Estimate the sizes of tables before creating them. Preferably, do this as part of database planning. Knowing the sizes, and uses, for database tables is an important part of database planning.
- Restrictions to Consider When Creating Tables
  There are restrictions to consider when you create tables.

19.2.1 Design Tables Before Creating Them

Usually, the application developer is responsible for designing the elements of an application, including the tables. Database administrators are responsible for establishing the attributes of the underlying tablespace that will hold the application tables.

Either the DBA or the applications developer, or both working jointly, can be responsible for the actual creation of the tables, depending upon the practices for a site. Working with the application developer, consider the following guidelines when designing tables:

- Use descriptive names for tables, columns, indexes, and clusters.
- Be consistent in abbreviations and in the use of singular and plural forms of table names and columns.
- Document the meaning of each table and its columns with the `COMMENT` command.
- Normalize each table.
- Select the appropriate data type for each column.
- Consider whether your applications would benefit from adding one or more virtual columns to some tables.
- Define columns that allow nulls last, to conserve storage space.
- Cluster tables whenever appropriate, to conserve storage space and optimize performance of SQL statements.

Before creating a table, you should also determine whether to use integrity constraints. Integrity constraints can be defined on the columns of a table to enforce the business rules of your database automatically.

19.2.2 Specify the Type of Table to Create

You can create different types of tables with Oracle Database.

Here are the types of tables that you can create:

Type of Table	Description
Ordinary (heap-organized) table	This is the basic, general purpose type of table which is the primary subject of this chapter. Its data is stored as an unordered collection (heap).
Clustered table	A clustered table is a table that is part of a cluster. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. Clusters and clustered tables are discussed in Managing Clusters.
Index-organized table	Unlike an ordinary (heap-organized) table, data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Besides storing the primary key column values of an index-organized table row, each index entry in the B-tree stores the nonkey column values as well. Index-organized tables are discussed in "Managing Index-Organized Tables ".

Type of Table	Description
Partitioned table	Partitioned tables enable your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Each partition can have separate physical attributes, such as compression enabled or disabled, type of compression, physical storage settings, and tablespace, thus providing a structure that can be better tuned for availability and performance. In addition, each partition can be managed individually, which can simplify and reduce the time required for backup and administration. Partitioned tables are discussed in *Oracle Database VLDB and Partitioning Guide*.
External table	An external table is a table that does not reside in the database, but resides outside the database in external files, such as operating system files or Hadoop Distributed File System (HDFS) files. External tables are discussed in Managing External Tables.
Hybrid partitioned table	A hybrid partitioned table is a partitioned table in which some partitions reside in the database and some partitions reside outside the database in external files, such as operating system files or Hadoop Distributed File System (HDFS) files. Hybrid partitioned tables are discussed in *Oracle Database VLDB and Partitioning Guide*.

19.2.3 Specify the Location of Each Table

It is advisable to specify the `TABLESPACE` clause in a `CREATE TABLE` statement to identify the tablespace that is to store the new table. For partitioned tables, you can optionally identify the tablespace that is to store each partition.

Ensure that you have the appropriate privileges and quota on any tablespaces that you use. If you do not specify a tablespace in a `CREATE TABLE` statement, the table is created in your default tablespace.

When specifying the tablespace to contain a new table, ensure that you understand implications of your selection. By properly specifying a tablespace during the creation of each table, you can increase the performance of the database system and decrease the time needed for database administration.

The following situations illustrate how not specifying a tablespace, or specifying an inappropriate one, can affect performance:

- If users' objects are created in the `SYSTEM` tablespace, the performance of the database can suffer, since both data dictionary objects and user objects must contend for the same data files. Users' objects should not be stored in the `SYSTEM` tablespace. To avoid this, ensure that all users are assigned default tablespaces when they are created in the database.
- If application-associated tables are arbitrarily stored in various tablespaces, the time necessary to complete administrative operations (such as backup and recovery) for the data of that application can be increased.

19.2.4 Consider Parallelizing Table Creation

You can use parallel execution when creating tables using a subquery (`AS SELECT`) in the `CREATE TABLE` statement. Because multiple processes work together to create the table, performance of the table creation operation is improved.

Parallelizing table creation is discussed in the section "Parallelizing Table Creation".

19.2.5 Consider Using NOLOGGING When Creating Tables

To create a table most efficiently use the `NOLOGGING` clause in the `CREATE TABLE...AS SELECT` statement. The `NOLOGGING` clause causes minimal redo information to be generated during the table creation.

Using the `NOLOGGING` clause has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the table is decreased.
- Performance improves for parallel creation of large tables.

The `NOLOGGING` clause also specifies that subsequent direct loads using SQL*Loader and direct load `INSERT` operations are not logged. Subsequent DML statements (`UPDATE`, `DELETE`, and conventional path insert) are unaffected by the `NOLOGGING` attribute of the table and generate redo.

If you cannot afford to lose the table after you have created it (for example, you will no longer have access to the data used to create the table) you should take a backup immediately after the table is created. In some situations, such as for tables that are created for temporary use, this precaution may not be necessary.

In general, the relative performance improvement of specifying `NOLOGGING` is greater for larger tables than for smaller tables. For small tables, `NOLOGGING` has little effect on the time it takes to create a table. However, for larger tables the performance improvement can be significant, especially when also parallelizing the table creation.

19.2.6 Consider Using Table Compression

As your database grows in size, consider using table compression to save space and improve performance.

- About Table Compression
  Compression saves disk space, reduces memory use in the database buffer cache, and can significantly speed query execution during reads.
- Examples Related to Table Compression
  Examples illustrate using table compression.
- Compression and Partitioned Tables
  A table can have both compressed and uncompressed partitions, and different partitions can use different compression methods. If the compression settings for a table and one of its partitions do not match, then the partition setting has precedence for the partition.
- Determining If a Table Is Compressed
  In the `*_TABLES` data dictionary views, compressed tables have `ENABLED` in the `COMPRESSION` column.
- Determining Which Rows Are Compressed
  To determine the compression level of a row, use the `GET_COMPRESSION_TYPE` function in the `DBMS_COMPRESSION` package.
- Changing the Compression Level
  You can change the compression level for a partition, table, or tablespace.

- Adding and Dropping Columns in Compressed Tables
  Some restrictions apply when adding columns to a compressed table or dropping columns from a compressed table.
- Exporting and Importing Hybrid Columnar Compression Tables
  Hybrid Columnar Compression tables can be imported using the `impdp` command of the Data Pump Import utility.
- Restoring a Hybrid Columnar Compression Table
  There may be times when a Hybrid Columnar Compression table must be restored from a backup. The table can be restored to a system that supports Hybrid Columnar Compression, or to a system that does not support Hybrid Columnar Compression.
- Notes and Restrictions for Compressed Tables
  Consider notes and restrictions related to compressed tables.
- Packing Compressed Tables
  If you use conventional DML on a table compressed with basic table compression or Hybrid Columnar Compression, then all inserted and updated rows are stored uncompressed or in a less-compressed format. To "pack" the compressed table so that these rows are compressed, use an `ALTER TABLE MOVE` statement.

19.2.6.1 About Table Compression

Compression saves disk space, reduces memory use in the database buffer cache, and can significantly speed query execution during reads.

Compression has a cost in CPU overhead for data loading and DML. However, this cost is offset by reduced I/O requirements. Because compressed table data stays compressed in memory, compression can also improve performance for DML operations, as more rows can fit in the database buffer cache (and flash cache if it is enabled).

Table compression is completely transparent to applications. It is useful in decision support systems (DSS), online transaction processing (OLTP) systems, and archival systems.

You can specify compression for a tablespace, a table, or a partition. If specified at the tablespace level, then all tables created in that tablespace are compressed by default.

Oracle Database supports several methods of table compression. They are summarized in Table 19-1.

Table 19-1 Table Compression Methods

Table Compression Method	Compression Level	CPU Overhead	Applications	Notes
Basic table compression	High	Minimal	DSS	None.
Advanced row compression	High	Minimal	OLTP, DSS	None.
Warehouse compression (Hybrid Columnar Compression)	Higher	Higher	DSS	The compression level and CPU overhead depend on compression level specified (LOW or HIGH).

Table 19-1 (Cont.) Table Compression Methods

Table Compression Method	Compression Level	CPU Overhead	Applications	Notes
Archive compression (Hybrid Columnar Compression)	Highest	Highest	Archiving	The compression level and CPU overhead depend on compression level specified (LOW or HIGH).

When you use basic table compression, warehouse compression, or archive compression, compression only occurs when data is bulk loaded or array inserted into a table.

Basic table compression supports limited data types and SQL operations.

Advanced row compression is intended for OLTP applications and compresses data manipulated by any SQL operation. When you use advanced row compression, compression occurs while data is being inserted, updated, or bulk loaded into a table. Operations that permit advanced row compression include:

- Single-row inserts and updates

  Inserts and updates are not compressed immediately. When updating an already compressed block, any columns that are not updated usually remain compressed. Updated columns are stored in an uncompressed format similar to any uncompressed block. The updated values are re-compressed when the block reaches a database-controlled threshold. Inserted data is also compressed when the data in the block reaches a database-controlled threshold.
- Array inserts

  Array inserts include `INSERT INTO SELECT` SQL statements without the `APPEND` hint, and array inserts from programmatic interfaces such as PL/SQL and the Oracle Call Interface (OCI).
- The following direct-path `INSERT` methods:
  - Direct path SQL*Loader
  - `CREATE TABLE AS SELECT` statements
  - Parallel `INSERT` statements
  - `INSERT` statements with an `APPEND` or `APPEND_VALUES` hint

  Inserts performed with these direct-path `INSERT` methods are compressed immediately.

Warehouse compression and archive compression achieve the highest compression levels because they use Hybrid Columnar Compression technology. Hybrid Columnar Compression technology uses a modified form of columnar storage instead of row-major storage. This enables the database to store similar data together, which improves the effectiveness of compression algorithms. For data that is updated, Hybrid Columnar Compression uses more CPU and moves the updated rows to row format so that future updates are faster. Because of this optimization, you should use it only for data that is updated infrequently.

The higher compression levels of Hybrid Columnar Compression are achieved only with data that is direct-path inserted or array inserted. Conventional inserts and updates are supported, but cause rows to be moved from columnar to row format, and reduce the compression level.

You can use Automatic Data Optimization (ADO) policies to move these rows back to the desired level of Hybrid Columnar Compression automatically.

With Hybrid Columnar Compression (warehouse and archive), for array inserts to be compressed immediately, the following conditions must be met:

- The table must be stored in a locally managed tablespace with Automatic Segment Space Management (ASSM) enabled.
- The database compatibility level must be at 12.2.0 or higher.

Regardless of the compression method, `DELETE` operations on a compressed block are identical to `DELETE` operations on a non-compressed block. Any space obtained on a data block, caused by SQL `DELETE` operations, is reused by subsequent SQL `INSERT` operations. With Hybrid Columnar Compression technology, when all the rows in a compression unit are deleted, the space in the compression unit is available for reuse.

Table 19-2 lists characteristics of each table compression method.

Table 19-2 Table Compression Characteristics

Table Compression Method	CREATE/ALTER TABLE Syntax	Direct-Path or Array Inserts	Notes
Basic table compression	`ROW STORE COMPRESS [BASIC]`	Rows are compressed with basic table compression.	`ROW STORE COMPRESS` and `ROW STORE COMPRESS BASIC` are equivalent. Rows inserted without using direct-path or array insert and updated rows are uncompressed.
Advanced row compression	`ROW STORE COMPRESS ADVANCED`	Rows are compressed with advanced row compression.	Rows inserted with or without using direct-path or array insert and updated rows are compressed using advanced row compression.
Warehouse compression (Hybrid Columnar Compression)	`COLUMN STORE COMPRESS FOR QUERY [LOW|HIGH]`	Rows are compressed with warehouse compression.	This compression method can result in high CPU overhead. Updated rows and rows inserted without using direct-path or array insert are stored in row format instead of column format, and thus have a lower compression level.
Archive compression (Hybrid Columnar Compression)	`COLUMN STORE COMPRESS FOR ARCHIVE [LOW|HIGH]`	Rows are compressed with archive compression.	This compression method can result in high CPU overhead. Updated rows and rows inserted without using direct-path or array insert are stored in row format instead of column format, and thus have a lower compression level.

You specify table compression with the `COMPRESS` clause of the `CREATE TABLE` statement. You can enable compression for an existing table by using these clauses in an `ALTER TABLE` statement. In this case, only data that is inserted or updated after compression is enabled is compressed. Using the `ALTER TABLE MOVE` statement also enables compression for data that is inserted and updated, but it compresses existing data as well. Similarly, you can disable table compression for an existing compressed table with the `ALTER TABLE`...`NOCOMPRESS` statement. In this case, all data that was already compressed remains compressed, and new data is inserted uncompressed.

The `COLUMN STORE COMPRESS FOR QUERY HIGH` option is the default data warehouse compression mode. It provides good compression and performance when using Hybrid Columnar Compression on Exadata storage. The `COLUMN STORE COMPRESS FOR QUERY LOW` option should be used in environments where load performance is critical. It loads faster than data compressed with the `COLUMN STORE COMPRESS FOR QUERY HIGH` option.

The `COLUMN STORE COMPRESS FOR ARCHIVE LOW` option is the default archive compression mode. It provides a high compression level and is ideal for infrequently-accessed data. The `COLUMN STORE COMPRESS FOR ARCHIVE HIGH` option should be used for data that is rarely accessed.

A compression advisor, provided by the `DBMS_COMPRESSION` package, helps you determine the expected compression level for a particular table with a particular compression method.

Note:

Hybrid Columnar Compression is dependent on the underlying storage system. See *Oracle Database Licensing Information* for more information.

See Also:

- *Oracle Database Concepts* for an overview of table compression
- "About Tablespaces with Default Compression Attributes"

19.2.6.2 Examples Related to Table Compression

Examples illustrate using table compression.

Example 19-1 Creating a Table with Advanced Row Compression

The following example enables advanced row compression on the table `orders`:

```
CREATE TABLE orders  ...  ROW STORE COMPRESS ADVANCED;
```

Data for the `orders` table is compressed during direct-path `INSERT`, array insert, and conventional DML.

Example 19-2 Creating a Table with Basic Table Compression

The following statements, which are equivalent, enable basic table compression on the `sales_history` table, which is a fact table in a data warehouse:

```
CREATE TABLE sales_history  ...  ROW STORE COMPRESS BASIC;

CREATE TABLE sales_history  ...  ROW STORE COMPRESS;
```

Frequent queries are run against this table, but no DML is expected.

Example 19-3 Using Direct-Path Insert to Insert Rows Into a Table

This example demonstrates using the `APPEND` hint to insert rows into the `sales_history` table using direct-path `INSERT`.

```
INSERT /*+ APPEND */ INTO sales_history SELECT * FROM sales WHERE cust_id=8890;
COMMIT;
```

Example 19-4 Using an Array Insert to Insert Rows Into a Table

This example demonstrates using an array insert in SQL to insert rows into the `sales_history` table.

```
INSERT INTO sales_history SELECT * FROM sales WHERE cust_id=8890;
COMMIT;
```

This example demonstrates using an array insert in PL/SQL to insert rows into the `hr.jobs_test` table.

```
DECLARE
   TYPE table_def IS TABLE OF hr.jobs%ROWTYPE;
   array table_def := table_def();
BEGIN
   SELECT * BULK COLLECT INTO array FROM hr.jobs;
   FORALL i in array.first .. array.last
   INSERT INTO hr.jobs_test VALUES array(i);
COMMIT;
END;
/
```

Note:

With Hybrid Columnar Compression (warehouse and archive), for array inserts performed in SQL, PL/SQL, or OCI to be compressed immediately, the table must be stored in a locally managed tablespace with Automatic Segment Space Management (ASSM) enabled, and the database compatibility level must be at 12.2.0 or higher.

Example 19-5 Creating a Table with Warehouse Compression

This example enables Hybrid Columnar Compression on the table `sales_history`:

```
CREATE TABLE sales_history  ...  COLUMN STORE COMPRESS FOR QUERY;
```

The table is created with the default `COLUMN STORE COMPRESS FOR QUERY HIGH` option. This option provides a higher level of compression than basic table compression or advanced row compression. It works well when frequent queries are run against this table and no DML is expected.

Example 19-6 Creating a Table with Archive Compression

The following example enables Hybrid Columnar Compression on the table `sales_history`:

```
CREATE TABLE sales_history  ...  COLUMN STORE COMPRESS FOR ARCHIVE;
```

The table is created with the default `COLUMN STORE COMPRESS FOR ARCHIVE LOW` option. This option provides a higher level of compression than basic, advanced row, or warehouse compression. It works well when load performance is critical and data is accessed infrequently. The default `COLUMN STORE COMPRESS FOR ARCHIVE LOW` option provides a lower level of compression than the `COLUMN STORE COMPRESS FOR ARCHIVE HIGH` option.

19.2.6.3 Compression and Partitioned Tables

A table can have both compressed and uncompressed partitions, and different partitions can use different compression methods. If the compression settings for a table and one of its partitions do not match, then the partition setting has precedence for the partition.

To change the compression method for a partition, do one of the following:

- To change the compression method for new data only, use `ALTER TABLE` ... `MODIFY PARTITION` ... `COMPRESS` ...
- To change the compression method for both new and existing data, use either `ALTER TABLE` ... `MOVE PARTITION` ... `COMPRESS` ... or online table redefinition.

When you execute these statements, specify the compression method. For example, run the following statement to change the compression method to advanced row compression for both new and existing data:

```
ALTER TABLE ... MOVE PARTITION ... ROW STORE COMPRESS ADVANCED...
```

19.2.6.4 Determining If a Table Is Compressed

In the `*_TABLES` data dictionary views, compressed tables have `ENABLED` in the `COMPRESSION` column.

For partitioned tables, this column is null, and the `COMPRESSION` column of the `*_TAB_PARTITIONS` views indicates the partitions that are compressed. In addition, the `COMPRESS_FOR` column indicates the compression method in use for the table or partition.

```
SQL> SELECT table_name, compression, compress_for FROM user_tables;

TABLE_NAME       COMPRESSION  COMPRESS_FOR
---------------- ------------ -----------------
T1               DISABLED
T2               ENABLED      BASIC
T3               ENABLED      ADVANCED
T4               ENABLED      QUERY HIGH
T5               ENABLED      ARCHIVE LOW


SQL> SELECT table_name, partition_name, compression, compress_for
  FROM user_tab_partitions;

TABLE_NAME  PARTITION_NAME   COMPRESSION   COMPRESS_FOR
----------- ---------------- -----------   ------------------------------
SALES       Q4_2004          ENABLED       ARCHIVE HIGH
  ...
SALES       Q3_2008          ENABLED       QUERY HIGH
SALES       Q4_2008          ENABLED       QUERY HIGH
SALES       Q1_2009          ENABLED       ADVANCED
SALES       Q2_2009          ENABLED       ADVANCED
```

19.2.6.5 Determining Which Rows Are Compressed

To determine the compression level of a row, use the `GET_COMPRESSION_TYPE` function in the `DBMS_COMPRESSION` package.

For example, the following query returns the compression type for a row in the `hr.employees` table:

```
SELECT DECODE(DBMS_COMPRESSION.GET_COMPRESSION_TYPE(
                 ownname    => 'HR',
                 tabname    => 'EMPLOYEES',
                 subobjname => '',
                 row_id     => 'AAAVEIAAGAAAABTAAD'),
   1,  'No Compression',
   2,  'Advanced Row Compression',
   4,  'Hybrid Columnar Compression for Query High',
   8,  'Hybrid Columnar Compression for Query Low',
   16, 'Hybrid Columnar Compression for Archive High',
   32, 'Hybrid Columnar Compression for Archive Low',
   4096, 'Basic Table Compression',
   'Unknown Compression Type') compression_type
FROM DUAL;
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for additional information about `GET_COMPRESSION_TYPE`

19.2.6.6 Changing the Compression Level

You can change the compression level for a partition, table, or tablespace.

For example, suppose a company uses warehouse compression for its sales data, but sales data older than six months is rarely accessed. If the sales data is stored in a table that is partitioned based on the age of the data, then the compression level for the older data can be changed to archive compression to free disk space.

To change the compression level for a partition or subpartition, you can use the following statements:

- `ALTER TABLE ... MOVE PARTITION ... ONLINE`
- `ALTER TABLE ... MOVE SUBPARTITION ... ONLINE`

These two statements support the `ONLINE` keyword, which enables DML operations to run uninterrupted on the partition or subpartition that is being moved. These statements also automatically keep all the indexes updated while the partition or subpartition is being moved. You can also use the `ALTER TABLE...MODIFY PARTITION` statement or online redefinition to change the compression level for a partition.

If a table is not partitioned, then you can use the `ALTER TABLE...MOVE...COMPRESS FOR...` statement to change the compression level. The `ALTER TABLE...MOVE` statement does not permit DML statements against the table while the command is running. However, you can also use online redefinition to compress a table, which keeps the table available for queries and DML statements during the redefinition.

To change the compression level for a tablespace, use the `ALTER TABLESPACE` statement.

See Also:

- "Moving a Table to a New Segment or Tablespace" for additional information about the `ALTER TABLE` command
- "Redefining Tables Online"
- *Oracle Database PL/SQL Packages and Types Reference* for additional information about the `DBMS_REDEFINITION` package

19.2.6.7 Adding and Dropping Columns in Compressed Tables

Some restrictions apply when adding columns to a compressed table or dropping columns from a compressed table.

The following restrictions apply when adding columns to compressed tables:

- Advanced row compression, warehouse compression, and archive compression: If a default value is specified for an added column and the table is already populated, then the conditions for optimized add column behavior must be met. These conditions are described in *Oracle Database SQL Language Reference*.

The following restrictions apply when dropping columns in compressed tables:

- Basic table compression: Dropping a column is not supported.
- Advanced row compression, warehouse compression, and archive compression: `DROP COLUMN` is supported, but internally the database sets the column `UNUSED` to avoid long-running decompression and recompression operations.

19.2.6.8 Exporting and Importing Hybrid Columnar Compression Tables

Hybrid Columnar Compression tables can be imported using the `impdp` command of the Data Pump Import utility.

By default, the `impdp` command preserves the table properties, and the imported table is a Hybrid Columnar Compression table. On tablespaces not supporting Hybrid Columnar Compression, the `impdp` command fails with an error. The tables can also be exported using the `expdp` command.

You can import the Hybrid Columnar Compression table as an uncompressed table using the `TRANSFORM=SEGMENT_ATTRIBUTES:n` option clause of the `impdp` command.

An uncompressed or advanced row-compressed table can be converted to Hybrid Columnar Compression format during import. To convert a non-Hybrid Columnar Compression table to a Hybrid Columnar Compression table, do the following:

1. Specify default compression for the tablespace using the `ALTER TABLESPACE ... SET DEFAULT COMPRESS` command.
2. Override the `SEGMENT_ATTRIBUTES` option of the imported table during import.

See Also:

- *Oracle Database Utilities* for additional information about the Data Pump Import utility
- *Oracle Database SQL Language Reference* for additional information about the `ALTER TABLESPACE` command

19.2.6.9 Restoring a Hybrid Columnar Compression Table

There may be times when a Hybrid Columnar Compression table must be restored from a backup. The table can be restored to a system that supports Hybrid Columnar Compression, or to a system that does not support Hybrid Columnar Compression.

When restoring a table with Hybrid Columnar Compression to a system that supports Hybrid Columnar Compression, restore the file using Oracle Recovery Manager (RMAN) as usual.

When a Hybrid Columnar Compression table is restored to a system that does not support Hybrid Columnar Compression, you must convert the table from Hybrid Columnar Compression to advanced row compression or an uncompressed format. To restore the table, do the following:

1. Ensure there is sufficient storage in environment to hold the data in uncompressed or advanced row compression format.
2. Use RMAN to restore the Hybrid Columnar Compression tablespace.
3. Complete one of the following actions to convert the table from Hybrid Columnar Compression to advanced row compression or an uncompressed format:
   - Use the following statement to change the data compression from Hybrid Columnar Compression to `ROW STORE COMPRESS ADVANCED`:

     ```
     ALTER TABLE table_name MOVE ROW STORE COMPRESS ADVANCED;
     ```

   - Use the following statement to change the data compression from Hybrid Columnar Compression to `NOCOMPRESS`:

     ```
     ALTER TABLE table_name MOVE NOCOMPRESS;
     ```

   - Use the following statement to change each partition to `NOCOMPRESS`:

     ```
     ALTER TABLE table_name MOVE PARTITION partition_name NOCOMPRESS;
     ```

     Change each partition separately.

     If DML is required on the partition while it is being moved, then include the `ONLINE` keyword:

     ```
     ALTER TABLE table_name MOVE PARTITION partition_name NOCOMPRESS ONLINE;
     ```

     Moving a partition online might take longer than moving a partition offline.

   - Use the following statement to move the data to `NOCOMPRESS` in parallel:

     ```
     ALTER TABLE table_name MOVE NOCOMPRESS PARALLEL;
     ```

See Also:

- *Oracle Database Backup and Recovery User's Guide* for additional information about RMAN
- *Oracle Database SQL Language Reference* for additional information about the `ALTER TABLE` command

19.2.6.10 Notes and Restrictions for Compressed Tables

Consider notes and restrictions related to compressed tables.

The following are notes and restrictions related to compressed tables:

- Advanced row compression, warehouse compression, and archive compression are not supported for the following types of tables:
  - Index-organized tables
  - External tables
  - Tables with `LONG` or `LONG RAW` columns
  - Temporary tables
  - Tables with `ROWDEPENDENCIES` enabled
  - Clustered tables
- Online segment shrink is not supported for tables compressed with the following compression methods:
  - Basic table compression using `ROW STORE COMPRESS BASIC`
  - Warehouse compression using `COLUMN STORE COMPRESS FOR QUERY`
  - Archive compression using `COLUMN STORE COMPRESS FOR ARCHIVE`
- The table compression methods described in this section do not apply to SecureFiles large objects (LOBs). SecureFiles LOBs have their own compression methods. See *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information.
- Compression technology uses CPU. Ensure that you have enough available CPU to handle the additional load.
- Tables created with basic table compression have the `PCT_FREE` parameter automatically set to `0` unless you specify otherwise.

19.2.6.11 Packing Compressed Tables

If you use conventional DML on a table compressed with basic table compression or Hybrid Columnar Compression, then all inserted and updated rows are stored uncompressed or in a less-compressed format. To "pack" the compressed table so that these rows are compressed, use an `ALTER TABLE MOVE` statement.

This operation takes an exclusive lock on the table, and therefore prevents any updates and loads until it completes. If this is not acceptable, then you can use online table redefinition.

When you move a partition or subpartition, you can use the `ALTER TABLE MOVE` statement to compress the partition or subpartition while still allowing DML operations to run interrupted on the partition or subpartition that is being moved.

See Also:

- *Oracle Database SQL Language Reference* for more details on the `ALTER TABLE`...`COMPRESS` and `ALTER TABLE`...`MOVE` statements, including restrictions
- *Oracle Database VLDB and Partitioning Guide* for more information on table partitioning
- "Redefining Tables Online"
- "Moving a Table to a New Segment or Tablespace" for more information about moving a table, partition, or subpartition

19.2.7 Managing Table Compression Using Enterprise Manager Cloud Control

You can manage table compression with Oracle Enterprise Manager Cloud Control.

- Table Compression and Enterprise Manager Cloud Control
  Enterprise Manager displays several central compression pages that summarize the compression features at the database and tablespace levels and contains links to different compression pages. The Compression pages display summaries of the compressed storage space at the database level and the tablespace level.
- Viewing the Compression Summary at the Database Level
  You can view the Compression Summary information at the database level.
- Viewing the Compression Summary at the Tablespace Level
  You can view the Compression Summary information at the tablespace level.
- Estimating the Compression Ratio
  You can run the Compression Advisor to calculate the compression ratio for a specific object.
- Compressing an Object
  You can compress an object such as a table.
- Viewing Compression Advice
  You can view compression advice from the Segment Advisor and take actions based on them.
- Initiating Automatic Data Optimization on an Object
  You can initiate Automatic Data Optimization on an object.

19.2.7.1 Table Compression and Enterprise Manager Cloud Control

Enterprise Manager displays several central compression pages that summarize the compression features at the database and tablespace levels and contains links to different compression pages. The Compression pages display summaries of the compressed storage space at the database level and the tablespace level.

On the database level, the Compression Summary for Database page shows the total database size (total size of all the objects, both compressed and uncompressed), the total size of compressed objects in the database, the total size of uncompressed objects in the database and the ratio of the total size of compressed objects to the total database size. This provides you with a general idea on how much storage space within a database is compressed. You can then take action based on the information displayed.

Likewise on the tablespace level, the Compression Summary for Tablespace page shows the total tablespace size (total size of all the objects, both compressed and uncompressed), the total size of compressed objects in the tablespace, the total size of uncompressed objects in the tablespace and the ratio of the total size of compressed objects to the total tablespace size.

You can use the Compression feature to perform the following tasks:

- View a summary of the compressed storage space for the top 100 tablespaces at the database level or the top 100 objects at the tablespace level. You can view a summary on how much storage space is compressed within each of top 100 tablespaces that use the most database storage, including the total size of the tablespace, the compressed size of a tablespace, the uncompressed size of tablespace, and the percentage of compressed storage within a tablespace. You can then perform compression tasks based on the information displayed.
- View the storage size that is compressed by each compression type for four object types: Table, Index, LOB (Large Objects), and DBFS (Oracle Database File System).
- Calculate the compression ratio for a specific object.
- Compress an object (tablespace, table, partition or LOB). This allows you to save storage space. You can run the Compression Advisor to ascertain how much space can be saved and then perform the compression action on the object.
- View compression advice from the Segment Advisor. You can access a link to the Segment Advisor to compress segments.

19.2.7.2 Viewing the Compression Summary at the Database Level

You can view the Compression Summary information at the database level.

1. From the **Administration** menu, choose **Storage**, then select **Compression**.

   Enterprise Manager displays the Compression Summary for Top 100 Tablespaces page.
2. You can view the summary information about the storage compression at the database level, including in the Space Usage section the total database size, the total size of compressed objects in the database, and the ratio of the total size of compressed objects to the total database size, and the uncompressed objects size. Similar information for segment counts is also shown here in the Segment Count section.
3. You can view the storage size that is used by each compression type for four object types: Table, Index, LOB (Large Objects), and DBFS (Oracle Database File System). Clicking each color in the chart displays a Compression Summary of Segments page, which shows compression information for the top 100 segments by size in the database for a particular object type and compression type.

19.2.7.3 Viewing the Compression Summary at the Tablespace Level

You can view the Compression Summary information at the tablespace level.

1. From the **Administration** menu, choose **Storage**, then select **Compression**.

   Enterprise Manager displays the Compression Summary for Top 100 Tablespaces page.

2. In the Top 100 Permanent Tablespaces by Size table, click on the row for the tablespace for which you want to view the compression summary.
3. Click **Show Compression Details**.

   Enterprise Manager displays the Compression Summary for Top 100 Objects in Tablespace page. From this page, you can view the total tablespace size, the total size of compressed objects in the tablespace, the ratio of the total size of compressed objects to the total tablespace size, and the uncompressed objects size in a tablespace.

   You can also view the compressed tablespace storage size by each compression type for four object types: Table, Index, LOB and DBFS. Clicking each color in the chart displays the Compression Summary of Segments dialog box, which shows compression information for the top 100 segments by size in the tablespace for a particular object type and compression type.

   Finally, you can view the compression summary for each of the top 100 segments that use the most tablespace storage.

19.2.7.4 Estimating the Compression Ratio

You can run the Compression Advisor to calculate the compression ratio for a specific object.

1. From the **Administration** menu, choose **Storage**, then select **Compression**.

   Enterprise Manager displays the Compression Summary for Top 100 Tablespaces page.
2. From the Top 100 Permanent Tablespaces by Size table, select a tablespace and click **Show Compression Details** to view the compression details for the selected tablespace.

   Enterprise Manager displays the Top 100 Objects By Size table.
3. Select an object and click **Estimate Compression Ratio** for the object.

   Enterprise Manager displays the Estimate Compression Ratio dialog box. Enter the following information:
   - Under the Input Parameters section, enter or select a Temporary Scratch Tablespace. You can enter the name directly or you can choose from the list that appears when you click the icon.
   - Enter the Compression Type. You can choose from Basic, Advanced, Query Low, Query High, Archive Low, or Archive High. For HCC compression types (Query Low, Query High, Archive Low, or Archive High.), be sure the table contains at least one million rows.
   - In the Schedule Job section, enter the Name of the job and a Description.
   - In the Schedule section, enter the job information such as when to Start, whether or not to Repeat the job, whether or not there should be a Grace Period, and Duration information.
   - Enter the Database Credentials and the Host Credentials in their respective sections.
   - Click **OK**.

The job runs either immediately or is scheduled, and you are returned to the Compression Summary for Top 100 Objects in Tablespace page.

19.2.7.5 Compressing an Object

You can compress an object such as a table.

1. From the **Administration** menu, choose **Storage**, then select **Compression**.

Enterprise Manager displays the Compression Summary for Top 100 Tablespaces page.

2. From the Top 100 Permanent Tablespaces by Size table, select a tablespace and click **Show Compression Details** to view Compression details for the selected tablespace.

   Enterprise Manager displays the Compression Summary for Top 100 Objects in Tablespace page.

3. Choose an object, such as a table, and click **Compress** to compress the object.

19.2.7.6 Viewing Compression Advice

You can view compression advice from the Segment Advisor and take actions based on them.

1. From the **Administration** menu, choose **Storage**, then select **Compression**.

   Enterprise Manager displays the Compression Summary for Top 100 Tablespaces page.

2. In the Compression Advice section, click the number that displays in the Segments with Compression Advice field.

   Enterprise Manager displays the Segment Advisor Recommendations page. You can use the Automatic Segment Advisor job to detect segment issues within maintenance windows. The recommendations are derived from the most recent runs of automatic and user-scheduled segment advisor jobs.

19.2.7.7 Initiating Automatic Data Optimization on an Object

You can initiate Automatic Data Optimization on an object.

1. From the **Administration** menu, choose **Storage**, then select **Compression**.

   Enterprise Manager displays the Compression Summary for Top 100 Tablespaces page.

2. From the Top 100 Permanent Tablespaces by Size table, select a tablespace and click **Show Compression Details** to view the compression details for the selected tablespace.

   Enterprise Manager displays the Compression Summary for Top 100 Objects in Tablespace page.

3. From the Top 100 Objects by Size table, select an object and click **Automatic Data Compression**.

   Enterprise Manager displays the Edit page for the object where you can initiate Automatic Data Optimization on the object.

19.2.8 Consider Using Segment-Level and Row-Level Compression Tiering

Segment-level compression tiering enables you to specify compression at the segment level within a table. Row-level compression tiering enables you to specify compression at the row level within a table. You can use a combination of these on the same table for fine-grained control over how the data in the table is stored and managed.

As user modifications to segments and rows change over time, it is often beneficial to change the compression level for them. For example, some segments and rows might be modified often for a short period of time after they are added to the database, but modifications might become less frequent over time.

You can use compression tiering to specify which segments and rows are compressed based on rules. For example, you can specify that rows that have not been modified in two weeks are compressed with advanced row compression. You can also specify that segments that have not been modified in six months are compressed with warehouse compression.

The following prerequisites must be met before you can use segment-level and row-level compression tiering:

- The `HEAT_MAP` initialization parameter must be set to `ON`.
- The `COMPATIBLE` initialization parameter must be set to `12.0.0` or higher.

To use segment-level compression tiering or row-level compression tiering, execute one of the following SQL statements and include an Automatic Data Optimization (ADO) policy that specifies the rules:

- `CREATE TABLE`
- `ALTER TABLE`

Example 19-7 Row-Level Compression Tiering

This example specifies row-level compression tiering for the `oe.orders` table. Oracle Database compresses rows using warehouse (`QUERY`) compression after 14 days with no modifications.

```
ALTER TABLE oe.orders ILM ADD POLICY
  COLUMN STORE COMPRESS FOR QUERY
  ROW
  AFTER 14 DAYS OF NO MODIFICATION;
```

Example 19-8 Segment-Level Compression Tiering

This example specifies segment-level compression tiering for the `oe.order_items` table. Oracle Database compresses segments using archive (`ARCHIVE HIGH`) compression after six months with no modifications to any rows in the segment and no queries accessing any rows in the segment.

```
ALTER TABLE oe.order_items ILM ADD POLICY
  COLUMN STORE COMPRESS FOR ARCHIVE HIGH
  SEGMENT
  AFTER 6 MONTHS OF NO ACCESS;
```

Note:

These examples specify Hybrid Columnar Compression, which is dependent on the underlying storage system. See *Oracle Database Licensing Information* for more information.

See Also:

- "Consider Using Table Compression" for information about different compression levels
- "Improving Query Performance with Oracle Database In-Memory"
- *Oracle Database VLDB and Partitioning Guide* for more information about segment-level and row-level compression tiering

19.2.9 Consider Using Attribute-Clustered Tables

An attribute-clustered table is a heap-organized table that stores data in close proximity on disk based on user-specified clustering directives.

> **Note:**
>
> This feature is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

The directives are as follows:

- The `CLUSTERING ... BY LINEAR ORDER` directive orders data in a table according to specified columns.

  `BY LINEAR ORDER` clustering, which is the default, is best when queries qualify the prefix of columns specified in the clustering clause. For example, if queries of `sh.sales` often specify either a customer ID or both customer ID and product ID, then you could cluster data in the table using the linear column order `cust_id`, `prod_id`. Note that the specified columns can be in multiple tables.
- The `CLUSTERING ... BY INTERLEAVED ORDER` directive orders data in one or more tables using a special algorithm, similar to a z-order function, that permits multicolumn I/O reduction.

  `BY INTERLEAVED ORDER` clustering is best when queries specify a variety of column combinations. The columns can be in one or more tables. For example, if queries of `sh.sales` specify different dimensions in different orders, then you could cluster data in the `sales` table according to columns in these dimensions.

Attribute clustering is available for the following types of operations:

- Direct-path `INSERT`

  See "Improving INSERT Performance with Direct-Path INSERT".
- Online redefinition

  See "Redefining Tables Online".
- Data movement operations, such as `ALTER TABLE ... MOVE` operations

  See "Moving a Table to a New Segment or Tablespace".
- Partition maintenance operations that create new segments, such as `ALTER TABLE ... MERGE PARTITION` operations

  See *Oracle Database VLDB and Partitioning Guide*.

Attribute clustering is ignored for conventional DML.

An attribute-clustered table has the following advantages:

- More optimized single block I/O is possible for table lookups when attribute clustering is aligned with common index access. For example, optimized I/O is possible for an index range scan on the leading column you chose for attribute clustering.
- Data ordering enables more optimal pruning for Exadata storage indexes and in-memory min/max pruning.
- You can cluster fact tables based on joined attributes from other tables.

- Attribute clustering can improve data compression and in this way indirectly improve table scan costs. When the same values are close to each other on disk, the database can more easily compress them.

Attribute-clustered tables are often used in data warehousing environments, but they are useful in any environment that can benefit from these advantages. Use the `CLUSTERING` clause in a `CREATE TABLE` SQL statement to create an attribute-clustered table.

See Also:

- *Oracle Database Concepts* for conceptual information about attribute-clustered tables
- *Oracle Database Data Warehousing Guide* for information about using attribute-clustered tables
- *Oracle Database SQL Language Reference*

19.2.10 Consider Using Zone Maps

A zone is a set of contiguous data blocks on disk. A zone map tracks the minimum and maximum of specified columns for all individual zones.

When a SQL statement contains predicates on columns stored in a zone map, the database compares the predicate values to the minimum and maximum stored in the zone to determine which zones to read during SQL execution. The primary benefit of zone maps is I/O reduction for table scans. I/O is reduced by skipping table blocks that are not needed in the query result. Use the `CREATE MATERIALIZED ZONEMAP` SQL statement to create a zone map.

Whenever attribute clustering is specified on a table, you can automatically create a zone map on the clustered columns. Due to clustering, minimum and maximum values of the columns are correlated with consecutive data blocks in the attribute-clustered table, which allows for more effective I/O pruning using the associated zone map.

Note:

Zone maps and attribute-clustered tables can be used together or separately.

Starting with Oracle Database Release 21c, you can automatically create and maintain basic zone maps for partitioned and non-partitioned heap tables. Use the `DBMS_AUTO_ZONEMAP.CONFIGURE` procedure to manage automatic zone map creation and maintenance.

See Also:

Oracle Database Licensing Information User Manual for details on which features are supported for different editions and services.

See Also:

- "Consider Using Attribute-Clustered Tables"
- *Oracle Database Concepts* for conceptual information about zone maps
- *Oracle Database Data Warehousing Guide* for information about using zone maps
- *Oracle Database SQL Language Reference* for information about the `CREATE MATERIALIZED ZONEMAP` statement

19.2.11 Consider Storing Tables in the In-Memory Column Store

The In-Memory Column Store is an optional portion of the system global area (SGA) that stores copies of tables, table partitions, and other database objects that is optimized for rapid scans. In the In-Memory Column Store, table data is stored by column rather than row in the SGA.

Note:

This feature is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

See Also:

- "Improving Query Performance with Oracle Database In-Memory"
- *Oracle Database Concepts*

19.2.12 Consider Using Invisible Columns

You can use invisible column to make changes to a table without disrupting applications that use the table.

- Understand Invisible Columns
  You can make individual table columns invisible. Any generic access of a table does not show the invisible columns in the table.
- Invisible Columns and Column Ordering
  There are special considerations for invisible columns and column ordering.

19.2.12.1 Understand Invisible Columns

You can make individual table columns invisible. Any generic access of a table does not show the invisible columns in the table.

For example, the following operations do not display invisible columns in the output:

- `SELECT * FROM` statements in SQL

- `DESCRIBE` commands in SQL*Plus
- `%ROWTYPE` attribute declarations in PL/SQL
- Describes in Oracle Call Interface (OCI)

You can use a `SELECT` statement to display output for an invisible column only if you explicitly specify the invisible column in the column list. Similarly, you can insert a value into an invisible column only if you explicitly specify the invisible column in the column list for the `INSERT` statement. If you omit the column list in the `INSERT` statement, then the statement can only insert values into visible columns.

You can make a column invisible during table creation or when you add a column to a table, and you can later alter the table to make the same column visible. You can also alter a table to make a visible column invisible.

You might use invisible columns if you want to make changes to a table without disrupting applications that use the table. After you add an invisible column to a table, queries and other operations that must access the invisible column must refer to the column explicitly by name. When you migrate the application to account for the invisible columns, you can make the invisible columns visible.

Virtual columns can be invisible. Also, you can use an invisible column as a partitioning key during table creation.

The following restrictions apply to invisible columns:

- The following types of tables cannot have invisible columns:
  - External tables
  - Cluster tables
  - Temporary tables
- Attributes of user-defined types cannot be invisible.

Note:

Invisible columns are not the same as system-generated hidden columns. You can make invisible columns visible, but you cannot make hidden columns visible.

See Also:

- "Creating Tables"
- "Adding Table Columns"
- "Modifying an Existing Column Definition"

19.2.12.2 Invisible Columns and Column Ordering

There are special considerations for invisible columns and column ordering.

The database usually stores columns in the order in which they were listed in the `CREATE TABLE` statement. If you add a new column to a table, then the new column becomes the last column in the table's column order.

When a table contains one or more invisible columns, the invisible columns are not included in the column order for the table. Column ordering is important when all of the columns in a table are accessed. For example, a `SELECT * FROM` statement displays columns in the table's column order. Because invisible columns are not included in this type of generic access of a table, they are not included in the column order.

When you make an invisible column visible, the column is included in the table's column order as the last column. When you make a visible column invisible, the invisible column is not included in the column order, and the order of the visible columns in the table might be re-arranged.

For example, consider the following table with an invisible column:

```
CREATE TABLE mytable (a INT, b INT INVISIBLE, c INT);
```

Because column `b` is invisible, this table has the following column order:

Column	Column Order
a	1
c	2

Next, make column `b` visible:

```
ALTER TABLE mytable MODIFY (b VISIBLE);
```

When you make column `b` visible, it becomes the last column in the table's column order. Therefore, the table has the following column order:

Column	Column Order
a	1
c	2
b	3

Consider another example that illustrates column ordering in tables with invisible columns. The following table does not contain any invisible columns:

```
CREATE TABLE mytable2 (x INT, y INT, z INT);
```

This table has the following column order:

Column	Column Order
x	1
y	2
z	3

Next, make column `y` invisible:

```
ALTER TABLE mytable2 MODIFY (y INVISIBLE);
```

When you make column `y` invisible, column `y` is no longer included in the table's column order, and it changes the column order of column `z`. Therefore, the table has the following column order:

Column	Column Order
x	1
z	2

Make column `y` visible again:

```
ALTER TABLE mytable2 MODIFY (y VISIBLE);
```

Column `y` is now last in the table's column order:

Column	Column Order
x	1
z	2
y	3

19.2.13 Consider Encrypting Columns That Contain Sensitive Data

You can encrypt individual table columns that contain sensitive data. Examples of sensitive data include social security numbers, credit card numbers, and medical records. Column encryption is transparent to your applications, with some restrictions.

Although encryption is not meant to solve all security problems, it does protect your data from users who try to circumvent the security features of the database and access database files directly through the operating system file system.

Column encryption uses the Transparent Data Encryption feature of Oracle Database, which requires that you create a keystore to store the master encryption key for the database. The keystore must be open before you can create a table with encrypted columns and before you can store or retrieve encrypted data. When you open the keystore, it is available to all sessions, and it remains open until you explicitly close it or until the database is shut down.

Transparent Data Encryption supports industry-standard encryption algorithms, including the following types of encryption algorithms Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms:

- Advanced Encryption Standard (AES)
- ARIA
- GHOST
- SEED
- Triple Data Encryption Standard (3DES)

See *Oracle Database Advanced Security Guide* for detailed information about the supported encryption algorithms.

You choose the algorithm to use when you create the table. All encrypted columns in the table use the same algorithm. The default is AES192. The encryption key length is implied by the algorithm name. For example, the AES128 algorithm uses 128-bit keys.

If you plan on encrypting many columns in one or more tables, you may want to consider encrypting an entire tablespace instead and storing these tables in that tablespace. Tablespace encryption, which also uses the Transparent Data Encryption feature but encrypts at the physical block level, can perform better than encrypting many columns. Another reason to encrypt at the tablespace level is to address the following limitations of column encryption:

- Certain data types, such as object data types, are not supported for column encryption.
- You cannot use the transportable tablespace feature for a tablespace that includes tables with encrypted columns.
- Other restrictions, which are detailed in *Oracle Database Advanced Security Guide*.

See Also:

- *Oracle Database Advanced Security Guide* for more information about Transparent Data Encryption
- *Oracle Database Enterprise User Security Administrator's Guide* for instructions for creating and opening keystores
- *Oracle Database SQL Language Reference* for information about the `CREATE TABLE` statement
- *Oracle Real Application Clusters Administration and Deployment Guide* for information on using a keystore in an Oracle Real Application Clusters environment

19.2.14 Understand Deferred Segment Creation

When you create heap-organized tables in a locally managed tablespace, the database defers table segment creation until the first row is inserted.

In addition, segment creation is deferred for any LOB columns of the table, any indexes created implicitly as part of table creation, and any indexes subsequently explicitly created on the table.

The advantages of this space allocation method are the following:

- It saves a significant amount of disk space in applications that create hundreds or thousands of tables upon installation, many of which might never be populated.
- It reduces application installation time.

There is a small performance penalty when the first row is inserted, because the new segment must be created at that time.

To enable deferred segment creation, compatibility must be set to `11.2.0` or higher.

The new clauses for the `CREATE TABLE` statement are:

- `SEGMENT CREATION DEFERRED`
- `SEGMENT CREATION IMMEDIATE`

These clauses override the default setting of the `DEFERRED_SEGMENT_CREATION` initialization parameter, `TRUE`, which defers segment creation. To disable deferred segment creation, set this parameter to `FALSE`.

Note that when you create a table with deferred segment creation, the new table appears in the `*_TABLES` views, but no entry for it appears in the `*_SEGMENTS` views until you insert the first row.

You can verify deferred segment creation by viewing the `SEGMENT_CREATED` column in `*_TABLES`, `*_INDEXES`, and `*_LOBS` views for nonpartitioned tables, and in `*_TAB_PARTITIONS`, `*_IND_PARTITIONS`, and `*_LOB_PARTITIONS` views for partitioned tables.

> **Note:**
>
> With this new allocation method, it is essential that you do proper capacity planning so that the database has enough disk space to handle segment creation when tables are populated. See "Capacity Planning for Database Objects ".

The following example creates two tables to demonstrate deferred segment creation. The first table uses the `SEGMENT CREATION DEFERRED` clause. No segments are created for it initially. The second table uses the `SEGMENT CREATION IMMEDIATE` clause and, therefore, segments are created for it immediately.

```
CREATE TABLE part_time_employees (
    empno NUMBER(8),
    name VARCHAR2(30),
    hourly_rate NUMBER (7,2)
    )
    SEGMENT CREATION DEFERRED;

CREATE TABLE hourly_employees (
    empno NUMBER(8),
    name VARCHAR2(30),
    hourly_rate NUMBER (7,2)
    )
   SEGMENT CREATION IMMEDIATE
   PARTITION BY RANGE(empno)
    (PARTITION empno_to_100 VALUES LESS THAN (100),
    PARTITION empno_to_200 VALUES LESS THAN (200));
```

The following query against `USER_SEGMENTS` returns two rows for `HOURLY_EMPLOYEES`, one for each partition, but returns no rows for `PART_TIME_EMPLOYEES because` segment creation for that table was deferred.

```
SELECT segment_name, partition_name FROM user_segments;

SEGMENT_NAME         PARTITION_NAME
-------------------- ------------------------------
HOURLY_EMPLOYEES     EMPNO_TO_100
HOURLY_EMPLOYEES     EMPNO_TO_200
```

The `USER_TABLES` view shows that `PART_TIME_EMPLOYEES` has no segments:

```
SELECT table_name, segment_created FROM user_tables;


TABLE_NAME                     SEGMENT_CREATED
------------------------------ ----------------------------------------
PART_TIME_EMPLOYEES            NO
HOURLY_EMPLOYEES               N/A
```

For the HOURLY_EMPLOYEES table, which is partitioned, the segment_created column is N/A because the USER_TABLES view does not provide that information for partitioned tables. It is available from the USER_TAB_PARTITIONS view, shown below.

```
SELECT table_name, segment_created, partition_name
 FROM user_tab_partitions;

TABLE_NAME           SEGMENT_CREATED      PARTITION_NAME
-------------------- -------------------- ------------------------------
HOURLY_EMPLOYEES     YES                  EMPNO_TO_100
HOURLY_EMPLOYEES     YES                  EMPNO_TO_200
```

The following statements add employees to these tables.

```
INSERT INTO hourly_employees VALUES (99, 'FRose', 20.00);
INSERT INTO hourly_employees VALUES (150, 'LRose', 25.00);

INSERT INTO part_time_employees VALUES (50, 'KReilly', 10.00);
```

Repeating the same SELECT statements as before shows that PART_TIME_EMPLOYEES now has a segment, due to the insertion of row data. HOURLY_EMPLOYEES remains as before.

```
SELECT segment_name, partition_name FROM user_segments;

SEGMENT_NAME         PARTITION_NAME
-------------------- ------------------------------
PART_TIME_EMPLOYEES
HOURLY_EMPLOYEES     EMPNO_TO_100
HOURLY_EMPLOYEES     EMPNO_TO_200


SELECT table_name, segment_created FROM user_tables;

TABLE_NAME           SEGMENT_CREATED
-------------------- --------------------
PART_TIME_EMPLOYEES  YES
HOURLY_EMPLOYEES     N/A
```

The USER_TAB_PARTITIONS view does not change.

See Also:

Oracle Database SQL Language Reference for notes and restrictions on deferred segment creation

19.2.15 Materializing Segments

The DBMS_SPACE_ADMIN package includes the MATERIALIZE_DEFERRED_SEGMENTS() procedure, which enables you to materialize segments for tables, table partitions, and dependent objects created with deferred segment creation enabled.

You can add segments as needed, rather than starting with more than you need and using database resources unnecessarily.

The following example materializes segments for the EMPLOYEES table in the HR schema.

```
BEGIN
  DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS(
    schema_name  => 'HR',
    table_name   => 'EMPLOYEES');
END;
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for details about this procedure

19.2.16 Estimate Table Size and Plan Accordingly

Estimate the sizes of tables before creating them. Preferably, do this as part of database planning. Knowing the sizes, and uses, for database tables is an important part of database planning.

You can use the combined estimated size of tables, along with estimates for indexes, undo space, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases.

You can use the estimated size and growth rate of an individual table to better determine the attributes of a tablespace and its underlying data files that are best suited for the table. This can enable you to more easily manage the table disk space and improve I/O performance of applications that use the table.

See Also:

"Capacity Planning for Database Objects "

19.2.17 Restrictions to Consider When Creating Tables

There are restrictions to consider when you create tables.

Here are some restrictions that may affect your table planning and usage:

- Tables containing object types cannot be imported into a pre-Oracle8 database.
- You cannot merge an exported table into a preexisting table having the same name in a different schema.
- You cannot move types and extent tables to a different schema when the original data still exists in the database.
- Oracle Database has a limit on the total number of columns that a table (or attributes that an object type) can have. See *Oracle Database Reference* for this limit.

  Further, when you create a table that contains user-defined type data, the database maps columns of user-defined type to relational columns for storing the user-defined type data. This causes additional relational columns to be created. This results in "hidden" relational columns that are not visible in a `DESCRIBE` table statement and are not returned by a `SELECT *` statement. Therefore, when you create an object table, or a relational table with columns of REF, varray, nested table, or object type, be aware that the total number of columns that the database actually creates for the table can be more than those you specify.

> **See Also:**
>
> *Oracle Database Object-Relational Developer's Guide* for more information about user-defined types

19.3 Creating Tables

Create tables using the SQL statement `CREATE TABLE`.

To create a new table in your schema, you must have the `CREATE TABLE` system privilege. To create a table in another user's schema, you must have the `CREATE ANY TABLE` system privilege. Additionally, the owner of the table must have a quota for the tablespace that contains the table, or the `UNLIMITED TABLESPACE` system privilege.

- Example: Creating a Table
  An example illustrates creating a table.
- Creating a Temporary Table
  Temporary tables are useful in applications where a result set is to be buffered (temporarily persisted), perhaps because it is constructed by running multiple DML operations. You can create either a global temporary table or a private temporary table.
- Parallelizing Table Creation
  When you specify the `AS SELECT` clause to create a table and populate it with data from another table, you can use parallel execution.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for exact syntax of the `CREATE TABLE` and other SQL statements discussed in this chapter
> - *Oracle Database JSON Developer's Guide* for an example of creating a table with JSON columns

19.3.1 Example: Creating a Table

An example illustrates creating a table.

When you issue the following statement, you create a table named `admin_emp` in the `hr` schema and store it in the `admin_tbs` tablespace:

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

```
CREATE TABLE hr.admin_emp (
         empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
         ssn        NUMBER(9) ENCRYPT USING 'AES256',
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         photo      BLOB,
         sal        NUMBER(7,2),
         hrly_rate  NUMBER(7,2) GENERATED ALWAYS AS (sal/2080),
         comm       NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL
                     CONSTRAINT admin_dept_fkey REFERENCES hr.departments
                     (department_id),
         comments   VARCHAR2(32767),
         status     VARCHAR2(10) INVISIBLE)
   TABLESPACE admin_tbs
   STORAGE ( INITIAL 50K);

COMMENT ON TABLE hr.admin_emp IS 'Enhanced employee table';
```

Note the following about this example:

- Integrity constraints are defined on several columns of the table.
- The `STORAGE` clause specifies the size of the first extent. See *Oracle Database SQL Language Reference* for details on this clause.
- Encryption is defined on one column (`ssn`), through the Transparent Data Encryption feature of Oracle Database. The keystore must therefore be open for this `CREATE TABLE` statement to succeed.
- The `photo` column is of data type `BLOB`, which is a member of the set of data types called large objects (LOBs). LOBs are used to store semi-structured data (such as an XML tree) and unstructured data (such as the stream of bits in a color image).
- One column is defined as a virtual column (`hrly_rate`). This column computes the employee's hourly rate as the yearly salary divided by 2,080. See *Oracle Database SQL Language Reference* for a discussion of rules for virtual columns.
- The `comments` column is a `VARCHAR2` column that is larger than 4000 bytes. Beginning with Oracle Database 12*c*, the maximum size for the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types is increased to 32767 bytes.

  To use extended data types, set the `MAX_STRING_SIZE` initialization parameter to `EXTENDED`. See *Oracle Database Reference* for information about setting this parameter.
- The `status` column is invisible.
- A `COMMENT` statement is used to store a comment for the table. You query the `*_TAB_COMMENTS` data dictionary views to retrieve such comments. See *Oracle Database SQL Language Reference* for more information.

See Also:

- *Oracle Database SQL Language Reference* for a description of the data types that you can specify for table columns
- "Managing Integrity Constraints"
- "Understand Invisible Columns"
- *Oracle Database Advanced Security Guide* for information about Transparent Data Encryption
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOBs.

19.3.2 Creating a Temporary Table

Temporary tables are useful in applications where a result set is to be buffered (temporarily persisted), perhaps because it is constructed by running multiple DML operations. You can create either a global temporary table or a private temporary table.

- Overview of Temporary Tables
  A **temporary table** holds data that exists only for the duration of a transaction or session.
- Considerations When Creating Temporary Tables
  Be aware of some considerations when you create temporary tables.
- Creating Global Temporary Tables
  Global temporary tables are permanent database objects that are stored on disk and visible to all sessions connected to the database.
- Creating Private Temporary Tables
  Private temporary tables are temporary database objects that are dropped at the end of a transaction or session. Private temporary tables are stored in memory and each one is visible only to the session that created it.

19.3.2.1 Overview of Temporary Tables

A **temporary table** holds data that exists only for the duration of a transaction or session.

Data in a temporary table is private to the session. Each session can only see and modify its own data.

You can create either a **global temporary table** or a **private temporary table**. The following table shows the essential differences between them.

Table 19-3 Temporary Table Characteristics

Characteristic	Global	Private
Naming rules	Same as for permanent tables	Must be prefixed with `ORA$PTT_`
Visibility of table definition	All sessions	Only the session that created the table
Storage of table definition	Disk	Memory only

Table 19-3 (Cont.) Temporary Table Characteristics

Characteristic	Global	Private
Types	Transaction-specific (`ON COMMIT DELETE ROWS`) or session-specific (`ON COMMIT PRESERVE ROWS`)	Transaction-specific (`ON COMMIT DROP DEFINITION`) or session-specific (`ON COMMIT PRESERVE DEFINITION`)

A third type of temporary table, known as a **cursor-duration temporary table**, is created by the database automatically for certain types of queries.

See Also:

Oracle Database SQL Tuning Guide to learn more about cursor-duration temporary tables

19.3.2.2 Considerations When Creating Temporary Tables

Be aware of some considerations when you create temporary tables.

Unlike permanent tables, temporary tables do not automatically allocate a segment when they are created. Instead, segments are allocated when the first `INSERT` (or `CREATE TABLE AS SELECT`) is performed. Therefore, if a `SELECT`, `UPDATE`, or `DELETE` is performed before the first `INSERT`, then the table appears to be empty.

DDL operations (except `TRUNCATE`) are allowed on an existing temporary table only if no session is currently bound to that temporary table.

If you rollback a transaction, the data you entered is lost, although the table definition persists.

A transaction-specific temporary table allows only one transaction at a time. If there are several autonomous transactions in a single transaction scope, each autonomous transaction can use the table only as soon as the previous one commits.

Because the data in a temporary table is, by definition, temporary, backup and recovery of temporary table data is not available in the event of a system failure. To prepare for such a failure, you should develop alternative methods for preserving temporary table data.

19.3.2.3 Creating Global Temporary Tables

Global temporary tables are permanent database objects that are stored on disk and visible to all sessions connected to the database.

- About Creating Global Temporary Tables
  The metadata of a global temporary table is visible to multiple users and their sessions, but its content is local to a session.
- Examples: Creating a Global Temporary Table
  Examples illustrate how to create a global temporary table.

19.3.2.3.1 About Creating Global Temporary Tables

The metadata of a global temporary table is visible to multiple users and their sessions, but its content is local to a session.

For example, assume a Web-based airlines reservations application allows a customer to create several optional itineraries. Each itinerary is represented by a row in a global temporary table. The application updates the rows to reflect changes in the itineraries. When the customer decides which itinerary she wants to use, the application moves the row for that itinerary to a persistent table.

During the session, the itinerary data is private. At the end of the session, the optional itineraries are dropped.

The definition of a global temporary table is visible to all sessions, but the data in a global temporary table is visible only to the session that inserts the data into the table.

Use the `CREATE GLOBAL TEMPORARY TABLE` statement to create a global temporary table. The `ON COMMIT` clause indicates if the data in the table is **transaction-specific** (the default) or **session-specific**, the implications of which are as follows:

ON COMMIT Setting	Implications
`DELETE ROWS`	This creates a global temporary table that is transaction specific. A session becomes bound to the global temporary table with a transactions first insert into the table. The binding goes away at the end of the transaction. The database truncates the table (delete all rows) after each commit.
`PRESERVE ROWS`	This creates a global temporary table that is session specific. A session gets bound to the global temporary table with the first insert into the table in the session. This binding goes away at the end of the session or by issuing a `TRUNCATE` of the table in the session. The database truncates the table when you terminate the session.

19.3.2.3.2 Examples: Creating a Global Temporary Table

Examples illustrate how to create a global temporary table.

This statement creates a global temporary table that is transaction specific:

```
CREATE GLOBAL TEMPORARY TABLE admin_work_area_trans
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT DELETE ROWS;
```

This statement creates a global temporary table that is session specific:

```
CREATE GLOBAL TEMPORARY TABLE admin_work_area_session
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT PRESERVE ROWS;
```

Indexes can be created on global temporary tables. They are also temporary and the data in the index has the same session or transaction scope as the data in the underlying table.

By default, rows in a global temporary table are stored in the default temporary tablespace of the user who creates it. However, you can assign a global temporary table to another tablespace upon creation of the global temporary table by using the `TABLESPACE` clause of

CREATE GLOBAL TEMPORARY TABLE. You can use this feature to conserve space used by global temporary tables. For example, if you must perform many small global temporary table operations and the default temporary tablespace is configured for sort operations and thus uses a large extent size, these small operations will consume lots of unnecessary disk space. In this case it is better to allocate a second temporary tablespace with a smaller extent size.

The following two statements create a temporary tablespace with a 64 KB extent size, and then a new global temporary table in that tablespace.

```
CREATE TEMPORARY TABLESPACE tbs_t1
    TEMPFILE 'tbs_t1.f' SIZE 50m REUSE AUTOEXTEND ON
    MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 64K;

CREATE GLOBAL TEMPORARY TABLE admin_work_area
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT DELETE ROWS
      TABLESPACE tbs_t1;
```

See Also:

- "About Temporary Tablespaces"
- *Oracle Database SQL Language Reference* for more information about using the CREATE TABLE statement to create a global temporary table, including restrictions that apply

19.3.2.4 Creating Private Temporary Tables

Private temporary tables are temporary database objects that are dropped at the end of a transaction or session. Private temporary tables are stored in memory and each one is visible only to the session that created it.

- About Creating Private Temporary Tables
  The metadata and content of a private temporary table is visible only within the session that created the it.
- Examples: Creating a Private Temporary Table
  These examples illustrate creating a private temporary table.

19.3.2.4.1 About Creating Private Temporary Tables

The metadata and content of a private temporary table is visible only within the session that created the it.

Private temporary tables are useful in the following situations:

- When an application stores temporary data in transient tables that are populated once, read few times, and then dropped at the end of a transaction or session
- When a session is maintained indefinitely and must create different temporary tables for different transactions
- When the creation of a temporary table must not start a new transaction or commit an existing transaction

- When different sessions of the same user must use the same name for a temporary table
- When a temporary table is required for a read-only database

For example, assume a reporting application uses only one schema, but the application uses multiple connections with the schema to run different reports. The sessions use private temporary tables for calculations during individual transactions, and each session creates a private temporary table with the same name. When each transaction commits, its temporary data is no longer needed. Both the definition of a private temporary table and the data in a private temporary table is visible only to the session that created the table.

Use the `CREATE PRIVATE TEMPORARY TABLE` statement to create a private temporary table. The `ON COMMIT` clause indicates if the data in the table is **transaction-specific** (the default) or **session-specific**, the implications of which are as follows:

ON COMMIT Setting	Implications
`DROP DEFINITION`	This creates a private temporary table that is transaction specific. All data in the table is lost, and the table is dropped at the end of transaction.
`PRESERVE DEFINITION`	This creates a private temporary table that is session specific. All data in the table is lost, and the table is dropped at the end of the session that created the table.

Note:

Names of private temporary tables must be prefixed according to the initialization parameter `private_temp_table_prefix`.

19.3.2.4.2 Examples: Creating a Private Temporary Table

These examples illustrate creating a private temporary table.

This statement creates a private temporary table that is transaction specific:

```
CREATE PRIVATE TEMPORARY TABLE ORA$PTT_sales_ptt_transaction
     (time_id      DATE,
      amount_sold  NUMBER(10,2))
   ON COMMIT DROP DEFINITION;
```

This statement creates a private temporary table that is session specific:

```
CREATE PRIVATE TEMPORARY TABLE ORA$PTT_sales_ptt_session
     (time_id      DATE,
      amount_sold  NUMBER(10,2))
   ON COMMIT PRESERVE DEFINITION;
```

By default, rows in a private temporary table are stored in the default temporary tablespace of the user who creates it. However, you can assign a private temporary table to another temporary tablespace during the creation of the temporary table by using the `TABLESPACE` clause of `CREATE PRIVATE TEMPORARY TABLE` statement.

See Also:

- "About Temporary Tablespaces"
- *Oracle Database SQL Language Reference* for more information about using the `CREATE TABLE` statement to create a private temporary table, including restrictions that apply

19.3.3 Parallelizing Table Creation

When you specify the `AS SELECT` clause to create a table and populate it with data from another table, you can use parallel execution.

The `CREATE TABLE...AS SELECT` statement contains two parts: a `CREATE` part (DDL) and a `SELECT` part (query). Oracle Database can parallelize both parts of the statement. The `CREATE` part is parallelized if *one* of the following is true:

- A `PARALLEL` clause is included in the `CREATE TABLE...AS SELECT` statement
- An `ALTER SESSION FORCE PARALLEL DDL` statement is specified

The query part is parallelized if *all* of the following are true:

- The query includes a parallel hint specification (`PARALLEL` or `PARALLEL_INDEX`) *or* the `CREATE` part includes the `PARALLEL` clause *or* the schema objects referred to in the query have a `PARALLEL` declaration associated with them.
- At least one of the tables specified in the query requires either a full table scan *or* an index range scan spanning multiple partitions.

If you parallelize the creation of a table, that table then has a parallel declaration (the `PARALLEL` clause) associated with it. Any subsequent DML or queries on the table, for which parallelization is possible, will attempt to use parallel execution.

The following simple statement parallelizes the creation of a table and stores the result in a compressed format, using table compression:

```
CREATE TABLE hr.admin_emp_dept
     PARALLEL COMPRESS
     AS SELECT * FROM hr.employees
     WHERE department_id = 10;
```

In this case, the `PARALLEL` clause tells the database to select an optimum number of parallel execution servers when creating the table.

See Also:

- *Oracle Database VLDB and Partitioning Guide* for detailed information on using parallel execution
- "Managing Processes for Parallel SQL Execution"

19.4 Loading Tables

There are several techniques for loading data into tables.

> **Note:**
>
> The default size of the first extent of any new segment for a partitioned table is 8 MB instead of 64 KB. This helps improve performance of inserts and queries on partitioned tables. Although partitioned tables will start with a larger initial size, once sufficient data is inserted, the space consumption will be the same as in previous releases. You can override this default by setting the `INITIAL` size in the storage clause for the table. This new default only applies to table partitions and LOB partitions.

- Methods for Loading Tables
  There are several means of inserting or initially loading data into your tables.
- Improving INSERT Performance with Direct-Path INSERT
  When loading large amounts of data, you can improve load performance by using direct-path `INSERT`.
- Using Conventional Inserts to Load Tables
  During **conventional INSERT operations**, the database reuses free space in the table, interleaving newly inserted data with existing data. During such operations, the database also maintains referential integrity constraints. Unlike direct-path `INSERT` operations, conventional `INSERT` operations do not require an exclusive lock on the table.
- Avoiding Bulk INSERT Failures with DML Error Logging
  You can avoid bulk `INSERT` failures by using the DML error logging feature.

19.4.1 Methods for Loading Tables

There are several means of inserting or initially loading data into your tables.

Most commonly used are the following:

Method	Description
SQL*Loader	This Oracle utility program loads data from external files into tables of an Oracle Database. Starting with Oracle Database 12*c*, SQL*Loader supports express mode. SQL*Loader express mode eliminates the need for a control file. Express mode simplifies loading data from external files. With express mode, SQL*Loader attempts to use the external table load method. If the external table load method is not possible, then SQL*Loader attempts to use direct path. If direct path is not possible, then SQL*Loader uses conventional path. SQL*Loader express mode automatically identifies the input datatypes based on the table column types and controls parallelism. SQL*Loader uses defaults to simplify usage, but you can override many of the defaults with command line parameters. You optionally can specify the direct path or the conventional path load method instead of using express mode. For information about SQL*Loader, see *Oracle Database Utilities*.
`CREATE TABLE` ... `AS SELECT` statement (CTAS)	Using this SQL statement you can create a table and populate it with data selected from another existing table, including an external table.
`INSERT` statement	The `INSERT` statement enables you to add rows to a table, either by specifying the column values or by specifying a subquery that selects data from another existing table, including an external table. One form of the `INSERT` statement enables direct-path `INSERT`, which can improve performance, and is useful for bulk loading. See "Improving INSERT Performance with Direct-Path INSERT". If you are inserting a lot of data and want to avoid statement termination and rollback if an error is encountered, you can insert with DML error logging. See "Avoiding Bulk INSERT Failures with DML Error Logging".
`MERGE` statement	The `MERGE` statement enables you to insert rows into or update rows of a table, by selecting rows from another existing table. If a row in the new data corresponds to an item that already exists in the table, then an `UPDATE` is performed, else an `INSERT` is performed.

Note:

Only a few details and examples of inserting data into tables are included in this book. Oracle documentation specific to data warehousing and application development provide more extensive information about inserting and manipulating data in tables. See:

- *Oracle Database Data Warehousing Guide*
- *Oracle Database SecureFiles and Large Objects Developer's Guide*

See Also:

"Managing External Tables"

19.4.2 Improving INSERT Performance with Direct-Path INSERT

When loading large amounts of data, you can improve load performance by using direct-path `INSERT`.

- About Direct-Path INSERT
  Direct-path insert operations are typically faster than conventional insert operations.
- How Direct-Path INSERT Works
  You can use direct-path `INSERT` on both partitioned and nonpartitioned tables.
- Loading Data with Direct-Path INSERT
  You can load data with direct-path `INSERT` by using direct-path `INSERT` SQL statements, inserting data in parallel mode, or by using the Oracle SQL*Loader utility in direct-path mode. A direct-path `INSERT` can be done in either serial or parallel mode.
- Logging Modes for Direct-Path INSERT
  Direct-path `INSERT` lets you choose whether to log redo and undo information during the insert operation.
- Additional Considerations for Direct-Path INSERT

19.4.2.1 About Direct-Path INSERT

Direct-path insert operations are typically faster than conventional insert operations.

Oracle Database inserts data into a table in one of two ways:

- During **conventional INSERT operations**, the database reuses free space in the table, interleaving newly inserted data with existing data. During such operations, the database also maintains referential integrity constraints.
- During **direct-path INSERT operations**, the database appends the inserted data after existing data in the table. Data is written directly into data files, bypassing the buffer cache. Free space in the table is not reused, and referential integrity constraints are ignored. Direct-path `INSERT` can perform significantly better than conventional insert.

The database can insert data either in serial mode, where one process executes the statement, or in parallel mode, where multiple processes work together simultaneously to run a single SQL statement. The latter is referred to as parallel execution.

The following are benefits of direct-path `INSERT`:

- During direct-path `INSERT`, you can disable the logging of redo and undo entries to reduce load time. Conventional insert operations, in contrast, must always log such entries, because those operations reuse free space and maintain referential integrity.
- Direct-path `INSERT` operations ensure atomicity of the transaction, even when run in parallel mode. Atomicity cannot be guaranteed during parallel direct path loads (using SQL*Loader).

When performing parallel direct path loads, one notable difference between SQL*Loader and `INSERT` statements is the following: If errors occur during parallel direct path loads with SQL*Loader, the load completes, but some indexes could be marked `UNUSABLE` at the end of the load. Parallel direct-path `INSERT`, in contrast, rolls back the statement if errors occur during index update.

Note:

A conventional `INSERT` operation checks for violations of `NOT NULL` constraints during the insert. Therefore, if a `NOT NULL` constraint is violated for a conventional `INSERT` operation, then the error is returned during the insert. A direct-path `INSERT` operation checks for violations of `NOT NULL` constraints before the insert. Therefore, if a `NOT NULL` constraint is violated for a direct-path `INSERT` operation, then the error is returned before the insert.

19.4.2.2 How Direct-Path INSERT Works

You can use direct-path `INSERT` on both partitioned and nonpartitioned tables.

- Serial Direct-Path INSERT into Partitioned or Nonpartitioned Tables
  The single process inserts data beyond the current high water mark of the table segment or of each partition segment. (The **high-water mark** is the level at which blocks have never been formatted to receive data.) When a `COMMIT` runs, the high-water mark is updated to the new value, making the data visible to users.
- Parallel Direct-Path INSERT into Partitioned Tables
  This situation is analogous to serial direct-path `INSERT`. Each parallel execution server is assigned one or more partitions, with no more than one process working on a single partition.
- Parallel Direct-Path INSERT into Nonpartitioned Tables
  Each parallel execution server allocates a new temporary segment and inserts data into that temporary segment. When a `COMMIT` runs, the parallel execution coordinator merges the new temporary segments into the primary table segment, where it is visible to users.

19.4.2.2.1 Serial Direct-Path INSERT into Partitioned or Nonpartitioned Tables

The single process inserts data beyond the current high water mark of the table segment or of each partition segment. (The **high-water mark** is the level at which blocks have never been formatted to receive data.) When a `COMMIT` runs, the high-water mark is updated to the new value, making the data visible to users.

19.4.2.2.2 Parallel Direct-Path INSERT into Partitioned Tables

This situation is analogous to serial direct-path `INSERT`. Each parallel execution server is assigned one or more partitions, with no more than one process working on a single partition.

Each parallel execution server inserts data beyond the current high-water mark of its assigned partition segment(s). When a `COMMIT` runs, the high-water mark of each partition segment is updated to its new value, making the data visible to users.

19.4.2.2.3 Parallel Direct-Path INSERT into Nonpartitioned Tables

Each parallel execution server allocates a new temporary segment and inserts data into that temporary segment. When a `COMMIT` runs, the parallel execution coordinator merges the new temporary segments into the primary table segment, where it is visible to users.

19.4.2.3 Loading Data with Direct-Path INSERT

You can load data with direct-path `INSERT` by using direct-path `INSERT` SQL statements, inserting data in parallel mode, or by using the Oracle SQL*Loader utility in direct-path mode. A direct-path `INSERT` can be done in either serial or parallel mode.

- Serial Mode Inserts with SQL Statements
  There are various ways to activate direct-path `INSERT` in serial mode with SQL.
- Parallel Mode Inserts with SQL Statements
  When you are inserting in parallel mode, direct-path `INSERT` is the default. However, you can insert in parallel mode using conventional `INSERT` by using the `NOAPPEND PARALLEL` hint.

19.4.2.3.1 Serial Mode Inserts with SQL Statements

There are various ways to activate direct-path `INSERT` in serial mode with SQL.

You can activate direct-path `INSERT` in serial mode with SQL in the following ways:

- If you are performing an `INSERT` with a subquery, specify the `APPEND` hint in each `INSERT` statement, either immediately after the `INSERT` keyword, or immediately after the `SELECT` keyword in the subquery of the `INSERT` statement.
- If you are performing an `INSERT` with the `VALUES` clause, specify the `APPEND_VALUES` hint in each `INSERT` statement immediately after the `INSERT` keyword. Direct-path `INSERT` with the `VALUES` clause is best used when there are hundreds of thousands or millions of rows to load. The typical usage scenario is for array inserts using OCI. Another usage scenario might be inserts in a `FORALL` statement in PL/SQL.

If you specify the `APPEND` hint (as opposed to the `APPEND_VALUES` hint) in an `INSERT` statement with a `VALUES` clause, the `APPEND` hint is ignored and a conventional insert is performed.

The following is an example of using the `APPEND` hint to perform a direct-path `INSERT`:

```
INSERT /*+ APPEND */ INTO sales_hist SELECT * FROM sales WHERE cust_id=8890;
```

The following PL/SQL code fragment is an example of using the `APPEND_VALUES` hint:

```
FORALL i IN 1..numrecords
  INSERT /*+ APPEND_VALUES */ INTO orderdata
  VALUES(ordernum(i), custid(i), orderdate(i),shipmode(i), paymentid(i));
COMMIT;
```

19.4.2.3.2 Parallel Mode Inserts with SQL Statements

When you are inserting in parallel mode, direct-path `INSERT` is the default. However, you can insert in parallel mode using conventional `INSERT` by using the `NOAPPEND PARALLEL` hint.

To run in parallel DML mode, the following requirements must be met:

- You must have Oracle Enterprise Edition installed.
- You must enable parallel DML in your session. To do this, submit the following statement:

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- You must meet at least one of the following requirements:
  - Specify the parallel attribute for the target table, either at create time or subsequently
  - Specify the `PARALLEL` hint for each insert operation

  – Set the database initialization parameter `PARALLEL_DEGREE_POLICY` to `AUTO`

To disable direct-path `INSERT`, specify the `NOAPPEND` hint in each `INSERT` statement. Doing so overrides parallel DML mode.

> **Note:**
>
> You cannot query or modify data inserted using direct-path `INSERT` immediately after the insert is complete. If you attempt to do so, an ORA-12838 error is generated. You must first issue a `COMMIT` statement before attempting to read or modify the newly-inserted data.

> **See Also:**
>
> - "Using Conventional Inserts to Load Tables"
> - *Oracle Database SQL Tuning Guide* for more information on using hints
> - *Oracle Database SQL Language Reference* for more information on the subquery syntax of `INSERT` statements and for additional restrictions on using direct-path `INSERT`

19.4.2.4 Logging Modes for Direct-Path INSERT

Direct-path `INSERT` lets you choose whether to log redo and undo information during the insert operation.

You specify the logging mode for direct-path `INSERT` in the following ways:

- You can specify logging mode for a table, partition, index, or `LOB` storage at create time (in a `CREATE` statement) or subsequently (in an `ALTER` statement).
- If you do not specify either `LOGGING` or `NOLOGGING` at these times:
  - The logging attribute of a partition defaults to the logging attribute of its table.
  - The logging attribute of a table or index defaults to the logging attribute of the tablespace in which it resides.
  - The logging attribute of `LOB` storage defaults to `LOGGING` if you specify `CACHE` for `LOB` storage. If you do not specify `CACHE`, then the logging attributes defaults to that of the tablespace in which the `LOB` values resides.
- You set the logging attribute of a tablespace in a `CREATE TABLESPACE` or `ALTER TABLESPACE` statements.

> **Note:**
>
> If the database or tablespace is in `FORCE LOGGING` mode, then direct-path `INSERT` always logs, regardless of the logging setting.

- Direct-Path INSERT with Logging
  In this mode, Oracle Database performs full redo logging for instance and media recovery.
- Direct-Path INSERT without Logging
  In this mode, Oracle Database inserts data without redo or undo logging. Instead, the database logs a small number of block range invalidation redo records and periodically updates the control file with information about the most recent direct write.

19.4.2.4.1 Direct-Path INSERT with Logging

In this mode, Oracle Database performs full redo logging for instance and media recovery.

If the database is in `ARCHIVELOG` mode, then you can archive redo logs to tape. If the database is in `NOARCHIVELOG` mode, then you can recover instance crashes but not disk failures.

19.4.2.4.2 Direct-Path INSERT without Logging

In this mode, Oracle Database inserts data without redo or undo logging. Instead, the database logs a small number of block range invalidation redo records and periodically updates the control file with information about the most recent direct write.

Direct-path `INSERT` without logging improves performance. However, if you subsequently must perform media recovery, the invalidation redo records mark a range of blocks as logically corrupt, because no redo data was logged for them. Therefore, it is important that you back up the data after such an insert operation.

You can significantly improve the performance of unrecoverable direct-path inserts by disabling the periodic update of the control files. You do so by setting the initialization parameter `DB_UNRECOVERABLE_SCN_TRACKING` to `FALSE`. However, if you perform an unrecoverable direct-path insert with these control file updates disabled, you will no longer be able to accurately query the database to determine if any data files are currently unrecoverable.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information about unrecoverable data files
- The section "Determining If a Backup Is Required After Unrecoverable Operations" in *Oracle Data Guard Concepts and Administration*

19.4.2.5 Additional Considerations for Direct-Path INSERT

When using direct-path `INSERT`, consider issues related to compressed tables, index maintenance, disk space, and locking.

- Compressed Tables and Direct-Path INSERT
  If a table is created with the basic table compression, then you must use direct-path `INSERT` to compress table data as it is loaded. If a table is created with advanced row, warehouse, or archive compression, then best compression ratios are achieved with direct-path `INSERT`.
- Index Maintenance with Direct-Path INSERT
  Oracle Database performs index maintenance at the end of direct-path `INSERT` operations on tables (partitioned or nonpartitioned) that have indexes.

- Space Considerations with Direct-Path INSERT
  Direct-path `INSERT` requires more space than conventional path `INSERT`.
- Locking Considerations with Direct-Path INSERT
  During direct-path `INSERT`, the database obtains exclusive locks on the table (or on all partitions of a partitioned table).

19.4.2.5.1 Compressed Tables and Direct-Path INSERT

If a table is created with the basic table compression, then you must use direct-path `INSERT` to compress table data as it is loaded. If a table is created with advanced row, warehouse, or archive compression, then best compression ratios are achieved with direct-path `INSERT`.

See "Consider Using Table Compression" for more information.

19.4.2.5.2 Index Maintenance with Direct-Path INSERT

Oracle Database performs index maintenance at the end of direct-path `INSERT` operations on tables (partitioned or nonpartitioned) that have indexes.

This index maintenance is performed by the parallel execution servers for parallel direct-path `INSERT` or by the single process for serial direct-path `INSERT`. You can avoid the performance impact of index maintenance by making the index unusable before the `INSERT` operation and then rebuilding it afterward.

See Also:

"Making an Index Unusable"

19.4.2.5.3 Space Considerations with Direct-Path INSERT

Direct-path `INSERT` requires more space than conventional path `INSERT`.

All serial direct-path `INSERT` operations, as well as parallel direct-path `INSERT` into partitioned tables, insert data above the high-water mark of the affected segment. This requires some additional space.

Parallel direct-path `INSERT` into nonpartitioned tables requires even more space, because it creates a temporary segment for each degree of parallelism. If the nonpartitioned table is not in a locally managed tablespace in automatic segment-space management mode, you can modify the values of the `NEXT` and `PCTINCREASE` storage parameter and `MINIMUM EXTENT` tablespace parameter to provide sufficient (but not excess) storage for the temporary segments. Choose values for these parameters so that:

- The size of each extent is not too small (no less than 1 MB). This setting affects the total number of extents in the object.
- The size of each extent is not so large that the parallel `INSERT` results in wasted space on segments that are larger than necessary.

After the direct-path `INSERT` operation is complete, you can reset these parameters to settings more appropriate for serial operations.

19.4.2.5.4 Locking Considerations with Direct-Path INSERT

During direct-path `INSERT`, the database obtains exclusive locks on the table (or on all partitions of a partitioned table).

As a result, users cannot perform any concurrent insert, update, or delete operations on the table, and concurrent index creation and build operations are not permitted. Concurrent queries, however, are supported, but the query will return only the information before the insert operation.

19.4.3 Using Conventional Inserts to Load Tables

During **conventional INSERT operations**, the database reuses free space in the table, interleaving newly inserted data with existing data. During such operations, the database also maintains referential integrity constraints. Unlike direct-path `INSERT` operations, conventional `INSERT` operations do not require an exclusive lock on the table.

Several other restrictions apply to direct-path `INSERT` operations that do not apply to conventional `INSERT` operations. See *Oracle Database SQL Language Reference* for information about these restrictions.

You can perform a conventional `INSERT` operation in serial mode or in parallel mode using the `NOAPPEND` hint.

The following is an example of using the `NOAPPEND` hint to perform a conventional `INSERT` in serial mode:

```
INSERT /*+ NOAPPEND */ INTO sales_hist SELECT * FROM sales WHERE cust_id=8890;
```

The following is an example of using the `NOAPPEND` hint to perform a conventional `INSERT` in parallel mode:

```
INSERT /*+ NOAPPEND PARALLEL */ INTO sales_hist
   SELECT * FROM sales;
```

To run in parallel DML mode, the following requirements must be met:

- You must have Oracle Enterprise Edition installed.
- You must enable parallel DML in your session. To do this, submit the following statement:

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- You must meet at least one of the following requirements:
  – Specify the parallel attribute for the target table, either at create time or subsequently
  – Specify the `PARALLEL` hint for each insert operation
  – Set the database initialization parameter `PARALLEL_DEGREE_POLICY` to `AUTO`

19.4.4 Avoiding Bulk INSERT Failures with DML Error Logging

You can avoid bulk `INSERT` failures by using the DML error logging feature.

- Inserting Data with DML Error Logging
  When you load a table using an `INSERT` statement with subquery, if an error occurs, the statement is terminated and rolled back in its entirety. This can be wasteful of time and

system resources. For such `INSERT` statements, you can avoid this situation by using the DML error logging feature.

- Error Logging Table Format
  The error logging table has a specific format.
- Creating an Error Logging Table
  You can create an error logging table manually, or you can use a PL/SQL package to automatically create one for you.
- Error Logging Restrictions and Caveats
  Some errors are not logged in error logging tables.

19.4.4.1 Inserting Data with DML Error Logging

When you load a table using an `INSERT` statement with subquery, if an error occurs, the statement is terminated and rolled back in its entirety. This can be wasteful of time and system resources. For such `INSERT` statements, you can avoid this situation by using the DML error logging feature.

To use DML error logging, you add a statement clause that specifies the name of an error logging table into which the database records errors encountered during DML operations. When you add this error logging clause to the `INSERT` statement, certain types of errors no longer terminate and roll back the statement. Instead, each error is logged and the statement continues. You then take corrective action on the erroneous rows at a later time.

DML error logging works with `INSERT`, `UPDATE`, `MERGE`, and `DELETE` statements. This section focuses on `INSERT` statements.

To insert data with DML error logging:

1. Create an error logging table. (Optional)

   You can create the table manually or use the `DBMS_ERRLOG` package to automatically create it for you. See "Creating an Error Logging Table" for details.
2. Execute an `INSERT` statement and include an error logging clause. This clause:
   - Optionally references the error logging table that you created. If you do not provide an error logging table name, the database logs to an error logging table with a default name. The default error logging table name is `ERR$_` followed by the first 25 characters of the name of the table that is being inserted into.
   - Optionally includes a **tag** (a numeric or string literal in parentheses) that gets added to the error log to help identify the statement that caused the errors. If the tag is omitted, a `NULL` value is used.
   - Optionally includes a `REJECT LIMIT` subclause.

     This subclause indicates the maximum number of errors that can be encountered before the `INSERT` statement terminates and rolls back. You can also specify `UNLIMITED`. The default reject limit is zero, which means that upon encountering the first error, the error is logged and the statement rolls back. For parallel DML operations, the reject limit is applied to each parallel execution server.

   Note:

   If the statement exceeds the reject limit and rolls back, the error logging table retains the log entries recorded so far.

See *Oracle Database SQL Language Reference* for error logging clause syntax information.

3. Query the error logging table and take corrective action for the rows that generated errors.

   See "Error Logging Table Format", later in this section, for details on the error logging table structure.

Example 19-9 Inserting Data with DML Error Logging

The following statement inserts rows into the `DW_EMPL` table and logs errors to the `ERR_EMPL` table. The tag '`daily_load`' is copied to each log entry. The statement terminates and rolls back if the number of errors exceeds 25.

```
INSERT INTO dw_empl
  SELECT employee_id, first_name, last_name, hire_date, salary, department_id
  FROM employees
  WHERE hire_date > sysdate - 7
  LOG ERRORS INTO err_empl ('daily_load') REJECT LIMIT 25
```

For more examples, see *Oracle Database SQL Language Reference* and *Oracle Database Data Warehousing Guide*.

19.4.4.2 Error Logging Table Format

The error logging table has a specific format.

The error logging table consists of two parts:

- A mandatory set of columns that describe the error. For example, one column contains the Oracle error number.

  Table 19-4 lists these error description columns.

- An optional set of columns that contain data from the row that caused the error. The column names match the column names from the table being inserted into (the "DML table").

  The number of columns in this part of the error logging table can be zero, one, or more, up to the number of columns in the DML table. If a column exists in the error logging table that has the same name as a column in the DML table, the corresponding data from the offending row being inserted is written to this error logging table column. If a DML table column does not have a corresponding column in the error logging table, the column is not logged. If the error logging table contains a column with a name that does not match a DML table column, the column is ignored.

  Because type conversion errors are one type of error that might occur, the data types of the optional columns in the error logging table must be types that can capture any value without data loss or conversion errors. (If the optional log columns were of the same types as the DML table columns, capturing the problematic data into the log could suffer the same data conversion problem that caused the error.) The database makes a best effort to log a meaningful value for data that causes conversion errors. If a value cannot be derived, `NULL` is logged for the column. An error on insertion into the error logging table causes the statement to terminate.

  Table 19-5 lists the recommended error logging table column data types to use for each data type from the DML table. These recommended data types are used when you create the error logging table automatically with the `DBMS_ERRLOG` package.

Table 19-4 Mandatory Error Description Columns

Column Name	Data Type	Description
`ORA_ERR_NUMBER$`	`NUMBER`	Oracle error number
`ORA_ERR_MESG$`	`VARCHAR2(2000)`	Oracle error message text
`ORA_ERR_ROWID$`	`ROWID`	Rowid of the row in error (for update and delete)
`ORA_ERR_OPTYP$`	`VARCHAR2(2)`	Type of operation: insert (`I`), update (`U`), delete (`D`) Note: Errors from the update clause and insert clause of a `MERGE` operation are distinguished by the `U` and `I` values.
`ORA_ERR_TAG$`	`VARCHAR2(2000)`	Value of the tag supplied by the user in the error logging clause

Table 19-5 Error Logging Table Column Data Types

DML Table Column Type	Error Logging Table Column Type	Notes
`NUMBER`	`VARCHAR2(4000)`	Able to log conversion errors
`CHAR/VARCHAR2(n)`	`VARCHAR2(4000)`	Logs any value without information loss
`NCHAR/NVARCHAR2(n)`	`NVARCHAR2(4000)`	Logs any value without information loss
`DATE/TIMESTAMP`	`VARCHAR2(4000)`	Logs any value without information loss. Converts to character format with the default date/time format mask
`RAW`	`RAW(2000)`	Logs any value without information loss
`ROWID`	`UROWID`	Logs any rowid type
`LONG/LOB`		Not supported
User-defined types		Not supported

19.4.4.3 Creating an Error Logging Table

You can create an error logging table manually, or you can use a PL/SQL package to automatically create one for you.

- Creating an Error Logging Table Automatically
  You use the `DBMS_ERRLOG` package to automatically create an error logging table.
- Creating an Error Logging Table Manually
  You use standard DDL to manually create the error logging table.

19.4.4.3.1 Creating an Error Logging Table Automatically

You use the `DBMS_ERRLOG` package to automatically create an error logging table.

The `CREATE_ERROR_LOG` procedure creates an error logging table with all of the mandatory error description columns plus all of the columns from the named DML table, and performs the data type mappings shown in Table 19-5.

The following statement creates the error logging table used in the previous example.

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('DW_EMPL', 'ERR_EMPL');
```

See *Oracle Database PL/SQL Packages and Types Reference* for details on `DBMS_ERRLOG`.

19.4.4.3.2 Creating an Error Logging Table Manually

You use standard DDL to manually create the error logging table.

See "Error Logging Table Format" for table structure requirements. You must include all mandatory error description columns. They can be in any order, but must be the first columns in the table.

19.4.4.4 Error Logging Restrictions and Caveats

Some errors are not logged in error logging tables.

Oracle Database logs the following errors during DML operations:

- Column values that are too large
- Constraint violations (`NOT NULL`, unique, referential, and check constraints)
- Errors raised during trigger execution
- Errors resulting from type conversion between a column in a subquery and the corresponding column of the table
- Partition mapping errors
- Certain `MERGE` operation errors (`ORA-30926`: Unable to get a stable set of rows for `MERGE` operation.)

Some errors are not logged, and cause the DML operation to terminate and roll back. For a list of these errors and for other DML logging restrictions, see the discussion of the `error_logging_clause` in the `INSERT` section of *Oracle Database SQL Language Reference*.

- Space Considerations
  Ensure that you consider space requirements before using DML error logging. You require available space not only for the table being inserted into, but also for the error logging table.
- Security
  The user who issues the `INSERT` statement with DML error logging must have `INSERT` privileges on the error logging table.

19.4.4.4.1 Space Considerations

Ensure that you consider space requirements before using DML error logging. You require available space not only for the table being inserted into, but also for the error logging table.

19.4.4.4.2 Security

The user who issues the `INSERT` statement with DML error logging must have `INSERT` privileges on the error logging table.

See Also:

Oracle Database SQL Language Reference and *Oracle Database Data Warehousing Guide* for DML error logging examples.

19.5 Optimizing the Performance of Bulk Updates

The `EXECUTE_UPDATE` procedure in the `DBMS_REDEFINITION` package can optimize the performance of bulk updates to a table. Performance is optimized because the updates are not logged in the redo log.

The `EXECUTE_UPDATE` procedure automatically uses the components of online table redefinition, such an interim table, a materialized view, and a materialized view log, to enable optimized bulk updates to a table. The `EXECUTE_UPDATE` procedure also removes fragmentation of the affected rows and ensures that the update is atomic. If the bulk updates raise any errors, then you can use the `ABORT_UPDATE` procedure to undo the changes made by the `EXECUTE_UPDATE` procedure.

The following restrictions apply to the `EXECUTE_UPDATE` procedure:

- All of the restrictions that apply to online table redefinition apply to the `EXECUTE_UPDATE` procedure and the `ABORT_UPDATE` procedure.
- You cannot run more than one `EXECUTE_UPDATE` procedure on a table at the same time.
- While the `EXECUTE_UPDATE` procedure is running on a table, do not make DML changes on the table from a different session. These DML changes are lost when the `EXECUTE_UPDATE` procedure completes.
- The table cannot have any triggers that fire on `UPDATE` statements.
- The `UPDATE` statement passed to the `EXECUTE_UPDATE` procedure cannot have a table with a partition-extended name.
- The table cannot have the following user-defined types: varrays, REFs, and nested tables.
- The table cannot have the following Oracle-supplied types: `ANYTYPE`, `ANYDATASET`, URI types, `SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`, and `Expression`.
- The table cannot have the following types of columns: hidden column, virtual column, unused column, pseudocolumns, or identity column.
- The table cannot be an object table.
- The table cannot have a Virtual Private Database (VPD) policy.
- The table cannot have check constraints.
- The table cannot be enabled for row archival.

To optimize the performance of bulk updates:

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".
2. Run the `EXECUTE_UPDATE` procedure, and specify the SQL statement that performs the bulk update.

If errors result, then use the ABORT_UPDATE procedure to undo the changes made by the EXECUTE_UPDATE procedure.

3. Perform a back up of the updated data.

   Because the EXECUTE_UPDATE procedure does not log changes in the redo log, recovery is not possible until you perform a back up of the database or of the tablespace that contains the updated table.

Example 19-10 Performing an Optimized Bulk Update of Product Data

This example performs a bulk update on the oe.order_items table. Specifically, it sets the unit_price of each order item with a product_id of 3106 to 45. If the bulk update fails, then the ABORT_UPDATE procedure cancels all of the changes performed by the EXECUTE_UPDATE procedure, which returns the data to its state before the procedure was run.

```
DECLARE
   update_stmt VARCHAR2(300) := 'UPDATE oe.order_items SET unit_price = 45
                                    WHERE product_id = 3106';
BEGIN
   DBMS_REDEFINITION.EXECUTE_UPDATE(update_stmt);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('No Data found for SELECT');
    DBMS_REDEFINITION.ABORT_UPDATE(update_stmt);
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('Reason for failure is'|| SQLERRM);
       IF (SQLCODE = 100)
       THEN
          DBMS_REDEFINITION.ABORT_UPDATE(update_stmt);
       END IF;
END;
/
```

19.6 Automatically Collecting Statistics on Tables

The PL/SQL package DBMS_STATS lets you generate and manage statistics for cost-based optimization. You can use this package to gather, modify, view, export, import, and delete statistics. You can also use this package to identify or name statistics that have been gathered.

Formerly, you enabled DBMS_STATS to automatically gather statistics for a table by specifying the MONITORING keyword in the CREATE (or ALTER) TABLE statement. The MONITORING and NOMONITORING keywords have been deprecated and statistics are collected automatically. If you do specify these keywords, they are ignored.

Monitoring tracks the approximate number of INSERT, UPDATE, and DELETE operations for the table since the last time statistics were gathered. Information about how many rows are affected is maintained in the SGA, until periodically (about every three hours) SMON incorporates the data into the data dictionary. This data dictionary information is made visible through the DBA_TAB_MODIFICATIONS,ALL_TAB_MODIFICATIONS, or USER_TAB_MODIFICATIONS views. The database uses these views to identify tables with stale statistics.

The default for the STATISTICS_LEVEL initialization parameter is TYPICAL, which enables automatic statistics collection. Automatic statistics collection and the DBMS_STATS package enable the optimizer to generate accurate execution plans. Setting the STATISTICS_LEVEL initialization parameter to BASIC disables the collection of many of the important statistics required by Oracle Database features and functionality. To disable monitoring of all tables, set the `STATISTICS_LEVEL` initialization parameter to `BASIC`. Automatic statistics collection and the `DBMS_STATS` package enable the optimizer to generate accurate execution plans.

See Also:

- *Oracle Database Reference* for detailed information on the `STATISTICS_LEVEL` initialization parameter
- *Oracle Database SQL Tuning Guide* for information on managing optimizer statistics
- *Oracle Database PL/SQL Packages and Types Reference* for information about using the `DBMS_STATS` package
- "About Automated Maintenance Tasks" for information on using the Scheduler to collect statistics automatically

19.7 Altering Tables

You alter a table using the `ALTER TABLE` statement. To alter a table, the table must be contained in your schema, or you must have either the `ALTER` object privilege for the table or the `ALTER ANY TABLE` system privilege.

Note:

Before altering a table, familiarize yourself with the consequences of doing so. The *Oracle Database SQL Language Reference* lists many of these consequences in the descriptions of the `ALTER TABLE` clauses.

If a view, materialized view, trigger, domain index, function-based index, check constraint, function, procedure of package depends on a base table, the alteration of the base table or its columns can affect the dependent object. See "Managing Object Dependencies" for information about how the database manages dependencies.

- Reasons for Using the ALTER TABLE Statement
  There are several reasons to use the `ALTER TABLE` statement.
- Altering Physical Attributes of a Table
  There are several considerations when you alter the physical attributes of a table.
- Moving a Table to a New Segment or Tablespace
  You can move a table to a new segment or tablespace to enable compression or to perform data maintenance.
- Manually Allocating Storage for a Table
  Oracle Database dynamically allocates additional extents for the data segment of a table, as required. However, perhaps you want to allocate an additional extent for a table explicitly. For example, in an Oracle Real Application Clusters environment, an extent of a table can be allocated explicitly for a specific instance.

- Modifying an Existing Column Definition
  Use the `ALTER TABLE...MODIFY` statement to modify an existing column definition. You can modify column data type, default value, column constraint, column expression (for virtual columns), column encryption, and visible/invisible property.
- Adding Table Columns
  To add a column to an existing table, use the `ALTER TABLE...ADD` statement.
- Renaming Table Columns
  Oracle Database lets you rename existing columns in a table. Use the `RENAME COLUMN` clause of the `ALTER TABLE` statement to rename a column.
- Dropping Table Columns
  You can drop columns that are no longer needed from a table, including an index-organized table. This provides a convenient means to free space in a database, and avoids your having to export/import data then re-create indexes and constraints.
- Placing a Table in Read-Only Mode
  You can place a table in read-only mode with the `ALTER TABLE`...`READ ONLY` statement, and return it to read/write mode with the `ALTER TABLE`...`READ WRITE` statement.

19.7.1 Reasons for Using the ALTER TABLE Statement

There are several reasons to use the `ALTER TABLE` statement.

You can use the `ALTER TABLE` statement to perform any of the following actions that affect a table:

- Modify physical characteristics (`INITRANS` or storage parameters)
- Move the table to a new segment or tablespace
- Explicitly allocate an extent or deallocate unused space
- Add, drop, or rename columns, or modify an existing column definition (data type, length, default value, `NOT NULL` integrity constraint, column expression (for virtual columns), and encryption properties.)
- Modify the logging attributes of the table
- Modify the `CACHE`/`NOCACHE` attributes
- Add, modify or drop integrity constraints associated with the table
- Enable or disable integrity constraints or triggers associated with the table
- Modify the degree of parallelism for the table
- Rename a table
- Put a table in read-only mode and return it to read/write mode
- Add or modify index-organized table characteristics
- Alter the characteristics of an external table
- Add or modify `LOB` columns
- Add or modify object type, nested table, or varray columns
- Modify table partitions

  Starting with Oracle Database 12*c*, you can perform some operations on more than two partitions or subpartitions at a time, such as split partition and merge partitions operations. See *Oracle Database VLDB and Partitioning Guide* for information.

Many of these operations are discussed in succeeding sections.

19.7.2 Altering Physical Attributes of a Table

There are several considerations when you alter the physical attributes of a table.

When altering the transaction entry setting `INITRANS` of a table, note that a new setting for `INITRANS` applies only to data blocks subsequently allocated for the table.

The storage parameters `INITIAL` and `MINEXTENTS` cannot be altered. All new settings for the other storage parameters (for example, `NEXT`, `PCTINCREASE`) affect only extents subsequently allocated for the table. The size of the next extent allocated is determined by the current values of `NEXT` and `PCTINCREASE`, and is not based on previous values of these parameters.

See Also:

The discussions of the physical attributes clause and the storage clause in *Oracle Database SQL Language Reference*

19.7.3 Moving a Table to a New Segment or Tablespace

You can move a table to a new segment or tablespace to enable compression or to perform data maintenance.

- About Moving a Table to a New Segment or Tablespace
  The `ALTER TABLE...MOVE [PARTITION|SUBPARTITION]` statement enables you to move a table, partition, or subpartition to change any physical storage attribute, such as compression, or the tablespace, assuming you have the appropriate quota in the target tablespace.
- Moving a Table
  Use the `ALTER TABLE...MOVE` statement to move a table to a new segment or tablespace.
- Moving a Table Partition or Subpartition Online
  Use the `ALTER TABLE...MOVE PARTITION` statement or `ALTER TABLE...MOVE SUBPARTITION` statement to move a table partition or subpartition, respectively.

19.7.3.1 About Moving a Table to a New Segment or Tablespace

The `ALTER TABLE...MOVE [PARTITION|SUBPARTITION]` statement enables you to move a table, partition, or subpartition to change any physical storage attribute, such as compression, or the tablespace, assuming you have the appropriate quota in the target tablespace.

`ALTER TABLE...MOVE` statements support the `ONLINE` keyword, which enables data manipulation language (DML) operations to run uninterrupted on the table, partition, or subpartition that is being moved. The following statements move a table, partition, or subpartition online:

- `ALTER TABLE ... MOVE ... ONLINE`
- `ALTER TABLE ... MOVE PARTITION ... ONLINE`
- `ALTER TABLE ... MOVE SUBPARTITION ... ONLINE`

Moving a table changes the rowids of the rows in the table. If you move a table and include the `ONLINE` keyword and the `UPDATE INDEXES` clause, then the indexes remain usable during the move operation. If you include the `UPDATE INDEXES` clause but not the `ONLINE` keyword, then the indexes are usable immediately after the move operation. The `UPDATE INDEXES` clause can only change the storage properties for the global indexes on the table or storage properties for the index partitions of any global partitioned index on the table. If you do not include the `UPDATE INDEXES` clause, then the changes to the rowids cause the indexes on the table to be marked `UNUSABLE`, and DML accessing the table using these indexes receive an ORA-01502 error. In this case, the indexes on the table must be dropped or rebuilt.

A move operation causes any statistics for the table to become invalid, and new statistics should be collected after moving the table.

If the table includes `LOB` column(s), then this statement can be used to move the table along with `LOB` data and `LOB` index segments (associated with this table) that are explicitly specified. If not specified, then the default is to not move the `LOB` data and `LOB` index segments.

19.7.3.2 Moving a Table

Use the `ALTER TABLE...MOVE` statement to move a table to a new segment or tablespace.

When you use the `ONLINE` keyword with this statement, data manipulation language (DML) operations can continue to run uninterrupted on the table that is being moved. If you do not include the `ONLINE` keyword, then concurrent DML operations are not possible on the data in the table during the move operation.

To move a table:

1. In SQL*Plus, connect as a user with the necessary privileges to alter the table.

   See *Oracle Database SQL Language Reference* for information about the privileges required to alter a table.

2. Run the `ALTER TABLE ... MOVE` statement.

Example 19-11 Moving a Table to a New Tablespace in Online Mode

The following statement moves the `hr.jobs` table online to a new segment and tablespace, specifying new storage parameters. The `ONLINE` keyword means that DML operations can run on the table uninterrupted during the move operation. The `hr_tbs` tablespace must exist.

```
ALTER TABLE hr.jobs MOVE ONLINE
      STORAGE ( INITIAL 20K
                NEXT 40K
                MINEXTENTS 2
                MAXEXTENTS 20
                PCTINCREASE 0 )
  TABLESPACE hr_tbs;
```

Example 19-12 Moving a Table and Updating the Table's Indexes

Assume the following statements created a table and its indexes:

```
CREATE TABLE dept_exp (
     DEPTNO NUMBER (2) NOT NULL,
     DNAME VARCHAR2 (14),
     LOC VARCHAR2 (13))
   TABLESPACE tbs_1;
```

```
CREATE INDEX i1_deptno ON dept_exp(deptno) TABLESPACE tbs_1;
CREATE INDEX i2_dname ON dept_exp(dname) TABLESPACE tbs_1;
```

The following statement moves the table to a new tablespace (`tbs_2`) and compresses the table. It also moves index `i2_dbname` to tablespace `tbs_2` and specifies that both the `i1_deptno` index and the `i2_dname` index are usable after the move operation.

```
ALTER TABLE dept_exp MOVE
   COMPRESS TABLESPACE tbs_2
   UPDATE INDEXES
      (i1_deptno TABLESPACE tbs_1,
       i2_dname TABLESPACE tbs_2);
```

Notice that this statement does not include the `ONLINE` keyword. However, the `ONLINE` keyword is supported if DML operations must be able to run on the table uninterrupted during the move operation, or if the indexes must be usable during the move operation.

Before running these statements, the `tbs_1` and `tbs_2` tablespaces must exist.

19.7.3.3 Moving a Table Partition or Subpartition Online

Use the `ALTER TABLE...MOVE PARTITION` statement or `ALTER TABLE...MOVE SUBPARTITION` statement to move a table partition or subpartition, respectively.

When you use the `ONLINE` keyword with either of these statements, DML operations can continue to run uninterrupted on the partition or subpartition that is being moved. If you do not include the `ONLINE` keyword, then DML operations are not permitted on the data in the partition or subpartition until the move operation is complete.

When you include the `UPDATE INDEXES` clause, these statements maintain both local and global indexes during the move. Therefore, using the `ONLINE` keyword with these statements eliminates the time it takes to regain partition performance after the move by maintaining global indexes and manually rebuilding indexes.

Some restrictions apply to moving table partitions and subpartitions. See *Oracle Database SQL Language Reference* for information about these restrictions.

To move a table partition or subpartition online:

1. In SQL*Plus, connect as a user with the necessary privileges to alter the table and move the partition or subpartition.

   See *Oracle Database SQL Language Reference* for information about the required privileges.

   See "Connecting to the Database with SQL*Plus".
2. Run the `ALTER TABLE ... MOVE PARTITION` or `ALTER TABLE ... MOVE SUBPARTITION` statement.

Example 19-13 Moving a Table Partition to a New Segment

The following statement moves the `sales_q4_2003` partition of the `sh.sales` table to a new segment with advanced row compression and index maintenance included:

```
ALTER TABLE sales MOVE PARTITION sales_q4_2003
  ROW STORE COMPRESS ADVANCED UPDATE INDEXES ONLINE;
```

See Also:

- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database SQL Language Reference*

19.7.4 Manually Allocating Storage for a Table

Oracle Database dynamically allocates additional extents for the data segment of a table, as required. However, perhaps you want to allocate an additional extent for a table explicitly. For example, in an Oracle Real Application Clusters environment, an extent of a table can be allocated explicitly for a specific instance.

You can allocate a new extent for a table using the `ALTER TABLE...ALLOCATE EXTENT` statement.

You can also explicitly deallocate unused space using the `DEALLOCATE UNUSED` clause of `ALTER TABLE`. This is described in "Reclaiming Unused Space".

19.7.5 Modifying an Existing Column Definition

Use the `ALTER TABLE...MODIFY` statement to modify an existing column definition. You can modify column data type, default value, column constraint, column expression (for virtual columns), column encryption, and visible/invisible property.

You can increase the length of an existing column, or decrease it, if all existing data satisfies the new length. Beginning with Oracle Database 12*c*, you can specify a maximum size of 32767 bytes for the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types. Before this release, the maximum size was 4000 bytes for the `VARCHAR2` and `NVARCHAR2` data types, and 2000 bytes for the `RAW` data type. To use extended data types, set the `MAX_STRING_SIZE` initialization parameter to `EXTENDED`.

You can change a column from byte semantics to `CHAR` semantics or vice versa. You must set the initialization parameter `BLANK_TRIMMING=TRUE` to decrease the length of a non-empty `CHAR` column.

If you are modifying a table to increase the length of a column of data type `CHAR`, then realize that this can be a time consuming operation and can require substantial additional storage, especially if the table contains many rows. This is because the `CHAR` value in each row must be blank-padded to satisfy the new column length.

If you modify the visible/invisible property of a column, then you cannot include any other column modification options in the same SQL statement.

Example 19-14 Changing the Length of a Column to a Size Larger Than 4000 Bytes

This example changes the length of the `product_description` column in the `oe.product_information` table to 32767 bytes.

```
ALTER TABLE oe.product_information MODIFY(product_description VARCHAR2(32767));
```

See Also:

- *Oracle Database SQL Language Reference* for additional information about modifying table columns and additional restrictions
- *Oracle Database Reference* for information about the `MAX_STRING_SIZE` initialization parameter

19.7.6 Adding Table Columns

To add a column to an existing table, use the `ALTER TABLE...ADD` statement.

The following statement alters the `hr.admin_emp` table to add a new column named `bonus`:

```
ALTER TABLE hr.admin_emp
      ADD (bonus NUMBER (7,2));
```

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

If a new column is added to a table, then the column is initially `NULL` unless you specify the `DEFAULT` clause. If you specify the `DEFAULT` clause for a nullable column for some table types, then the default value is stored as metadata, but the column itself is not populated with data. However, subsequent queries that specify the new column are rewritten so that the default value is returned in the result set. This behavior optimizes the resource usage and storage requirements for the operation.

You can add a column with a `NOT NULL` constraint only if the table does not contain any rows, or you specify a default value.

Note:

- If you enable basic table compression on a table, then you can add columns only if you do not specify default values.
- If you enable advanced row compression on a table, then you can add columns to that table with or without default values.
- If the new column is a virtual column, its value is determined by its column expression. (Note that a virtual column's value is calculated only when it is queried.)

See Also:

- *Oracle Database SQL Language Reference* for rules and restrictions for adding table columns
- "Consider Using Table Compression"
- *Oracle Database Concepts*
- "Example: Creating a Table" for an example of a virtual column

19.7.7 Renaming Table Columns

Oracle Database lets you rename existing columns in a table. Use the `RENAME COLUMN` clause of the `ALTER TABLE` statement to rename a column.

The new name must not conflict with the name of any existing column in the table. No other clauses are allowed with the `RENAME COLUMN` clause.

The following statement renames the `comm` column of the `hr.admin_emp` table.

```
ALTER TABLE hr.admin_emp
      RENAME COLUMN comm TO commission;
```

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

As noted earlier, altering a table column can invalidate dependent objects. However, when you rename a column, the database updates associated data dictionary tables to ensure that function-based indexes and check constraints remain valid.

Oracle Database also lets you rename column constraints. This is discussed in "Renaming Constraints".

Note:

The `RENAME TO` clause of `ALTER TABLE` appears similar in syntax to the `RENAME COLUMN` clause, but is used for renaming the table itself.

19.7.8 Dropping Table Columns

You can drop columns that are no longer needed from a table, including an index-organized table. This provides a convenient means to free space in a database, and avoids your having to export/import data then re-create indexes and constraints.

Note:

You cannot drop all columns from a table, nor can you drop columns from a table owned by `SYS`. Any attempt to do so results in an error.

- Removing Columns from Tables
  When you issue an `ALTER TABLE...DROP COLUMN` statement, the column descriptor and the data associated with the target column are removed from each row in the table. You can drop multiple columns with one statement.
- Marking Columns Unused
  If you are concerned about the length of time it could take to drop column data from all of the rows in a large table, you can use the `ALTER TABLE...SET UNUSED` statement.
- Removing Unused Columns
  The `ALTER TABLE...DROP UNUSED COLUMNS` statement is the only action allowed on unused columns. It physically removes unused columns from the table and reclaims disk space.
- Dropping Columns in Compressed Tables
  If you enable advanced row compression on a table, then you can drop table columns. If you enable basic table compression only, then you cannot drop columns.

See Also:

Oracle Database SQL Language Reference for information about additional restrictions and options for dropping columns from a table

19.7.8.1 Removing Columns from Tables

When you issue an `ALTER TABLE...DROP COLUMN` statement, the column descriptor and the data associated with the target column are removed from each row in the table. You can drop multiple columns with one statement.

The following statements are examples of dropping columns from the `hr.admin_emp` table. The first statement drops only the `sal` column:

```
ALTER TABLE hr.admin_emp DROP COLUMN sal;
```

The next statement drops both the `bonus` and `comm` columns:

```
ALTER TABLE hr.admin_emp DROP (bonus, commission);
```

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

19.7.8.2 Marking Columns Unused

If you are concerned about the length of time it could take to drop column data from all of the rows in a large table, you can use the `ALTER TABLE...SET UNUSED` statement.

This statement marks one or more columns as unused, but does not actually remove the target column data or restore the disk space occupied by these columns. However, a column that is marked as unused is not displayed in queries or data dictionary views, and its name is removed so that a new column can reuse that name. In most cases, constraints, indexes, and statistics defined on the column are also removed. The exception is that any internal indexes for LOB columns that are marked unused are not removed.

To mark the `hiredate` and `mgr` columns as unused, execute the following statement:

```
ALTER TABLE hr.admin_emp SET UNUSED (hiredate, mgr);
```

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

You can later remove columns that are marked as unused by issuing an `ALTER TABLE...DROP UNUSED COLUMNS` statement. Unused columns are also removed from the target table whenever an explicit drop of any particular column or columns of the table is issued.

The data dictionary views `USER_UNUSED_COL_TABS`, `ALL_UNUSED_COL_TABS`, or `DBA_UNUSED_COL_TABS` can be used to list all tables containing unused columns. The `COUNT` field shows the number of unused columns in the table.

```
SELECT * FROM DBA_UNUSED_COL_TABS;

OWNER                       TABLE_NAME                  COUNT
--------------------------- --------------------------- -----
HR                          ADMIN_EMP                       2
```

For external tables, the `SET UNUSED` statement is transparently converted into an `ALTER TABLE DROP COLUMN` statement. Because external tables consist of metadata only in the database, the `DROP COLUMN` statement performs equivalently to the `SET UNUSED` statement.

19.7.8.3 Removing Unused Columns

The `ALTER TABLE...DROP UNUSED COLUMNS` statement is the only action allowed on unused columns. It physically removes unused columns from the table and reclaims disk space.

In the `ALTER TABLE` statement that follows, the optional clause `CHECKPOINT` is specified. This clause causes a checkpoint to be applied after processing the specified number of rows, in this case 250. Checkpointing cuts down on the amount of undo logs accumulated during the drop column operation to avoid a potential exhaustion of undo space.

```
ALTER TABLE hr.admin_emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

19.7.8.4 Dropping Columns in Compressed Tables

If you enable advanced row compression on a table, then you can drop table columns. If you enable basic table compression only, then you cannot drop columns.

See Also:

"Consider Using Table Compression"

19.7.9 Placing a Table in Read-Only Mode

You can place a table in read-only mode with the `ALTER TABLE`...`READ ONLY` statement, and return it to read/write mode with the `ALTER TABLE`...`READ WRITE` statement.

An example of a table for which read-only mode makes sense is a configuration table. If your application contains configuration tables that are not modified after installation and that must not be modified by users, your application installation scripts can place these tables in read-only mode.

To place a table in read-only mode, you must have the `ALTER TABLE` privilege on the table or the `ALTER ANY TABLE` privilege. In addition, the `COMPATIBLE` initialization parameter must be set to `11.2.0` or higher.

The following example places the `SALES` table in read-only mode:

```
ALTER TABLE SALES READ ONLY;
```

The following example returns the table to read/write mode:

```
ALTER TABLE SALES READ WRITE;
```

When a table is in read-only mode, operations that attempt to modify table data are disallowed. A `SELECT` *`column_list`* `ON` *`table_name`* statement on a table must always return the same data set after a table or partition has been placed in read-only mode.

The following operations are not permitted on a read-only table:

- All DML operations on the read-only table or on a read-only partition
- `TRUNCATE TABLE`
- `SELECT FOR UPDATE`
- `ALTER TABLE RENAME`/`DROP COLUMN`
- `DROP` of a read-only partition or a partition of a read only table
- `ALTER TABLE SET COLUMN UNUSED`

- `ALTER TABLE DROP`/`TRUNCATE`/`EXCHANGE (SUB)PARTITION`
- `ALTER TABLE UPGRADE INCLUDING DATA` or `ALTER TYPE CASCADE INCLUDING TABLE DATA` for a type with read-only table dependents
- Online redefinition
- `FLASHBACK TABLE`

The following operations are permitted on a read-only table:

- `SELECT`
- `CREATE`/`ALTER`/`DROP INDEX`
- `ALTER TABLE ADD`/`MODIFY COLUMN`
- `ALTER TABLE ADD`/`MODIFY`/`DROP`/`ENABLE`/`DISABLE CONSTRAINT`
- `ALTER TABLE` for physical property changes
- `ALTER TABLE DROP UNUSED COLUMNS`
- `ALTER TABLE ADD`/`COALESCE`/`MERGE`/`MODIFY`/`MOVE`/`RENAME`/`SPLIT (SUB)PARTITION`
- `ALTER TABLE MOVE`
- `ALTER TABLE ENABLE ROW MOVEMENT` and `ALTER TABLE SHRINK`
- `RENAME TABLE` and `ALTER TABLE RENAME TO`
- `DROP TABLE`
- `ALTER TABLE DEALLOCATE UNUSED`
- `ALTER TABLE ADD`/`DROP SUPPLEMENTAL LOG`

See Also:

- *Oracle Database SQL Language Reference* for more information about the `ALTER TABLE` statement
- *Oracle Database VLDB and Partitioning Guide* for more information about read-only partitions

19.8 Redefining Tables Online

You can modify the logical or physical structure of a table.

- About Redefining Tables Online
  In any database system, it is occasionally necessary to modify the logical or physical structure of a table to improve the performance of queries or DML, accommodate application changes, or Manage storage. You can redefine tables online with the `DBMS_REDEFINITION` package.
- Features of Online Table Redefinition
  Online table redefinition enables you to modify a table in several different ways while the table remains online.

- Privileges Required for the DBMS_REDEFINITION Package
  Execute privileges on the `DBMS_REDEFINITION` package are required to run subprograms in the package. Execute privileges on the `DBMS_REDEFINITION` package are granted to `EXECUTE_CATALOG_ROLE`.
- Restrictions for Online Redefinition of Tables
  Several restrictions apply to online redefinition of tables.
- Performing Online Redefinition with the REDEF_TABLE Procedure
  You can use the `REDEF_TABLE` procedure in the `DBMS_REDEFINITION` package to perform online redefinition of a table's storage properties.
- Redefining Tables Online with Multiple Procedures in DBMS_REDEFINITION
  You can use multiple procedures in the `DBMS_REDEFINITION` package to redefine tables online.
- Results of the Redefinition Process
  There are several results of the redefinition process.
- Performing Intermediate Synchronization
  During the redefinition process, you can synchronize the interim table with the original table if there were a large number of DML statements executed on the original table.
- Refreshing Dependent Materialized Views During Online Table Redefinition
  To refresh dependent fast refreshable materialized views during online table redefinition, set the `refresh_dep_mviews` parameter to `Y` in the `REDEF_TABLE` procedure or the `START_REDEF_TABLE` procedure.
- Monitoring Online Table Redefinition Progress
  You can query the `V$ONLINE_REDEF` view to monitor the progress of an online table redefinition operation.
- Restarting Online Table Redefinition After a Failure
  If online table redefinition fails, then you can check the `DBA_REDEFINITION_STATUS` view to see the error information and restartable information.
- Rolling Back Online Table Redefinition
  You can enable roll back of a table after online table redefinition to return the table to its original definition and preserve DML changes made to the table.
- Terminating Online Table Redefinition and Cleaning Up After Errors
  You can terminate the online redefinition process. Doing so drops temporary logs and tables associated with the redefinition process. After this procedure is called, you can drop the interim table and its dependent objects.
- Online Redefinition of One or More Partitions
  You can redefine online one or more partitions of a table. This is useful if, for example, you want to move partitions to a different tablespace and keep the partitions available for DML during the operation.
- Online Table Redefinition Examples
  Examples illustrate online redefinition of tables.

19.8.1 About Redefining Tables Online

In any database system, it is occasionally necessary to modify the logical or physical structure of a table to improve the performance of queries or DML, accommodate application changes, or Manage storage. You can redefine tables online with the `DBMS_REDEFINITION` package.

Oracle Database provides a mechanism to make table structure modifications without significantly affecting the availability of the table. The mechanism is called **online table**

redefinition. Redefining tables online provides a substantial increase in availability compared to traditional methods of redefining tables.

When a table is redefined online, it is accessible to both queries and DML during much of the redefinition process. Typically, the table is locked in the exclusive mode only during a very small window that is independent of the size of the table and complexity of the redefinition, and that is completely transparent to users. However, if there are many concurrent DML operations during redefinition, then a longer wait might be necessary before the table can be locked.

Online table redefinition requires an amount of free space that is approximately equivalent to the space used by the table being redefined. More space may be required if new columns are added.

You can perform online table redefinition with the Oracle Enterprise Manager Cloud Control (Cloud Control) Reorganize Objects wizard or with the `DBMS_REDEFINITION` package.

Note:

To invoke the Reorganize Objects wizard:

1. On the Tables page of Cloud Control, click in the **Select** column to select the table to redefine.
2. In the Actions list, select **Reorganize**.
3. Click **Go**.

See Also:

Oracle Database PL/SQL Packages and Types Reference for a description of the `DBMS_REDEFINITION` package

19.8.2 Features of Online Table Redefinition

Online table redefinition enables you to modify a table in several different ways while the table remains online.

Online table redefinition enables you to:

- Modify the storage parameters of a table or cluster
- Move a table or cluster to a different tablespace

  Note:

  If it is not important to keep a table available for DML when moving it to another tablespace, then you can use the simpler `ALTER TABLE MOVE` command. See "Moving a Table to a New Segment or Tablespace".

- Add, modify, or drop one or more columns in a table or cluster

- Add or drop partitioning support (non-clustered tables only)
- Change partition structure
- Change physical properties of a single table partition or subpartition, including moving it to a different tablespace in the same schema

  Starting with Oracle Database 12*c*, you can move a partition or subpartition online without using online table redefinition. DML operations can continue to run uninterrupted on the partition or subpartition that is being moved. See "Moving a Table to a New Segment or Tablespace".
- Change physical properties of a materialized view log or an Oracle Database Advanced Queuing queue table

  Note:

  The `REDEF_TABLE` procedure in the `DBMS_REDEFINITION` package does not support changing physical properties of an Oracle Database Advanced Queuing queue table.
- Add support for parallel queries
- Re-create a table or cluster to reduce fragmentation

  Note:

  In many cases, online segment shrink is an easier way to reduce fragmentation. See "Reclaiming Unused Space".
- Change the organization of a normal table (heap organized) to an index-organized table, or do the reverse.
- Convert a relational table into a table with object columns, or do the reverse.
- Convert an object table into a relational table or a table with object columns, or do the reverse.
- Compress, or change the compression type for, a table, partition, index key, or LOB columns.
- Convert LOB columns from BasicFiles LOB storage to SecureFiles LOB storage, or do the reverse.
- You can enable roll back of a table after online table redefinition to return the table to its original definition and preserve DML changes made to the table.
- You can refresh dependent fast refreshable materialized views during online table redefinition by setting the `refresh_dep_mviews` parameter to `Y` in the `REDEF_TABLE` procedure or the `START_REDEF_TABLE` procedure.
- You can query the `V$ONLINE_REDEF` view to monitor the progress of an online table redefinition operation.
- When online table redefinition fails, often you can correct the problem that caused the failure and restart the online redefinition process where it last stopped.

You can combine two or more of the usage examples above into one operation. See "Example 8" in "Online Table Redefinition Examples" for an example.

19.8.3 Privileges Required for the DBMS_REDEFINITION Package

Execute privileges on the `DBMS_REDEFINITION` package are required to run subprograms in the package. Execute privileges on the `DBMS_REDEFINITION` package are granted to `EXECUTE_CATALOG_ROLE`.

In addition, for a user to redefine a table in the user's schema using the package, the user must be granted the following privileges:

- `CREATE TABLE`
- `CREATE MATERIALIZED VIEW`

The `CREATE TRIGGER` privilege is also required to execute the `COPY_TABLE_DEPENDENTS` procedure.

For a user to redefine a table in other schemas using the package, the user must be granted the following privileges:

- `CREATE ANY TABLE`
- `ALTER ANY TABLE`
- `DROP ANY TABLE`
- `LOCK ANY TABLE`
- `SELECT ANY TABLE`

The following additional privileges are required to execute `COPY_TABLE_DEPENDENTS` on tables in other schemas:

- `CREATE ANY TRIGGER`
- `CREATE ANY INDEX`

19.8.4 Restrictions for Online Redefinition of Tables

Several restrictions apply to online redefinition of tables.

The following restrictions apply to the online redefinition of tables:

- If the table is to be redefined using primary key or pseudo-primary keys (unique keys or constraints with all component columns having *not null* constraints), then the post-redefinition table must have the same primary key or pseudo-primary key columns. If the table is to be redefined using rowids, then the table must not be an index-organized table.
- After redefining a table that has a materialized view log, the subsequent refresh of any dependent materialized view must be a complete refresh.

  There is an exception to this restriction. When online table redefinition uses the `REDEF_TABLE` or `START_REDEF_TABLE` procedure, and the `refresh_dep_mviews` parameter is set to `Y` in the procedure, any dependent materialized views configured for incremental refresh are refreshed during the online table redefinition operation.
- Tables that are replicated in an n-way master configuration can be redefined, but horizontal subsetting (subset of rows in the table), vertical subsetting (subset of columns in the table), and column transformations are not allowed.
- The overflow table of an index-organized table cannot be redefined online independently.

- Tables for which Flashback Data Archive is enabled cannot be redefined online. You cannot enable Flashback Data Archive for the interim table.
- Tables with `LONG` columns can be redefined online, but those columns must be converted to `CLOBS`. Also, `LONG RAW` columns must be converted to `BLOBS`. Tables with `LOB` columns are acceptable.
- On a system with sufficient resources for parallel execution, and in the case where the interim table is not partitioned, redefinition of a `LONG` column to a `LOB` column can be executed in parallel, provided that:
  - The segment used to store the `LOB` column in the interim table belongs to a locally managed tablespace with Automatic Segment Space Management (ASSM) enabled.
  - There is a simple mapping from one `LONG` column to one `LOB` column, and the interim table has only one `LOB` column.

  In the case where the interim table is partitioned, the normal methods for parallel execution for partitioning apply.
- Tables in the `SYS` and `SYSTEM` schema cannot be redefined online.
- Temporary tables cannot be redefined.
- A subset of rows in the table cannot be redefined.
- Only simple deterministic expressions, sequences, and `SYSDATE` can be used when mapping the columns in the interim table to those of the original table. For example, subqueries are not allowed.
- If new columns are being added as part of the redefinition and there are no column mappings for these columns, then they must not be declared `NOT NULL` until the redefinition is complete.
- There cannot be any referential constraints between the table being redefined and the interim table.
- Table redefinition cannot be done `NOLOGGING`.
- For materialized view logs and queue tables, online redefinition is restricted to changes in physical properties. No horizontal or vertical subsetting is permitted, nor are any column transformations. The only valid value for the column mapping string is `NULL`.
- You cannot perform online redefinition on a partition that includes one or more nested tables.
- You can convert a `VARRAY` to a nested table with the `CAST` operator in the column mapping. However, you cannot convert a nested table to a `VARRAY`.
- When the columns in the `col_mapping` parameter of the `DBMS_REDEFINITION.START_REDEF_TABLE` procedure include a sequence, the `orderby_cols` parameter must be `NULL`.
- For tables with a Virtual Private Database (VPD) security policy, when the `copy_vpd_opt` parameter is specified as `DBMS_REDEFINITION.CONS_VPD_AUTO`, the following restrictions apply:
  - The column mapping string between the original table and interim table must be `NULL` or `'*'`.
  - No VPD policies can exist on the interim table.

  See "Handling Virtual Private Database (VPD) Policies During Online Redefinition". Also, see *Oracle Database Security Guide* for information about VPD policies.

- Online redefinition cannot run on multiple tables concurrently in separate `DBMS_REDEFINITION` sessions if the tables are related by reference partitioning.

  See *Oracle Database VLDB and Partitioning Guide* for more information about reference partitioning.
- Online redefinition of an object table or `XMLType` table can cause a dangling `REF` in other tables if those other tables have a `REF` column that references the redefined table.

  See *Oracle Database SQL Language Reference* for more information about dangling `REF`s.
- Tables that use Oracle Label Security (OLS) cannot be redefined online.

  See *Oracle Label Security Administrator's Guide*.
- Tables with fine-grained access control cannot be redefined online.
- Tables that use Oracle Real Application Security cannot be redefined online.

  See *Oracle Database Security Guide*.

19.8.5 Performing Online Redefinition with the REDEF_TABLE Procedure

You can use the `REDEF_TABLE` procedure in the `DBMS_REDEFINITION` package to perform online redefinition of a table's storage properties.

The `REDEF_TABLE` procedure enables you to perform online redefinition a table's storage properties in a single step when you want to change the following properties:

- Tablespace changes, including a tablespace change for a table, partition, index, or LOB columns
- Compression type changes, including a compression type change for a table, partition, index key, or LOB columns
- For LOB columns, a change to `SECUREFILE` or `BASICFILE` storage

When your online redefinition operation is not limited to these changes, you must perform online redefinition of the table using multiple steps. The steps include invoking multiple procedures in the `DBMS_REDEFINITION` package, including the following procedures: `CAN_REDEF_TABLE`, `START_REDEF_TABLE`, `COPY_TABLE_DEPENDENTS`, and `FINISH_REDEF_TABLE`.

Note:

Online table redefinition rollback is not supported when the `REDEF_TABLE` procedure is used to redefine a table.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for procedure details
- Example 1 in "Online Table Redefinition Examples"
- "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information

19.8.6 Redefining Tables Online with Multiple Procedures in DBMS_REDEFINITION

You can use multiple procedures in the `DBMS_REDEFINITION` package to redefine tables online.

- Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION
  You can use multiple procedures in the `DBMS_REDEFINITION` package to perform online redefinition of a table.
- Constructing a Column Mapping String
  The column mapping string that you pass as an argument to `START_REDEF_TABLE` contains a comma-delimited list of column mapping pairs.
- Handling Virtual Private Database (VPD) Policies During Online Redefinition
  If the original table being redefined has VPD policies specified for it, then you can use the `copy_vpd_opt` parameter in the `START_REDEF_TABLE` procedure to handle these policies during online redefinition.
- Creating Dependent Objects Automatically
  You use the `COPY_TABLE_DEPENDENTS` procedure to automatically create dependent objects on the interim table.
- Creating Dependent Objects Manually
  If you manually create dependent objects on the interim table with SQL*Plus or Cloud Control, then you must use the `REGISTER_DEPENDENT_OBJECT` procedure to register the dependent objects. Registering dependent objects enables the redefinition completion process to restore dependent object names to what they were before redefinition.

19.8.6.1 Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION

You can use multiple procedures in the `DBMS_REDEFINITION` package to perform online redefinition of a table.

To redefine a table online using multiple steps:

1. Choose the redefinition method: by key or by rowid

   By key—Select a primary key or pseudo-primary key to use for the redefinition. Pseudo-primary keys are unique keys with all component columns having `NOT NULL` constraints. For this method, the versions of the tables before and after redefinition should have the same primary key columns. This is the preferred and default method of redefinition.

   By rowid—Use this method if no key is available. In this method, a hidden column named `M_ROW$$` is added to the post-redefined version of the table. It is recommended that this column be dropped or marked as unused after the redefinition is complete. The final phase of redefinition automatically sets this column unused. You can then use the `ALTER TABLE` ... `DROP UNUSED COLUMNS` statement to drop it.

   You cannot use this method on index-organized tables.

2. Verify that the table can be redefined online by invoking the `CAN_REDEF_TABLE` procedure. If the table is not a candidate for online redefinition, then this procedure raises an error indicating why the table cannot be redefined online.

3. Create an empty interim table (in the same schema as the table to be redefined) with all of the desired logical and physical attributes. If columns are to be dropped, then do not include them in the definition of the interim table. If a column is to be added, then add the

column definition to the interim table. If a column is to be modified, then create it in the interim table with the properties that you want.

It is not necessary to create the interim table with all the indexes, constraints, grants, and triggers of the table being redefined, because these will be defined in step 7 when you copy dependent objects.

4. If you are redefining a partitioned table with the rowid method, then enable row movement on the interim table.

   ```
   ALTER TABLE ... ENABLE ROW MOVEMENT;
   ```

5. (Optional) If you are redefining a large table and want to improve the performance of the next step by running it in parallel, issue the following statements:

   ```
   ALTER SESSION FORCE PARALLEL DML PARALLEL degree-of-parallelism;
   ALTER SESSION FORCE PARALLEL QUERY PARALLEL degree-of-parallelism;
   ```

6. Start the redefinition process by calling `START_REDEF_TABLE`, providing the following:

   - The schema and table name of the table to be redefined in the `uname` and `orig_table` parameters, respectively
   - The interim table name in the `int_table` parameter
   - A column mapping string that maps the columns of table to be redefined to the columns of the interim table in the `col_mapping` parameter

     See "Constructing a Column Mapping String" for details.
   - The redefinition method in the `options_flag` parameter

     Package constants are provided for specifying the redefinition method. `DBMS_REDEFINITION.CONS_USE_PK` is used to indicate that the redefinition should be done using primary keys or pseudo-primary keys. `DBMS_REDEFINITION.CONS_USE_ROWID` is use to indicate that the redefinition should be done using rowids. If this argument is omitted, the default method of redefinition (`CONS_USE_PK`) is assumed.
   - Optionally, the columns to be used in ordering rows in the `orderby_cols` parameter
   - The partition name or names in the `part_name` parameter when redefining one partition or multiple partitions of a partitioned table

     See "Online Redefinition of One or More Partitions" for details.
   - The method for handling Virtual Private Database (VPD) policies defined on the table in the `copy_vpd_opt` parameter

     See "Handling Virtual Private Database (VPD) Policies During Online Redefinition" for details.

   Because this process involves copying data, it may take a while. The table being redefined remains available for queries and DML during the entire process.

   Note:

   - You can query the `DBA_REDEFINITION_OBJECTS` view to list the objects currently involved in online redefinition.
   - If `START_REDEF_TABLE` fails for any reason, you must call `ABORT_REDEF_TABLE`, otherwise subsequent attempts to redefine the table will fail.

7. Copy dependent objects (such as triggers, indexes, materialized view logs, grants, and constraints) and statistics from the table being redefined to the interim table, using one of the following two methods. Method 1 is the preferred method because it is more automatic, but there may be times that you would choose to use method 2. Method 1 also enables you to copy table statistics to the interim table.
   - Method 1: Automatically Creating Dependent Objects

     Use the `COPY_TABLE_DEPENDENTS` procedure to automatically create dependent objects on the interim table. This procedure also **registers** the dependent objects. Registering the dependent objects enables the identities of these objects and their copied counterparts to be automatically swapped later as part of the redefinition completion process. The result is that when the redefinition is completed, the names of the dependent objects will be the same as the names of the original dependent objects.

     For more information, see "Creating Dependent Objects Automatically".
   - Method 2: Manually Creating Dependent Objects

     You can manually create dependent objects on the interim table and then register them. For more information, see "Creating Dependent Objects Manually".

   Note:

   In Oracle9*i*, you were *required* to manually create the triggers, indexes, grants, and constraints on the interim table, and there may still be situations where you want to or must do so. In such cases, any referential constraints involving the interim table (that is, the interim table is either a parent or a child table of the referential constraint) must be created disabled. When online redefinition completes, the referential constraint is automatically enabled. In addition, until the redefinition process is either completed or terminated, any trigger defined on the interim table does not execute.

8. Execute the `FINISH_REDEF_TABLE` procedure to complete the redefinition of the table. During this procedure, the original table is locked in exclusive mode for a very short time, independent of the amount of data in the original table. However, `FINISH_REDEF_TABLE` will wait for all pending DML to commit before completing the redefinition.

   You can use the `dml_lock_timeout` parameter in the `FINISH_REDEF_TABLE` procedure to specify how long the procedure waits for pending DML to commit. The parameter specifies the number of seconds to wait before the procedure ends gracefully. When you specify a non-`NULL` value for this parameter, you can restart the `FINISH_REDEF_TABLE` procedure, and it continues from the point at which it timed out. When the parameter is set to `NULL`, the procedure does not time out. In this case, if you stop the procedure manually, then you must terminate the online table redefinition using the `ABORT_REDEF_TABLE` procedure and start over from step 6.

9. Wait for any long-running queries against the interim (former) table to complete, and then drop the interim table.

   If you drop the interim table while there are active queries running against it, you may encounter error `ORA-08103 object no longer exists`.

See Also:

- "Online Table Redefinition Examples"
- *Oracle Database PL/SQL Packages and Types Reference* for package details

19.8.6.2 Constructing a Column Mapping String

The column mapping string that you pass as an argument to `START_REDEF_TABLE` contains a comma-delimited list of column mapping pairs.

Each pair has the following syntax:

```
[expression]  column_name
```

The *`column_name`* term indicates a column in the interim table. The optional *`expression`* can include columns from the table being redefined, constants, operators, function or method calls, and so on, in accordance with the rules for expressions in a SQL `SELECT` statement. However, only simple deterministic subexpressions—that is, subexpressions whose results do not vary between one evaluation and the next—plus sequences and `SYSDATE` can be used. No subqueries are permitted. In the simplest case, the expression consists of just a column name from the table being redefined.

If an expression is present, its value is placed in the designated interim table column during redefinition. If the expression is omitted, it is assumed that both the table being redefined and the interim table have a column named *`column_name`*, and the value of that column in the table being redefined is placed in the same column in the interim table.

For example, if the `override` column in the table being redefined is to be renamed to `override_commission`, and every override commission is to be raised by 2%, the correct column mapping pair is:

```
override*1.02  override_commission
```

If you supply '`*`' or `NULL` as the column mapping string, it is assumed that all the columns (with their names unchanged) are to be included in the interim table. Otherwise, only those columns specified explicitly in the string are considered. The order of the column mapping pairs is unimportant.

For examples of column mapping strings, see "Online Table Redefinition Examples".

Data Conversions

When mapping columns, you can convert data types, with some restrictions.

If you provide '`*`' or `NULL` as the column mapping string, only the implicit conversions permitted by SQL are supported. For example, you can convert from `CHAR` to `VARCHAR2`, from `INTEGER` to `NUMBER`, and so on.

To perform other data type conversions, including converting from one object type to another or one collection type to another, you must provide a column mapping pair with an expression that performs the conversion. The expression can include the `CAST` function, built-in functions like `TO_NUMBER`, conversion functions that you create, and so on.

19.8.6.3 Handling Virtual Private Database (VPD) Policies During Online Redefinition

If the original table being redefined has VPD policies specified for it, then you can use the `copy_vpd_opt` parameter in the `START_REDEF_TABLE` procedure to handle these policies during online redefinition.

You can specify the following values for this parameter:

Parameter Value	Description
`DBMS_REDEFINITION.CONS_VPD_NONE`	Specify this value if there are no VPD policies on the original table. This value is the default. If this value is specified, and VPD policies exist for the original table, then an error is raised.
`DBMS_REDEFINITION.CONS_VPD_AUTO`	Specify this value to copy the VPD policies automatically from the original table to the new table during online redefinition.
`DBMS_REDEFINITION.CONS_VPD_MANUAL`	Specify this value to copy the VPD policies manually from the original table to the new table during online redefinition.

If there are no VPD policies specified for the original table, then specify the default value of `DBMS_REDEFINITION.CONS_VPD_NONE` for the `copy_vpd_opt` parameter.

Specify `DBMS_REDEFINITION.CONS_VPD_AUTO` for the `copy_vpd_opt` parameter when the column names and column types are the same for the original table and the interim table. To use this value, the column mapping string between original table and interim table must be `NULL` or `'*'`. When you use `DBMS_REDEFINITION.CONS_VPD_AUTO` for the `copy_vpd_opt` parameter, only the table owner and the user invoking online redefinition can access the interim table during online redefinition.

Specify `DBMS_REDEFINITION.CONS_VPD_MANUAL` for the `copy_vpd_opt` parameter when either of the following conditions are true:

- There are VPD policies specified for the original table, and there are column mappings between the original table and the interim table.
- You want to add or modify VPD policies during online redefinition of the table.

To copy the VPD policies manually, you specify the VPD policies for the interim table before you run the `START_REDEF_TABLE` procedure. When online redefinition of the table is complete, the redefined table has the modified policies.

See Also:

- "Restrictions for Online Redefinition of Tables" for restrictions related to tables with VPD policies
- "Online Table Redefinition Examples" for an example that redefines a table with VPD policies
- *Oracle Database Security Guide*

19.8.6.4 Creating Dependent Objects Automatically

You use the COPY_TABLE_DEPENDENTS procedure to automatically create dependent objects on the interim table.

You can discover if errors occurred while copying dependent objects by checking the num_errors output argument. If the ignore_errors argument is set to TRUE, the COPY_TABLE_DEPENDENTS procedure continues copying dependent objects even if an error is encountered when creating an object. You can view these errors by querying the DBA_REDEFINITION_ERRORS view.

Reasons for errors include:

- A lack of system resources
- A change in the logical structure of the table that would require recoding the dependent object.

  See Example 3 in "Online Table Redefinition Examples" for a discussion of this type of error.

If ignore_errors is set to FALSE, the COPY_TABLE_DEPENDENTS procedure stops copying objects as soon as any error is encountered.

After you correct any errors you can again attempt to copy the dependent objects by reexecuting the COPY_TABLE_DEPENDENTS procedure. Optionally you can create the objects manually and then register them as explained in "Creating Dependent Objects Manually". The COPY_TABLE_DEPENDENTS procedure can be used multiple times as necessary. If an object has already been successfully copied, it is not copied again.

19.8.6.5 Creating Dependent Objects Manually

If you manually create dependent objects on the interim table with SQL*Plus or Cloud Control, then you must use the REGISTER_DEPENDENT_OBJECT procedure to register the dependent objects. Registering dependent objects enables the redefinition completion process to restore dependent object names to what they were before redefinition.

The following are examples changes that require you to create dependent objects manually:

- Moving an index to another tablespace
- Modifying the columns of an index
- Modifying a constraint
- Modifying a trigger
- Modifying a materialized view log

When you run the REGISTER_DEPENDENT_OBJECT procedure, you must specify that type of the dependent object with the dep_type parameter. You can specify the following constants in this parameter:

- DEMS_REDEFINITION.CONS_INDEX when the dependent object is an index
- DEMS_REDEFINITION.CONS_CONSTRAINT when the dependent object type is a constraint
- DEMS_REDEFINITION.CONS_TRIGGER when the dependent object is a trigger
- DEMS_REDEFINITION.CONS_MVLOG when the dependent object is a materialized view log

You would also use the `REGISTER_DEPENDENT_OBJECT` procedure if the `COPY_TABLE_DEPENDENTS` procedure failed to copy a dependent object and manual intervention is required.

You can query the `DBA_REDEFINITION_OBJECTS` view to determine which dependent objects are registered. This view shows dependent objects that were registered explicitly with the `REGISTER_DEPENDENT_OBJECT` procedure or implicitly with the `COPY_TABLE_DEPENDENTS` procedure. Only current information is shown in the view.

The `UNREGISTER_DEPENDENT_OBJECT` procedure can be used to unregister a dependent object on the table being redefined and on the interim table.

Note:

- Manually created dependent objects do not have to be identical to their corresponding original dependent objects. For example, when manually creating a materialized view log on the interim table, you can log different columns. In addition, the interim table can have more or fewer dependent objects.
- If the table being redefined includes named LOB segments, then the LOB segment names are replaced by system-generated names during online redefinition. To avoid this, you can create the interim table with new LOB segment names.

See Also:

Example 4 in "Online Table Redefinition Examples" for an example that registers a dependent object

19.8.7 Results of the Redefinition Process

There are several results of the redefinition process.

The following are the end results of the redefinition process:

- The original table is redefined with the columns, indexes, constraints, grants, triggers, and statistics of the interim table, assuming that either `REDEF_TABLE` or `COPY_TABLE_DEPENDENTS` was used.
- Dependent objects that were registered, either explicitly using `REGISTER_DEPENDENT_OBJECT` or implicitly using `COPY_TABLE_DEPENDENTS`, are renamed automatically so that dependent object names on the redefined table are the same as before redefinition.

  Note:

  If no registration is done or no automatic copying is done, then you must manually rename the dependent objects.

- The referential constraints involving the interim table now involve the redefined table and are enabled.
- Any indexes, triggers, materialized view logs, grants, and constraints defined on the original table (before redefinition) are transferred to the interim table and are dropped when the user drops the interim table. Any referential constraints involving the original table before the redefinition now involve the interim table and are disabled.
- Some PL/SQL objects, views, synonyms, and other table-dependent objects may become invalidated. Only those objects that depend on elements of the table that were changed are invalidated. For example, if a PL/SQL procedure queries only columns of the redefined table that were unchanged by the redefinition, the procedure remains valid. See "Managing Object Dependencies" for more information about schema object dependencies.

19.8.8 Performing Intermediate Synchronization

During the redefinition process, you can synchronize the interim table with the original table if there were a large number of DML statements executed on the original table.

After the redefinition process has been started by calling `START_REDEF_TABLE` and before `FINISH_REDEF_TABLE` has been called, a large number of DML statements might have been executed on the original table. If you know that this is the case, then it is recommended that you periodically synchronize the interim table with the original table.

When you start an online table redefinition operation with the `START_REDEF_TABLE` procedure, it creates an internal materialized view to facilitate synchronization. This internal materialized view is refreshed to synchronize the interim table with the original table.

To synchronize the interim table with the original table:

- Run the `SYNC_INTERIM_TABLE` procedure in the `DBMS_REDEFINITION` package.

Calling this procedure reduces the time taken by `FINISH_REDEF_TABLE` to complete the redefinition process. There is no limit to the number of times that you can call `SYNC_INTERIM_TABLE`.

The small amount of time that the original table is locked during `FINISH_REDEF_TABLE` is independent of whether `SYNC_INTERIM_TABLE` has been called.

19.8.9 Refreshing Dependent Materialized Views During Online Table Redefinition

To refresh dependent fast refreshable materialized views during online table redefinition, set the `refresh_dep_mviews` parameter to `Y` in the `REDEF_TABLE` procedure or the `START_REDEF_TABLE` procedure.

A dependent materialized view is any materialized view that is defined on the table being redefined. Performing a complete refresh of dependent materialized views after online table redefinition can be time consuming. You can incrementally refresh fast refreshable materialized views during online table redefinition to make the operation more efficient.

The following restrictions apply to refreshing a dependent materialized view:

- The materialized view must be fast refreshable.
- `ROWID` materialized views are not supported.
- Materialized join views are not supported.

A complete refresh of dependent ROWID materialized views and materialized join views is required after online table redefinition.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".
2. Perform an online redefinition of a table using one of the following methods:
   - Running the REDEF_TABLE procedure and ensuring that the refresh_dep_mviews parameter is set to Y.
     With this method, fast refresh of dependent materialized views is performed once at the end of the redefinition operation.
   - Starting online table redefinition with the START_REDEF_TABLE procedure and ensuring that the refresh_dep_mviews parameter is set to Y. This method ends with the FINISH_REDEF_TABLE procedure.
     With this method, fast refresh of dependent materialized views is performed when the START_REDEF_TABLE procedure is run, each time the SYNC_INTERIM_TABLE procedure is run, and when the FINISH_REDEF_TABLE procedure is run.

Note:

– You can check the value of the refresh_dep_mviews parameter for an online table redefinition operation by querying the DBA_REDEFINITION_STATUS view.

– You can check on the progress of a refresh that is run automatically during online table redefinition by querying the REFRESH_STATEMENT_SQL_ID and REFRESH_STATEMENT columns in the V$ONLINE_REDEF view. You can use the SQL_ID value returned in the REFRESH_STATEMENT_SQL_ID column to monitor the progress of a refresh in views such as the V$SQL view and the V$SQL_MONITOR view.

– If you want to change the value of the refresh_dep_mviews parameter during an online table redefinition operation, then you can use the DBMS_REDEFINITION.SET_PARAM procedure to reset the parameter.

Example 19-15 Refreshing Dependent Materialized Views While Running the REDEF_TABLE Procedure

```
hr.employees

BEGIN
  DBMS_REDEFINITION.REDEF_TABLE(
    uname                  => 'HR',
    tname                  => 'EMPLOYEES',
    table_compression_type => 'ROW STORE COMPRESS ADVANCED',
    refresh_dep_mviews     => 'Y');
END;
/
```

Example 19-16 Refreshing Dependent Materialized Views While Starting with the START_REDEF_TABLE Procedure

Assume that you want to redefine the `oe.orders` table. The table definition is:

```
CREATE TABLE oe.orders(
     order_id      NUMBER(12),
     order_date    TIMESTAMP WITH LOCAL TIME ZONE,
     order_mode    VARCHAR2(8),
     customer_id   NUMBER(6),
     order_status  NUMBER(2),
     order_total   NUMBER(8,2),
     sales_rep_id  NUMBER(6),
     promotion_id  NUMBER(6));
```

This example redefines the table to increase the size of the `order_mode` column to `16`. The interim table definition is:

```
CREATE TABLE oe.int_orders(
     order_id      NUMBER(12),
     order_date    TIMESTAMP WITH LOCAL TIME ZONE,
     order_mode    VARCHAR2(16),
     customer_id   NUMBER(6),
     order_status  NUMBER(2),
     order_total   NUMBER(8,2),
     sales_rep_id  NUMBER(6),
     promotion_id  NUMBER(6));
```

Also assume that this table has dependent materialized views. The table has a materialized view log created with the following statement:

```
CREATE MATERIALIZED VIEW LOG ON oe.orders WITH PRIMARY KEY, ROWID;
```

The `oe.orders` table has the following dependent materialized views:

```
CREATE MATERIALIZED VIEW oe.orders_pk REFRESH FAST AS
 SELECT * FROM oe.orders;

CREATE MATERIALIZED VIEW oe.orders_rowid REFRESH FAST WITH ROWID AS
 SELECT * FROM oe.orders;
```

The `oe.orders_pk` materialized view is a fast refreshable, primary key materialized view. Therefore, it can be refreshed during online table redefinition.

The `oe.orders_rowid` materialized view is fast refreshable, but it is a `ROWID` materialized view. Therefore, it cannot be refreshed during online table redefinition.

Complete the following steps to perform online table redefinition on the `oe.orders` table while refreshing the `oe.orders_pk` materialized view:

1. Start the redefinition process.

   ```
   BEGIN
     DBMS_REDEFINITION.START_REDEF_TABLE(
       uname              => 'oe',
   ```

```
    orig_table         => 'orders',
    int_table          => 'int_orders',
    options_flag       => DBMS_REDEFINITION.CONS_USE_PK,
    refresh_dep_mviews => 'Y');
END;
/
```

2. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on `oe.int_orders`.)

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'oe',
    orig_table       => 'orders',
    int_table        => 'int_orders',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => TRUE,
    num_errors       => num_errors);
END;
/
```

3. Check the redefinition status:

```
SELECT REDEFINITION_ID, REFRESH_DEP_MVIEWS
   FROM DBA_REDEFINITION_STATUS
   WHERE BASE_TABLE_OWNER = 'OE' AND BASE_TABLE_NAME = 'ORDERS';
```

4. Perform DML on the original table. For example:

```
INSERT INTO oe.orders VALUES(3000,sysdate,'direct',102,1,42283.2,154,NULL);
COMMIT;
```

5. Synchronize the interim table `oe.int_orders`. This step refreshes the dependent materialized view `oe.orders_pk`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'OE',
    orig_table => 'ORDERS',
    int_table  => 'INT_ORDERS');
END;
/
```

6. Check the refresh status of the dependent materialized views for the `oe.orders` table:

```
SELECT m.OWNER, m.MVIEW_NAME, m.STALENESS, m.LAST_REFRESH_DATE
   FROM ALL_MVIEWS m, ALL_MVIEW_DETAIL_RELATIONS d
   WHERE m.OWNER=d.OWNER AND
         m. MVIEW_NAME=d.MVIEW_NAME AND
         d.DETAILOBJ_OWNER = 'OE' AND
         d.DETAILOBJ_NAME = 'ORDERS';
```

The `oe.orders_pk` materialized view was refreshed during the previous step, so it has `FRESH` for its `STALENESS` status. The `oe.orders_rowid` materialized view was not refreshed during the previous step, so it has `NEEDS_COMPLILE` for its `STALENESS` status.

7. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'OE',
    orig_table => 'ORDERS',
    int_table  => 'INT_ORDERS');
END;
/
```

You can query the `oe.orders_pk` materialized view to confirm that the new row inserted into the `oe.orders` table exist in the materialized view because it was refreshed during online table redefinition.

Related Topics

- Monitoring Online Table Redefinition Progress
  You can query the `V$ONLINE_REDEF` view to monitor the progress of an online table redefinition operation.

19.8.10 Monitoring Online Table Redefinition Progress

You can query the `V$ONLINE_REDEF` view to monitor the progress of an online table redefinition operation.

During the process of redefining a table online, some operations can take a long time to execute. While these operations are executing, you can query the `V$ONLINE_REDEF` view for detailed information about the progress of the operation. For example, it can take a long time for the `DBMS_REDEFINITION.START_REDEF_TABLE` procedure to load data into the interim table.

The `V$ONLINE_REDEF` view provides a percentage complete value for the operation in the `PROGRESS` column. This view shows the current step in the total number of steps required to complete the operation in the `OPERATION` column. For example, if there are 10 steps in the operation, then this column might show `Step 6 out of 10`. The view also includes a `SUBOPERATION` column and a `DETAILED_MESSAGE` column for more granular information about the current operation.

During the online table redefinition process, an internal materialized view is created, and this materialized view is refreshed during some operations to keep the original table and the interim table synchronized. You can check on the progress of a refresh that is run automatically during online table redefinition by querying the `REFRESH_STATEMENT_SQL_ID` and `REFRESH_STATEMENT` columns in the `V$ONLINE_REDEF` view. You can use the `SQL_ID` value returned in the `REFRESH_STATEMENT_SQL_ID` column to monitor the progress of a refresh in views such as the `V$SQL` view and the `V$SQL_MONITOR` view.

1. Connect to the database in a session that is separate from the session that is performing online table redefinition.
2. Query the `V$ONLINE_REDEF` view.

Example 19-17 Monitoring Online Table Redefinition Progress

This example redefines the Oracle-supplied `sh.customers` table by adding a `cust_alt_phone_number` column.

```
CREATE TABLE customers (
    cust_id                 NUMBER          NOT NULL,
    cust_first_name         VARCHAR2(20)    NOT NULL,
    cust_last_name          VARCHAR2(40)    NOT NULL,
    cust_gender             CHAR(1)         NOT NULL,
    cust_year_of_birth      NUMBER(4)       NOT NULL,
    cust_marital_status     VARCHAR2(20),
    cust_street_address    VARCHAR2(40)    NOT NULL,
    cust_postal_code       VARCHAR2(10)    NOT NULL,
    cust_city              VARCHAR2(30)    NOT NULL,
    cust_city_id           NUMBER          NOT NULL,
    cust_state_province    VARCHAR2(40)    NOT NULL,
    cust_state_province_id NUMBER          NOT NULL,
    country_id             NUMBER          NOT NULL,
    cust_main_phone_number VARCHAR2(25)    NOT NULL,
    cust_income_level      VARCHAR2(30),
    cust_credit_limit      NUMBER,
    cust_email             VARCHAR2(50),
    cust_total             VARCHAR2(14)    NOT NULL,
    cust_total_id          NUMBER          NOT NULL,
    cust_src_id            NUMBER,
    cust_eff_from          DATE,
    cust_eff_to            DATE,
    cust_valid             VARCHAR2(1));
```

This table contains a large amount of data, and some of the operations in the online table redefinition process will take time. This example monitors various operations by querying the `V$ONLINE_REDEF` view.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Create an interim table `sh.int_customers`.

   ```
   CREATE TABLE sh.int_customers (
       cust_id                 NUMBER          NOT NULL,
       cust_first_name         VARCHAR2(20)    NOT NULL,
       cust_last_name          VARCHAR2(40)    NOT NULL,
       cust_gender             CHAR(1)         NOT NULL,
       cust_year_of_birth      NUMBER(4)       NOT NULL,
       cust_marital_status     VARCHAR2(20),
       cust_street_address     VARCHAR2(40)    NOT NULL,
       cust_postal_code        VARCHAR2(10)    NOT NULL,
       cust_city               VARCHAR2(30)    NOT NULL,
       cust_city_id            NUMBER          NOT NULL,
       cust_state_province     VARCHAR2(40)    NOT NULL,
       cust_state_province_id  NUMBER          NOT NULL,
       country_id              NUMBER          NOT NULL,
       cust_main_phone_number  VARCHAR2(25)    NOT NULL,
       cust_income_level       VARCHAR2(30),
       cust_credit_limit       NUMBER,
       cust_email              VARCHAR2(50),
       cust_total              VARCHAR2(14)    NOT NULL,
       cust_total_id           NUMBER          NOT NULL,
       cust_src_id             NUMBER,
       cust_eff_from           DATE,
       cust_eff_to             DATE,
       cust_valid              VARCHAR2(1),
       cust_alt_phone_number   VARCHAR2(25));
   ```

3. Start the redefinition process, and monitor the progress of the operation.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
    uname        => 'sh',
    orig_table   => 'customers',
    int_table    => 'int_customers',
    options_flag => DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

As this operation is running, and in a session that is separate from the session that is performing online table redefinition, query the `V$ONLINE_REDEF` view to monitor its progress:

```
SELECT * FROM V$ONLINE_REDEF;
```

Output from this query might show the following:

- `START_REDEF_TABLE` for `OPERATION`
- `complete refresh the materialized view` for `SUBOPERATION`
- `step 6 out of 7` for `PROGRESS`

4. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on `sh.int_customers`.)

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'sh',
    orig_table       => 'customers',
    int_table        => 'int_customers',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => TRUE,
    num_errors       => num_errors);
END;
/
```

As this operation is running, and in a session that is separate from the session that is performing online table redefinition, query the `V$ONLINE_REDEF` view to monitor its progress:

```
SELECT * FROM V$ONLINE_REDEF;
```

Output from this query might show the following:

- `COPY_TABLE_DEPENDENTS` for `OPERATION`
- `copy the indexes` for `SUBOPERATION`
- `step 3 out of 7` for `PROGRESS`

Note that the `ignore_errors` argument is set to `TRUE` for this call. The reason is that the interim table was created with a primary key constraint, and when `COPY_TABLE_DEPENDENTS` attempts to copy the primary key constraint and index from the original table, errors occur. You can ignore these errors.

5. Synchronize the interim table `hr.int_emp_redef`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'sh',
    orig_table => 'customers',
    int_table  => 'int_customers');
END;
/
```

6. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'sh',
    orig_table => 'customers',
    int_table  => 'int_customers');
END;
/
```

Related Topics

- Refreshing Dependent Materialized Views During Online Table Redefinition
  To refresh dependent fast refreshable materialized views during online table redefinition, set the `refresh_dep_mviews` parameter to `Y` in the `REDEF_TABLE` procedure or the `START_REDEF_TABLE` procedure.

19.8.11 Restarting Online Table Redefinition After a Failure

If online table redefinition fails, then you can check the `DBA_REDEFINITION_STATUS` view to see the error information and restartable information.

If `RESTARTABLE` is `Y`, then you can correct the error and restart the online redefinition process where it last stopped. If `RESTARTABLE` is `N`, you must stop the redefinition operation.

In some cases, it is possible to restart the online redefinition of a table after a failure. Restarting the operation means that the online redefinition process begins where it stopped because of the failure, and no work is lost. For example, if a `SYNC_INTERIM_TABLE` procedure call fails because of an “unable to extent table in tablespace” error, then the problem can be corrected by increasing the size of the tablespace that ran out of space and rerunning the `SYNC_INTERIM_TABLE` procedure call.

If online table redefinition fails, then you can complete the following steps to restart it:

1. Query the `DBA_REDEFINITION_STATUS` view to determine the cause of the failure and the action required to correct it.

   For example, run the following query:

```
SELECT BASE_TABLE_NAME,
       INTERIM_OBJECT_NAME,
       OPERATION,
       STATUS,
       RESTARTABLE,
       ACTION
   FROM DBA_REDEFINITION_STATUS;
```

   If the `RESTARTABLE` value is `Y`, then the operation can be restarted. If the `RESTARTABLE` value is `N`, then the operation cannot be restarted, and redefinition must be performed again from the beginning.

2. Perform the action specified in the query results from the previous step.
3. Restart the online redefinition with the operation specified in the query results, and run all of the subsequent operations to finish online redefinition of the table.

Example 19-18   SYNC_INTERIM_TABLE Procedure Call Failure

This example illustrates restarting an online redefinition operation that failed on a `SYNC_INTERIM_TABLE` procedure call with the following error:

```
BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('U1', 'ORIG', 'INT');
END;
/
ORA-42009: error occurred while synchronizing the redefinition
ORA-01653: unable to extend table U1.INT by 8 in tablespace my_tbs
ORA-06512: at "SYS.DBMS_REDEFINITION", line 148
ORA-06512: at "SYS.DBMS_REDEFINITION", line 2807
ORA-06512: at line 2
```

1. Query the `DBA_REDEFINITION_STATUS` view:

   ```
   SELECT BASE_TABLE_NAME,  INT_TABLE_NAME, OPERATION, STATUS, RESTARTABLE, 
   ACTION
      FROM DBA_REDEFINITION_STATUS;

   BASE_TABLE_NAME INT_OBJ_NAME OPERATION          STATUS  RESTARTABLE ACTION
   --------------- ------------ ------------------ ------- ----------- 
   ---------
   ORIG            INT          SYNC_INTERIM_TABLE FAILED  Y           Fix 
   error
   ```

   The online redefinition operation can be restarted because `RESTARTABLE` is `Y` in the query results. To restart the operation, correct the error returned when the operation failed and restart the operation. In this example, the error is “ORA-01653: unable to extend table `U1.INT` by 8 in tablespace `my_tbs`”.
2. Increase the size of the `my_tbs` tablespace by adding a data file to it:

   ```
   ALTER TABLESPACE my_tbs
      ADD DATAFILE '/u02/oracle/data/my_tbs2.dbf' SIZE 100M;
   ```
3. Rerun `SYNC_INTERIM_TABLE` procedure call:

   ```
   BEGIN
      DBMS_REDEFINITION.SYNC_INTERIM_TABLE('U1', 'ORIG', 'INT');
   END;
   /
   ```

Example 19-19 Materialized View Log Problem

After the redefinition is started on the original table, there can be a problem with the materialized view log. For example, the materialized view log might be accidentally dropped or corrupted for some reason. In such cases, errors similar to the following are returned:

```
ERROR at line 1:
ORA-42010: error occurred while synchronizing the redefinition
ORA-12034: materialized view log on "HR"."T1" younger than last refresh
```

Assume a table that was created with the following SQL statement is being redefined:

```
CREATE TABLE hr.t1(
      c1 NUMBER PRIMARY KEY,
      c2 NUMBER)
   TABLESPACE example_tbs;
```

Assume an interim table was created with the following SQL statement that changes the table's tablespace:

```
CREATE TABLE hr.int_t1(
      c1 NUMBER PRIMARY KEY,
      c2 NUMBER)
   TABLESPACE hr_tbs;
```

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Start the redefinition process.

   ```
   BEGIN
     DBMS_REDEFINITION.START_REDEF_TABLE(
       uname          => 'hr',
       orig_table     => 't1',
       int_table      => 'int_t1');
   END;
   /
   ```

3. Drop the materialized view log on the original table.

   ```
   DROP MATERIALIZED VIEW LOG ON hr.t1;
   ```

4. Create a new materialized view log on the original table.

   ```
   CREATE MATERIALIZED VIEW LOG ON hr.t1
      WITH COMMIT SCN PURGE
      IMMEDIATE ASYNCHRONOUS;
   ```

5. Synchronize the interim table `hr.int_t1`.

   ```
   BEGIN
     DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
   ```

```
    uname      => 'hr',
    orig_table => 't1',
    int_table  => 'int_t1');
END;
/
BEGIN
*
ERROR at line 1:
ORA-42010: error occurred while synchronizing the redefinition
ORA-12034: materialized view log on "HR"."T1" younger than last refresh
```

6. Because an error was returned, check the `DBA_REDEFINITION_STATUS` view.

```
COLUMN BASE_OBJECT_NAME FORMAT A11
COLUMN OPERATION FORMAT A10
COLUMN STATUS FORMAT A10
COLUMN RESTARTABLE FORMAT A11
COLUMN ERR_TXT FORMAT A15
COLUMN ACTION FORMAT A18

SELECT BASE_OBJECT_NAME, OPERATION, STATUS, RESTARTABLE, ERR_TXT, ACTION
   FROM DBA_REDEFINITION_STATUS
   ORDER BY BASE_TABLE_NAME, BASE_OBJECT_NAME;


BASE_OBJECT OPERATION  STATUS     RESTARTABLE ERR_TXT         ACTION
----------- ---------- ---------- ----------- ---------------
------------------
T1          SYNC_REDEF Failure    N           ORA-12034: mate Abort
redefinition
            _TABLE                            rialized view l
                                              og on "HR"."T1"
                                               younger than l
                                              ast refresh
```

The online redefinition operation cannot be restarted because `RESTARTABLE` is `N` in the query results, and the `ACTION` column indicates that the online table redefinition operation must be terminated.

7. Terminate the online table redefinition operation.

```
BEGIN
  DBMS_REDEFINITION.ABORT_REDEF_TABLE(
    uname      => 'hr',
    orig_table => 't1',
    int_table  => 'int_t1');
END;
/
```

19.8.12 Rolling Back Online Table Redefinition

You can enable roll back of a table after online table redefinition to return the table to its original definition and preserve DML changes made to the table.

- About Online Table Redefinition Rollback
  After online table redefinition, you can roll back the table to its definition before online table redefinition while preserving all data manipulation language (DML) changes made to the table.
- Performing Online Table Redefinition Rollback
  The `ROLLBACK` procedure in the `DBMS_REDEFINITION` package returns a table that was redefined online to its original definition while preserving DML changes.

19.8.12.1 About Online Table Redefinition Rollback

After online table redefinition, you can roll back the table to its definition before online table redefinition while preserving all data manipulation language (DML) changes made to the table.

In some cases, you might want to undo an online redefinition of a table. For example, the performance of operations on the table might be worse after the redefinition than it was before the redefinition. In these cases, you can roll back the table to its original definition while preserving all of the DML changes made to the table after it was redefined. Online table redefinition rollback is used mainly when redefinition changes the storage characteristics of the table, and the changes unexpectedly result in degraded performance.

To enable rollback of online table redefinition, the `ENABLE_ROLLBACK` parameter must be set to `TRUE` in the `DBMS_REDEFINITION.START_TABLE_REDEF` procedure. When this parameter is set to true, Oracle Database maintains the interim table created during redefinition after redefinition is complete. You can run the `SYNC_INTERIM_TABLE` procedure to synchronize the interim table periodically to apply DML changes made to the redefined table to the interim table. An internal materialized view and materialized view log enables maintenance of the interim table. If you decide to roll back the online table redefinition, then the interim table is synchronized, and Oracle Database switches back to it so that the table has its original definition.

The following restrictions apply to online table redefinition rollback:

- When there is no one to one mapping of the original table's columns to interim table's columns, there must be no operators or functions in column mappings during redefinition.

  There can be operators and functions in column mappings when there is a one to one mapping of the original table's columns to interim table's columns.
- When rollback is enabled for a redefinition, the table cannot be redefined again until the online table redefinition is rolled back or terminated.

19.8.12.2 Performing Online Table Redefinition Rollback

The `ROLLBACK` procedure in the `DBMS_REDEFINITION` package returns a table that was redefined online to its original definition while preserving DML changes.

To use the `ROLLBACK` procedure, online table redefinition rollback must be enabled during online table redefinition. If you decide to retain the changes made by online table redefinition, then you can run the `ABORT_ROLLBACK` procedure.

1. Perform an online redefinition of a table, starting with the `START_REDEF_TABLE` procedure and ending with the `FINISH_REDEF_TABLE` procedure.

   The `ENABLE_ROLLBACK` parameter must be set to `TRUE` in the `START_REDEF_TABLE` procedure. The default for this parameter is `FALSE`.
2. Optional: Periodically, run the `SYNC_INTERIM_TABLE` procedure to apply DML changes made to the redefined table to the interim table.

You can improve the performance of the online table redefinition rollback if you periodically apply the DML changes to the interim table.

3. Choose one of the following options:
   - If you want to undo the changes made by online table redefinition and return to the original table definition, then run the `ROLLBACK` procedure in the `DBMS_REDEFINITION` package.
   - If you want to retain the changes made by online table redefinition, then run the `ABORT_ROLLBACK` procedure in the `DBMS_REDEFINITION` package.
     Terminating the rollback stops maintenance of the interim table and removes the materialized view and materialized view log that enabled rollback.

Example 19-20 Rolling Back Online Table Redefinition

This example illustrates online redefinition of a table by changing the storage characteristics for the table. Specifically, this example compresses the tablespace for the table during online redefinition. Assume that you want to evaluate the performance of the table after online redefinition is complete. If the table does not perform as well as expected, then you want to be able to roll back the changes made by online redefinition.

Assume that the following statements created the original tablespace and table:

```
CREATE TABLESPACE tst_rollback_tbs
   DATAFILE 'tst_rollback_tbs.dbf' SIZE 10M
   ONLINE;

CREATE TABLE hr.tst_rollback
     (rllbck_id    NUMBER(6) PRIMARY KEY,
      rllbck_name  VARCHAR2(20))
   TABLESPACE tst_rollback_tbs
   STORAGE (INITIAL 2M);
```

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Create a compressed tablespace for the interim table.

   ```
   CREATE TABLESPACE tst_cmp_rollback_tbs
      DEFAULT ROW STORE COMPRESS ADVANCED
      DATAFILE 'tst_cmp_rollback_tbs.dbf' SIZE 10M
      ONLINE;
   ```

3. Create an interim table `hr.int_tst_rollback`.

   ```
   CREATE TABLE hr.int_tst_rollback
        (rllbck_id    NUMBER(6) PRIMARY KEY,
         rllbck_name  VARCHAR2(20))
      TABLESPACE tst_cmp_rollback_tbs
      STORAGE (INITIAL 2M);
   ```

   Ensure that the interim table uses the compressed tablespace created in the previous step.

4. Start the redefinition process.

   ```
   BEGIN
     DBMS_REDEFINITION.START_REDEF_TABLE(
   ```

```
    uname          => 'hr',
    orig_table     => 'tst_rollback',
    int_table      => 'int_tst_rollback',
    options_flag   => DBMS_REDEFINITION.CONS_USE_PK,
    enable_rollback => TRUE);
END;
/
```

Ensure that `enable_rollback` is set to `TRUE` so that the changes made by online redefinition can be rolled back.

5. Copy dependent objects.

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'hr',
    orig_table       => 'tst_rollback',
    int_table        => 'int_tst_rollback',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => TRUE,
    num_errors       => num_errors);
END;
/
```

6. Query the `DBA_REDEFINITION_ERRORS` view to check for errors.

```
SET LONG  8000
SET PAGES 8000
COLUMN OBJECT_NAME HEADING 'Object Name' FORMAT A20
COLUMN BASE_TABLE_NAME HEADING 'Base Table Name' FORMAT A10
COLUMN DDL_TXT HEADING 'DDL That Caused Error' FORMAT A40

SELECT OBJECT_NAME, BASE_TABLE_NAME, DDL_TXT FROM
         DBA_REDEFINITION_ERRORS;
```

You can ignore errors related to the primary key and indexes.

7. Synchronize the interim table `hr.int_tst_rollback`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'hr',
    orig_table => 'tst_rollback',
    int_table  => 'int_tst_rollback');
END;
/
```

8. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'hr',
    orig_table => 'tst_rollback',
    int_table  => 'int_tst_rollback');
END;
/
```

The table `hr.tst_rollbck` is locked in the exclusive mode only for a small window toward the end of this step. After this call the table `hr.tst_rollback` is redefined such that it has all the attributes of the `hr.int_tst_rollback` table. In this example, the tablespace for the `hr.tst_rollbck` table is now compressed.

9. During the evaluation period, you can periodically synchronize the interim table `hr.int_tst_rollback`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'hr',
    orig_table => 'tst_rollback',
    int_table  => 'int_tst_rollback');
END;
/
```

   Synchronizing the tables updates the original table with the DML changes made to the redefined table. When you synchronize the tables periodically, a rollback operation is more efficient because fewer DML changes must be made to the original table. You can query the `STATUS` column of the `DBA_REDEFINITION_STATUS` view to determine the status of the rollback operation.

10. Perform one of the following actions:

    - Assume that the redefined table did not perform as well as expected, and roll back the changes made by online redefinition.

```
BEGIN
  DBMS_REDEFINITION.ROLLBACK(
    uname      => 'hr',
    orig_table => 'tst_rollback',
    int_table  => 'int_tst_rollback');
END;
/
```

    - Assume that the redefined table performed as expected, and terminate the rollback to retain the changes made by online table redefinition and clean up the database objects that enable rollback.

```
BEGIN
  DBMS_REDEFINITION.ABORT_ROLLBACK(
    uname      => 'hr',
    orig_table => 'tst_rollback',
    int_table  => 'int_tst_rollback');
END;
/
```

19.8.13 Terminating Online Table Redefinition and Cleaning Up After Errors

You can terminate the online redefinition process. Doing so drops temporary logs and tables associated with the redefinition process. After this procedure is called, you can drop the interim table and its dependent objects.

To terminate the online redefinition process in the event that an error is raised during the redefinition process, or if you choose to terminate the redefinition process manually:

- Run the `ABORT_REDEF_TABLE` procedure.

If the online redefinition process must be restarted, if you do not first call `ABORT_REDEF_TABLE`, then subsequent attempts to redefine the table will fail.

Note:

It is not necessary to call the `ABORT_REDEF_TABLE` procedure if the redefinition process stops because the `FINISH_REDEF_TABLE` procedure has timed out. The `dml_lock_timeout` parameter in the `FINISH_REDEF_TABLE` procedure controls the time-out period. See step 8 in "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information

19.8.14 Online Redefinition of One or More Partitions

You can redefine online one or more partitions of a table. This is useful if, for example, you want to move partitions to a different tablespace and keep the partitions available for DML during the operation.

You can redefine multiple partitions in a table at one time. If you do, then multiple interim tables are required during the table redefinition process. Ensure that you have enough free space and undo space to complete the table redefinition.

When you redefine multiple partitions, you can specify that the redefinition continues even if it encounters an error for a particular partition. To do so, set the `continue_after_errors` parameter to `TRUE` in redefinition procedures in the `DBMS_REDEFINITION` package. You can check the `DBA_REDEFINITION_STATUS` view to see if any errors were encountered during the redefinition process. The `STATUS` column in this view shows whether the redefinition process succeeded or failed for each partition.

You can also redefine an entire table one partition at a time to reduce resource requirements. For example, to move a very large table to a different tablespace, you can move it one partition at a time to minimize the free space and undo space required to complete the move.

Redefining partitions differs from redefining a table in the following ways:

- There is no need to copy dependent objects. It is not valid to use the `COPY_TABLE_DEPENDENTS` procedure when redefining a single partition.
- You must manually create and register any local indexes on the interim table.

  See "Creating Dependent Objects Manually".
- The column mapping string for `START_REDEF_TABLE` must be `NULL`.

Note:

Starting with Oracle Database 12*c*, you can use the simpler `ALTER TABLE...MOVE PARTITION ... ONLINE` statement to move a partition or subpartition online without using online table redefinition. DML operations can continue to run uninterrupted on the partition or subpartition that is being moved. See "Moving a Table to a New Segment or Tablespace".

- Rules for Online Redefinition of a Single Partition
  The underlying mechanism for redefinition of a single partition is the **exchange partition** capability of the database (`ALTER TABLE...EXCHANGE PARTITION`).

See Also:

Oracle Database VLDB and Partitioning Guide

19.8.14.1 Rules for Online Redefinition of a Single Partition

The underlying mechanism for redefinition of a single partition is the **exchange partition** capability of the database (`ALTER TABLE`...`EXCHANGE PARTITION`).

Rules and restrictions for online redefinition of a single partition are therefore governed by this mechanism. Here are some general restrictions:

- No logical changes (such as adding or dropping a column) are permitted.
- No changes to the partitioning method (such as changing from range partitioning to hash partitioning) are permitted.

Here are the rules for defining the interim table:

- If the partition being redefined is a range, hash, or list partition, then the interim table must be nonpartitioned.
- If the partition being redefined is a range partition of a composite range-hash partitioned table, then the interim table must be a hash partitioned table. In addition, the partitioning key of the interim table must be identical to the subpartitioning key of the range-hash partitioned table, and the number of partitions in the interim table must be identical to the number of subpartitions in the range partition being redefined.
- If the partition being redefined is a range partition of a composite range-list partitioned table, then the interim table must be a list partitioned table. In addition, the partitioning key of the interim table must be identical to the subpartitioning key of the range-list partitioned table, and the values lists of the interim table's list partitions must exactly match the values lists of the list subpartitions in the range partition being redefined.
- If you define the interim table as compressed, then you must use the by-key method of redefinition, not the by-rowid method.

These additional rules apply if the table being redefined is a partitioned index-organized table:

- The interim table must also be index-organized.
- The original and interim tables must have primary keys on the same columns, in the same order.
- If prefix compression is enabled, then it must be enabled for both the original and interim tables, with the same prefix length.
- Both the original and interim tables must have overflow segments, or neither can have them. Likewise for mapping tables.

See Also:

- The section "Exchanging Partitions" in *Oracle Database VLDB and Partitioning Guide*
- "Online Table Redefinition Examples" for examples that redefine tables with partitions

19.8.15 Online Table Redefinition Examples

Examples illustrate online redefinition of tables.

For the following examples, see *Oracle Database PL/SQL Packages and Types Reference* for descriptions of all `DBMS_REDEFINITION` subprograms.

Example	Description
Example 1	Redefines a table's storage properties in a single step with the `REDEF_TABLE` procedure.
Example 2	Redefines a table by adding new columns and adding partitioning.
Example 3	Demonstrates redefinition with object data types.
Example 4	Demonstrates redefinition with manually registered dependent objects.
Example 5	Redefines multiple partitions, moving them to different tablespaces.
Example 6	Redefines a table with virtual private database (VPD) policies without changing the properties of any of the table's columns.
Example 7	Redefines a table with VPD policies and changes the properties of one of the table's columns.
Example 8	Redefines a table by making multiple changes using online redefinition.

Example 1

This example illustrates online redefinition of a table's storage properties using the `REDEF_TABLE` procedure.

The original table, named `print_ads`, is defined in the `pm` schema as follows:

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
AD_ID                                              NUMBER(6)
AD_TEXT                                            CLOB
```

In this table, the LOB column `ad_text` uses BasicFiles LOB storage.

An index for the table was created with the following SQL statement:

```
CREATE INDEX pm.print_ads_ix
    ON print_ads (ad_id)
  TABLESPACE example;
```

The table is redefined as follows:

- The table is compressed with advanced row compression.
- The table's tablespace is changed from `EXAMPLE` to `NEWTBS`. This example assumes that the `NEWTBS` tablespace exists.
- The index is compressed with `COMPRESS 1` compression.
- The index's tablespace is changed from `EXAMPLE` to `NEWIDXTBS`. This example assumes that the `NEWIDXTBS` tablespace exists.
- The LOB column in the table is compressed with `COMPRESS HIGH` compression.
- The tablespace for the LOB column is changed from `EXAMPLE` to `NEWLOBTBS`. This example assumes that the `NEWLOBTBS` tablespace exists.
- The LOB column is changed to SecureFiles LOB storage.

The steps in this redefinition are illustrated below.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Run the `REDEF_TABLE` procedure:

```
BEGIN
  DBMS_REDEFINITION.REDEF_TABLE(
    uname                      => 'PM',
    tname                      => 'PRINT_ADS',
    table_compression_type     => 'ROW STORE COMPRESS ADVANCED',
    table_part_tablespace      => 'NEWTBS',
    index_key_compression_type => 'COMPRESS 1',
    index_tablespace           => 'NEWIDXTBS',
    lob_compression_type       => 'COMPRESS HIGH',
    lob_tablespace             => 'NEWLOBTBS',
    lob_store_as               => 'SECUREFILE');
END;
/
```

Note:

If an errors occurs, then the interim table is dropped, and the `REDEF_TABLE` procedure must be re-executed.

Example 2

This example illustrates online redefinition of a table by adding new columns and adding partitioning.

The original table, named `emp_redef`, is defined in the `hr` schema as follows:

```
Name      Type
--------- ----------------------------
EMPNO     NUMBER(5)     <- Primary key
ENAME     VARCHAR2(15)
JOB       VARCHAR2(10)
DEPTNO    NUMBER(3)
```

The table is redefined as follows:

- New columns mgr, hiredate, sal, and bonus are added.
- The new column bonus is initialized to 0 (zero).
- The column deptno has its value increased by 10.
- The redefined table is partitioned by range on empno.

The steps in this redefinition are illustrated below.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Verify that the table is a candidate for online redefinition. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

```
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
    uname        => 'hr',
    tname        =>'emp_redef',
    options_flag => DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

3. Create an interim table hr.int_emp_redef.

```
CREATE TABLE hr.int_emp_redef
        (empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         sal        NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL,
         bonus      NUMBER (7,2) DEFAULT(0))
     PARTITION BY RANGE(empno)
       (PARTITION emp1000 VALUES LESS THAN (1000) TABLESPACE admin_tbs,
        PARTITION emp2000 VALUES LESS THAN (2000) TABLESPACE admin_tbs2);
```

   Ensure that the specified tablespaces exist.

4. Start the redefinition process.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
    uname        => 'hr',
    orig_table   => 'emp_redef',
    int_table    => 'int_emp_redef',
    col_mapping  => 'empno empno, ename ename, job job, deptno+10 deptno,
                     0 bonus',
    options_flag => DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

5. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on hr.int_emp_redef.)

```
DECLARE
num_errors PLS_INTEGER;
```

```
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'hr',
    orig_table       => 'emp_redef',
    int_table        => 'int_emp_redef',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => TRUE,
    num_errors       => num_errors);
END;
/
```

Note that the `ignore_errors` argument is set to `TRUE` for this call. The reason is that the interim table was created with a primary key constraint, and when `COPY_TABLE_DEPENDENTS` attempts to copy the primary key constraint and index from the original table, errors occur. You can ignore these errors, but you must run the query shown in the next step to see if there are other errors.

6. Query the `DBA_REDEFINITION_ERRORS` view to check for errors.

```
SET LONG  8000
SET PAGES 8000
COLUMN OBJECT_NAME HEADING 'Object Name' FORMAT A20
COLUMN BASE_TABLE_NAME HEADING 'Base Table Name' FORMAT A10
COLUMN DDL_TXT HEADING 'DDL That Caused Error' FORMAT A40

SELECT OBJECT_NAME, BASE_TABLE_NAME, DDL_TXT FROM
         DBA_REDEFINITION_ERRORS;

Object Name          Base Table DDL That Caused Error
-------------------- ---------- ----------------------------------------
SYS_C006796          EMP_REDEF  CREATE UNIQUE INDEX "HR"."TMP$$_SYS_C006
                                7960" ON "HR"."INT_EMP_REDEF" ("EMPNO")
                                  PCTFREE 10 INITRANS 2 MAXTRANS 255
                                  STORAGE(INITIAL 65536 NEXT 1048576 MIN
                                EXTENTS 1 MAXEXTENTS 2147483645
                                  PCTINCREASE 0 FREELISTS 1 FREELIST GRO
                                UPS 1
                                  BUFFER_POOL DEFAULT)
                                  TABLESPACE "ADMIN_TBS"
SYS_C006794          EMP_REDEF  ALTER TABLE "HR"."INT_EMP_REDEF" MODIFY
                                ("ENAME" CONSTRAINT "TMP$$_SYS_C0067940"
                                 NOT NULL ENABLE NOVALIDATE)
SYS_C006795          EMP_REDEF  ALTER TABLE "HR"."INT_EMP_REDEF" MODIFY
                                ("DEPTNO" CONSTRAINT "TMP$$_SYS_C0067950
                                " NOT NULL ENABLE NOVALIDATE)
SYS_C006796          EMP_REDEF  ALTER TABLE "HR"."INT_EMP_REDEF" ADD CON
                                STRAINT "TMP$$_SYS_C0067960" PRIMARY KEY
                                 ("EMPNO")
                                  USING INDEX PCTFREE 10 INITRANS 2 MAXT
                                RANS 255
                                  STORAGE(INITIAL 65536 NEXT 1048576 MIN
                                EXTENTS 1 MAXEXTENTS 2147483645
                                  PCTINCREASE 0 FREELISTS 1 FREELIST GRO
                                UPS 1
                                  BUFFER_POOL DEFAULT)
                                  TABLESPACE "ADMIN_TBS"  ENABLE NOVALID
                                ATE
```

These errors are caused by the existing primary key constraint on the interim table and can be ignored. Note that with this approach, the names of the primary key constraint and index on the post-redefined table are changed. An alternate approach, one that avoids errors and name changes, would be to define the interim table without a primary key constraint. In this case, the primary key constraint and index are copied from the original table.

Note:

The best approach is to define the interim table with a primary key constraint, use `REGISTER_DEPENDENT_OBJECT` to register the primary key constraint and index, and then copy the remaining dependent objects with `COPY_TABLE_DEPENDENTS`. This approach avoids errors and ensures that the redefined table always has a primary key and that the dependent object names do not change.

7. (Optional) Synchronize the interim table `hr.int_emp_redef`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'hr',
    orig_table => 'emp_redef',
    int_table  => 'int_emp_redef');
END;
/
```

8. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'hr',
    orig_table => 'emp_redef',
    int_table  => 'int_emp_redef');
END;
/
```

   The table `hr.emp_redef` is locked in the exclusive mode only for a small window toward the end of this step. After this call the table `hr.emp_redef` is redefined such that it has all the attributes of the `hr.int_emp_redef` table.

   Consider specifying a non-`NULL` value for the `dml_lock_timeout` parameter in this procedure. See step 8 in "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information.

9. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 3

This example redefines a table to change columns into object attributes. The redefined table gets a new column that is an object type.

The original table, named `customer`, is defined as follows:

```
Name         Type
------------ -------------
CID          NUMBER          <- Primary key
NAME         VARCHAR2(30)
STREET       VARCHAR2(100)
CITY         VARCHAR2(30)
```

```
STATE         VARCHAR2(2)
ZIP           NUMBER(5)
```

The type definition for the new object is:

```
CREATE TYPE addr_t AS OBJECT (
   street VARCHAR2(100),
   city VARCHAR2(30),
   state VARCHAR2(2),
   zip NUMBER(5, 0) );
/
```

Here are the steps for this redefinition:

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Verify that the table is a candidate for online redefinition. Specify that the redefinition is to be done using primary keys or pseudo-primary keys.

   ```
   BEGIN
     DBMS_REDEFINITION.CAN_REDEF_TABLE(
       uname         => 'steve',
       tname         =>'customer',
       options_flag => DBMS_REDEFINITION.CONS_USE_PK);
   END;
   /
   ```

3. Create the interim table `int_customer`.

   ```
   CREATE TABLE int_customer(
     CID    NUMBER,
     NAME   VARCHAR2(30),
     ADDR   addr_t);
   ```

   Note that no primary key is defined on the interim table. When dependent objects are copied in step 6, the primary key constraint and index are copied.

4. Because `customer` is a very large table, specify parallel operations for the next step.

   ```
   ALTER SESSION FORCE PARALLEL DML PARALLEL 4;
   ALTER SESSION FORCE PARALLEL QUERY PARALLEL 4;
   ```

5. Start the redefinition process using primary keys.

   ```
   BEGIN
     DBMS_REDEFINITION.START_REDEF_TABLE(
      uname        => 'steve',
      orig_table   => 'customer',
      int_table    => 'int_customer',
      col_mapping => 'cid cid, name name,
         addr_t(street, city, state, zip) addr');
   END;
   /
   ```

   Note that `addr_t(street, city, state, zip)` is a call to the object constructor.

6. Copy dependent objects.

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'steve',
    orig_table       => 'customer',
    int_table        => 'int_customer',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => FALSE,
    num_errors       => num_errors,
    copy_statistics  => TRUE);
END;
/
```

Note that for this call, the final argument indicates that table statistics are to be copied to the interim table.

7. Optionally synchronize the interim table.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'steve',
    orig_table => 'customer',
    int_table  => 'int_customer');
END;
/
```

8. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'steve',
    orig_table => 'customer',
    int_table  => 'int_customer');
END;
/
```

Consider specifying a non-`NULL` value for the `dml_lock_timeout` parameter in this procedure. See step 8 in "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information.

9. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 4

This example addresses the situation where a dependent object must be manually created and registered.

The table to be redefined is defined as follows:

```
CREATE TABLE steve.t1
  (c1 NUMBER);
```

The table has an index for column c1:

```
CREATE INDEX steve.index1 ON steve.t1(c1);
```

Consider the case where column `c1` becomes column `c2` after the redefinition. In this case, `COPY_TABLE_DEPENDENTS` tries to create an index on the interim table corresponding to `index1`, and tries to create it on a column c1, which does not exist in the interim table. This results in an error. You must therefore manually create the index on column c2 and register it.

Here are the steps for this redefinition:

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Ensure that t1 is a candidate for online redefinition with CAN_REDEF_TABLE, and then begin the redefinition process with START_REDEF_TABLE.

```
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
   uname        => 'steve',
   tname        => 't1',
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID);
END;
/
```

3. Create the interim table int_t1 and create an index int_index1 on column c2.

```
CREATE TABLE steve.int_t1
  (c2 NUMBER);

CREATE INDEX steve.int_index1 ON steve.int_t1(c2);
```

4. Start the redefinition process.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
    uname        => 'steve',
    orig_table   => 't1',
    int_table    => 'int_t1',
    col_mapping  => 'c1 c2',
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID);
END;
/
```

5. Register the original (index1) and interim (int_index1) dependent objects.

```
BEGIN
 DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(
   uname         => 'steve',
   orig_table    => 't1',
   int_table     => 'int_t1',
   dep_type      => DBMS_REDEFINITION.CONS_INDEX,
   dep_owner     => 'steve',
   dep_orig_name => 'index1',
   dep_int_name  => 'int_index1');
END;
/
```

6. Copy the dependent objects.

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'steve',
    orig_table       => 't1',
    int_table        => 'int_t1',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => TRUE,
    num_errors       => num_errors);
END;
/
```

7. Optionally synchronize the interim table.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'steve',
    orig_table => 't1',
    int_table  => 'int_t1');
END;
/
```

8. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'steve',
    orig_table => 't1',
    int_table  => 'int_t1');
END;
/
```

9. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 5

This example demonstrates redefining multiple partitions. It moves two of the partitions of a range-partitioned sales table to new tablespaces. The table containing the partitions to be redefined is defined as follows:

```
CREATE TABLE steve.salestable
  (s_productid NUMBER,
  s_saledate DATE,
  s_custid NUMBER,
  s_totalprice NUMBER)
  TABLESPACE users
  PARTITION BY RANGE(s_saledate)
  (PARTITION sal10q1 VALUES LESS THAN (TO_DATE('01-APR-2010', 'DD-MON-YYYY')),
  PARTITION sal10q2 VALUES LESS THAN (TO_DATE('01-JUL-2010', 'DD-MON-YYYY')),
  PARTITION sal10q3 VALUES LESS THAN (TO_DATE('01-OCT-2010', 'DD-MON-YYYY')),
  PARTITION sal10q4 VALUES LESS THAN (TO_DATE('01-JAN-2011', 'DD-MON-YYYY')));
```

This example moves the `sal10q1` partition to the `sales1` tablespace and the `sal10q2` partition to the `sales2` tablespace. The `sal10q3` and `sal10q4` partitions are not moved.

To move the partitions, the tablespaces `sales1` and `sales2` must exist. The following examples create these tablespaces:

```
CREATE TABLESPACE sales1 DATAFILE '/u02/oracle/data/sales01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

CREATE TABLESPACE sales2 DATAFILE '/u02/oracle/data/sales02.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

Note:

You can also complete this operation by executing two `ALTER TABLE ... MOVE PARTITION ... ONLINE` statements. See "Moving a Table to a New Segment or Tablespace".

The table has a local partitioned index that is defined as follows:

```
CREATE INDEX steve.sales_index ON steve.salestable
   (s_saledate, s_productid, s_custid) LOCAL;
```

Here are the steps. In the following procedure calls, note the extra argument: partition name (`part_name`).

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Ensure that `salestable` is a candidate for redefinition.

```
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
   uname        => 'steve',
   tname        => 'salestable',
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
   part_name    => 'sal10q1, sal10q2');
END;
/
```

3. Create the interim tables in the new tablespaces. Because this is a redefinition of a range partition, the interim tables are nonpartitioned.

```
CREATE TABLE steve.int_salestb1
  (s_productid NUMBER,
  s_saledate DATE,
  s_custid NUMBER,
  s_totalprice NUMBER)
  TABLESPACE sales1;

CREATE TABLE steve.int_salestb2
  (s_productid NUMBER,
  s_saledate DATE,
  s_custid NUMBER,
  s_totalprice NUMBER)
  TABLESPACE sales2;
```

4. Start the redefinition process using rowid.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
   uname        => 'steve',
   orig_table   => 'salestable',
   int_table    => 'int_salestb1, int_salestb2',
   col_mapping  => NULL,
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
   part_name    => 'sal10q1, sal10q2',
   continue_after_errors => TRUE);
```

```
END;
/
```

Notice that the `part_name` parameter specifies both of the partitions and that the `int_table` parameter specifies the interim table for each partition. Also, the `continue_after_errors` parameter is set to `TRUE` so that the redefinition process continues even if it encounters an error for a particular partition.

5. Manually create any local indexes on the interim tables.

```
CREATE INDEX steve.int_sales1_index ON steve.int_salestb1
(s_saledate, s_productid, s_custid)
TABLESPACE sales1;

CREATE INDEX steve.int_sales2_index ON steve.int_salestb2
(s_saledate, s_productid, s_custid)
TABLESPACE sales2;
```

6. Optionally synchronize the interim tables.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
   uname      => 'steve',
   orig_table => 'salestable',
   int_table  => 'int_salestb1, int_salestb2',
   part_name    => 'sal10q1, sal10q2',
   continue_after_errors => TRUE);
END;
/
```

7. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
   uname      => 'steve',
   orig_table => 'salestable',
   int_table  => 'int_salestb1, int_salestb2',
   part_name    => 'sal10q1, sal10q2',
   continue_after_errors => TRUE);
END;
/
```

Consider specifying a non-`NULL` value for the `dml_lock_timeout` parameter in this procedure. See step 8 in "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information.

8. Wait for any long-running queries against the interim tables to complete, and then drop the interim tables.

9. (Optional) Query the `DBA_REDEFINITION_STATUS` view to ensure that the redefinition succeeded for each partition.

```
SELECT BASE_TABLE_OWNER, BASE_TABLE_NAME, OPERATION, STATUS
  FROM DBA_REDEFINITION_STATUS;
```

If redefinition failed for any partition, then query the `DBA_REDEFINITION_ERRORS` view to determine the cause of the failure. Correct the conditions that caused the failure, and rerun online redefinition.

The following query shows that two of the partitions in the table have been moved to the new tablespaces:

```
SELECT PARTITION_NAME, TABLESPACE_NAME FROM DBA_TAB_PARTITIONS
 WHERE TABLE_NAME = 'SALESTABLE';
```

```
PARTITION_NAME                 TABLESPACE_NAME
------------------------------ ------------------------------
SAL10Q1                        SALES1
SAL10Q2                        SALES2
SAL10Q3                        USERS
SAL10Q4                        USERS

4 rows selected.
```

Example 6

This example illustrates online redefinition of a table with virtual private database (VPD) policies. The example disables all triggers for a table without changing any of the column names or column types in the table.

The table to be redefined is defined as follows:

```
CREATE TABLE hr.employees(
   employee_id    NUMBER(6)  PRIMARY KEY,
   first_name     VARCHAR2(20),
   last_name      VARCHAR2(25)
              CONSTRAINT     emp_last_name_nn  NOT NULL,
   email          VARCHAR2(25)
              CONSTRAINT     emp_email_nn  NOT NULL,
   phone_number   VARCHAR2(20),
   hire_date      DATE
              CONSTRAINT     emp_hire_date_nn  NOT NULL,
   job_id         VARCHAR2(10)
              CONSTRAINT     emp_job_nn  NOT NULL,
   salary         NUMBER(8,2),
   commission_pct NUMBER(2,2),
   manager_id     NUMBER(6),
   department_id  NUMBER(4),
                  CONSTRAINT     emp_salary_min
                     CHECK (salary > 0),
                  CONSTRAINT     emp_email_uk
                     UNIQUE (email));
```

If you installed the `HR` sample schema, then this table exists in your database.

Assume that the following `auth_emp_dep_100` function is created for the VPD policy:

```
CREATE OR REPLACE FUNCTION hr.auth_emp_dep_100(
  schema_var IN VARCHAR2,
  table_var IN VARCHAR2
 )
 RETURN VARCHAR2
 AS
  return_val VARCHAR2 (400);
  unm        VARCHAR2(30);
 BEGIN
  SELECT USER INTO unm FROM DUAL;
  IF (unm = 'HR') THEN
  return_val := NULL;
 ELSE
  return_val := 'DEPARTMENT_ID = 100';
  END IF;
 RETURN return_val;
END auth_emp_dep_100;
/
```

The following ADD_POLICY procedure specifies a VPD policy for the original table hr.employees using the auth_emp_dep_100 function:

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'hr',
    object_name      => 'employees',
    policy_name      => 'employees_policy',
    function_schema  => 'hr',
    policy_function  => 'auth_emp_dep_100',
    statement_types  => 'select, insert, update, delete'
   );
 END;
/
```

In this example, the hr.employees table is redefined to disable all of its triggers. No column names or column types are changed during redefinition. Therefore, specify DBMS_REDEFINITION.CONS_VPD_AUTO for the copy_vpd_opt in the START_REFEF_TABLE procedure.

The steps in this redefinition are illustrated below.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table and the required privileges for managing VPD policies.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package" and EXECUTE privilege on the DBMS_RLS package.

   See "Connecting to the Database with SQL*Plus".

2. Verify that the table is a candidate for online redefinition. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

   ```
   BEGIN
     DBMS_REDEFINITION.CAN_REDEF_TABLE('hr','employees',
         DBMS_REDEFINITION.CONS_USE_PK);
   END;
   /
   ```

3. Create an interim table hr.int_employees.

   ```
   CREATE TABLE hr.int_employees(
      employee_id    NUMBER(6),
      first_name     VARCHAR2(20),
      last_name      VARCHAR2(25),
      email          VARCHAR2(25),
      phone_number   VARCHAR2(20),
      hire_date      DATE,
      job_id         VARCHAR2(10),
      salary         NUMBER(8,2),
      commission_pct NUMBER(2,2),
      manager_id     NUMBER(6),
      department_id  NUMBER(4));
   ```

4. Start the redefinition process.

   ```
   BEGIN
     DBMS_REDEFINITION.START_REDEF_TABLE (
       uname          => 'hr',
       orig_table     => 'employees',
       int_table      => 'int_employees',
       col_mapping    => NULL,
       options_flag   => DBMS_REDEFINITION.CONS_USE_PK,
       orderby_cols   => NULL,
   ```

```
    part_name      => NULL,
    copy_vpd_opt   => DBMS_REDEFINITION.CONS_VPD_AUTO);
END;
/
```

When the `copy_vpd_opt` parameter is set to `DBMS_REDEFINITION.CONS_VPD_AUTO`, only the table owner and the user invoking online redefinition can access the interim table during online redefinition.

Also, notice that the `col_mapping` parameter is set to `NULL`. When the `copy_vpd_opt` parameter is set to `DBMS_REDEFINITION.CONS_VPD_AUTO`, the `col_mapping` parameter must be `NULL` or `'*'`. See "Handling Virtual Private Database (VPD) Policies During Online Redefinition".

5. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on `hr.int_employees`.)

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname             => 'hr',
    orig_table        => 'employees',
    int_table         => 'int_employees',
    copy_indexes      => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers     => TRUE,
    copy_constraints  => TRUE,
    copy_privileges   => TRUE,
    ignore_errors     => FALSE,
    num_errors        => num_errors);
END;
/
```

6. Disable all of the triggers on the interim table.

```
ALTER TABLE hr.int_employees
  DISABLE ALL TRIGGERS;
```

7. (Optional) Synchronize the interim table `hr.int_employees`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'hr',
    orig_table => 'employees',
    int_table  => 'int_employees');
END;
/
```

8. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'hr',
    orig_table => 'employees',
    int_table  => 'int_employees');
END;
/
```

The table `hr.employees` is locked in the exclusive mode only for a small window toward the end of this step. After this call the table `hr.employees` is redefined such that it has all the attributes of the `hr.int_employees` table.

Consider specifying a non-NULL value for the dml_lock_timeout parameter in this procedure. See step 8 in "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information.

9. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 7

This example illustrates online redefinition of a table with virtual private database (VPD) policies. The example changes the name of a column in the table.

The table to be redefined is defined as follows:

```
CREATE TABLE oe.orders(
   order_id       NUMBER(12)  PRIMARY KEY,
   order_date     TIMESTAMP WITH LOCAL TIME ZONE CONSTRAINT order_date_nn NOT NULL,
   order_mode     VARCHAR2(8),
   customer_id    NUMBER(6) CONSTRAINT order_customer_id_nn NOT NULL,
   order_status   NUMBER(2),
   order_total    NUMBER(8,2),
   sales_rep_id   NUMBER(6),
   promotion_id   NUMBER(6),
   CONSTRAINT     order_mode_lov
                  CHECK (order_mode in ('direct','online')),
   CONSTRAINT     order_total_min
                  check (order_total >= 0));
```

If you installed the OE sample schema, then this table exists in your database.

Assume that the following auth_orders function is created for the VPD policy:

```
CREATE OR REPLACE FUNCTION oe.auth_orders(
  schema_var IN VARCHAR2,
  table_var IN VARCHAR2
 )
 RETURN VARCHAR2
 AS
  return_val VARCHAR2 (400);
  unm        VARCHAR2(30);
 BEGIN
  SELECT USER INTO unm FROM DUAL;
  IF (unm = 'OE') THEN
  return_val := NULL;
 ELSE
  return_val := 'SALES_REP_ID = 159';
  END IF;
 RETURN return_val;
END auth_orders;
/
```

The following ADD_POLICY procedure specifies a VPD policy for the original table oe.orders using the auth_orders function:

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'oe',
    object_name      => 'orders',
    policy_name      => 'orders_policy',
    function_schema  => 'oe',
    policy_function  => 'auth_orders',
    statement_types  => 'select, insert, update, delete');
```

```
 END;
/
```

In this example, the table is redefined to change the `sales_rep_id` column to `sale_pid`. When one or more column names or column types change during redefinition, you must specify `DBMS_REDEFINITION.CONS_VPD_MANUAL` for the `copy_vpd_opt` in the `START_REFEF_TABLE` procedure.

The steps in this redefinition are illustrated below.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table and the required privileges for managing VPD policies.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package" and `EXECUTE` privilege on the `DBMS_RLS` package.

   See "Connecting to the Database with SQL*Plus".

2. Verify that the table is a candidate for online redefinition. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

   ```
   BEGIN
     DBMS_REDEFINITION.CAN_REDEF_TABLE(
       uname         => 'oe',
       tname         => 'orders',
       options_flag => DBMS_REDEFINITION.CONS_USE_PK);
   END;
   /
   ```

3. Create an interim table `oe.int_orders`.

   ```
   CREATE TABLE oe.int_orders(
      order_id      NUMBER(12),
      order_date    TIMESTAMP WITH LOCAL TIME ZONE,
      order_mode    VARCHAR2(8),
      customer_id   NUMBER(6),
      order_status  NUMBER(2),
      order_total   NUMBER(8,2),
      sales_pid     NUMBER(6),
      promotion_id  NUMBER(6));
   ```

   Note that the `sales_rep_id` column is changed to the `sales_pid` column in the interim table.

4. Start the redefinition process.

   ```
   BEGIN
     DBMS_REDEFINITION.START_REDEF_TABLE (
       uname          => 'oe',
       orig_table     => 'orders',
       int_table      => 'int_orders',
       col_mapping    => 'order_id order_id, order_date order_date, order_mode
                          order_mode, customer_id customer_id, order_status
                          order_status, order_total order_total, sales_rep_id
                          sales_pid, promotion_id promotion_id',
       options_flag   => DBMS_REDEFINITION.CONS_USE_PK,
       orderby_cols   => NULL,
       part_name      => NULL,
       copy_vpd_opt   => DBMS_REDEFINITION.CONS_VPD_MANUAL);
   END;
   /
   ```

Because a column name is different in the original table and the interim table, DBMS_REDEFINITION.CONS_VPD_MANUAL must be specified for the copy_vpd_opt parameter. See "Handling Virtual Private Database (VPD) Policies During Online Redefinition".

5. Create the VPD policy on the interim table.

   In this example, complete the following steps:

   a. Create a new function called auth_orders_sales_pid for the VPD policy that specifies the sales_pid column instead of the sales_rep_id column:

```
CREATE OR REPLACE FUNCTION oe.auth_orders_sales_pid(
  schema_var IN VARCHAR2,
  table_var IN VARCHAR2
 )
 RETURN VARCHAR2
 AS
  return_val VARCHAR2 (400);
  unm        VARCHAR2(30);
 BEGIN
  SELECT USER INTO unm FROM DUAL;
  IF (unm = 'OE') THEN
  return_val := NULL;
 ELSE
  return_val := 'SALES_PID = 159';
  END IF;
 RETURN return_val;
END auth_orders_sales_pid;
/
```

   b. Run the ADD_POLICY procedure and specify the new function auth_orders_sales_pid and the interim table int_orders:

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'oe',
    object_name      => 'int_orders',
    policy_name      => 'orders_policy',
    function_schema  => 'oe',
    policy_function  => 'auth_orders_sales_pid',
    statement_types  => 'select, insert, update, delete');
 END;
/
```

6. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on oe.int_orders.)

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname            => 'oe',
    orig_table       => 'orders',
    int_table        => 'int_orders',
    copy_indexes     => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => TRUE,
    num_errors       => num_errors);
END;
/
```

Note that the `ignore_errors` argument is set to `TRUE` for this call. The reason is that the original table has an index and a constraint related to the `sales_rep_id` column, and this column is changed to `sales_pid` in the interim table. The next step shows the errors and describes how to create the index and the constraint on the interim table.

7. Query the `DBA_REDEFINITION_ERRORS` view to check for errors.

```
SET LONG  8000
SET PAGES 8000
COLUMN OBJECT_NAME HEADING 'Object Name' FORMAT A20
COLUMN BASE_TABLE_NAME HEADING 'Base Table Name' FORMAT A10
COLUMN DDL_TXT HEADING 'DDL That Caused Error' FORMAT A40

SELECT OBJECT_NAME, BASE_TABLE_NAME, DDL_TXT FROM
         DBA_REDEFINITION_ERRORS;

Object Name          Base Table DDL That Caused Error
-------------------- ---------- ----------------------------------------
ORDERS_SALES_REP_FK  ORDERS     ALTER TABLE "OE"."INT_ORDERS" ADD CONSTR
                                AINT "TMP$$_ORDERS_SALES_REP_FK1" FOREIG
                                N KEY ("SALES_REP_ID")
                                          REFERENCES "HR"."EMPLOYEES"
                                ("EMPLOYE
                                E_ID") ON DELETE SET NULL DISABLE
ORD_SALES_REP_IX     ORDERS     CREATE INDEX "OE"."TMP$$_ORD_SALES_REP_I
                                X0" ON "OE"."INT_ORDERS" ("SALES_REP_ID"
                                )
                                  PCTFREE 10 INITRANS 2 MAXTRANS 255 COM
                                PUTE STATISTICS
                                  STORAGE(INITIAL 65536 NEXT 1048576 MIN
                                EXTENTS 1 MAXEXTENTS 2147483645
                                  PCTINCREASE 0 FREELISTS 1 FREELIST GRO
                                UPS 1
                                  BUFFER_POOL DEFAULT)
                                  TABLESPACE "EXAMPLE"
TMP$$_ORDERS_SALES_R ORDERS     ALTER TABLE "OE"."INT_ORDERS" ADD CONSTR
EP_FK0                          AINT "TMP$$_TMP$$_ORDERS_SALES_RE0" FORE
                                IGN KEY ("SALES_REP_ID")
                                          REFERENCES "HR"."INT_EMPLOYEES"
                                ("EMP
                                LOYEE_ID") ON DELETE SET NULL DISABLE
```

If necessary, correct the errors reported in the output.

In this example, original table has an index and a foreign key constraint on the `sales_rep_id` column. The index and the constraint could not be copied to the interim table because the name of the column changed from `sales_rep_id` to `sales_pid`.

To correct the problems, add the index and the constraint on the interim table by completing the following steps:

a. Add the index:

```
ALTER TABLE oe.int_orders
  ADD (CONSTRAINT orders_sales_pid_fk
      FOREIGN KEY (sales_pid)
      REFERENCES hr.employees(employee_id)
      ON DELETE SET NULL);
```

b. Add the foreign key constraint:

```
CREATE INDEX ord_sales_pid_ix ON oe.int_orders (sales_pid);
```

8. (Optional) Synchronize the interim table `oe.int_orders`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => 'oe',
    orig_table => 'orders',
    int_table  => 'int_orders');
END;
/
```

9. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => 'oe',
    orig_table => 'orders',
    int_table  => 'int_orders');
END;
/
```

   The table `oe.orders` is locked in the exclusive mode only for a small window toward the end of this step. After this call the table `oe.orders` is redefined such that it has all the attributes of the `oe.int_orders` table.

   Consider specifying a non-`NULL` value for the `dml_lock_timeout` parameter in this procedure. See step 8 in "Performing Online Redefinition with Multiple Procedures in DBMS_REDEFINITION" for more information.

10. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 8

This example illustrates making multiple changes to a table using online redefinition.

The table to be redefined is defined as follows:

```
CREATE TABLE testredef.original(
   col1 NUMBER PRIMARY KEY,
   col2 VARCHAR2(10),
   col3 CLOB,
   col4 DATE)
ORGANIZATION INDEX;
```

The table is redefined as follows:

- The table is compressed with advanced row compression.
- The LOB column is changed to SecureFiles LOB storage.
- The table's tablespace is changed from `example` to `testredeftbs`, and the table's block size is changed from 8KB to 16KB.

  This example assumes that the database block size is 8KB. This example also assumes that the `DB_16K_CACHE_SIZE` initialization parameter is set and that the `testredef` tablespace was created with a 16KB block size. For example:

```
CREATE TABLESPACE testredeftbs
  DATAFILE '/u01/app/oracle/oradata/testredef01.dbf' SIZE 500M   EXTENT MANAGEMENT
LOCAL AUTOALLOCATE
  SEGMENT SPACE MANAGEMENT AUTO
  BLOCKSIZE 16384;
```

- The table is partitioned on the `col1` column.
- The `col5` column is added.

- The col2 column is dropped.
- Columns col3 and col4 are renamed, and their position in the table is changed.
- The type of the col3 column is changed from DATE to TIMESTAMP.
- The table is changed from an index-organized table (IOT) to a heap-organized table.
- The table is defragmented.

  To demonstrate defragmentation, the table must be populated. For the purposes of this example, you can use this PL/SQL block to populate the table:

```
DECLARE
  V_CLOB CLOB;
BEGIN
   FOR I IN 0..999 LOOP
      V_CLOB := NULL;
      FOR J IN 1..1000 LOOP
         V_CLOB := V_CLOB||TO_CHAR(I,'0000');
      END LOOP;
      INSERT INTO testredef.original VALUES(I,TO_CHAR(I),V_CLOB,SYSDATE+I);
      COMMIT;
   END LOOP;
   COMMIT;
END;
/
```

  Run the following SQL statement to fragment the table by deleting every third row:

```
DELETE FROM testredef.original WHERE (COL1/3) <> TRUNC(COL1/3);
```

  You can confirm the fragmentation by using the DBMS_SPACE.SPACE_USAGE procedure.

  See Also:

  Oracle Database PL/SQL Packages and Types Reference for more information about the DBMS_SPACE.SPACE_USAGE procedure

The steps in this redefinition are illustrated below.

1. In SQL*Plus, connect as a user with the required privileges for performing online redefinition of a table.

   Specifically, the user must have the privileges described in "Privileges Required for the DBMS_REDEFINITION Package".

   See "Connecting to the Database with SQL*Plus".

2. Verify that the table is a candidate for online redefinition. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

```
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
    uname        => 'testredef',
    tname        => 'original',
    options_flag => DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

3. Create an interim table testredef.interim.

```
CREATE TABLE testredef.interim(
    col1 NUMBER,
    col3 TIMESTAMP,
    col4 CLOB,
    col5 VARCHAR2(3))
    LOB(col4) STORE AS SECUREFILE (NOCACHE FILESYSTEM_LIKE_LOGGING)
    PARTITION BY RANGE (COL1) (
       PARTITION par1 VALUES LESS THAN (333),
       PARTITION par2 VALUES LESS THAN (666),
       PARTITION par3 VALUES LESS THAN (MAXVALUE))
   TABLESPACE testredeftbs
   ROW STORE COMPRESS ADVANCED;
```

4. Start the redefinition process.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
     uname        => 'testredef',
     orig_table   => 'original',
     int_table    => 'interim',
     col_mapping  => 'col1 col1, TO_TIMESTAMP(col4) col3, col3 col4',
     options_flag => DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

5. Copy the dependent objects.

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname             => 'testredef',
    orig_table        => 'original',
    int_table         => 'interim',
    copy_indexes      => DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers     => TRUE,
    copy_constraints  => TRUE,
    copy_privileges   => TRUE,
    ignore_errors     => TRUE,
    num_errors        => num_errors);
END;
/
```

6. Optionally synchronize the interim table.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname       => 'testredef',
    orig_table  => 'original',
    int_table   => 'interim');
END;
/
```

7. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname       => 'testredef',
    orig_table  => 'original',
    int_table   => 'interim');
END;
/
```

See Also:

Oracle Database Sample Schemas

19.9 Researching and Reversing Erroneous Table Changes

To enable you to research and reverse erroneous changes to tables, Oracle Database provides a group of features that you can use to view past states of database objects or to return database objects to a previous state without using point-in-time media recovery. These features are known as **Oracle Flashback features**.

To research an erroneous change, you can use multiple Oracle Flashback queries to view row data at specific points in time. A more efficient approach would be to use Oracle Flashback Version Query to view all changes to a row over a period of time. With this feature, you append a `VERSIONS` clause to a `SELECT` statement that specifies a system change number (SCN) or timestamp range between which you want to view changes to row values. The query also can return associated metadata, such as the transaction responsible for the change.

After you identify an erroneous transaction, you can use Oracle Flashback Transaction Query to identify other changes that were made by the transaction. You can then use Oracle Flashback Transaction to reverse the erroneous transaction. (Note that Oracle Flashback Transaction must also reverse all dependent transactions—subsequent transactions involving the same rows as the erroneous transaction.) You also have the option of using Oracle Flashback Table, described in "Recovering Tables Using Oracle Flashback Table".

Note:

You must be using automatic undo management to use Oracle Flashback features. See "Introduction to Automatic Undo Management ".

See Also:

Oracle Database Development Guide for information about Oracle Flashback features.

19.10 Recovering Tables Using Oracle Flashback Table

Oracle Flashback Table enables you to restore a table to its state as of a previous point in time.

It provides a fast, online solution for recovering a table that has been accidentally modified or deleted by a user or application. In many cases, Oracle Flashback Table eliminates the need for you to perform more complicated point-in-time recovery operations.

Oracle Flashback Table:

- Restores all data in a specified table to a previous point in time described by a timestamp or SCN.
- Performs the restore operation online.

- Automatically maintains all of the table attributes, such as indexes, triggers, and constraints that are necessary for an application to function with the flashed-back table.
- Maintains any remote state in a distributed environment. For example, all of the table modifications required by replication if a replicated table is flashed back.
- Maintains data integrity as specified by constraints. Tables are flashed back provided none of the table constraints are violated. This includes any referential integrity constraints specified between a table included in the `FLASHBACK TABLE` statement and another table that is not included in the `FLASHBACK TABLE` statement.
- Even after a flashback operation, the data in the original table is not lost. You can later revert to the original state.

Note:

You must be using automatic undo management to use Oracle Flashback Table. See "Introduction to Automatic Undo Management ".

See Also:

Oracle Database Backup and Recovery User's Guide for more information about the `FLASHBACK TABLE` statement.

19.11 Dropping Tables

To drop a table that you no longer need, use the `DROP TABLE` statement.

The table must be contained in your schema or you must have the `DROP ANY TABLE` system privilege.

Note:

Before dropping a table, familiarize yourself with the consequences of doing so:

- Dropping a table removes the table definition from the data dictionary. All rows of the table are no longer accessible.
- All indexes and triggers associated with a table are dropped.
- All views and PL/SQL program units dependent on a dropped table remain, yet become invalid (not usable). See "Managing Object Dependencies" for information about how the database manages dependencies.
- All synonyms for a dropped table remain, but return an error when used.
- All extents allocated for a table that is dropped are returned to the free space of the tablespace and can be used by any other object requiring new extents or new objects. All rows corresponding to a clustered table are deleted from the blocks of the cluster. Clustered tables are the subject of Managing Clusters.

The following statement drops the `hr.int_admin_emp` table:

```
DROP TABLE hr.int_admin_emp;
```

If the table to be dropped contains any primary or unique keys referenced by foreign keys of other tables and you intend to drop the `FOREIGN KEY` constraints of the child tables, then include the `CASCADE` clause in the `DROP TABLE` statement, as shown below:

```
DROP TABLE hr.admin_emp CASCADE CONSTRAINTS;
```

When you drop a table, normally the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the `FLASHBACK TABLE` statement if you find that you dropped the table in error. If you should want to immediately release the space associated with the table at the time you issue the `DROP TABLE` statement, include the `PURGE` clause as shown in the following statement:

```
DROP TABLE hr.admin_emp PURGE;
```

Perhaps instead of dropping a table, you want to truncate it. The `TRUNCATE` statement provides a fast, efficient method for deleting all rows from a table, but it does not affect any structures associated with the table being truncated (column definitions, constraints, triggers, and so forth) or authorizations. The `TRUNCATE` statement is discussed in "Truncating Tables and Clusters".

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating and Modifying Tables*.

19.12 Using Flashback Drop and Managing the Recycle Bin

When you drop a table, the database does not immediately remove the space associated with the table. The database renames the table and places it and any associated objects in a recycle bin, where, in case the table was dropped in error, it can be recovered at a later time. This feature is called Flashback Drop, and the `FLASHBACK TABLE` statement is used to restore the table.

Before discussing the use of the `FLASHBACK TABLE` statement for this purpose, it is important to understand how the recycle bin works, and how you manage its contents.

- What Is the Recycle Bin?
  The recycle bin is actually a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and so on are not removed and still occupy space.
- Enabling and Disabling the Recycle Bin
  When the recycle bin is enabled, dropped tables and their dependent objects are placed in the recycle bin. When the recycle bin is disabled, dropped tables and their dependent objects are *not* placed in the recycle bin; they are dropped, and you must use other means to recover them (such as recovering from backup).
- Viewing and Querying Objects in the Recycle Bin
  Oracle Database provides two views for obtaining information about objects in the recycle bin.

- Purging Objects in the Recycle Bin
  If you decide that you are never going to restore an item from the recycle bin, then you can use the `PURGE` statement to remove the items and their associated objects from the recycle bin and release their storage space. You need the same privileges as if you were dropping the item.
- Restoring Tables from the Recycle Bin
  Use the `FLASHBACK TABLE` ... `TO BEFORE DROP` statement to recover objects from the recycle bin.

19.12.1 What Is the Recycle Bin?

The recycle bin is actually a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and so on are not removed and still occupy space.

They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

Each user can be thought of as having his own recycle bin, because, unless a user has the `SYSDBA` privilege, the only objects that the user has access to in the recycle bin are those that the user owns. A user can view his objects in the recycle bin using the following statement:

```
SELECT * FROM RECYCLEBIN;
```

Only the `DROP TABLE` SQL statement places objects in the recycle bin. It adds the table and its associated objects so that they can be recovered as a group. In addition to the table itself, the associated objects that are added to the recycle bin can include the following types of objects:

- Nested tables
- LOB segments
- Indexes
- Constraints (excluding foreign key constraints)
- Triggers
- Clusters

When you drop a tablespace including its contents, the objects in the tablespace are not placed in the recycle bin and the database purges any entries in the recycle bin for objects located in the tablespace. The database also purges any recycle bin entries for objects in a tablespace when you drop the tablespace, not including contents, and the tablespace is otherwise empty. Likewise:

- When you drop a user, any objects belonging to the user are not placed in the recycle bin and any objects in the recycle bin are purged.
- When you drop a cluster, its member tables are not placed in the recycle bin and any former member tables in the recycle bin are purged.
- When you drop a type, any dependent objects such as subtypes are not placed in the recycle bin and any former dependent objects in the recycle bin are purged.

Object Naming in the Recycle Bin

When a dropped table is moved to the recycle bin, the table and its associated objects are given system-generated names. This is necessary to avoid name conflicts that may arise if multiple tables have the same name. This could occur under the following circumstances:

- A user drops a table, re-creates it with the same name, then drops it again.
- Two users have tables with the same name, and both users drop their tables.

The renaming convention is as follows:

```
BIN$unique_id$version
```

where:

- *unique_id* is a 26-character globally unique identifier for this object, which makes the recycle bin name unique across all databases
- *version* is a version number assigned by the database

19.12.2 Enabling and Disabling the Recycle Bin

When the recycle bin is enabled, dropped tables and their dependent objects are placed in the recycle bin. When the recycle bin is disabled, dropped tables and their dependent objects are *not* placed in the recycle bin; they are dropped, and you must use other means to recover them (such as recovering from backup).

Disabling the recycle bin does not purge or otherwise affect objects already in the recycle bin. The recycle bin is enabled by default.

You enable and disable the recycle bin by changing the `recyclebin` initialization parameter. This parameter is not dynamic, so a database restart is required when you change it with an `ALTER SYSTEM` statement.

To enable the recycle bin:

1. Issue one of the following statements:

   ```
   ALTER SESSION SET recyclebin = ON;

   ALTER SYSTEM SET recyclebin = ON SCOPE = SPFILE;
   ```

2. If you used `ALTER SYSTEM`, restart the database.

To disable the recycle bin:

1. Issue one of the following statements:

   ```
   ALTER SESSION SET recyclebin = OFF;

   ALTER SYSTEM SET recyclebin = OFF SCOPE = SPFILE;
   ```

2. If you used `ALTER SYSTEM`, restart the database.

See Also:

Oracle Multitenant Administrator's Guide for a description of dynamic and static initialization parameters

19.12.3 Viewing and Querying Objects in the Recycle Bin

Oracle Database provides two views for obtaining information about objects in the recycle bin.

View	Description
USER_RECYCLEBIN	This view can be used by users to see their own dropped objects in the recycle bin. It has a synonym RECYCLEBIN, for ease of use.
DBA_RECYCLEBIN	This view gives administrators visibility to all dropped objects in the recycle bin

One use for these views is to identify the name that the database has assigned to a dropped object, as shown in the following example:

```
SELECT object_name, original_name FROM dba_recyclebin
   WHERE owner = 'HR';

OBJECT_NAME                    ORIGINAL_NAME
------------------------------ --------------------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 EMPLOYEES
```

You can also view the contents of the recycle bin using the SQL*Plus command SHOW RECYCLEBIN.

```
SQL> show recyclebin

ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
EMPLOYEES        BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TABLE        2003-10-27:14:00:19
```

You can query objects that are in the recycle bin, just as you can query other objects. However, you must specify the name of the object as it is identified in the recycle bin. For example:

```
SELECT * FROM "BIN$yrMKlZaVMhfgNAgAIMenRA==$0";
```

19.12.4 Purging Objects in the Recycle Bin

If you decide that you are never going to restore an item from the recycle bin, then you can use the PURGE statement to remove the items and their associated objects from the recycle bin and release their storage space. You need the same privileges as if you were dropping the item.

When you use the PURGE statement to purge a table, you can use the name that the table is known by in the recycle bin or the original name of the table. The recycle bin name can be obtained from either the DBA_ or USER_RECYCLEBIN view as shown in "Viewing and Querying Objects in the Recycle Bin". The following hypothetical example purges the table hr.int_admin_emp, which was renamed to BIN$jsleilx392mk2=293$0 when it was placed in the recycle bin:

```
PURGE TABLE "BIN$jsleilx392mk2=293$0";
```

You can achieve the same result with the following statement:

```
PURGE TABLE int_admin_emp;
```

You can use the PURGE statement to purge all the objects in the recycle bin that are from a specified tablespace or only the tablespace objects belonging to a specified user, as shown in the following examples:

```
PURGE TABLESPACE example;
PURGE TABLESPACE example USER oe;
```

Users can purge the recycle bin of their own objects, and release space for objects, by using the following statement:

```
PURGE RECYCLEBIN;
```

If you have the `SYSDBA` privilege or the `PURGE DBA_RECYCLEBIN` system privilege, then you can purge the entire recycle bin by specifying `DBA_RECYCLEBIN`, instead of `RECYCLEBIN` in the previous statement.

You can also use the `PURGE` statement to purge an index from the recycle bin or to purge from the recycle bin all objects in a specified tablespace.

See Also:

Oracle Database SQL Language Reference for more information on the `PURGE` statement

19.12.5 Restoring Tables from the Recycle Bin

Use the `FLASHBACK TABLE ... TO BEFORE DROP` statement to recover objects from the recycle bin.

You can specify either the name of the table in the recycle bin or the original table name. An optional `RENAME TO` clause lets you rename the table as you recover it. The recycle bin name can be obtained from either the `DBA_` or `USER_RECYCLEBIN` view as shown in "Viewing and Querying Objects in the Recycle Bin". To use the `FLASHBACK TABLE ... TO BEFORE DROP` statement, you need the same privileges required to drop the table.

The following example restores `int_admin_emp` table and assigns to it a new name:

```
FLASHBACK TABLE int_admin_emp TO BEFORE DROP
   RENAME TO int2_admin_emp;
```

The system-generated recycle bin name is very useful if you have dropped a table multiple times. For example, suppose you have three versions of the `int2_admin_emp` table in the recycle bin and you want to recover the second version. You can do this by issuing two `FLASHBACK TABLE` statements, or you can query the recycle bin and then flashback to the appropriate system-generated name, as shown in the following example. Including the create time in the query can help you verify that you are restoring the correct table.

```
SELECT object_name, original_name, createtime FROM recyclebin;

OBJECT_NAME                    ORIGINAL_NAME   CREATETIME
------------------------------ --------------- -------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:21:05:52
BIN$yrMKlZaVMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:21:25:13
BIN$yrMKlZaQMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:22:05:53

FLASHBACK TABLE "BIN$yrMKlZaVMhfgNAgAIMenRA==$0" TO BEFORE DROP;
```

Restoring Dependent Objects

When you restore a table from the recycle bin, dependent objects such as indexes do not get their original names back; they retain their system-generated recycle bin names. You must manually rename dependent objects to restore their original names. If you plan to manually restore original names for dependent objects, ensure that you make note of each dependent object's system-generated recycle bin name *before* you restore the table.

The following is an example of restoring the original names of some of the indexes of the dropped table `JOB_HISTORY`, from the `HR` sample schema. The example assumes that you are logged in as the `HR` user.

1. After dropping `JOB_HISTORY` and before restoring it from the recycle bin, run the following query:

   ```
   SELECT OBJECT_NAME, ORIGINAL_NAME, TYPE FROM RECYCLEBIN;

   OBJECT_NAME                    ORIGINAL_NAME             TYPE
   ------------------------------ ------------------------- --------
   BIN$DBo9UChtZSbgQFeMiAdCcQ==$0 JHIST_JOB_IX              INDEX
   BIN$DBo9UChuZSbgQFeMiAdCcQ==$0 JHIST_EMPLOYEE_IX         INDEX
   BIN$DBo9UChvZSbgQFeMiAdCcQ==$0 JHIST_DEPARTMENT_IX       INDEX
   BIN$DBo9UChwZSbgQFeMiAdCcQ==$0 JHIST_EMP_ID_ST_DATE_PK   INDEX
   BIN$DBo9UChxZSbgQFeMiAdCcQ==$0 JOB_HISTORY               TABLE
   ```

2. Restore the table with the following command:

   ```
   FLASHBACK TABLE JOB_HISTORY TO BEFORE DROP;
   ```

3. Run the following query to verify that all `JOB_HISTORY` indexes retained their system-generated recycle bin names:

   ```
   SELECT INDEX_NAME FROM USER_INDEXES WHERE TABLE_NAME = 'JOB_HISTORY';

   INDEX_NAME
   ------------------------------
   BIN$DBo9UChwZSbgQFeMiAdCcQ==$0
   BIN$DBo9UChtZSbgQFeMiAdCcQ==$0
   BIN$DBo9UChuZSbgQFeMiAdCcQ==$0
   BIN$DBo9UChvZSbgQFeMiAdCcQ==$0
   ```

4. Restore the original names of the first two indexes as follows:

   ```
   ALTER INDEX "BIN$DBo9UChtZSbgQFeMiAdCcQ==$0" RENAME TO JHIST_JOB_IX;
   ALTER INDEX "BIN$DBo9UChuZSbgQFeMiAdCcQ==$0" RENAME TO JHIST_EMPLOYEE_IX;
   ```

   Note that double quotes are required around the system-generated names.

19.13 Managing Index-Organized Tables

An index-organized table's storage organization is a variant of a primary B-tree index. Unlike a heap-organized table, data is stored in primary key order.

- What Are Index-Organized Tables?
  An **index-organized table** has a storage organization that is a variant of a primary B-tree. Unlike an ordinary (heap-organized) table whose data is stored as an unordered collection (heap), data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Each leaf block in the index structure stores both the key and nonkey columns.
- Creating Index-Organized Tables
  Index-organized tables provide fast primary key access and high availability.
- Maintaining Index-Organized Tables
  Index-organized tables differ from ordinary tables only in physical organization. Logically, they are manipulated in the same manner as ordinary tables. You can specify an index-

organized table just as you would specify a regular table in `INSERT`, `SELECT`, `DELETE`, and `UPDATE` statements.

- Creating Secondary Indexes on Index-Organized Tables
  A secondary index is an index on an index-organized table. The secondary index is an independent schema object and is stored separately from the index-organized table.
- Analyzing Index-Organized Tables
  Just like ordinary tables, index-organized tables are analyzed using the `DBMS_STATS` package, or the `ANALYZE` statement.
- Using the ORDER BY Clause with Index-Organized Tables
  If an `ORDER BY` clause only references the primary key column or a prefix of it, then the optimizer avoids the sorting overhead, as the rows are returned sorted on the primary key columns.
- Converting Index-Organized Tables to Regular Tables
  You can convert index-organized tables to regular (heap organized) tables using the Oracle import or export utilities, or the `CREATE TABLE...AS SELECT` statement.

19.13.1 What Are Index-Organized Tables?

An **index-organized table** has a storage organization that is a variant of a primary B-tree. Unlike an ordinary (heap-organized) table whose data is stored as an unordered collection (heap), data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Each leaf block in the index structure stores both the key and nonkey columns.

The structure of an index-organized table provides the following benefits:

- Fast random access on the primary key because an index-only scan is sufficient. And, because there is no separate table storage area, changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure.
- Fast range access on the primary key because the rows are clustered in primary key order.
- Lower storage requirements because duplication of primary keys is avoided. They are not stored both in the index and underlying table, as is true with heap-organized tables.

Index-organized tables have full table functionality. They support features such as constraints, triggers, LOB and object columns, partitioning, parallel operations, online reorganization, and replication. And, they offer these additional features:

- Prefix compression
- Overflow storage area and specific column placement
- Secondary indexes, including bitmap indexes.

Index-organized tables are ideal for OLTP applications, which require fast primary key access and high availability. For example, queries and DML on an orders table used in electronic order processing are predominantly based on primary key access, and heavy volume of concurrent DML can cause row chaining and inefficient space usage in indexes, resulting in a frequent need to reorganize. Because an index-organized table can be reorganized online and without invalidating its secondary indexes, the window of unavailability is greatly reduced or eliminated.

Index-organized tables are suitable for modeling application-specific index structures. For example, content-based information retrieval applications containing text, image and audio data require inverted indexes that can be effectively modeled using index-organized tables. A fundamental component of an internet search engine is an inverted index that can be modeled using index-organized tables.

These are but a few of the applications for index-organized tables.

See Also:

- *Oracle Database Concepts* for a more thorough description of index-organized tables
- *Oracle Database VLDB and Partitioning Guide* for information about partitioning index-organized tables

19.13.2 Creating Index-Organized Tables

Index-organized tables provide fast primary key access and high availability.

- About Creating Index-Organized Tables
  You use the `CREATE TABLE` statement to create index-organized tables.
- Example: Creating an Index-Organized Table
  An example illustrates creating an index-organized table.
- Restrictions for Index-Organized Tables
  Several restrictions apply when you are creating an index-organized table.
- Creating Index-Organized Tables That Contain Object Types
  Index-organized tables can store object types.
- Choosing and Monitoring a Threshold Value
  Choose a threshold value that can accommodate your key columns, as well as the first few nonkey columns (if they are frequently accessed).
- Using the INCLUDING Clause
  In addition to specifying `PCTTHRESHOLD`, you can use the `INCLUDING` clause to control which nonkey columns are stored with the key columns in an index-organized table.
- Parallelizing Index-Organized Table Creation
  The `CREATE TABLE...AS SELECT` statement enables you to create an index-organized table and load data from an existing table into it. By including the `PARALLEL` clause, the load can be done in parallel.
- Using Prefix Compression
  Creating an index-organized table using prefix compression (also known as key compression) enables you to eliminate repeated occurrences of key column prefix values.

19.13.2.1 About Creating Index-Organized Tables

You use the `CREATE TABLE` statement to create index-organized tables.

When you create an index-organized table, but you must provide additional information:

- An `ORGANIZATION INDEX` qualifier, which indicates that this is an index-organized table
- A primary key, specified through a column constraint clause (for a single column primary key) or a table constraint clause (for a multiple-column primary key).

Optionally, you can specify the following:

- An OVERFLOW clause, which preserves dense clustering of the B-tree index by enabling the storage of some of the nonkey columns in a separate overflow data segment.
- A PCTTHRESHOLD value, which, when an overflow segment is being used, defines the maximum size of the portion of the row that is stored in the index block, as a percentage of block size. Rows columns that would cause the row size to exceed this maximum are stored in the overflow segment. The row is broken at a column boundary into two pieces, a head piece and tail piece. The head piece fits in the specified threshold and is stored along with the key in the index leaf block. The tail piece is stored in the overflow area as one or more row pieces. Thus, the index entry contains the key value, the nonkey column values that fit the specified threshold, and a pointer to the rest of the row.
- An INCLUDING clause, which can be used to specify the nonkey columns that are to be stored in the index block with the primary key.

19.13.2.2 Example: Creating an Index-Organized Table

An example illustrates creating an index-organized table.

The following statement creates an index-organized table:

```
CREATE TABLE admin_docindex(
        token char(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(2000),
        CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    OVERFLOW TABLESPACE admin_tbs2;
```

This example creates an index-organized table named admin_docindex, with a primary key composed of the columns token and doc_id. The OVERFLOW and PCTTHRESHOLD clauses specify that if the length of a row exceeds 20% of the index block size, then the column that exceeded that threshold and all columns after it are moved to the overflow segment. The overflow segment is stored in the admin_tbs2 tablespace.

See Also:

Oracle Database SQL Language Reference for more information about the syntax to create an index-organized table

19.13.2.3 Restrictions for Index-Organized Tables

Several restrictions apply when you are creating an index-organized table.

The following are restrictions on creating index-organized tables.

- The maximum number of columns is 1000.
- The maximum number of columns in the index portion of a row is 255, including both key and nonkey columns. If more than 255 columns are required, you must use an overflow segment.
- The maximum number of columns that you can include in the primary key is 32.

- PCTTHRESHOLD must be in the range of 1–50. The default is 50.
- All key columns must fit within the specified threshold.
- If the maximum size of a row exceeds 50% of the index block size and you do not specify an overflow segment, the CREATE TABLE statement fails.
- Index-organized tables cannot have virtual columns.
- When a table has a foreign key, and the parent of the foreign key is an index-organized table, a session that updates a row that contains the foreign key can hang when another session is updating a non-key column in the parent table.

  For example, consider a scenario in which a departments table is an index-organized table, and department_id is its primary key. There is an employees table with a department_id column that is a foreign key of the departments table. Assume a session is updating the department_name in a row in the departments table for which the department_id is 20 while another session is updating a row in the employees table for which the department_id is 20. In this case, the session updating the employees table can hang until the session updating the departments table commits or rolls back.
- Index-organized tables that contain one or more LOB columns cannot be moved in parallel.

19.13.2.4 Creating Index-Organized Tables That Contain Object Types

Index-organized tables can store object types.

The following example creates object type admin_typ, then creates an index-organized table containing a column of object type admin_typ:

```
CREATE OR REPLACE TYPE admin_typ AS OBJECT
    (col1 NUMBER, col2 VARCHAR2(6));
CREATE TABLE admin_iot (c1 NUMBER primary key, c2 admin_typ)
    ORGANIZATION INDEX;
```

You can also create an index-organized table of object types. For example:

```
CREATE TABLE admin_iot2 OF admin_typ (col1 PRIMARY KEY)
    ORGANIZATION INDEX;
```

Another example, that follows, shows that index-organized tables store nested tables efficiently. For a nested table column, the database internally creates a storage table to hold all the nested table rows.

```
CREATE TYPE project_t AS OBJECT(pno NUMBER, pname VARCHAR2(80));
/
CREATE TYPE project_set AS TABLE OF project_t;
/
CREATE TABLE proj_tab (eno NUMBER, projects PROJECT_SET)
    NESTED TABLE projects STORE AS emp_project_tab
                ((PRIMARY KEY(nested_table_id, pno))
    ORGANIZATION INDEX)
    RETURN AS LOCATOR;
```

The rows belonging to a single nested table instance are identified by a nested_table_id column. If an ordinary table is used to store nested table columns, the nested table rows typically get de-clustered. But when you use an index-organized table, the nested table rows can be clustered based on the nested_table_id column.

See Also:

- *Oracle Database SQL Language Reference* for details of the syntax used for creating index-organized tables
- *Oracle Database VLDB and Partitioning Guide* for information about creating partitioned index-organized tables
- *Oracle Database Object-Relational Developer's Guide* for information about object types

19.13.2.5 Choosing and Monitoring a Threshold Value

Choose a threshold value that can accommodate your key columns, as well as the first few nonkey columns (if they are frequently accessed).

After choosing a threshold value, you can monitor tables to verify that the value you specified is appropriate. You can use the `ANALYZE TABLE ... LIST CHAINED ROWS` statement to determine the number and identity of rows exceeding the threshold value.

See Also:

- "Listing Chained Rows of Tables and Clusters" for more information about chained rows
- *Oracle Database SQL Language Reference* for syntax of the `ANALYZE` statement

19.13.2.6 Using the INCLUDING Clause

In addition to specifying `PCTTHRESHOLD`, you can use the `INCLUDING` clause to control which nonkey columns are stored with the key columns in an index-organized table.

The database accommodates all nonkey columns up to and including the column specified in the `INCLUDING` clause in the index leaf block, provided it does not exceed the specified threshold. All nonkey columns beyond the column specified in the `INCLUDING` clause are stored in the overflow segment. If the `INCLUDING` and `PCTTHRESHOLD` clauses conflict, `PCTTHRESHOLD` takes precedence.

Note:

Oracle Database moves all primary key columns of an indexed-organized table to the beginning of the table (in their key order) to provide efficient primary key–based access. As an example:

```
CREATE TABLE admin_iot4(a INT, b INT, c INT, d INT,
                primary key(c,b))
    ORGANIZATION INDEX;
```

The stored column order is: `c b a d` (instead of: `a b c d`). The last primary key column is `b`, based on the stored column order. The `INCLUDING` column can be the last primary key column (`b` in this example), or any nonkey column (that is, any column after `b` in the stored column order).

The following `CREATE TABLE` statement is similar to the one shown earlier in "Example: Creating an Index-Organized Table" but is modified to create an index-organized table where the `token_offsets` column value is always stored in the overflow area:

```
CREATE TABLE admin_docindex2(
        token CHAR(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(2000),
        CONSTRAINT pk_admin_docindex2 PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    INCLUDING token_frequency
    OVERFLOW TABLESPACE admin_tbs2;
```

Here, only nonkey columns before `token_offsets` (in this case a single column only) are stored with the key column values in the index leaf block.

19.13.2.7 Parallelizing Index-Organized Table Creation

The `CREATE TABLE...AS SELECT` statement enables you to create an index-organized table and load data from an existing table into it. By including the `PARALLEL` clause, the load can be done in parallel.

The following statement creates an index-organized table in parallel by selecting rows from the conventional table `hr.jobs`:

```
CREATE TABLE admin_iot3(i PRIMARY KEY, j, k, l)
     ORGANIZATION INDEX
     PARALLEL
     AS SELECT * FROM hr.jobs;
```

This statement provides an alternative to parallel bulk-load using SQL*Loader.

19.13.2.8 Using Prefix Compression

Creating an index-organized table using prefix compression (also known as key compression) enables you to eliminate repeated occurrences of key column prefix values.

Prefix compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys in each index block while improving performance.

You can enable prefix compression using the `COMPRESS` clause while:

- Creating an index-organized table
- Moving an index-organized table

You can also specify the prefix length (as the number of key columns), which identifies how the key columns are broken into a prefix and suffix entry.

```
CREATE TABLE admin_iot5(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS;
```

The preceding statement is equivalent to the following statement:

```
CREATE TABLE admin_iot6(i INT, j INT, k INT, l INT, PRIMARY KEY(i, j, k))
    ORGANIZATION INDEX COMPRESS 2;
```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4) the repeated occurrences of (1,2), (1,3) are compressed away.

You can also override the default prefix length used for compression as follows:

```
CREATE TABLE admin_iot7(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS 1;
```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4), the repeated occurrences of 1 are compressed away.

You can disable compression as follows:

```
ALTER TABLE admin_iot5 MOVE NOCOMPRESS;
```

One application of prefix compression is in a time-series application that uses a set of time-stamped rows belonging to a single item, such as a stock price. Index-organized tables are attractive for such applications because of the ability to cluster rows based on the primary key. By defining an index-organized table with primary key (stock symbol, time stamp), you can store and manipulate time-series data efficiently. You can achieve more storage savings by compressing repeated occurrences of the item identifier (for example, the stock symbol) in a time series by using an index-organized table with prefix compression.

See Also:

Oracle Database Concepts for more information about prefix compression

19.13.3 Maintaining Index-Organized Tables

Index-organized tables differ from ordinary tables only in physical organization. Logically, they are manipulated in the same manner as ordinary tables. You can specify an index-organized table just as you would specify a regular table in `INSERT`, `SELECT`, `DELETE`, and `UPDATE` statements.

- Altering Index-Organized Tables
  All of the alter options available for ordinary tables are available for index-organized tables. This includes `ADD`, `MODIFY`, and `DROP COLUMNS` and `CONSTRAINTS`. However, the primary key constraint for an index-organized table cannot be dropped, deferred, or disabled.
- Moving (Rebuilding) Index-Organized Tables
  Because index-organized tables are primarily stored in a B-tree index, you can encounter fragmentation as a consequence of incremental updates. However, you can use the `ALTER TABLE...MOVE` statement to rebuild the index and reduce this fragmentation.

19.13.3.1 Altering Index-Organized Tables

All of the alter options available for ordinary tables are available for index-organized tables. This includes `ADD`, `MODIFY`, and `DROP COLUMNS` and `CONSTRAINTS`. However, the primary key constraint for an index-organized table cannot be dropped, deferred, or disabled.

You can use the `ALTER TABLE` statement to modify physical and storage attributes for both primary key index and overflow data segments. All the attributes specified before the `OVERFLOW` keyword are applicable to the primary key index segment. All attributes specified after the `OVERFLOW` key word are applicable to the overflow data segment. For example, you can set the `INITRANS` of the primary key index segment to 4 and the overflow of the data segment `INITRANS` to 6 as follows:

```
ALTER TABLE admin_docindex INITRANS 4 OVERFLOW INITRANS 6;
```

You can also alter `PCTTHRESHOLD` and `INCLUDING` column values. A new setting is used to break the row into head and overflow tail pieces during subsequent operations. For example, the `PCTHRESHOLD` and `INCLUDING` column values can be altered for the `admin_docindex` table as follows:

```
ALTER TABLE admin_docindex PCTTHRESHOLD 15 INCLUDING doc_id;
```

By setting the `INCLUDING` column to `doc_id`, all the columns that follow `token_frequency` and `token_offsets`, are stored in the overflow data segment.

For index-organized tables created without an overflow data segment, you can add an overflow data segment by using the `ADD OVERFLOW` clause. For example, you can add an overflow segment to table `admin_iot3` as follows:

```
ALTER TABLE admin_iot3 ADD OVERFLOW TABLESPACE admin_tbs2;
```

19.13.3.2 Moving (Rebuilding) Index-Organized Tables

Because index-organized tables are primarily stored in a B-tree index, you can encounter fragmentation as a consequence of incremental updates. However, you can use the `ALTER TABLE...MOVE` statement to rebuild the index and reduce this fragmentation.

The following statement rebuilds the index-organized table `admin_docindex`:

```
ALTER TABLE admin_docindex MOVE;
```

You can rebuild index-organized tables online using the `ONLINE` keyword. The overflow data segment, if present, is rebuilt when the `OVERFLOW` keyword is specified. For example, to rebuild the `admin_docindex` table but not the overflow data segment, perform a move online as follows:

```
ALTER TABLE admin_docindex MOVE ONLINE;
```

To rebuild the `admin_docindex` table along with its overflow data segment perform the move operation as shown in the following statement. This statement also illustrates moving both the table and overflow data segment to new tablespaces.

```
ALTER TABLE admin_docindex MOVE TABLESPACE admin_tbs2
    OVERFLOW TABLESPACE admin_tbs3;
```

In this last statement, an index-organized table with a LOB column (CLOB) is created. Later, the table is moved with the `LOB` index and data segment being rebuilt and moved to a new tablespace.

```
CREATE TABLE admin_iot_lob
   (c1 number (6) primary key,
    admin_lob CLOB)
   ORGANIZATION INDEX
   LOB (admin_lob) STORE AS (TABLESPACE admin_tbs2);
.
.
.
ALTER TABLE admin_iot_lob MOVE LOB (admin_lob) STORE AS (TABLESPACE admin_tbs3);
```

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for information about LOBs in index-organized tables

19.13.4 Creating Secondary Indexes on Index-Organized Tables

A secondary index is an index on an index-organized table. The secondary index is an independent schema object and is stored separately from the index-organized table.

- About Secondary Indexes on Index-Organized Tables
  You can create secondary indexes on an index-organized tables to provide multiple access paths.
- Creating a Secondary Index on an Index-Organized Table
  You can create a secondary index on an index-organized table.
- Maintaining Physical Guesses in Logical Rowids
  A logical rowid can include a guess, which identifies the block location of a row at the time the guess is made. Instead of doing a full key search, the database uses the guess to search the block directly.
- Specifying Bitmap Indexes on Index-Organized Tables
  Bitmap indexes on index-organized tables are supported, provided the index-organized table is created with a mapping table.

19.13.4.1 About Secondary Indexes on Index-Organized Tables

You can create secondary indexes on an index-organized tables to provide multiple access paths.

Secondary indexes on index-organized tables differ from indexes on ordinary tables in two ways:

- They store logical rowids instead of physical rowids. This is necessary because the inherent movability of rows in a B-tree index results in the rows having no permanent physical addresses. If the physical location of a row changes, its logical rowid remains valid. One effect of this is that a table maintenance operation, such as `ALTER TABLE ... MOVE`, does not make the secondary index unusable.
- The logical rowid also includes a physical guess which identifies the database block address at which the row is likely to be found. If the physical guess is correct, a secondary index scan would incur a single additional I/O once the secondary key is found. The performance would be similar to that of a secondary index-scan on an ordinary table.

Unique and non-unique secondary indexes, function-based secondary indexes, and bitmap indexes are supported as secondary indexes on index-organized tables.

19.13.4.2 Creating a Secondary Index on an Index-Organized Table

You can create a secondary index on an index-organized table.

The following statement shows the creation of a secondary index on the `docindex` index-organized table where `doc_id` and `token` are the key columns:

```
CREATE INDEX Doc_id_index on Docindex(Doc_id, Token);
```

This secondary index allows the database to efficiently process a query, such as the following, the involves a predicate on `doc_id`:

```
SELECT Token FROM Docindex WHERE Doc_id = 1;
```

19.13.4.3 Maintaining Physical Guesses in Logical Rowids

A logical rowid can include a guess, which identifies the block location of a row at the time the guess is made. Instead of doing a full key search, the database uses the guess to search the block directly.

However, as new rows are inserted, guesses can become stale. The indexes are still usable through the primary key-component of the logical rowid, but access to rows is slower.

1. Collect index statistics with the `DBMS_STATS` package to monitor the staleness of guesses.

   The database checks whether the existing guesses are still valid and records the percentage of rows with valid guesses in the data dictionary.
2. Query the `PCT_DIRECT_ACCESS` column of the `DBA_INDEXES` view (and related views) to show the statistics related to existing guesses.
3. To obtain fresh guesses, you can rebuild the secondary index.

Rebuilding a secondary index on an index-organized table involves reading the base table, unlike rebuilding an index on an ordinary table.

A quicker, more light weight means of fixing the guesses is to use the ALTER INDEX ... UPDATE BLOCK REFERENCES statement. This statement is performed online, while DML is still allowed on the underlying index-organized table.

After you rebuild a secondary index, or otherwise update the block references in the guesses, collect index statistics again.

19.13.4.4 Specifying Bitmap Indexes on Index-Organized Tables

Bitmap indexes on index-organized tables are supported, provided the index-organized table is created with a mapping table.

This is done by specifying the MAPPING TABLE clause in the CREATE TABLE statement that you use to create the index-organized table, or in an ALTER TABLE statement to add the mapping table later.

See Also:

Oracle Database Concepts for a description of mapping tables

19.13.5 Analyzing Index-Organized Tables

Just like ordinary tables, index-organized tables are analyzed using the DBMS_STATS package, or the ANALYZE statement.

- Collecting Optimizer Statistics for Index-Organized Tables
  To collect optimizer statistics, use the DBMS_STATS package.
- Validating the Structure of Index-Organized Tables
  Use the ANALYZE statement to validate the structure of your index-organized table or to list any chained rows.

19.13.5.1 Collecting Optimizer Statistics for Index-Organized Tables

To collect optimizer statistics, use the DBMS_STATS package.

For example, the following statement gathers statistics for the index-organized countries table in the hr schema:

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('HR','COUNTRIES');
```

The DBMS_STATS package analyzes both the primary key index segment and the overflow data segment, and computes logical as well as physical statistics for the table.

- The logical statistics can be queried using USER_TABLES, ALL_TABLES or DBA_TABLES.
- You can query the physical statistics of the primary key index segment using USER_INDEXES, ALL_INDEXES or DBA_INDEXES (and using the primary key index name). For example, you can obtain the primary key index segment physical statistics for the table admin_docindex as follows:

  ```
  SELECT LAST_ANALYZED, BLEVEL,LEAF_BLOCKS, DISTINCT_KEYS
     FROM DBA_INDEXES WHERE INDEX_NAME= 'PK_ADMIN_DOCINDEX';
  ```

- You can query the physical statistics for the overflow data segment using the `USER_TABLES`, `ALL_TABLES` or `DBA_TABLES`. You can identify the overflow entry by searching for `IOT_TYPE = 'IOT_OVERFLOW'`. For example, you can obtain overflow data segment physical attributes associated with the `admin_docindex` table as follows:

```
SELECT LAST_ANALYZED, NUM_ROWS, BLOCKS, EMPTY_BLOCKS
   FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW'
          and IOT_NAME= 'ADMIN_DOCINDEX';
```

See Also:

- *Oracle Database SQL Tuning Guide* for more information about collecting optimizer statistics
- *Oracle Database PL/SQL Packages and Types Reference* for more information about of the `DBMS_STATS` package

19.13.5.2 Validating the Structure of Index-Organized Tables

Use the `ANALYZE` statement to validate the structure of your index-organized table or to list any chained rows.

These operations are discussed in the following sections located elsewhere in this book:

- "Validating Tables, Indexes, Clusters, and Materialized Views"
- "Listing Chained Rows of Tables and Clusters"

Note:

There are special considerations when listing chained rows for index-organized tables. These are discussed in the *Oracle Database SQL Language Reference*.

19.13.6 Using the ORDER BY Clause with Index-Organized Tables

If an `ORDER BY` clause only references the primary key column or a prefix of it, then the optimizer avoids the sorting overhead, as the rows are returned sorted on the primary key columns.

The following queries avoid sorting overhead because the data is already sorted on the primary key:

```
SELECT * FROM admin_docindex2 ORDER BY token, doc_id;
SELECT * FROM admin_docindex2 ORDER BY token;
```

If, however, you have an `ORDER BY` clause on a suffix of the primary key column or non-primary-key columns, additional sorting is required (assuming no other secondary indexes are defined).

```
SELECT * FROM admin_docindex2 ORDER BY doc_id;
SELECT * FROM admin_docindex2 ORDER BY token_frequency;
```

19.13.7 Converting Index-Organized Tables to Regular Tables

You can convert index-organized tables to regular (heap organized) tables using the Oracle import or export utilities, or the `CREATE TABLE...AS SELECT` statement.

To convert an index-organized table to a regular table:

- Export the index-organized table data using conventional path.
- Create a regular table definition with the same definition.
- Import the index-organized table data, making sure `IGNORE=y` (ensures that object exists error is ignored).

> **Note:**
>
> Before converting an index-organized table to a regular table, be aware that index-organized tables cannot be exported using pre-Oracle8 versions of the Export utility.

> **See Also:**
>
> *Oracle Database Utilities* for more details about using the original `IMP` and `EXP` utilities and the Data Pump import and export utilities

19.14 Managing Partitioned Tables

Partitioned tables enable your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Each partition can have separate physical attributes, such as compression enabled or disabled, type of compression, physical storage settings, and tablespace, thus providing a structure that can be better tuned for availability and performance. In addition, each partition can be managed individually, which can simplify and reduce the time required for backup and administration.

See *Oracle Database VLDB and Partitioning Guide* for more information about managing partitioned tables.

19.15 Managing External Tables

External tables are the tables that do not reside in the database. They reside outside the database, in Object storage or external files, such as operating system files or Hadoop Distributed File System (HDFS) files.

- About External Tables
  Oracle Database allows you read-only access to data in external tables. **External tables** are defined as tables that do not reside in the database, and they can be in any format for which an access driver is provided.
- Creating External Tables
  You create external tables using the `CREATE TABLE` statement with an `ORGANIZATION EXTERNAL` clause. This statement creates only metadata in the data dictionary.

- Altering External Tables
  You can modify an external table with the `ALTER TABLE` statement.
- Preprocessing External Tables
  External tables can be preprocessed by user-supplied preprocessor programs. By using a preprocessing program, users can use data from a file that is not in a format supported by the driver.
- Overriding Parameters for External Tables in a Query
  The `EXTERNAL MODIFY` clause of a `SELECT` statement modifies external table parameters.
- Using Inline External Tables
  Inline external tables enable the runtime definition of an external table as part of a SQL statement, without creating the external table as persistent object in the data dictionary.
- Partitioning External Tables
  For large amounts of data, partitioning for external tables provides fast query performance and enhanced data maintenance.
- Dropping External Tables
  For an external table, the `DROP TABLE` statement removes only the table metadata in the database. It has no affect on the actual data, which resides outside of the database.
- System and Object Privileges for External Tables
  System and object privileges for external tables are a subset of those for regular table.

19.15.1 About External Tables

Oracle Database allows you read-only access to data in external tables. **External tables** are defined as tables that do not reside in the database, and they can be in any format for which an access driver is provided.

By providing the database with metadata describing an external table, the database is able to expose the data in the external table as if it were data residing in a regular database table. The external data can be queried directly and in parallel using SQL.

You can, for example, select, join, or sort external table data. You can also create views and synonyms for external tables. However, no DML operations (`UPDATE`, `INSERT`, or `DELETE`) are possible, and no indexes can be created, on external tables.

External tables provide a framework to unload the result of an arbitrary `SELECT` statement into a platform-independent Oracle-proprietary format that can be used by Oracle Data Pump. External tables provide a valuable means for performing basic extraction, transformation, and loading (ETL) tasks that are common for data warehousing.

You define the metadata for external tables with the `CREATE TABLE...ORGANIZATION EXTERNAL` statement. This external table definition can be thought of as a view that allows running any SQL query against external data without requiring that the external data first be loaded into the database. An access driver is the mechanism used to read the external data in the table. When you use external tables to unload data, the metadata is automatically created based on the data types in the `SELECT` statement.

Oracle Database provides access drivers for external tables. The default access driver is `ORACLE_LOADER`, which allows the reading of data from external files using the Oracle loader technology. The `ORACLE_LOADER` access driver provides data mapping capabilities which are a subset of the control file syntax of SQL*Loader utility. Another access driver, `ORACLE_DATAPUMP`, lets you unload data—that is, read data from the database and insert it into an external table, represented by one or more external files—and then reload it into an Oracle Database.

Starting with Oracle Database 12*c* Release 2 (12.2), new access drivers `ORACLE_HIVE` and `ORACLE_HDFS` are available. The `ORACLE_HIVE` access driver can extract data stored in Apache Hive. The `ORACLE_HDFS` access driver can extract data stored in a Hadoop Distributed File System (HDFS).

Starting with Oracle Database 18c, inline external tables are supported. Inline external tables enable the runtime definition of an external table as part of a SQL statement, without creating the external table as persistent object in the data dictionary.

Starting with Oracle Database Release 19.10, Object storage is supported as a source for external table data. This enables Oracle databases to use data stored in Cloud applications. Additional data formats such as ORC, Parquet, and Avro are supported with the `ORACLE_BIGDATA` access driver.

Note:

The `ANALYZE` statement is not supported for gathering statistics for external tables. Use the `DBMS_STATS` package instead.

See Also:

- *Oracle Database SQL Language Reference* for restrictions that apply to external tables
- *Oracle Database Utilities* for information about access drivers
- *Oracle Database Data Warehousing Guide* for information about using external tables for ETL in a data warehousing environment
- *Oracle Database SQL Tuning Guide* for information about using the `DBMS_STATS` package
- *Oracle Database Utilities* for information about the `ORACLE_HIVE` and `ORACLE_HDFS` drivers and for more information about external tables

19.15.2 Creating External Tables

You create external tables using the `CREATE TABLE` statement with an `ORGANIZATION EXTERNAL` clause. This statement creates only metadata in the data dictionary.

Note:

- Starting with Oracle Database 12*c* Release 2 (12.2), you can partition external tables for fast query performance and enhanced data maintenance for large amounts of data.
- External tables can have virtual columns. However, a virtual column in an external table cannot be defined using the *`evaluation_edition_clause`* or the *`unusable_edition_clause`*.

Example 19-21   Creating an External Table and Loading Data

This example creates an external table and then uploads the data to a database table. Alternatively, you can unload data through the external table framework by specifying the `AS` *`subquery`* clause of the `CREATE TABLE` statement. External table data pump unload can use only the `ORACLE_DATAPUMP` access driver.

The data for the external table resides in the two text files `empxt1.dat` and `empxt2.dat`.

The file `empxt1.dat` contains the following sample data:

```
360,Jane,Janus,ST_CLERK,121,17-MAY-2001,3000,0,50,jjanus
361,Mark,Jasper,SA_REP,145,17-MAY-2001,8000,.1,80,mjasper
362,Brenda,Starr,AD_ASST,200,17-MAY-2001,5500,0,10,bstarr
363,Alex,Alda,AC_MGR,145,17-MAY-2001,9000,.15,80,aalda
```

The file `empxt2.dat` contains the following sample data:

```
401,Jesse,Cromwell,HR_REP,203,17-MAY-2001,7000,0,40,jcromwel
402,Abby,Applegate,IT_PROG,103,17-MAY-2001,9000,.2,60,aapplega
403,Carol,Cousins,AD_VP,100,17-MAY-2001,27000,.3,90,ccousins
404,John,Richardson,AC_ACCOUNT,205,17-MAY-2001,5000,0,110,jrichard
```

The following SQL statements create an external table named `admin_ext_employees` in the `hr` schema and load data from the external table into the `hr.employees` table.

```
CONNECT  /  AS SYSDBA;
-- Set up directories and grant access to hr
CREATE OR REPLACE DIRECTORY admin_dat_dir
    AS '/flatfiles/data';
CREATE OR REPLACE DIRECTORY admin_log_dir
    AS '/flatfiles/log';
CREATE OR REPLACE DIRECTORY admin_bad_dir
    AS '/flatfiles/bad';
GRANT READ ON DIRECTORY admin_dat_dir TO hr;
GRANT WRITE ON DIRECTORY admin_log_dir TO hr;
GRANT WRITE ON DIRECTORY admin_bad_dir TO hr;
-- hr connects. Provide the user password (hr) when prompted.
CONNECT hr
-- create the external table
CREATE TABLE admin_ext_employees
                   (employee_id       NUMBER(4),
                    first_name        VARCHAR2(20),
                    last_name         VARCHAR2(25),
                    job_id            VARCHAR2(10),
                    manager_id        NUMBER(4),
                    hire_date         DATE,
                    salary            NUMBER(8,2),
                    commission_pct    NUMBER(2,2),
                    department_id     NUMBER(4),
                    email             VARCHAR2(25)
                   )
     ORGANIZATION EXTERNAL
     (
       TYPE ORACLE_LOADER
       DEFAULT DIRECTORY admin_dat_dir
       ACCESS PARAMETERS
       (
         records delimited by newline
         badfile admin_bad_dir:'empxt%a_%p.bad'
         logfile admin_log_dir:'empxt%a_%p.log'
         fields terminated by ','
         missing field values are null
         ( employee_id, first_name, last_name, job_id, manager_id,
           hire_date char date_format date mask "dd-mon-yyyy",
           salary, commission_pct, department_id, email
         )
       )
       LOCATION ('empxt1.dat', 'empxt2.dat')
     )
     PARALLEL
     REJECT LIMIT UNLIMITED;
-- enable parallel for loading (good if lots of data to load)
ALTER SESSION ENABLE PARALLEL DML;
-- load the data in hr employees table
INSERT INTO employees (employee_id, first_name, last_name, job_id, manager_id,
                       hire_date, salary, commission_pct, department_id, email)
            SELECT * FROM admin_ext_employees;
```

The following paragraphs contain descriptive information about this example.

The first few statements in this example create the directory objects for the operating system directories that contain the data sources, and for the bad record and log files specified in the access parameters. You must also grant `READ` or `WRITE` directory object privileges, as appropriate.

Note:

When creating a directory object or BFILEs, ensure that the following conditions are met:

- The operating system file must not be a symbolic or hard link.
- The operating system directory path named in the Oracle Database directory object must be an existing OS directory path.
- The operating system directory path named in the directory object should not contain any symbolic links in its components.

The `TYPE` specification indicates the access driver of the external table. The access driver is the API that interprets the external data for the database. If you omit the `TYPE` specification, `ORACLE_LOADER` is the default access driver. You must specify the `ORACLE_DATAPUMP` access driver if you specify the `AS` *`subquery`* clause to unload data from one Oracle Database and reload it into the same or a different Oracle Database.

The access parameters, specified in the `ACCESS PARAMETERS` clause, are opaque to the database. These access parameters are defined by the access driver, and are provided to the access driver by the database when the external table is accessed. See *Oracle Database Utilities* for a description of the `ORACLE_LOADER` access parameters.

The `PARALLEL` clause enables parallel query on the data sources. The granule of parallelism is by default a data source, but parallel access within a data source is implemented whenever possible. For example, if `PARALLEL=3` were specified, then multiple parallel execution servers could be working on a data source. But, parallel access within a data source is provided by the access driver only if all of the following conditions are met:

- The media allows random positioning within a data source.
- It is possible to find a record boundary from a random position.

- The data files are large enough to make it worthwhile to break up into multiple chunks.

> **Note:**
>
> Specifying a `PARALLEL` clause is of value *only* when dealing with large amounts of data. Otherwise, it is not advisable to specify a `PARALLEL` clause, and doing so can be detrimental.

The `REJECT LIMIT` clause specifies that there is no limit on the number of errors that can occur during a query of the external data. For parallel access, the `REJECT LIMIT` applies to each parallel execution server independently. For example, if a `REJECT LIMIT` of 10 is specified, then each parallel query process can allow up to 10 rejections. Therefore, with a parallel degree of two and a `REJECT LIMIT` of 10, the statement might fail with between 10 and 20 rejections. If one parallel server processes all 10 rejections, then the limit is reached, and the statement is terminated. However, one parallel execution server could process nine rejections and another parallel execution server could process nine rejections and the statement will succeed with 18 rejections. Hence, the only precisely enforced values for `REJECT LIMIT` on parallel query are `0` and `UNLIMITED`.

In this example, the `INSERT INTO TABLE` statement generates a dataflow from the external data source to the Oracle Database SQL engine where data is processed. As data is parsed by the access driver from the external table sources and provided to the external table interface, the external data is converted from its external representation to its Oracle Database internal data type.

Example 19-22 Creating an External Table for Data Stored in Oracle Cloud

This example creates an external table that enables you to access data stored in Oracle Cloud Infrastructure Object Storage (Object Storage).

Before you create the external table, you must create a credential object to store your object storage credentials. The credential object stores, in an encrypted format, the user name and password (or API signing key) required to access Object Storage credentials. The credential password must match the Auth Token created for the user name in your Cloud service. Ensure that the identity specified by the credential has access to the underlying data in Object Store. Note that this step is required only once, unless your object store credentials change.

The following example creates a credential object named `MY_OCI_CRED`.

```
BEGIN
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'MY_OCI_CRED',
    username => 'oss_user@example.com',
    password => 'password');
END;
/
```

Create an external table on top of your source files using the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure. This procedure supports external files in the supported cloud object storage services. The credential is a table level property; therefore, the external files must be on the same object store.

The following statement creates an external table to access data stored in Object Storage. The external table is based on the source file `channels.txt`.

```
BEGIN
   DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
    table_name =>'CHANNELS_EXT',
    credential_name =>'MY_OCI_CRED',
    file_uri_list =>'https://objectstorage.us-phoenix-1.oraclecloud.com/n/
namespace-string/b/bucketname/o/channels.txt',
    format => json_object('delimiter' value ','),
    column_list => 'CHANNEL_ID NUMBER, CHANNEL_DESC VARCHAR2(20),
CHANNEL_CLASS VARCHAR2(20)' );
END;
/
```

where:

- `credential_name` is the name of the credential object created, `MY_OCI_CRED`.
- `file_uri_list` is a comma delimited list of the source files you want to query.
- `format` defines the options you can specify to describe the format of the source file.
- `column_list` is a comma delimited list of the column definitions in the source files.

See Also:

- *Oracle Database SQL Language Reference* provides details of the syntax of the `CREATE TABLE` statement for creating external tables and specifies restrictions on the use of clauses
- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_CLOUD package and its procedures

19.15.3 Altering External Tables

You can modify an external table with the `ALTER TABLE` statement.

You can use any of the `ALTER TABLE` clauses shown in Table 19-6 to change the characteristics of an external table. No other clauses are permitted.

Table 19-6 ALTER TABLE Clauses for External Tables

ALTER TABLE Clause	Description	Example
`REJECT LIMIT`	Changes the reject limit. The default value is `0`.	`ALTER TABLE admin_ext_employees REJECT LIMIT 100;`

Table 19-6 (Cont.) ALTER TABLE Clauses for External Tables

ALTER TABLE Clause	Description	Example

`PROJECT COLUMN`	Determines how the access driver validates rows in subsequent queries:
	• `PROJECT COLUMN REFERENCED`: the access driver processes only the columns in the select list of the query. This setting may not provide a consistent set of rows when querying a different column list from the same external table. This is the default setting for big data access drivers `ORACLE_HDFS` and `ORACLE_HIVE` access drivers.
	• `PROJECT COLUMN ALL`: the access driver processes all of the columns defined on the external table. This setting always provides a consistent set of rows when querying an external table. This is the default. This is also the default setting for `ORACLE_LOADER` and `ORACLE_DATAPUMP` access drivers.
		`ALTER TABLE admin_ext_employees PROJECT COLUMN REFERENCED;`
		
		`ALTER TABLE admin_ext_employees PROJECT COLUMN ALL;`

`DEFAULT DIRECTORY`	Changes the default directory specification.	`ALTER TABLE admin_ext_employees DEFAULT DIRECTORY admin_dat2_dir;`
`ACCESS PARAMETERS`	Allows access parameters to be changed without dropping and re-creating the external table metadata.	`ALTER TABLE admin_ext_employees ACCESS PARAMETERS (FIELDS TERMINATED BY ';');`
`LOCATION`	Allows data sources to be changed without dropping and re-creating the external table metadata.	`ALTER TABLE admin_ext_employees LOCATION ('empxt3.txt', 'empxt4.txt');`
`PARALLEL`	No difference from regular tables. Allows degree of parallelism to be changed.	No new syntax
`ADD COLUMN`	No difference from regular tables. Allows a column to be added to an external table. Virtual columns are not permitted.	No new syntax

Table 19-6 (Cont.) ALTER TABLE Clauses for External Tables

ALTER TABLE Clause	Description	Example
`MODIFY COLUMN`	No difference from regular tables. Allows an external table column to be modified. Virtual columns are not permitted.	No new syntax
`SET UNUSED`	Transparently converted into an `ALTER TABLE DROP COLUMN` command. Because external tables consist of metadata only in the database, the `DROP COLUMN` command performs equivalently to the `SET UNUSED` command.	No new syntax
`DROP COLUMN`	No difference from regular tables. Allows an external table column to be dropped.	No new syntax
`RENAME TO`	No difference from regular tables. Allows external table to be renamed.	No new syntax

19.15.4 Preprocessing External Tables

External tables can be preprocessed by user-supplied preprocessor programs. By using a preprocessing program, users can use data from a file that is not in a format supported by the driver.

Caution:

There are security implications to consider when using the `PREPROCESSOR` clause. See *Oracle Database Security Guide* for more information.

For example, a user may want to access data stored in a compressed format. Specifying a decompression program for the `ORACLE_LOADER` access driver allows the data to be decompressed as the access driver processes the data.

To use the preprocessing feature, you must specify the `PREPROCESSOR` clause in the access parameters of the `ORACLE_LOADER` access driver. The preprocessor must be a directory object, and the user accessing the external table must have `EXECUTE` privileges for the directory object. The following example includes the `PREPROCESSOR` clause and specifies the directory and preprocessor program.

```
CREATE TABLE sales_transactions_ext
(PROD_ID NUMBER,
 CUST_ID NUMBER,
 TIME_ID DATE,
 CHANNEL_ID CHAR,
 PROMO_ID NUMBER,
 QUANTITY_SOLD NUMBER,
 AMOUNT_SOLD NUMBER(10,2),
 UNIT_COST NUMBER(10,2),
 UNIT_PRICE NUMBER(10,2))
```

```
ORGANIZATION external
(TYPE oracle_loader
 DEFAULT DIRECTORY data_file_dir
 ACCESS PARAMETERS
  (RECORDS DELIMITED BY NEWLINE
   CHARACTERSET AL32UTF8
   PREPROCESSOR exec_file_dir:'zcat'
   BADFILE log_file_dir:'sh_sales.bad_xt'
   LOGFILE log_file_dir:'sh_sales.log_xt'
   FIELDS TERMINATED BY "|" LDRTRIM
  ( PROD_ID,
    CUST_ID,
    TIME_ID,
    CHANNEL_ID,
    PROMO_ID,
    QUANTITY_SOLD,
    AMOUNT_SOLD,
    UNIT_COST,
    UNIT_PRICE))
 location ('sh_sales.dat.gz')
)REJECT LIMIT UNLIMITED;
```

The `PREPROCESSOR` clause is not available for databases that use Oracle Database Vault.

Note:

On the Windows platform, a preprocessor program must have a `.bat` or `.cmd` extension.

See Also:

- *Oracle Database Utilities* provides information more information about the `PREPROCESSOR` clause
- *Oracle Database Security Guide* for more information about the security implications of the `PREPROCESSOR` clause

19.15.5 Overriding Parameters for External Tables in a Query

The `EXTERNAL MODIFY` clause of a `SELECT` statement modifies external table parameters.

You can override the following clauses for an external table in an `EXTERNAL MODIFY` clause:

- `DEFAULT DIRECTORY`
- `LOCATION`
- `ACCESS PARAMETERS`
- `REJECT LIMIT`

You can modify more than one clause in a single query. A bind variable can be specified for `LOCATION` and `REJECT LIMIT`, but not for `DEFAULT DIRECTORY` or `ACCESS PARAMETERS`.

The modifications only apply to the query. They do not affect the table permanently.

For partitioned external tables, only table-level clauses can be overridden.

1. Connect to the database as a user with the privileges required to query the external table.
2. Issue a `SELECT` statement on the external table with the `EXTERNAL MODIFY` clause.

Example 19-23 Overriding Parameters for External Tables in a Query

Assume an external table named `sales_external` has a `REJECT LIMIT` set to `25`. The following query modifies this setting to `REJECT LIMIT UNLIMITED`:

```
SELECT * FROM sales_external EXTERNAL MODIFY (LOCATION ('sales_9.csv')
   REJECT LIMIT UNLIMITED);
```

19.15.6 Using Inline External Tables

Inline external tables enable the runtime definition of an external table as part of a SQL statement, without creating the external table as persistent object in the data dictionary.

With inline external tables, the same syntax that is used to create an external table with a `CREATE TABLE` statement can be used in a `SELECT` statement at runtime. Specify inline external tables in the `FROM` clause of a query block. Queries that include inline external tables can also include regular tables for joins, aggregation, and so on.

The following SQL statement performs a runtime query on external data:

```
SELECT * FROM   EXTERNAL (
    (time_id        DATE NOT NULL,
     prod_id        INTEGER NOT NULL,
     quantity_sold  NUMBER(10,2),
     amount_sold    NUMBER(10,2))
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY data_dir1
    ACCESS PARAMETERS (
      RECORDS DELIMITED BY NEWLINE
      FIELDS TERMINATED BY '|')
   LOCATION ('sales_9.csv') REJECT LIMIT UNLIMITED) sales_external;
```

Although no table named `sales_external` was created previously, this query reads the external data and returns the results.

Note:

Inline external tables do not support partitioning. The query can control which directories and files to scan, so that pruning can be accomplished by omitting files that are not needed for the query.

19.15.7 Partitioning External Tables

For large amounts of data, partitioning for external tables provides fast query performance and enhanced data maintenance.

- About Partitioning External Tables
  Partitioning data in external tables is similar to partitioning tables stored in the database, but there are some differences. The files for the partitioned external table can be stored on a file system, in Apache Hive storage, or in a Hadoop Distributed File System (HDFS).
- Restrictions for Partitioned External Tables
  Some restrictions apply to partitioned external tables.
- Creating a Partitioned External Table
  You create a non-composite partitioned external table by issuing a `CREATE TABLE` statement with the `ORGANIZATION EXTERNAL` clause and the `PARTITION BY` clause. To create a composite partitioned external table, the `SUBPARTITION BY` clause must also be included.
- Altering a Partitioned External Table
  You can use the `ALTER TABLE` statement to modify table-level external parameters, but not the partition-level and subpartition-level parameters, of a partitioned external table.

19.15.7.1 About Partitioning External Tables

Partitioning data in external tables is similar to partitioning tables stored in the database, but there are some differences. The files for the partitioned external table can be stored on a file system, in Apache Hive storage, or in a Hadoop Distributed File System (HDFS).

Before attempting to partition external tables, you should understand the concepts related to partitioning in *Oracle Database VLDB and Partitioning Guide*.

The main reason to partition external tables is to take advantage of the same performance improvements provided by partitioning tables stored in the database. Specifically, partition pruning and partition-wise joins can improve query performance. Partition pruning means that queries can focus on a subset of the data in an external table instead of all of the data because the query can apply to only one partition. Partition-wise joins can be applied when two tables are being joined and both tables are partitioned on the join key, or when a reference partitioned table is joined with its parent table. Partition-wise joins break a large join into smaller joins that occur between each of the partitions, completing the overall join in less time.

Most of the partitioning strategies that are supported for tables in the database are supported for external tables. External tables can be partitioned by range or list, and composite partitioning is supported. However, hash partitioning is not supported for external tables.

For a partitioned table that is stored in the database, storage for each partition is specified with a tablespace. For a partitioned external table, storage for each partition is specified by indicating the directory and files for each partition.

Clauses for Creating Partitioned External Tables

The clauses for creating a non-partitioned external table are the following:

- `TYPE` - Specifies the access driver for the type of external table (`ORACLE_LOADER`, `ORACLE_DATAPUMP`, `ORACLE_HIVE`, and `ORACLE_HDFS`).
- `DEFAULT DIRECTORY` - Specifies with a directory object the default directory to use for all input and output files that do not explicitly name a directory object.

- ACCESS PARAMETERS - Describe the external data source.
- LOCATION - Specifies the files for the external table.
- REJECT LIMIT - Specifies the number of errors that can occur during a query of the external data.

When you create a partitioned external table, you must include a PARTITION clause that defines each partition. The following table describes the clauses allowed at each level during external table creation.

Table 19-7 External Table Clauses and Partitioning

Clause	Table Level	Partition Level	Subpartition Level
TYPE	Allowed	Not Allowed	Not Allowed
DEFAULT DIRECTORY	Allowed	Allowed	Allowed
ACCESS PARAMETERS	Allowed	Not Allowed	Not Allowed
LOCATION	Not allowed	Allowed	Allowed
REJECT LIMIT	Allowed	Not allowed	Not allowed

For a non-composite partitioned table, files for a partition must be specified in the LOCATION clause for the partition. For a composite partitioned table, files for a subpartition must be specified in the LOCATION clause for the subpartition. When a partition has subpartitions, the LOCATION clause can be specified for subpartitions but not for the partition. If the LOCATION clause is omitted for a partition or subpartition, then an empty partition or subpartition is created.

In the LOCATION clause, the files are named in the form *directory:file*, and one clause can specify multiple files. The *directory* portion is optional. The following rules apply for the directory used by a partition or subpartition:

- When a directory is specified in the LOCATION clause for a partition or subpartition, then it applies to that location only.
- In the LOCATION clause for a specific partition, for each file that does not have a directory specification, use the directory specified in the DEFAULT DIRECTORY clause for the partition or table level, in order.

  For example, when the ORGANIZATION EXTERNAL clause of a CREATE TABLE statement includes a DEFAULT DIRECTORY clause, and a PARTITION clause in the statement does not specify a directory for a file in its LOCATION clause, the file uses the directory specified in the DEFAULT DIRECTORY clause for the table.
- In the LOCATION clause for a specific subpartition, for each file that does not have a directory specification, use the directory specified in the DEFAULT DIRECTORY clause for the subpartition, partition, or table level, in order.

  For example, when a PARTITION clause includes a DEFAULT DIRECTORY clause, and a SUBPARITION clause in the partition does not specify a directory for a file in its LOCATION clause, the file uses the directory specified in the DEFAULT DIRECTORY clause for the partition.
- The default directory for a partition or subpartition cannot be specified in a LOCATION clause. It can only be specified in a DEFAULT DIRECTORY clause.

See Also:

Example 19-25 illustrates the directory rules

Using the ORACLE_HIVE Access Driver

Apache Hive has its own partitioning. To create partitioned external tables, use the `CREATE_EXTDDL_FOR_HIVE` procedure in the `DBMS_HADOOP` package. This procedure generates data definition language (DDL) statements that you can use to create a partitioned external table that corresponds with the partitioning in the Apache Hive storage.

The `DBMS_HADOOP` package also includes the `SYNC_PARTITIONS_FOR_HIVE` procedure. This procedure automatically synchronizes the partitioning of the partitioned external table in the Apache Hive storage with the partitioning metadata of the same table stored in the Oracle Database.

Related Topics

- Altering External Tables
  You can modify an external table with the `ALTER TABLE` statement.
- *Oracle Database Utilities*
- *Oracle Database PL/SQL Packages and Types Reference*

19.15.7.2 Restrictions for Partitioned External Tables

Some restrictions apply to partitioned external tables.

The following are restrictions for partitioned external tables:

- All restrictions that apply to non-partitioned external tables also apply to partitioned external tables.
- Partitioning restrictions that apply to tables stored in the database also apply to partitioned external tables, such as the maximum number of partitions.
- Oracle Database cannot guarantee that the external files for partitions contain data that satisfies partitioning definitions.
- Only the `DEFAULT DIRECTORY` and `LOCATION` clauses can be specified in a `PARTITION` or `SUBPARTITION` clause.
- When altering a partitioned external table with the `ALTER TABLE` statement, the following clauses are not supported: `MODIFY PARTITION`, `EXCHANGE PARTITION`, `MOVE PARTITION`, `MERGE PARTITIONS`, `SPLIT PARTITION`, `COALESCE PARTITION`, and `TRUNCATE PARTITION`.
- Reference partitioning, automatic list partitioning, and interval partitioning are not supported.
- Subpartition templates are not supported.
- The `ORACLE_DATAPUMP` access driver cannot populate external files for partitions using a `CREATE TABLE AS SELECT` statement.
- Incremental statistics are not gathered for partitioned external tables.
- In addition to restrictions on partitioning methods that can be used for the other drivers, range and composite partitioning are not supported for the `ORACLE_HIVE` access driver.

- A SELECT statement with the EXTERNAL MODIFY clause cannot override partition-level or subpartition-level clauses. Only external clauses supported at the table level can be overridden with the EXTERNAL MODIFY clause. Because the LOCATION clause is not allowed at the table level for a partitioned external table, it cannot be overridden with the EXTERNAL MODIFY clause.

See Also:

- "About External Tables"
- *Oracle Database SQL Language Reference* provides details of the syntax of the CREATE TABLE statement for creating external tables and specifies restrictions on the use of clauses

19.15.7.3 Creating a Partitioned External Table

You create a non-composite partitioned external table by issuing a CREATE TABLE statement with the ORGANIZATION EXTERNAL clause and the PARTITION BY clause. To create a composite partitioned external table, the SUBPARTITION BY clause must also be included.

The PARTITION BY clause and the SUBPARTITION BY clause specify the locations of the external files for each partition and subpartition.

To create a partitioned external table, the database must be at 12.2.0 compatibility level or higher.

1. Connect to the database as a user with the privileges required to create the external table.

   See *Oracle Database SQL Language Reference* for information about the required privileges.

2. Issue a CREATE TABLE statement with the ORGANIZATION EXTERNAL clause and the PARTITION BY clause. For a composite partitioned table, include the SUBPARTITION BY clause also.

Example 19-24 Creating a Partitioned External Table with Access Parameters Common to All Partitions

This example creates an external table named orders_external_range that is partitioned by the date data in the order_date column. The ACCESS PARAMETERS clause is specified at the table level for the ORACLE_LOADER access driver. The data_dir1 directory object is the default directory object used for the partitions month1, month2, and month3. The pmax partition specifies the data_dir2 directory object in the DEFAULT DIRECTORY clause, so the data_dir2 directory object is used for the pmax partition.

```
-- Set up directories and grant access to oe
CREATE OR REPLACE DIRECTORY data_dir1
    AS '/flatfiles/data1';
CREATE OR REPLACE DIRECTORY data_dir2
    AS '/flatfiles/data2';
CREATE OR REPLACE DIRECTORY bad_dir
    AS '/flatfiles/bad';
CREATE OR REPLACE DIRECTORY log_dir
    AS '/flatfiles/log';
```

```
GRANT READ ON DIRECTORY data_dir1 TO oe;
GRANT READ ON DIRECTORY data_dir2 TO oe;
GRANT WRITE ON DIRECTORY bad_dir TO oe;
GRANT WRITE ON DIRECTORY log_dir TO oe;
-- oe connects. Provide the user password (oe) when prompted.
CONNECT oe
-- create the partitioned external table
CREATE TABLE orders_external_range(
    order_id           NUMBER(12),
    order_date         DATE NOT NULL,
    customer_id        NUMBER(6) NOT NULL,
    order_status       NUMBER(2),
    order_total        NUMBER(8,2),
    sales_rep_id       NUMBER(6))
ORGANIZATION EXTERNAL(
   TYPE ORACLE_LOADER
   DEFAULT DIRECTORY data_dir1
   ACCESS PARAMETERS(
      RECORDS DELIMITED BY NEWLINE
      BADFILE bad_dir: 'sh%a_%p.bad'
      LOGFILE log_dir: 'sh%a_%p.log'
      FIELDS TERMINATED BY '|'
      MISSING FIELD VALUES ARE NULL))
PARALLEL
REJECT LIMIT UNLIMITED
PARTITION BY RANGE (order_date)
  (PARTITION month1 VALUES LESS THAN (TO_DATE('31-12-2014', 'DD-MM-YYYY'))
        LOCATION ('sales_1.csv'),
    PARTITION month2 VALUES LESS THAN (TO_DATE('31-01-2015', 'DD-MM-YYYY'))
        LOCATION ('sales_2.csv'),
    PARTITION month3 VALUES LESS THAN (TO_DATE('28-02-2015', 'DD-MM-YYYY'))
        LOCATION ('sales_3.csv'),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
        DEFAULT DIRECTORY data_dir2 LOCATION('sales_4.csv'));
```

In the previous example, the default directory `data_dir2` is specified for the `pmax` partition. You can also specify the directory for a specific location in this partition in the `LOCATION` clause in the following way:

```
PARTITION pmax VALUES LESS THAN (MAXVALUE)
   LOCATION ('data_dir2:sales_4.csv')
```

Note that, in this case, the directory `data_dir2` is specified for the location `sales_4.csv`, but the `data_dir2` directory is not the default directory for the partition. Therefore, the default directory for the `pmax` partition is the same as the default directory for the table, which is `data_dir1`.

Example 19-25 Creating a Composite List-Range Partitioned External Table

This example creates an external table named `accounts` that is partitioned by the data in the `region` column. This partition is subpartitioned using range on the data in the `balance` column. The `ACCESS PARAMETERS` clause is specified at the table level for the `ORACLE_LOADER` access driver. A `LOCATION` clause is specified for each subpartition.

There is a table-level `DEFAULT DIRECTORY` clause set to the `data_dir1` directory object, and this directory object is used for all of the subpartitions, except for the following:

- There is a partition-level `DEFAULT DIRECTORY` clause set to the `data_dir2` directory object for partition `p_southcentral`. In that partition, the following subpartitions use this default directory: `p_sc_low`, `p_sc_high`, and `p_sc_extraordinary`.
- In partition `p_southcentral`, the subpartition `p_sc_average` has a subpartition-level `DEFAULT DIRECTORY` clause set to the `data_dir3` directory object, and this subpartition uses the `data_dir3` directory object.
- As previously stated, the default directory for the `p_sc_high` subpartition is `data_dir2`. The `p_sc_high` subpartition does not have a `DEFAULT DIRECTORY` clause, and the default directory `data_dir2` is inherited from the `DEFAULT DIRECTORY` specified in the `PARTITION BY` clause for the partition `p_southcentral`. The files in the `p_sc_high` subpartition use the following directories:
  - The `psch1.csv` file uses `data_dir2`, the default directory for the subpartition.
  - The `psch2.csv` file uses the `data_dir4` directory because the `data_dir4` directory is specified for that location.

```
-- Set up the directories and grant access to oe
CREATE OR REPLACE DIRECTORY data_dir1
    AS '/stage/data1_dir';
CREATE OR REPLACE DIRECTORY data_dir2
    AS '/stage/data2_dir';
CREATE OR REPLACE DIRECTORY data_dir3
    AS '/stage/data3_dir';
CREATE OR REPLACE DIRECTORY data_dir4
    AS '/stage/data4_dir';
CREATE OR REPLACE DIRECTORY bad_dir
    AS '/stage/bad_dir';
CREATE OR REPLACE DIRECTORY log_dir
    AS '/stage/log_dir';
GRANT READ ON DIRECTORY data_dir1 TO oe;
GRANT READ ON DIRECTORY data_dir2 TO oe;
GRANT READ ON DIRECTORY data_dir3 TO oe;
GRANT READ ON DIRECTORY data_dir4 TO oe;
GRANT WRITE ON DIRECTORY bad_dir TO oe;
GRANT WRITE ON DIRECTORY log_dir TO oe;
-- oe connects. Provide the user password (oe) when prompted.
CONNECT oe
-- create the partitioned external table
CREATE TABLE accounts
( id             NUMBER,
  account_number NUMBER,
  customer_id    NUMBER,
  balance        NUMBER,
  branch_id      NUMBER,
  region         VARCHAR(2),
  status         VARCHAR2(1)
)
ORGANIZATION EXTERNAL(
   TYPE ORACLE_LOADER
   DEFAULT DIRECTORY data_dir1
   ACCESS PARAMETERS(
      RECORDS DELIMITED BY NEWLINE
      BADFILE bad_dir: 'sh%a_%p.bad'
      LOGFILE log_dir: 'sh%a_%p.log'
      FIELDS TERMINATED BY '|'
      MISSING FIELD VALUES ARE NULL))
PARALLEL
REJECT LIMIT UNLIMITED
PARTITION BY LIST (region)
SUBPARTITION BY RANGE (balance)
( PARTITION p_northwest VALUES ('OR', 'WA')
  ( SUBPARTITION p_nw_low VALUES LESS THAN (1000) LOCATION ('pnwl.csv'),
    SUBPARTITION p_nw_average VALUES LESS THAN (10000) LOCATION ('pnwa.csv'),
    SUBPARTITION p_nw_high VALUES LESS THAN (100000) LOCATION ('pnwh.csv'),
    SUBPARTITION p_nw_extraordinary VALUES LESS THAN (MAXVALUE) LOCATION 
('pnwe.csv')
  ),
  PARTITION p_southwest VALUES ('AZ', 'UT', 'NM')
  ( SUBPARTITION p_sw_low VALUES LESS THAN (1000) LOCATION ('pswl.csv'),
    SUBPARTITION p_sw_average VALUES LESS THAN (10000) LOCATION ('pswa.csv'),
    SUBPARTITION p_sw_high VALUES LESS THAN (100000) LOCATION ('pswh.csv'),
    SUBPARTITION p_sw_extraordinary VALUES LESS THAN (MAXVALUE) LOCATION 
('pswe.csv')
  ),
  PARTITION p_northeast VALUES ('NY', 'VM', 'NJ')
  ( SUBPARTITION p_ne_low VALUES LESS THAN (1000) LOCATION ('pnel.csv'),
    SUBPARTITION p_ne_average VALUES LESS THAN (10000) LOCATION ('pnea.csv'),
    SUBPARTITION p_ne_high VALUES LESS THAN (100000) LOCATION ('pneh.csv'),
    SUBPARTITION p_ne_extraordinary VALUES LESS THAN (MAXVALUE) LOCATION 
('pnee.csv')
  ),
  PARTITION p_southeast VALUES ('FL', 'GA')
  ( SUBPARTITION p_se_low VALUES LESS THAN (1000) LOCATION ('psel.csv'),
    SUBPARTITION p_se_average VALUES LESS THAN (10000) LOCATION ('psea.csv'),
    SUBPARTITION p_se_high VALUES LESS THAN (100000) LOCATION ('pseh.csv'),
    SUBPARTITION p_se_extraordinary VALUES LESS THAN (MAXVALUE) LOCATION 
('psee.csv')
  ),
  PARTITION p_northcentral VALUES ('SD', 'WI')
  ( SUBPARTITION p_nc_low VALUES LESS THAN (1000) LOCATION ('pncl.csv'),
    SUBPARTITION p_nc_average VALUES LESS THAN (10000) LOCATION ('pnca.csv'),
    SUBPARTITION p_nc_high VALUES LESS THAN (100000) LOCATION ('pnch.csv'),
    SUBPARTITION p_nc_extraordinary VALUES LESS THAN (MAXVALUE) LOCATION 
('pnce.csv')
  ),
  PARTITION p_southcentral VALUES ('OK', 'TX') DEFAULT DIRECTORY data_dir2
  ( SUBPARTITION p_sc_low VALUES LESS THAN (1000) LOCATION ('pscl.csv'),
    SUBPARTITION p_sc_average VALUES LESS THAN (10000)
       DEFAULT DIRECTORY data_dir3 LOCATION ('psca.csv'),
    SUBPARTITION p_sc_high VALUES LESS THAN (100000)
       LOCATION ('psch1.csv','data_dir4:psch2.csv'),
    SUBPARTITION p_sc_extraordinary VALUES LESS THAN (MAXVALUE)
       LOCATION ('psce.csv')
  )
);
```

See Also:

Oracle Database VLDB and Partitioning Guide

19.15.7.4 Altering a Partitioned External Table

You can use the `ALTER TABLE` statement to modify table-level external parameters, but not the partition-level and subpartition-level parameters, of a partitioned external table.

The locations of external files are specified in the `PARTITION BY` and `SUBPARTITION BY` clauses. External files for a partition are specified in the partition’s `PARTITION BY` clause. External files for a subpartition are specified in the subpartition's `SUBPARTITION BY` clause.

The only exception is that the `LOCATION` clause cannot be specified at the table level during the creation of a partitioned external table. Therefore, the `LOCATION` clause cannot be added at the table level in an `ALTER TABLE` statement that modifies a partitioned external table.

At the partition level, only `ADD`, `DROP`, and `RENAME` operations are supported. An `ALTER TABLE` statement cannot modify the attributes of existing partitions and subpartitions. However, you can include the `DEFAULT DIRECTORY` and `LOCATION` clauses in a `PARTITION` clause or `SUBPARTITION` clause when you add a new partition or subpartition.

1. Connect to the database as a user with the privileges required to alter the external table.
2. Issue an `ALTER TABLE` statement.

Example 19-26 Renaming a Partition of a Partitioned External Table

This example renames a partition of the partitioned external table named `orders_external_range`.

```
ALTER TABLE orders_external_range RENAME PARTITION pmax TO other_months;
```

19.15.8 Dropping External Tables

For an external table, the `DROP TABLE` statement removes only the table metadata in the database. It has no affect on the actual data, which resides outside of the database.

19.15.9 System and Object Privileges for External Tables

System and object privileges for external tables are a subset of those for regular table.

Only the following system privileges are applicable to external tables:

- `ALTER ANY TABLE`
- `CREATE ANY TABLE`
- `DROP ANY TABLE`
- `READ ANY TABLE`
- `SELECT ANY TABLE`

Only the following object privileges are applicable to external tables:

- `ALTER`

- `READ`
- `SELECT`

However, object privileges associated with a directory are:

- `READ`
- `WRITE`

For external tables, `READ` privileges are required on directory objects that contain data sources, while `WRITE` privileges are required for directory objects containing bad, log, or discard files.

19.16 Managing Hybrid Partitioned Tables

A hybrid partitioned table is a partitioned table in which some partitions reside in the database and some partitions reside outside the database in external files, such as operating system files or Hadoop Distributed File System (HDFS) files.

Note:

The restrictions that apply to external tables also apply to hybrid partitioned tables.

See Also:

- "Partitioning External Tables"
- *Oracle Database VLDB and Partitioning Guide* for more information about managing hybrid partitioned tables

19.17 Managing Immutable Tables

Immutable tables provide protection against unauthorized data modification.

- About Immutable Tables
  Immutable tables are read-only tables that prevent unauthorized data modifications by insiders and accidental data modifications resulting from human errors.
- Guidelines for Managing Immutable Tables
  You can follow guidelines for working with immutable tables.
- Creating Immutable Tables
  Use the `CREATE IMMUTABLE TABLE` statement to create an immutable table. The immutable table is created in the specified schema and the table metadata is added to the data dictionary.
- Altering Immutable Tables
  You can modify the retention period for an immutable table and the retention period for rows within the immutable table.
- Deleting Rows from Immutable Tables
  Only rows that are outside the specified retention period can be deleted from an immutable table.

- Dropping Immutable Tables
  An immutable table can be dropped if it is empty or after it has not been modified for a period of time that is defined by its retention period.
- Immutable Tables Data Dictionary Views
  Data dictionary views provide information about immutable tables in the database.

19.17.1 About Immutable Tables

Immutable tables are read-only tables that prevent unauthorized data modifications by insiders and accidental data modifications resulting from human errors.

Unauthorized modifications can be attempted by compromised or rogue employees who have access to insider credentials.

New rows can be added to an immutable table, but existing rows cannot be modified. You must specify a retention period both for the immutable table and for rows within the immutable table. Rows become obsolete after the specified row retention period. Only obsolete rows can be deleted from the immutable table.

Immutable tables contain system-generated hidden columns. The columns are the same as those for blockchain tables. When a row is inserted, a non-NULL value is set for the `ORABCTAB_CREATION_TIME$` and `ORABCTAB_USER_NUMBER$` columns. The value of the remaining system-generated hidden columns is set to `NULL`.

Using immutable tables requires no changes to existing applications.

Table 19-8 Differences Between Immutable Tables and Blockchain Tables

Immutable Tables	Blockchain Tables
Immutable tables prevent unauthorized changes by rogue or compromised insiders who have access to user credentials.	In addition to preventing unauthorized changes by rogue or compromised insiders, blockchain tables provide the following functionality: • detects unauthorized changes made by bypassing Oracle Database software • detects end user impersonation and insertion of data in a user's name but without their authorization • prevents data tampering and ensures that data was actually inserted in to the table
Rows are not chained together.	Each row, except the first row, is chained to the previous row by using a cryptographic hash. The hash value of a row is computed based on the row data and the hash value of the previous row in the chain. Any modification to a row breaks the chain, thereby indicating that the row was tampered.
Inserting rows does not require additional processing at commit time.	Additional processing time is required, at commit time, to chain rows.

Related Topics

- Hidden Columns in Blockchain Tables
  Each row in a blockchain table contains hidden columns whose values are managed by the database.

19.17.2 Guidelines for Managing Immutable Tables

You can follow guidelines for working with immutable tables.

- Specify the Retention Period for the Immutable Table
  Use the `NO DROP` clause in a `CREATE IMMUTABLE TABLE` statement to set the retention period for the immutable table.
- Specify the Retention Period for Rows in the Immutable Table
  Use the `NO DELETE` clause in a `CREATE IMMUTABLE TABLE` statement to specify the retention period for rows in the immutable table.
- Restrictions for Immutable Tables
  Using immutable tables is subject to certain restrictions.

19.17.2.1 Specify the Retention Period for the Immutable Table

Use the `NO DROP` clause in a `CREATE IMMUTABLE TABLE` statement to set the retention period for the immutable table.

Unless the immutable table is empty, it cannot be dropped while it is within the specified retention period.

Specify one of the following clauses:

- `NO DROP`

  Immutable table cannot be dropped.

- `NO DROP UNTIL` *n* `DAYS IDLE`

  Immutable table cannot be dropped if the newest row is less than *n* days old. The minimum value that is allowed for *n* is 0. However, to ensure security of immutable tables, it is recommended that you set the minimum value to at least 16.

  To set the table retention period to 0 days, the initialization parameter `BLOCKCHAIN_TABLE_MAX_NO_DROP` must be set to 0. This setting is useful for testing immutable tables. Note that when this parameter is set to 0, the only allowed retention period is 0 days. To subsequently set a non-zero retention period, you must reset the value of the `BLOCKCHAIN_TABLE_MAX_NO_DROP` initialization parameter.

Use the `ALTER TABLE` statement to increase the retention period for a immutable table. You cannot reduce the retention period.

19.17.2.2 Specify the Retention Period for Rows in the Immutable Table

Use the `NO DELETE` clause in a `CREATE IMMUTABLE TABLE` statement to specify the retention period for rows in the immutable table.

The retention period controls when rows can be deleted from an immutable table.

Use one of the following options to specify the row retention period:

- `NO DELETE [LOCKED]`

  Rows cannot be deleted from the immutable table when `NO DELETE` is used.

  To ensure that rows are never deleted from the immutable table, use the `NO DELETE LOCKED` clause in the `CREATE IMMUTABLE TABLE` statement. The `LOCKED` keyword specifies that the row retention setting cannot be modified.

- `NO DELETE UNTIL n DAYS AFTER INSERT [LOCKED]`

  A row cannot be deleted until *n* days after it was added. Use the `ALTER TABLE` statement with the `NO DELETE UNTIL` clause to modify this setting and increase the retention period. You cannot reduce the retention period.

  The minimum value for *n* is 16 days. If `LOCKED` is included, you cannot subsequently modify the row retention.

19.17.2.3 Restrictions for Immutable Tables

Using immutable tables is subject to certain restrictions.

- The following data types are not supported with immutable tables: `ROWID`, `UROWID`, `LONG`, object type, REF, varray, nested table, `TIMESTAMP WITH TIME ZONE`, `TIMESTAMP WITH LOCAL TIME ZONE`, `BFILE`, and `XMLType`.

  `XMLType` tables are also not supported.
- Immutable tables can not be index-organized tables, `ORGANIZATION CUBE`, `ORGANIZATION EXTERNAL`, or hybrid partitioned.
- Immutable tables can not be index-organized tables, `ORGANIZATION CUBE`, `ORGANIZATION EXTERNAL`, or hybrid partitioned.
- The maximum number of user-created columns is 980.
- When immutable tables are exported and imported using Oracle Data Pump, they are exported and imported as regular tables without the system-generated hidden columns.
- The following operations are not supported with immutable tables:
  - Creating immutable tables in the CDB root or application root
  - Updating rows, merging rows, adding columns, renaming columns, dropping columns, and dropping partitions
  - Truncating the immutable table
  - Sharded tables
  - Direct-path loading
  - Flashback table
  - Defining `BEFORE ROW` triggers that fire for update operations (other triggers are allowed)
  - Creating Automatic Data Optimization (ADO) policies
  - Creating Oracle Label Security (OLS) policies
  - Online redefinition using the `DBMS_REDEFINITION` package
  - Transient Logical Standby and rolling upgrades

    DDL and DML on immutable tables are not supported and not replicated.
  - Logical Standby and Oracle GoldenGate

    DDL and DML on immutable tables succeed on the primary database but are not replicated to standby databases.
  - Converting a regular table to an immutable table or vice versa
- Flashback Database and point-in-time recovery of a database undo the changes made to all tables, including immutable tables. For example, if a database is flashed back to SCN

1000, or a backup is restored and recovered up to SCN 1000, all changes since SCN 1000 are removed from the database.

Oracle Database does not prevent flashback and point-in-time recovery operations since they may be required to undo physical and logical corruptions.

- Correctly enforcing retention policies in immutable tables relies on the system time. The privilege to change the system time must not be granted widely, and changes to the system time must be audited and reviewed.

19.17.3 Creating Immutable Tables

Use the `CREATE IMMUTABLE TABLE` statement to create an immutable table. The immutable table is created in the specified schema and the table metadata is added to the data dictionary.

The `COMPATIBLE` initialization parameter must be set to 19.11.0.0 or higher.

The `CREATE TABLE` system privilege is required to create immutable tables in your own schema. The `CREATE ANY TABLE` system privilege is required to create immutable tables in another user's schema. The `NO DROP` and `NO DELETE` clauses are mandatory in a `CREATE IMMUTABLE TABLE` statement.

Example 19-27 Creating an Immutable Table

The following example creates an immutable table named `trade_ledger` in your user schema. A row cannot be deleted until 100 days after it has been inserted. The immutable table can be dropped only after 40 days of inactivity

```
CREATE IMMUTABLE TABLE trade_ledger (id NUMBER, luser VARCHAR2(40), value 
NUMBER)
       NO DROP UNTIL 40 DAYS IDLE
       NO DELETE UNTIL 100 DAYS AFTER INSERT;
```

19.17.4 Altering Immutable Tables

You can modify the retention period for an immutable table and the retention period for rows within the immutable table.

The immutable table must be contained in your schema, or you must have either the `ALTER` object privilege for the immutable table or the `ALTER ANY TABLE` system privilege.

Use the `ALTER TABLE` statement with the `NO DROP` or `NO DELETE` clauses to alter the definition of an immutable table.

Note that you cannot reduce the retention period for an immutable table.

Example 19-28 Modifying the Retention Period for an Immutable Table

The following statement modifies the definition of the immutable table `trade_ledger` and specifies that it cannot be dropped if the newest row is less than 50 days old. The previous value for `NO DROP` clause was 30 days.

```
ALTER TABLE trade_ledger NO DROP UNTIL 50 DAYS IDLE;
```

Example 19-29 Modifying the Retention Period for Immutable Tables Rows

The following statement modifies the definition of the immutable table `trade_ledger` and specifies that a row cannot be deleted until 120 days after it was created.

```
ALTER TABLE trade_ledger NO DELETE UNTIL 120 DAYS AFTER INSERT;
```

19.17.5 Deleting Rows from Immutable Tables

Only rows that are outside the specified retention period can be deleted from an immutable table.

The `SYS` user or the owner of the schema can delete immutable table rows.

Use the `DBMS_IMMUTABLE_TABLE.DELETE_EXPIRED_ROWS` procedure to delete all rows that are beyond the specified retention period or obsolete rows that were created before a specified time.

Example 19-30 Deleting all Expired Rows from an Immutable Table

The following example, when connected as `SYS`, deletes all rows in the immutable table `trade_ledger` that are outside the retention window. The number of rows deleted is stored in the output parameter `num_rows`.

```
DECLARE
   num_rows NUMBER;
BEGIN
   DBMS_IMMUTABLE_TABLE.DELETE_EXPIRED_ROWS('EXAMPLES','TRADE_LEDGER', NULL, 
num_rows);
   DBMS_OUTPUT.PUT_LINE('Number_of_rows_deleted = ' || num_rows);
END;
/
```

Example 19-31 Deleting Eligible Rows Based on their Creation Time

The following example when connected as `SYS`, deletes obsolete rows that were created 30 days before the current system date. The number of rows deleted is stored in the output parameter `num_rows`.

```
DECLARE
       num_rows NUMBER;
BEGIN
       DBMS_IMMUTABLE_TABLE.DELETE_EXPIRED_ROWS('EXAMPLES','TRADE_LEDGER', 
SYSDATE-30, num_rows);
       DBMS_OUTPUT.PUT_LINE('Number_of_rows_deleted=' || num_rows);
END;
/
```

19.17.6 Dropping Immutable Tables

An immutable table can be dropped if it is empty or after it has not been modified for a period of time that is defined by its retention period.

The immutable table must be contained in your schema or you must have the `DROP ANY TABLE` system privilege.

Use the DROP TABLE statement to drop an immutable table. Dropping an immutable table removes its definition from the data dictionary, deletes all its rows, and deletes any indexes and triggers defined on the table.

The following statement drops the immutable table named trade_ledger in the examples schema:

```
DROP TABLE examples.trade_ledger;
```

19.17.7 Immutable Tables Data Dictionary Views

Data dictionary views provide information about immutable tables in the database.

Query one of the following views for information about immutable tables: DBA_IMMUTABLE_TABLES, USER_IMMUTABLE_TABLES, or ALL_IMMUTABLE_TABLES. The information includes the row retention period and table retention period. The DBA view describes all the immutable tables in the database, ALL view describes all immutable tables accessible to the user, and USER view is limited to immutable tables owned by the user.

Example 19-32 Displaying Immutable Table Information

The following query displays details of the immutable table trade_ledger in the examples schema.

```
SELECT row_retention "Row Retention Period", row_retention_locked "Row
Retention Lock", table_inactivity_retention "Table Retention Period"
FROM dba_immutable_tables
WHERE table_name = 'TRADE_LEDGER';

Row Retention Period Row Retention Locked Table Retention Period
-------------------- -------------------- ----------------------
                 110                   NO                     16
```

19.18 Managing Blockchain Tables

Blockchain tables protect data that records important actions, assets, entities, and documents from unauthorized modification or deletion by criminals, hackers, and fraud. Blockchain tables prevent unauthorized changes made using the database and detect unauthorized changes that bypass the database.

- About Blockchain Tables
  Blockchain tables are insert-only tables that organize rows into a number of chains. Each row in a chain, except the first row, is chained to the previous row in the chain by using a cryptographic hash.
- Guidelines for Managing Blockchain Tables
  You can follow guidelines for creating and using blockchain tables.
- Creating Blockchain Tables
  You create a blockchain table using the CREATE BLOCKCHAIN TABLE statement. This statement creates the blockchain table in the specified schema and the table metadata in the data dictionary.
- Altering Blockchain Tables
  You can modify the retention period for the blockchain table and for rows within the blockchain table.

- Adding Certificates Used to Sign Blockchain Table Rows
  Certificates can be used to verify the signature of a blockchain table row.
- Adding the Certificate of a Certificate Authority to the Database
  The digital certificate used to sign blockchain table rows is issued by a Certificate Authority. The digital certificate of this Certificate Authority is used in the process of verifying the validity of a user's digital certificate.
- Deleting Certificates
  Delete any certificates that are no longer required to verify the signature of blockchain table rows.
- Adding a Signature to Blockchain Table Rows
  Signing a row sets a user signature for a previously created row. A signature is optional and provides additional security against tampering.
- Validating Data in Blockchain Tables
  The PL/SQL procedure `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` verifies that rows in a blockchain table were not modified since they were inserted. Being tamper-resistant is a key requirement for blockchain tables.
- Verifying the Integrity of Blockchain Tables
  Maintain the integrity of blockchain tables by continuously verifying that the blockchain table data has not been compromised.
- Deleting Rows from Blockchain Tables
  Only rows that are outside the retention period can be deleted from a blockchain table.
- Dropping Blockchain Tables
  A blockchain table can be dropped if it contains no rows or after it has not been modified for a period of time that is defined by its retention period.
- Determining the Data Format for Row Content to Compute Row Hash
  To compute the hash value for a row, the data format for row content is determined using the `DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_HASH` procedure.
- Determining the Data Format to Compute Row Signature
  You can determine the data format for the row content that is used to compute the signature of a row. The row signature is computed based on the hash value of that row.
- Displaying the Byte Values of Data in Blockchain Tables
  You can retrieve the byte values of data, both rows and columns, in a blockchain table.
- Blockchain Tables Data Dictionary Views
  Data dictionary views provide information about blockchain tables.

19.18.1 About Blockchain Tables

Blockchain tables are insert-only tables that organize rows into a number of chains. Each row in a chain, except the first row, is chained to the previous row in the chain by using a cryptographic hash.

Rows in a blockchain table are tamper-resistant. Each row contains a cryptographic hash value which is based on the data in that row and the hash value of the previous row in the chain. If a row is tampered with, the hash value of the row changes and this causes the hash value of the next row in the chain to change. For enhanced fraud protection, an optional user signature can be added to a row. If you sign a blockchain table row, a digital certificate must be used. While verifying the chains in a blockchain table, the database needs the certificate to verify the row signature.

Blockchain tables can be indexed and partitioned. You can control whether and when rows are deleted from a blockchain table. You can also control whether the blockchain table can be dropped. Blockchain tables can be used along with (regular) tables in transactions and queries.

Blockchain tables can be used to implement blockchain applications where the participants trust the Oracle Database, but want a means to verify that their data has not been tampered. The participants are different database users who trust Oracle Database to maintain a verifiable, tamper-resistant blockchain of transactions. All participants must have privileges to insert data into the blockchain table. The contents of the blockchain are defined and managed by the application. By leveraging a trusted provider with verifiable crypto-secure data management practices, such applications can avoid the distributed consensus requirements. This provides most of the protection of the distributed peer-to-peer blockchains, but with much higher throughput and lower transaction latency compared to peer-to-peer blockchains using distributed consensus.

Use blockchain tables when immutability of data is critical for your centralized applications and you need to maintain a tamper-resistant ledger of current and historical transactions. A blockchain table is a building block. You must define the triggers or stored procedures required to perform the tasks that will implement a centralized blockchain. Information Lifecycle Management (ILM) is used to manage the lifecycle of data in blockchain tables. When the data in one or more partitions of a blockchain table is old, it can be moved to cheaper storage using ILM techniques.

- Benefits of Using Blockchain Tables
  Blockchain tables address data protection challenges faced by enterprises and governments by focusing on protecting data from criminals, hackers, and fraud.
- Chaining Rows in Blockchain Tables
  A row in a blockchain table is chained to the previous row in the chain, and the chain of rows is verifiable by all participants.
- Hidden Columns in Blockchain Tables
  Each row in a blockchain table contains hidden columns whose values are managed by the database.

19.18.1.1 Benefits of Using Blockchain Tables

Blockchain tables address data protection challenges faced by enterprises and governments by focusing on protecting data from criminals, hackers, and fraud.

Traditional data security technologies focus on preventing unauthorized users from accessing vital data. Techniques include using passwords, privileges, encryption, and firewalls. Blockchain tables provide enhanced data security by preventing unauthorized modification or deletion of data that records important actions, assets, entities, and documents. Unauthorized modification of important records can result in loss of assets, loss of business, and possible legal issues.

Blockchain tables provide the following benefits:

- Prevents unauthorized modification of data by insiders or criminals who use stolen insider insider credentials

  This is achieved by making the table insert-only. The database does not permit modification or deletion of existing data by removing the ability of users to perform the following actions:

  – Update or delete rows
  – Change the blockchain table definition
  – Convert the blockchain table to an updatable table or vice-versa

  - Modify table metadata in database dictionary
- Prevents undetected modification of data by hackers

  The is achieved by using the following techniques:
  - Cryptographic chaining of blockchain table rows on insert by using database-calculated row hashes, which include the current row’s data and previous row’s hash and a corresponding PL/SQL function to verify the chains

    A change to any row causes the chain to break thereby indicating that rows were subject to tampering. Cryptographic chaining is effective even when hackers take control of the database or operating system. Data changes that bypass SQL can be detected using an Oracle-provided PL/SQL function.
  - Cryptographic digest of the blockchain table generated on request and signed with the database schema owner’s private key for non-repudiation

    The cryptographic digest is computed based on the content of the blockchain table (metadata columns for the last row of every chain in the table). Therefore, any data modification results in a change to the digest value. You can periodically compute a cryptographic digest and distribute it to safe repositories or interested parties. To detect cover-ups of unauthorized changes made by the database operator or highly sophisticated hackers, you can verify the digest on a range of rows between two timestamps by using an Oracle-provided PL/SQL function.
- Prevents undetected, unauthorized modifications of data by using stolen end-user credentials

  The is achieved by using the following techniques:
  - Cryptographic signing of new data by the end user by using an Oracle-provided PL/SQL function to insert a signature over the row data

    End users can use a digital certificate and a private key to cryptographically sign rows that they insert into a blockchain table. This guards against impersonation by another end user, hackers with stolen end-user credentials, or unauthorized users who bypass application credential checking. It can also help to verify that data being recorded has not been modified in transit or in the application, and that it was actually inserted by the end user. User’s signature provides non-repudiation because the end user cannot claim that the data was inserted by some other user if it is signed by their private key and verified by the public key using a PKI certificate provided by the user.
  - Cryptographic signing of the blockchain table digest by the table schema owner

    The signed digest for a blockchain table ensures that the end user data was received and recorded, signature provided by the end user matches the recorded data, and cryptographic digest includes the new data. It prevents repudiation of data by end users.
- Prevents unauthorized modifications made by using the database and detects unauthorized changes that bypass the database
- Integrates blockchain technology in the Oracle database, thereby enhancing data protection with minimal changes to existing applications and no new infrastructure requirements
- Enables users to mix database and regular tables in queries and transactions
- Enables the use of advanced Oracle database functionality, including analytics capabilities, on cryptographically secured data
- Enables users to maintain history of all transactions in regular tables by creating a paired blockchain table for an audit trail and recording the history in the blockchain table by using a triggered stored procedure on the original table

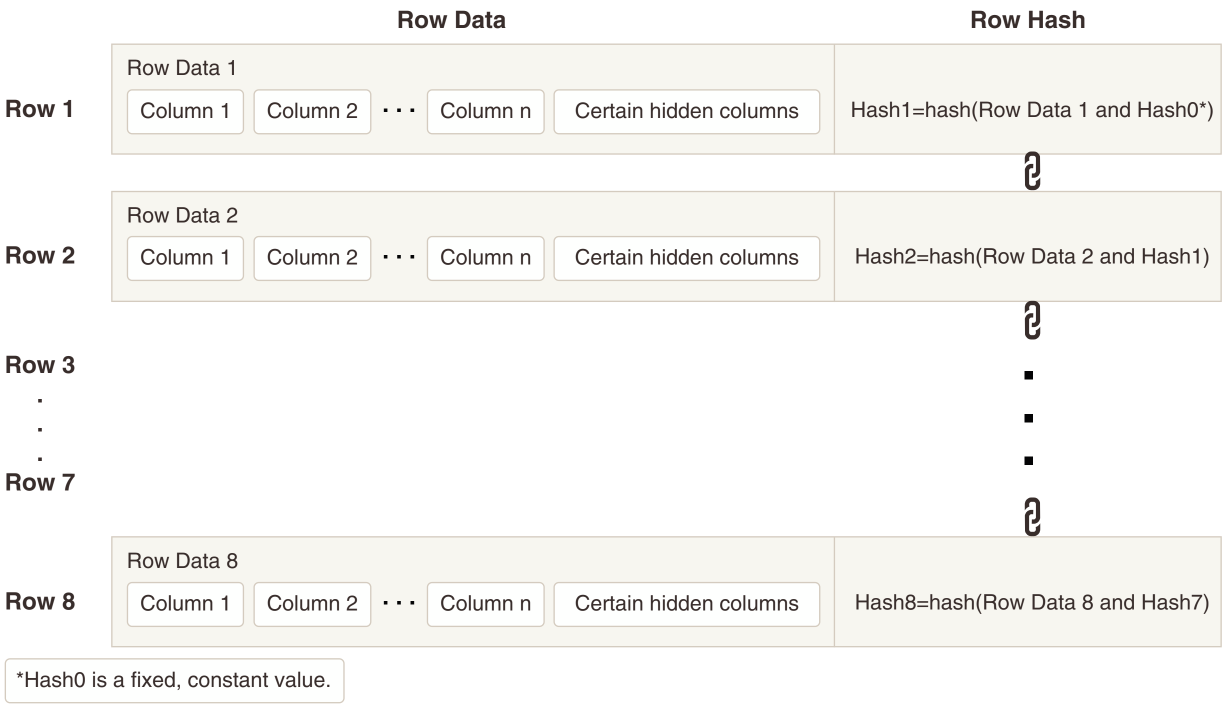
19.18.1.2 Chaining Rows in Blockchain Tables

A row in a blockchain table is chained to the previous row in the chain, and the chain of rows is verifiable by all participants.

For each Oracle Real Application Clusters (Oracle RAC) instance, a blockchain table contains thirty two chains, ranging from 0 through 31. A chain contains multiple rows and is identified by a unique combination of instance ID and chain ID. A row consists of user columns and hidden columns (created by the database). When a row is inserted, it is assigned a unique sequence number within the chain, and linked to the previous row in the chain. The sequence number of a row is 1 higher than the sequence number of the previous row in the chain. Each row, except the first row in a chain, has a unique previous row. A row can be uniquely identified using a combination of the instance ID, chain ID, and sequence number. It is recommended that you create an index on the combination of instance ID, chain ID, sequence number.

Figure 19-1 illustrates how rows are chained. The **row data** for a row consists of the user columns and certain hidden columns. The **hash value** of a row is computed based on the row data and the hash value of the previous row in the chain. The SHA2-512 hashing algorithm is used to compute the hash value. Rows are chained together by using the hash value.

Figure 19-1 Rows in a Single Chain of a Blockchain Table



A single transaction can insert rows into multiple blockchain tables. Rows in a blockchain table that are inserted by a single transaction are added to the same chain, and their positions on the chain respect the order in which they were inserted into the blockchain table. The chain for the rows is selected automatically by the database when the transaction commits.

When multiple users insert rows simultaneously into the same chain in a blockchain table, the sequence for adding the rows depends on the commit order of the transactions that inserted these rows.

Rows are linked to the blockchain when the transaction commits. Inserting a large number of rows in a single transaction results in a higher commit latency. Therefore, it is better to avoid inserting a very large number of rows in a single transaction.

Related Topics

- Blockchain Tables Reference
  You can independently verify the hash value and signature of a row by using its row content.

19.18.1.3 Hidden Columns in Blockchain Tables

Each row in a blockchain table contains hidden columns whose values are managed by the database.

Hidden columns are populated when an inserted row is committed. They are used to implement sequencing of rows and verify that data is tamper-resistant. You can create indexes on hidden columns. Hidden columns can only be displayed by explicitly including the column names in the query.

Table 19-9 Hidden Columns in Blockchain Tables

Column Name	Data Type	Description
`ORABCTAB_INST_ID$`	`NUMBER (22)`	Instance ID of the database instance into which the row is inserted.
`ORABCTAB_CHAIN_ID$`	`NUMBER (22)`	Chain ID of the chain, in the database instance, into which the row is inserted. Valid values for chain ID are 0 through 31.
`ORABCTAB_SEQ_NUM$`	`NUMBER(22)`	Sequence number of the row on the chain. Each row inserted into a chain of a blockchain table is assigned a unique sequence number that starts with 1. The sequence number of a row is 1 higher than the sequence number of the previous row in the chain. Missing rows can be detected using this column. The combination of instance ID, chain ID, and sequence number uniquely identifies a row in the blockchain table.
`ORABCTAB_CREATION_TIME$`	`TIMESTAMP WITH TIME ZONE`	Time, in UTC format, when a row is created.
`ORABCTAB_USER_NUMBER$`	`NUMBER (22)`	User ID of the database user who inserted the row.
`ORABCTAB_HASH$`	`RAW(2000)`	Hash value of the row. The hash value is computed based on the row content of the row and the hash value of the previous row in the chain.
`ORABCTAB_SIGNATURE$`	`RAW(2000)`	User signature of the row. The signature is computed using the hash value of the row.

Table 19-9 (Cont.) Hidden Columns in Blockchain Tables

Column Name	Data Type	Description
`ORABCTAB_SIGNATURE_ALG$`	`NUMBER(22)`	Signature algorithm used to produce the user signature of a signed row.
`ORABCTAB_SIGNATURE_CERT$`	`RAW(16)`	GUID of the certificate associated with the signature on a signed row.
`ORABCTAB_SPARE$`	`RAW(2000)`	This column is reserved for future use.

19.18.2 Guidelines for Managing Blockchain Tables

You can follow guidelines for creating and using blockchain tables.

Note:

The guidelines for creating tables are also applicable to blockchain tables. Additional guidelines are described in this section.

- For each chain in a database instance, periodically save the current hash and the corresponding sequence number outside the database. This enables you to verify that no chain in the blockchain table has been shortened or overwritten.
- In an Oracle Data Guard environment, consider using the maximum protection mode or maximum availability mode to avoid loss of data.
- Specify the Retention Period for the Blockchain Table
  Use the `NO DROP` clause in a `CREATE BLOCKCHAIN TABLE` statement to specify the retention period for the blockchain table.
- Specify the Retention Period for Rows in the Blockchain Table
  Use the `NO DELETE` clause in a `CREATE BLOCKCHAIN TABLE` statement to specify the retention period for rows in the blockchain table.
- Exporting and Importing Blockchain Tables with Oracle Data Pump
  To export or import blockchain tables, review these minimum requirements, restrictions, and guidelines.
- Restrictions for Blockchain Tables
  Using blockchain tables is subject to certain restrictions.

19.18.2.1 Specify the Retention Period for the Blockchain Table

Use the `NO DROP` clause in a `CREATE BLOCKCHAIN TABLE` statement to specify the retention period for the blockchain table.

If a blockchain table contains rows, it cannot be dropped while it is within the specified retention period.

Include one of the following clauses to specify retention period:

- `NO DROP`

Blockchain table cannot be dropped.

- `NO DROP UNTIL n DAYS IDLE`

  Blockchain table cannot be dropped if the newest row is less than *n* days old. The minimum value that is allowed for *n* is 0. However, to ensure the security of blockchain tables, it is recommended that you set the minimum value to at least 16.

  To set the table retention period to zero days, the dynamic initialization parameter `BLOCKCHAIN_TABLE_MAX_NO_DROP` must be set to its default value or zero. This setting is useful for testing blockchain tables. Note that when this parameter is set to zero, the only allowed retention period is 0 days. To subsequently set a non-zero retention period, you must reset the value of the `BLOCKCHAIN_TABLE_MAX_NO_DROP` parameter.

  Use the `ALTER TABLE` statement to increase the retention period for a blockchain table. You cannot reduce the retention period.

19.18.2.2 Specify the Retention Period for Rows in the Blockchain Table

Use the `NO DELETE` clause in a `CREATE BLOCKCHAIN TABLE` statement to specify the retention period for rows in the blockchain table.

The retention period controls when rows can be deleted from a blockchain table. Use one of the following options to specify the retention period:

- `NO DELETE [LOCKED]`

  Rows cannot be deleted from the blockchain table when `NO DELETE` is used.

  To ensure that rows are never deleted from the blockchain table, use the `NO DELETE LOCKED` clause in the `CREATE BLOCKCHAIN TABLE` statement. The `LOCKED` keyword specifies that the row retention setting cannot be modified.

- `NO DELETE UNTIL n DAYS AFTER INSERT [LOCKED]`

  Rows cannot be deleted until *n* days after the most recent row was added. You can use the `ALTER TABLE` statement with the `NO DELETE UNTIL` clause to modify this setting and increase the retention period. You cannot reduce the retention period.

  The minimum value for *n* is 16 days. If `LOCKED` is included, you cannot subsequently modify the row retention.

19.18.2.3 Exporting and Importing Blockchain Tables with Oracle Data Pump

To export or import blockchain tables, review these minimum requirements, restrictions, and guidelines.

If you use Oracle Data Pump with blockchain tables, then you can use only `CONVENTIONAL access_method`.

Blockchain tables are exported only under the following conditions:

- The `VERSION` parameter for the export is explicitly set to `21.0.0.0.0` or later.
- The `VERSION` parameter is set to (or defaults to) `COMPATIBLE`, and the database compatibility is set to `21.0.0.0.0` or later.
- The `VERSION` parameter is set to `LATEST`, and the database release is set to `21.0.0.0.0` or later.

If you attempt to use Oracle Data Pump options that are not supported with blockchain tables, then you receive errors when you attempt to use those options.

The following options of Oracle Data Pump are not supported with blockchain tables:

- `ACCESS_METHOD=[DIRECT_PATH, EXTERNAL_TABLE, INSERT_AS_SELECT]`
- `TABLE_EXISTS_ACTION=[REPLACE | APPEND | TRUNCATE]`

  These options result in errors when you attempt to use them to import data into an existing blockchain table.
- `CONTENT=DATA_ONLY`

  This option results in error when you attempt to import data into a blockchain table.
- `PARTITION_OPTIONS= [DEPARTITIONING | MERGE]`

  If you request departitioning using this option with blockchain tables, then the blockchain tables are skipped during departitioning.
- `NETWORK IMPORT`
- `TRANSPORTABLE`
- `SAMPLE, QUERY and REMAP_DATA`

19.18.2.4 Restrictions for Blockchain Tables

Using blockchain tables is subject to certain restrictions.

- The following data types are not supported with blockchain tables: `ROWID`, `UROWID`, `LONG`, object type, REF, varray, nested table, `TIMESTAMP WITH TIME ZONE`, `TIMESTAMP WITH LOCAL TIME ZONE`, `BFILE`, and `XMLType`.

  `XMLType` tables are not supported.
- Immutable tables can not be index-organized tables, `ORGANIZATION CUBE`, `ORGANIZATION EXTERNAL`, or hybrid partitioned.
- The maximum number of user-created columns is 980.
- When using Oracle Golden Gate or the `DBMS_ROLLING` package, DML statements (inserts, updates, and deletes) on blockchain tables succeed on the primary or source database, but are not replicated to the standby or target database.
- The following operations are not supported with blockchain tables:
  - Creating blockchain tables in the CDB root or application root
  - Updating rows, merging rows, adding columns, dropping columns, and renaming columns
  - Truncating the blockchain table
  - Dropping partitions
  - Sharded tables
  - Distributed transactions

    Inserting data into a blockchain table using Active Data Guard DML redirection is not supported because it internally relies on the distributed transaction framework using in-memory database links.
  - Direct-path loading and inserting data using parallel DML
  - Flashback table
  - Defining `BEFORE ROW` triggers that fire for update operations (other triggers are allowed)

  - XA transactions
  - Creating Automatic Data Optimization (ADO) policies
  - Creating Oracle Virtual Private Database (VPD) policies
  - Creating Oracle Label Security (OLS) policies
  - Online redefinition using the `DBMS_REDEFINITION` package
  - Transient Logical Standby and rolling upgrades

    DDL and DML on blockchain tables are not supported and not replicated.
  - Logical Standby and Oracle GoldenGate

    DDL and DML on blockchain tables succeed on the primary database but are not replicated to standby databases.
  - Converting a regular table to a blockchain table or vice versa
- Flashback Database and point-in-time recovery of a database undo the changes made to all tables, including blockchain tables. For example, if a database is flashed back to SCN 1000, or a backup is restored and recovered up to SCN 1000, all changes since SCN 1000 are removed from the database.

  Oracle Database does not prevent flashback and point-in-time recovery operations since they may be required to undo physical and logical corruptions. To detect any loss of data in a blockchain table, you must periodically publish a signed blockchain digest.
- Correctly enforcing retention policies in blockchain tables relies on the system time. The privilege to change the system time must not be granted widely, and changes to the system time must be audited and reviewed.

19.18.3 Creating Blockchain Tables

You create a blockchain table using the `CREATE BLOCKCHAIN TABLE` statement. This statement creates the blockchain table in the specified schema and the table metadata in the data dictionary.

Note:

Blockchain tables cannot be created in the root container and in an application root container.

The `CREATE TABLE` system privilege is required to create blockchain tables in your own schema. The `CREATE ANY TABLE` system privilege is required to create blockchain tables in another user's schema.

The `NO DROP`, `NO DELETE`, `HASHING USING`, and `VERSION` clauses are mandatory in a `CREATE BLOCKCHAIN TABLE` statement.

Example 19-33 Creating a Simple Blockchain Table

This example creates a blockchain table named `bank_ledger`, with the specified columns, in your schema. Rows can never be deleted. The blockchain table can be dropped only after 31 days of inactivity.

```
CREATE BLOCKCHAIN TABLE bank_ledger (bank VARCHAR2(128), deposit_date DATE,
deposit_amount NUMBER)
```

```
         NO DROP UNTIL 31 DAYS IDLE
         NO DELETE LOCKED
         HASHING USING "SHA2_512" VERSION "v1";
```

Example 19-34   Creating a Partitioned Blockchain Table

This example creates a blockchain table `bctab_part` with the specified columns and partitions. The table can be dropped only after 16 days of inactivity. Rows cannot be deleted until 25 days after they were inserted. The blockchain table is partitioned on the `trans_date` column.

```
CREATE BLOCKCHAIN TABLE bctab_part (trans_id number primary key, sender 
varchar2(50), recipient varchar2(50), trans_date DATE, amount number)
     NO DROP UNTIL 16 DAYS IDLE
     NO DELETE UNTIL 25 DAYS AFTER INSERT
     HASHING USING "SHA2_512" VERSION "v1"
     PARTITION BY RANGE(trans_date)
      (PARTITION p1 VALUES LESS THAN (TO_DATE('30-09-2019','dd-mm-yyyy')),
       PARTITION p2 VALUES LESS THAN (TO_DATE('31-12-2019','dd-mm-yyyy')),
       PARTITION p3 VALUES LESS THAN (TO_DATE('31-03-2020','dd-mm-yyyy')),
       PARTITION p4 VALUES LESS THAN (TO_DATE('30-06-2020','dd-mm-yyyy'))
      );
```

Example 19-35   Displaying Blockchain Table Columns (including hidden columns)

This example displays the details of columns, including hidden columns, in a blockchain table.

```
    Col ID Column Name                    Data Type                      Data 
Length
---------- ------------------------------ ------------------------------ 
-----------
         1 BANK                                                          
VARCHAR2                             128
         2 DEPOSIT_DATE                                                  
DATE                                   7
         3 DEPOSIT_AMOUNT                                                
NUMBER                                22
         4 ORABCTAB_INST_ID$                                             
NUMBER                                22
         5 ORABCTAB_CHAIN_ID$                                            
NUMBER                                22
         6 ORABCTAB_SEQ_NUM$                                             
NUMBER                                22
         7 ORABCTAB_CREATION_TIME$        TIMESTAMP(6) WITH TIME 
ZONE           13
         8 ORABCTAB_USER_NUMBER$                                         
NUMBER                                22
         9 ORABCTAB_HASH$                                                
RAW                                 2000
        10 ORABCTAB_SIGNATURE$                                           
RAW                                 2000
        11 ORABCTAB_SIGNATURE_ALG$                                       
NUMBER                                22
        12 ORABCTAB_SIGNATURE_CERT$                                      
RAW                                   16
        13 ORABCTAB_SPARE$
```

```
RAW                                    2000

13 rows selected.
```

19.18.4 Altering Blockchain Tables

You can modify the retention period for the blockchain table and for rows within the blockchain table.

The retention period cannot be reduced while altering a blockchain table definition. For example, assume you create a blockchain table and set the retention period to 30 days. You cannot subsequently alter it and set the retention period to 20 days.

- Use the `ALTER TABLE` statement with the `NO DROP` or `NO DELETE` clauses. Using the `NO DELETE LOCKED` clause specifies that rows can never be deleted from the blockchain table.

  The following statement modifies the definition of the blockchain table `bank_ledger` and specifies that it cannot be dropped if the newest row is less than 16 days old.

  ```
  ALTER TABLE bank_ledger NO DROP UNTIL 16 DAYS IDLE;
  ```

  The following statement modifies the definition of the blockchain table `bctab` and specifies that rows cannot be deleted until 20 days after they were created. The `LOCKED` clause indicates that this setting can never be modified.

  ```
  ALTER TABLE bctab NO DELETE UNTIL 20 DAYS AFTER INSERT LOCKED;
  ```

19.18.5 Adding Certificates Used to Sign Blockchain Table Rows

Certificates can be used to verify the signature of a blockchain table row.

You need to obtain an X.509 digital certificate from a Certificate Authority (CA). This certificate is added to the database, as a BLOB, and then used to add and verify the signature of one or more blockchain table rows. Multiple certificates can be used to sign rows in one blockchain table. Use OpenSSL APIs to manipulate digital certificates.

The digital certificate to be added must be stored as a `BLOB` in the database. The `BLOB` can be within a directory object.

- Use the `DBMS_USER_CERTS.ADD_CERTIFICATE` procedure to add a certificate.

  When a certificate is added to the database, it is assigned a unique certificate ID. This ID is the output of the `DBMS_USER_CERTS.ADD_CERTIFICATE` procedure. The certificate ID is used when adding and verifying signatures for a blockchain table row. You must remember this certificate ID, else you cannot use the associated digital certificate.

Example 19-36 Adding a Digital Certificate to the Database

This example adds the digital certificate that is stored, in binary format, in the file `u1_cert.der`. This file is stored in the `MY_DIR` directory object. Procedures in the `DBMS_LOB` package are used to open the certificate and read its contents into the variable `buffer`. The variable `cert_id` stores the procedure output, the certificate ID.

```
DECLARE
    file       BFILE;
```

```
    buffer     BLOB;
    amount     NUMBER := 32767;
    cert_id  RAW(16);
BEGIN
    file := BFILENAME('MY_DIR', 'u1_cert.der');
    DBMS_LOB.FILEOPEN(file);
    DBMS_LOB.READ(file, amount, 1, buffer);
    DBMS_LOB.FILECLOSE(file);
    DBMS_USER_CERTS.ADD_CERTIFICATE(buffer, cert_id);
    DBMS_OUTPUT.PUT_LINE('Certificate ID = ' || cert_id);
END;
/
Certificate ID = 9D267F1C280B60D8E053E5885A0A25FA

PL/SQL procedure successfully completed.
```

Example 19-37 Viewing Information About Certificates

This example displays information about the existing certificates by querying the `DBA_CERTIFICATES` data dictionary view. Other views that contain information about certificates are `CDB_CERTIFICATES` and `USER_CERTIFICATES`.

```
SELECT user_name, distinguished_name, UTL_RAW.LENGTH(certificate_id)
CERT_ID_LEN, DBMS_LOB.GETLENGTH(certificate) CERT_LEN
FROM DBA_CERTIFICATES ORDER BY user_name;

USER_NAME  DISTINGUISHED_NAME
----------
---------------------------------------------------------------------
CERT_ID_LEN   CERT_LEN
------------- ----------
U1          CN=USER1,OU=Americas,O=oracle,L=redwoodshores,ST=CA,C=US
            16        835

U2          CN=USER2,OU=IT-Department,O=Global-Security,L=London,ST=London,C=GB
            16       1465
```

19.18.6 Adding the Certificate of a Certificate Authority to the Database

The digital certificate used to sign blockchain table rows is issued by a Certificate Authority. The digital certificate of this Certificate Authority is used in the process of verifying the validity of a user's digital certificate.

A Certificate Revocation List (CRL) stores the list of digital certificates that were revoked by the Certificate Authority (CA) before their specified expiration date. The CRL and the digital certificate of the CA are used during the process of validating user digital certificates. Before a user signs a row using a digital certificate, the CRL is checked to verify that the digital certificate has not been revoked. The digital certificate of the CA is used to verify the authenticity of the CRL.

You must download the CRL associated with your Certificate Authority and store it in the `WALLET_ROOT/`*`PDB_GUID`*`/bctable/crl` directory. `WALLET_ROOT` is an initialization parameter that specifies the path to the root of a directory tree containing a subdirectory for each pluggable database (PDB) and *`PDB_GUID`* is the GUID of the pluggable database (PDB) that contains the blockchain table.

The certificate to be added to the database must be stored as a BLOB in the database.

- Use the `DBMS_USER_CERTS.ADD_CERTIFICATE` procedure to add the digital certificate of a Certificate Authority to the database.

  The database assigns a unique certificate ID to the new certificate. This ID is the output parameter of the `DBMS_USER_CERTS.ADD_CERTIFICATE` procedure. You must note down this certificate ID for later use.

After the certificate is added, rename the downloaded CRL as *`ca_cert_id`*`.crl`, where *`ca_cert_id`* is the unique certificate ID of the Certificate Authority. The file name must be in this format for the database to be able to use it to verify the validity of user digital certificates.

19.18.7 Deleting Certificates

Delete any certificates that are no longer required to verify the signature of blockchain table rows.

To delete a certificate from the database, you must either have the `SYSBA` privilege or be the owner of the certificate. You must also know the GUID that was generated when the certificate was added to the database.

- Use the `DBMS_USER_CERTS.DROP_CERTIFICATE` procedure to delete a certificate.

Example 19-38 Deleting a Certificate

This example deletes the certificate whose GUID is 9CCC45ABA31D5DC2E0532A26C40A860F.

```
declare
    certificate_guid RAW(16):='9CCC45ABA31D5DC2E0532A26C40A860F';
begin
    DBMS_USER_CERTS.DROP_CERTIFICATE(certificate_guid);
end;
```

19.18.8 Adding a Signature to Blockchain Table Rows

Signing a row sets a user signature for a previously created row. A signature is optional and provides additional security against tampering.

You must use a digital certificate when adding a signature to a blockchain table row. The signature is validated using the specified digital certificate and signature algorithm. The signature algorithms supported are SIGN_ALGO_RSA_SHA2_256, SIGN_ALGO_RSA_SHA2_384, and SIGN_ALGO_RSA_SHA2_512.

Before adding a user signature to a row, Oracle Database verifies that the current user owns the row being updated, the hash (if provided) matches the stored hash value of the row, and the digital certificate used to sign the row is valid. The database checks the Certificate Revocation List (CRL) file for the list of digital certificates that were revoked by the Certificate Authority before their scheduled expiration date. If the certificate of the Certificate Authority who issued the user's digital certificate is not added to the database, or the CRL file is not in the specified location, the user certificate is assumed to be valid.

The prerequisites for signing a blockchain table row are as follows:

- You must have the `INSERT` privilege on the blockchain table.
- The existing signature of the row to which a signature is being added must be NULL.

- The CRL of the Certificate Authority who issued the digital certificate used to sign the row must be stored in the `WALLET_ROOT/`*`PDB_GUID`*`/bctable/crl` directory.

  `WALLET_ROOT` is an initialization parameter that specifies the path to the root of a directory tree containing a subdirectory for each pluggable database (PDB) and `PDB_GUID` represents the GUID of the pluggable database (PDB) that contains the blockchain table.
- The name of the CRL file must be in the format *`ca_cert_id`*`.crl`, where *`ca_cert_id`* represents the unique certificate ID of the Certificate Authority.

To add a signature to an existing blockchain table row:

- Run the `DBMS_BLOCKCHAIN_TABLE.SIGN_ROW` procedure.

  Specify the following input values: blockchain table name, schema that contains the blockchain table, instance ID, chain ID, sequence ID, user signature, certificate ID of the user's digital certificate, and signature algorithm.

Note:

The `DBMS_BLOCKCHAIN_TABLE.SIGN_ROW` procedure depends on information specific to a pluggable database (PDB) and is applicable only to rows that were inserted in the current PDB by users, applications, or utilities other than Oracle Data Pump. For example, suppose you insert a row into a blockchain table in the PDB `my_pdb1`, commit the transaction, use Oracle Data Pump to export the blockchain table, and use Oracle Data Pump to import the blockchain table into the PDB `my_pdb2`. If you try to sign this row in the PDB `my_pdb2` by using the `DBMS_BLOCKCHAIN_TABLE.SIGN_ROW` procedure, an exception is raised.

Before you use Oracle Data Pump to create a copy of the blockchain table, you must sign all rows in a blockchain table that need to be signed.

Example 19-39 Signing a Blockchain Table Row

This example adds a signature to the row in the `bank_ledger` table with bank name as 'my_bank'. This table is in the `examples` schema. The signature is computed outside the database, by using standard OpenSSL commands, and stored in binary format in the file `ulr1_sign.dat`. The signature algorithm used is `DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_RSA_SHA2_512`. The variable `cert_guid` represents the GUID of the certificate that was added to the database and used to generate the signature.

```
DECLARE
        inst_id binary_integer;
        chain_id binary_integer;
        sequence_no binary_integer;
        file BFILE;
        amount NUMBER;
        signature RAW(2000);
        cert_guid RAW (16) := HEXTORAW('9CCC45ABA31D5DC2E0532A26C40A860F');
BEGIN
        SELECT ORABCTAB_INST_ID$, ORABCTAB_CHAIN_ID$, ORABCTAB_SEQ_NUM$ INTO
inst_id, chain_id, sequence_no
                FROM bank_ledger WHERE bank='my_bank';
        file := bfilename('MY_DIR1', 'u1r1_sign.dat');
        DBMS_LOB.FILEOPEN(file);
        dbms_lob.READ(file, amount, 1, signature);
        dbms_lob.FILECLOSE(file);
        DBMS_BLOCKCHAIN_TABLE.SIGN_ROW('EXAMPLES','BANK_LEDGER', inst_id,
chain_id, sequence_no, NULL, signature, cert_guid,
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_RSA_SHA2_512);
END;
/

PL/SQL procedure successfully completed.

SQL> SELECT bank, UTL_RAW.LENGTH(ORABCTAB_SIGNATURE$) sign_len,
ORABCTAB_SIGNATURE_ALG$,
  2    UTL_RAW.LENGTH(ORABCTAB_SIGNATURE_CERT$) sign_cert_guid_len
  3    FROM examples.bank_ledger ORDER BY bank;

BANK               SIGN_LEN ORABCTAB_SIGNATURE_ALG$ SIGN_CERT_GUID_LEN
---------------- ---------- ----------------------- ------------------
my_bank                 512                       1                 16
bank2                   256                       3                 16
```

Related Topics

- Adding the Certificate of a Certificate Authority to the Database
  The digital certificate used to sign blockchain table rows is issued by a Certificate Authority. The digital certificate of this Certificate Authority is used in the process of verifying the validity of a user's digital certificate.
- *Oracle Database PL/SQL Packages and Types Reference*

19.18.9 Validating Data in Blockchain Tables

The PL/SQL procedure `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` verifies that rows in a blockchain table were not modified since they were inserted. Being tamper-resistant is a key requirement for blockchain tables.

You must have the `SELECT` privilege on the blockchain table to run this procedure.

- Use the `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` procedure to verify the integrity of the hash column in a blockchain table. If a row contains a signature, the signature can be verified.

  You can validate all rows in the blockchain table or specify criteria to filter rows that must be validated. Rows can be filtered using the instance ID, chain ID, or row creation time.

Example 19-40 Validating Blockchain Table Rows In a Specific Instance

The following PL/SQL block verifies that the rows in the blockchain table `bank_ledger`, with instance IDs between 1 and 4, have not been tampered with since they were created.

Because the `verify_signature` parameter of the `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` procedure is omitted, the default value of TRUE is used. The row contents and the row signature (if present) are verified. If the `verify_signature` parameter is set to FALSE, the row contents are verified, but the row signature is not. You may chose to skip signature verification to conserve the additional time and resources spent on this process.

```
DECLARE
        verify_rows NUMBER;
        instance_id NUMBER;
BEGIN
```

```
        FOR instance_id IN 1 .. 4 LOOP
             DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS('EXAMPLES','BANK_LEDGER',
NULL, NULL, instance_id, NULL, verify_rows);
             DBMS_OUTPUT.PUT_LINE('Number of rows verified in instance Id '||
instance_id || ' = '|| verify_rows);
        END LOOP;
END;
/

Number of rows verified in instance Id 1 = 3
Number of rows verified in instance Id 2 = 12
Number of rows verified in instance Id 3 = 8
Number of rows verified in instance Id 4 = 10

PL/SQL procedure successfully completed.
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` procedure

19.18.10 Verifying the Integrity of Blockchain Tables

Maintain the integrity of blockchain tables by continuously verifying that the blockchain table data has not been compromised.

To verify the integrity of blockchain table data:

1. Verify the links between all the chains in the blockchain table by using the `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` procedure.

   For rows that contain a user signature, the row signature is also verified.

2. Generate a signature and signed digest for the blockchain table using the `DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST` function.

   Assume that this signature and signed digest were generated at time T1. Store these generated details, including the generated date and time, in your repository. The repository must be outside the database that stores the blockchain table. It can be another relational database.

3. At another point in time, generate a signature and signed digest for the blockchain table using the `DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST` function.

   Assume that this signature and signed digest were generated at time T2. Store the generated details, with generation date and time, in your repository.

4. Verify the integrity of rows that were created between time T1 and T2 by running the `DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN` procedure.

   The inputs to this procedure are the signed digests generated at times T1 and T2. The integrity of rows is verified using the time information that is part of the signed digest.

5. Repeat the process in Steps 2 through 4, at different time periods, to verify the integrity of rows inserted between different time periods.

For example, compute the signed digest at times T3 and T4 and then verify the integrity of rows created in the period between times T3 and T4.

It is recommended that you verify of the integrity of blockchain table data at regular intervals. This technique of continuous comparison and verification, between different periods of times, provides a guarantee that the rows in the blockchain table are not compromised.

- Generating a Signed Digest for Blockchain Tables
  The signed digest consists of metadata and data about the last row in each chain of a blockchain table. It can be used when verifying the integrity of blockchain table data.
- Verifying Blockchain Table Rows Created in a Specified Time Period
  Verifying rows created between specified time periods enables you to validate the integrity of the blockchain table during that period.

19.18.10.1 Generating a Signed Digest for Blockchain Tables

The signed digest consists of metadata and data about the last row in each chain of a blockchain table. It can be used when verifying the integrity of blockchain table data.

The database computes a signature that is based on the contents of the signed digest. The signature uses the private key and certificate of the blockchain table owner. You can use third-party tools to verify the signature generated by the database. Ensure that you store the signature and signed digest generated at various times in your repository.

An important aspect of maintaining the integrity of blockchain table data is to ensure that all rows are intact. Computing a signed digest provides a snapshot of the metadata and data about the last row in all chains at a particular time. You must store this information in a repository. Signed digests generated at various times comprise the input to the `DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN` procedure. Use this procedure to verify the integrity of rows created between two specified times.

Prerequisites

The certificate of blockchain table owner must be added to database using `DBMS_USER_CERTS.ADD_CERTIFICATE` procedure. The PKI private key and certificate of blockchain table owner must be stored in a wallet that is located in the *`WALLET_ROOT`*`/`*`pdb_guid`*`/bctable/` directory, where *`pdb_guid`* is the GUID of the PDB that contains the blockchain table. *`WALLET_ROOT`* specifies the path to the root of a directory tree containing a subdirectory for each PDB, under which a directory structure is used to store the various wallets associated with the PDB.

To generate a signed digest and signature for a blockchain table:

- Use the `DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST` function.
  The function first checks if the certificate of the blockchain table owner is valid. It then computes a signed digest of data type `BLOB` and a PL/SQL array version of the signed digest. The signed digest contains metadata and data of the last row in each chain of the blockchain table. The PL/SQL array identifies the last row in each chain of the signed digest. The function returns a signature that is based on the signed digest.

Note:

A signed digest contains table information specific to a pluggable database (PDB). Therefore, you can use this signed digest only in the PDB in which it was created and only for the table that was used to create the digest.

Example 19-41   Generating a Signed Digest and Signature for Blockchain Tables

This example computes the signed digest and generates a signature for the blockchain `EXAMPLES.BANK_LEDGER`. The signed digest is in binary format and consists of metadata and data of the last rows in each chain. It is stored in `signed_bytes`. The PL/SQL array version of the signed digest is stored in the output parameter `signed_row_array`. The GUID of the certificate used to generate the signature is stored in `certificate_guid`. The algorithm used is `DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_RSA_SHA2_512`.

```
DECLARE
   signed_bytes              BLOB:=EMPTY_BLOB();
   signed_row_array        SYS.ORABCTAB_ROW_ARRAY_T;
   certificate_guid        RAW(2000);
   signature               RAW(2000);
BEGIN
   signature := DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST('EXAMPLES',
                   'BANK_LEDGER', signed_bytes, signed_row_array,
                   certificate_guid,
dbms_blockchain_table.SIGN_ALGO_RSA_SHA2_512);
   DBMS_OUTPUT.PUT_LINE('Certificate GUID = ' || certificate_guid);
   DBMS_OUTPUT.PUT_LINE('Signature length = ' || UTL_RAW.LENGTH(signature));
   DBMS_OUTPUT.PUT_LINE('Number of chains = ' || signed_row_array.count);
   DBMS_OUTPUT.PUT_LINE('Signature content buffer length = ' ||
DBMS_LOB.GETLENGTH(signed_bytes));
END;
/

Certificate GUID = AF27H7FE3EEA473GE0783FE56A0AFCEB
Signature length = 256
Number of chains = 10
Signature content buffer length = 1248

PL/SQL procedure successfully completed.
```

Related Topics

- Format of the Signed Digest in Blockchain Tables
  The signed digest consists of metadata and data about the last row in each chain of a blockchain table.

19.18.10.2 Verifying Blockchain Table Rows Created in a Specified Time Period

Verifying rows created between specified time periods enables you to validate the integrity of the blockchain table during that period.

The `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` procedure verifies the integrity of all rows in the blockchain table. Instead of verifying the entire table every time, you can just verify the rows that were created since the most recent verification. For example, if you ran `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` two days ago, you can verify only the rows added since that verification.

To verify blockchain table rows created within a specified time period:

- Use the `DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN` procedure.

  The inputs to this procedure are two signed digests that were generated at different times by using the `DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST` function. The time period for verification is determined by using the information in the signed digests. The minimum row creation time from the first signed digest and the maximum row creation time from the second signed digest are considered. The output is the number of rows verified.

Note:

Both signed digests must be generated in the current pluggable database (PDB) and for the same blockchain table. For example, assume that you create a signed digest for a blockchain table in the PDB `my_pdb1`, use Oracle Data Pump to export the blockchain table, and then use Oracle Data Pump to import the blockchain table into the PDB `my_pdb2`. The signed digest created in the PDB `my_pdb1` cannot be used in the PDB `my_pdb2`. You need to create a new signed digest in the PDB `my_pdb2`.

Example 19-42 Verifying Blockchain Table Rows Created Between a Specified Time Period

This example verifies the rows created in the `EXAMPLES.BANK_LEDGER` blockchain table between a specified time period. The signed digest of the blockchain table at two different times is stored in `signed_bytes1` and `signed_bytes2`. The value of `signed_bytes1` is read from the `signed_digest_repo` table which is a repository for signed digests and signatures. The value of `signed_bytes2` is computed using the `DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST` function. Rows created between the minimum row creation time in `signed_bytes1` and the maximum row creation time in `signed_bytes2` are considered for the verification.

```
DECLARE
    signature               RAW(2000);
    sign_row_array          SYS.ORABCTAB_ROW_ARRAY_T;
    signed_bytes1           BLOB;
    certificate_guid        RAW(2000);
    signed_bytes2           BLOB;
    rows_verified           NUMBER;
BEGIN
    SELECT signed_digest INTO signed_bytes1 FROM signed_digest_repo WHERE
time>=SYSDATE-1;
    signature :=
DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST('EXAMPLES',
                      'BANK_LEDGER', signed_bytes2, sign_row_array,
certificate_guid);

DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN(signed_bytes2,signed_bytes1,
rows_verified);
    dbms_output.put_line('Rows verified = ' || rows_verified);
END;
/

Rows verified = 10
PL/SQL procedure successfully completed.
```

19.18.11 Deleting Rows from Blockchain Tables

Only rows that are outside the retention period can be deleted from a blockchain table.

The `SYS` user or the owner of the schema can delete rows from a blockchain table.

The PL/SQL procedure `DBMS_BLOCKCHAIN_TABLE.DELETE_EXPIRED_ROWS` deletes rows that are beyond the retention period from a blockchain table. You can either delete all rows outside the retention period or rows that were created before a specified date.

Example 19-43 Deleting Eligible Rows from a Blockchain Table

The following example, when connected as `SYS`, deletes all rows in the blockchain table `bank_ledger` that are outside the retention window. The number of rows deleted is stored in the output parameter `num_rows`.

```
DECLARE
        num_rows NUMBER;
BEGIN
        DBMS_BLOCKCHAIN_TABLE.DELETE_EXPIRED_ROWS('EXAMPLES','BANK_LEDGER',
NULL, num_rows);
        DBMS_OUTPUT.PUT_LINE('Number_of_rows_deleted=' || num_rows);
END;
/
Number_of_rows_deleted=2

PL/SQL procedure successfully completed.
```

Example 19-44 Deleting Eligible Rows Based on their Creation Time

The following example, when connected as `SYS`, deletes rows outside the retention period that were created before 10-OCT-2019. The number of rows deleted is stored in the output parameter `num_rows`.

```
DECLARE
        num_rows NUMBER;
BEGIN
        DBMS_BLOCKCHAIN_TABLE.DELETE_EXPIRED_ROWS('EXAMPLES','BANK_LEDGER',
TO_DATE('10-OCT-19','DD-MON-YY'), num_rows);
        DBMS_OUTPUT.PUT_LINE('Number_of_rows_deleted=' || num_rows);
END;
    Number_of_rows_deleted=5

    PL/SQL procedure successfully completed.
```

19.18.12 Dropping Blockchain Tables

A blockchain table can be dropped if it contains no rows or after it has not been modified for a period of time that is defined by its retention period.

The blockchain table must be contained in your schema or you must have the `DROP ANY TABLE` system privilege. It is recommended that you include the `PURGE` option when you drop a blockchain table.

- Use the `DROP TABLE` statement to drop a blockchain table. Dropping a blockchain table removes its definition from the data dictionary, deletes all its rows, and deletes any indexes and triggers defined on the blockchain table.

  The following command drops the blockchain table named `my_blockchain_table` in the `examples` schema:

  ```
  DROP TABLE examples.my_blockchain_table PURGE;
  ```

19.18.13 Determining the Data Format for Row Content to Compute Row Hash

To compute the hash value for a row, the data format for row content is determined using the `DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_HASH` procedure.

If you want to independently verify the hash value of a row that was computed by the database, first determine the data format for its row content (in bytes). Then use the SHA2-512 hashing algorithm on the combination of row content and hash value of the previous row in the chain.

To enable the database character set and the national character set to be changed without invalidating row hashes in blockchain tables, each row hash in a blockchain table is computed over normalized values for each column with a character data type or a character `LOB` data type. Specifically, the value in a `VARCHAR2` column or a `CHAR` column is converted to an AL32UTF8 representation before being hashed. The value in an `NVARCHAR2` column, an `NCHAR` column, a `CLOB` column, or an `NCLOB` column is converted to an AL16UTF16 representation before being hashed. A column value already in AL32UTF8 or AL16UTF16 is not converted but may be checked for illegal character codes.

`CHAR` and `NCHAR` values are further normalized by removing trailing blanks. A `CHAR` or `NCHAR` value consisting of all blanks is normalized to a single blank to avoid becoming a null.

Use the `DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_HASH` procedure to determine the data format for row content when computing the row hash. This procedure returns the bytes, in column position order, for the specified row followed by the hash value (in data format) of the previous row in the chain.

To specify a row, you must provide the instance ID, chain ID, and sequence number of the row.

Example 19-45 Verifying the Stored Row Hash Value

This example retrieves the data format for the row content of the most recently-added row in a specific database instance and chain of the `BANK_LEDGER` table. The `DBMS_CRYPTO.HASH` function is used to compute the hash value. To independently verify the hash value, the computed hash value is compared with the value retrieved from the database.

You must have the permissions required to run the `DBMS_CRYPTO` package. The value for data format must be 1.

```
set serveroutput on;

DECLARE
        row_data BLOB;
        row_id ROWID;
        row_hash RAW(64);
```

```
        computed_hash RAW(64);
        buffer RAW(4000);
        inst_id BINARY_INTEGER;
        chain_id BINARY_INTEGER;
        sequence_no BINARY_INTEGER;
BEGIN
        -- Get the row details and hash value of the most recently inserted 
row with the specified instance ID and chain ID
        SELECT MAX(ORABCTAB_SEQ_NUM$) INTO sequence_no
               FROM EXAMPLES.BANK_LEDGER
               WHERE ORABCTAB_INST_ID$=1 AND ORABCTAB_CHAIN_ID$=4;
        SELECT ORABCTAB_INST_ID$, ORABCTAB_CHAIN_ID$, ORABCTAB_SEQ_NUM$, 
ORABCTAB_HASH$ INTO inst_id, chain_id, sequence_no, row_hash
               FROM EXAMPLES.BANK_LEDGER
               WHERE ORABCTAB_INST_ID$=1 AND ORABCTAB_CHAIN_ID$=4 AND 
ORABCTAB_SEQ_NUM$ = sequence_no;
        -- Compute the row hash externally from row column bytes
        DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_HASH('EXAMPLES', 
'BANK_LEDGER', inst_id, chain_id, sequence_no, 1, row_data);
        computed_hash := DBMS_CRYPTO.HASH(row_data, DBMS_CRYPTO.HASH_SH512);
        -- Verify that the row's hash and externally computed hash are same
        if UTL_RAW.COMPARE(row_hash, computed_hash) = 0 THEN
                DBMS_OUTPUT.PUT_LINE('Hash verification successful');
        else
                DBMS_OUTPUT.PUT_LINE('Hash verification failed');
END IF;
END;

Hash verification successful

PL/SQL procedure successfully completed.
```

Related Topics

- Blockchain Tables Reference
  You can independently verify the hash value and signature of a row by using its row content.

19.18.14 Determining the Data Format to Compute Row Signature

You can determine the data format for the row content that is used to compute the signature of a row. The row signature is computed based on the hash value of that row.

Use the `DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_SIGNATURE` procedure to determine the data format for row content to compute the row signature. This procedure returns the bytes for the specified row, in row content format.

Example 19-46 Computing the Signature of a Row in the Blockchain Table

This example computes the bytes for the row with `bank` value 'my_bank' in the `examples.bank_ledger` table. The bytes are stored in the variable `row_data`.

```
DECLARE
        row_data BLOB;
        buffer RAW(4000);
        inst_id BINARY_INTEGER;
        chain_id BINARY_INTEGER;
        sequence_no BINARY_INTEGER;
        row_len BINARY_INTEGER;
BEGIN
        SELECT ORABCTAB_INST_ID$, ORABCTAB_CHAIN_ID$, ORABCTAB_SEQ_NUM$ INTO
inst_id, chain_id, sequence_no
             FROM EXAMPLES.BANK_LEDGER where bank='my_bank';

DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_SIGNATURE('EXAMPLES','BANK_LEDGER',ins
t_id, chain_id, sequence_no, 1, row_data);
        row_len := DBMS_LOB.GETLENGTH(row_data);
        DBMS_LOB.READ(row_data, row_len, 1, buffer);
END;
/

PL/SQL procedure successfully completed.
```

19.18.15 Displaying the Byte Values of Data in Blockchain Tables

You can retrieve the byte values of data, both rows and columns, in a blockchain table.

Use one of the following procedures to view the byte values of data in a blockchain table or a regular table:

- Use the `DBMS_TABLE_DATA.GET_BYTES_FOR_COLUMN` procedure to determine the column data, in bytes, for a single column.

  The following example determines the byte value for one `bank` column in the `bank_ledger` table of the `examples` schema.

  ```
  DECLARE
          row_id ROWID;
          col_data BLOB;
          buffer RAW(4000);
          data_len BINARY_INTEGER;
  begin
          SELECT rowid INTO row_id FROM bank_ledger WHERE bank='my_bank';
          DBMS_TABLE_DATA.GET_BYTES_FOR_COLUMN('EXAMPLES', 'BANK_LEDGER',
  row_id,'BANK', col_data);
          data_len := dbms_lob.getlength(col_data);
          DBMS_LOB.READ(col_data, data_len, 1, buffer);
          DBMS_OUTPUT.PUT_LINE('len=' || data_len || ', data=' ||
  RAWTOHEX(buffer));
  end;
  ```

- Use the `DBMS_TABLE_DATA.GET_BYTES_FOR_COLUMNS` procedure to determine the column data, in bytes, for a set of columns. The set of columns is provided to the procedure using a `VARRAY`.
- Use the `DBMS_TABLE_DATA.GET_BYTES_FOR_ROW` procedure to determine the row data, in bytes, for a single row.

The following example displays the row data, in bytes, for a specific row in the table `bank_ledger`.

```
DECLARE
        row_data blob;
        data_len binary_integer;
        row_id rowid;
        inst_id binary_integer;
        chain_id binary_integer;
        sequence_no binary_integer;
BEGIN
        SELECT rowid INTO row_id FROM bank_ledger WHERE bank='my_bank';
        DBMS_TABLE_DATA.GET_BYTES_FOR_ROW('EXAMPLES','BANK_LEDGER',row_id,
row_data);
        data_len := DBMS_LOB.GETLENGTH(row_data);
        DBMS_OUTPUT.PUT_LINE('Row data-length=' || data_len);
END;
/

Row data-length=908.

PL/SQL procedure successfully completed.
```

Related Topics

- Blockchain Tables Reference
  You can independently verify the hash value and signature of a row by using its row content.

19.18.16 Blockchain Tables Data Dictionary Views

Data dictionary views provide information about blockchain tables.

Query one of the following views: `DBA_BLOCKCHAIN_TABLES`, `ALL_BLOCKCHAIN_TABLES`, or `USER_BLOCKCHAIN_TABLES` for information about blockchain tables. Information includes the row retention period, table retention period, and hashing algorithm used to chain rows. The `DBA` view describes all the blockchain tables in the database, `ALL` view describes all blockchain tables accessible to the user, and `USER` view is limited to blockchain tables owned by the user.

Example 19-47 Displaying Blockchain Table Information

The following command displays the details of blockchain table `bank_ledger` in the `examples` schema.

```
SELECT row_retention "Row Retention Period", row_retention_locked "Row
Retention Lock", table_inactivity_retention "Table Retention Period",
hash_algorithm "Hash Algorithm"
FROM dba_blockchain_tables WHERE table_name='BANK_LEDGER';

Row Retention Period Row Retention Lock   Table  Retention Period Hash
Algorithm
-------------------- ------------------  ------------------------
--------------
                  16 YES                                       31 SHA2_512
```

19.19 Tables Data Dictionary Views

You can query a set of data dictionary views for information about tables.

View	Description

`DBA_TABLES`
`ALL_TABLES`
`USER_TABLES`
	`DBA` view describes all relational tables in the database. `ALL` view describes all tables accessible to the user. `USER` view is restricted to tables owned by the user. Some columns in these views contain statistics that are generated by the `DBMS_STATS` package or `ANALYZE` statement.

`DBA_TAB_COLUMNS`
`ALL_TAB_COLUMNS`
`USER_TAB_COLUMNS`
	These views describe the columns of tables, views, and clusters in the database. Some columns in these views contain statistics that are generated by the `DBMS_STATS` package or `ANALYZE` statement.

`DBA_ALL_TABLES`
`ALL_ALL_TABLES`
`USER_ALL_TABLES`
	These views describe all relational and object tables in the database. Object tables are not specifically discussed in this book.

`DBA_TAB_COMMENTS`
`ALL_TAB_COMMENTS`
`USER_TAB_COMMENTS`
	These views display comments for tables and views. Comments are entered using the `COMMENT` statement.

`DBA_COL_COMMENTS`
`ALL_COL_COMMENTS`
`USER_COL_COMMENTS`
	These views display comments for table and view columns. Comments are entered using the `COMMENT` statement.

`DBA_EXTERNAL_TABLES`
`ALL_EXTERNAL_TABLES`
`USER_EXTERNAL_TABLES`
	These views list the specific attributes of external tables in the database.

`DBA_EXTERNAL_LOCATIONS`
`ALL_EXTERNAL_LOCATIONS`
`USER_EXTERNAL_LOCATIONS`
	These views list the data sources for external tables.

`DBA_XTERNAL_PART_TABLES`
`ALL_XTERNAL_PART_TABLES`
`USER_XTERNAL_PART_TABLES`
	These views list the specific attributes of partitioned external tables in the database.

`DBA_XTERNAL_TAB_PARTITIONS`
`ALL_XTERNAL_TAB_PARTITIONS`
`USER_XTERNAL_TAB_PARTITIONS`
	These views list the partition-level information for partitioned external tables in the database.

`DBA_XTERNAL_TAB_SUBPARTITIONS`
`ALL_XTERNAL_TAB_SUBPARTITIONS`
`USER_XTERNAL_TAB_SUBPARTITIONS`
	These views list the subpartition-level information for partitioned external tables in the database.

`DBA_XTERNAL_LOC_PARTITIONS`
`ALL_XTERNAL_LOC_PARTITIONS`
`USER_XTERNAL_LOC_PARTITIONS`
	These views list the data sources for partitions in external tables.

View	Description
DBA_XTERNAL_LOC_SUBPARTITIONS ALL_XTERNAL_LOC_SUBPARTITIONS USER_XTERNAL_LOC_SUBPARTITIONS	These views list the data sources for subpartitions in external tables.
DBA_TAB_HISTOGRAMS ALL_TAB_HISTOGRAMS USER_TAB_HISTOGRAMS	These views describe histograms on tables and views.
DBA_TAB_STATISTICS ALL_TAB_STATISTICS USER_TAB_STATISTICS	These views contain optimizer statistics for tables.
DBA_TAB_COL_STATISTICS ALL_TAB_COL_STATISTICS USER_TAB_COL_STATISTICS	These views provide column statistics and histogram information extracted from the related TAB_COLUMNS views.
DBA_TAB_MODIFICATIONS ALL_TAB_MODIFICATIONS USER_TAB_MODIFICATIONS	These views describe tables that have been modified since the last time table statistics were gathered on them. They are not populated immediately, but after a time lapse (usually 3 hours).
DBA_ENCRYPTED_COLUMNS ALL_ENCRYPTED_COLUMNS USER_ENCRYPTED_COLUMNS	These views list table columns that are encrypted, and for each column, lists the encryption algorithm in use.
DBA_UNUSED_COL_TABS ALL_UNUSED_COL_TABS USER_UNUSED_COL_TABS	These views list tables with unused columns, as marked by the ALTER TABLE ... SET UNUSED statement.
DBA_PARTIAL_DROP_TABS ALL_PARTIAL_DROP_TABS USER_PARTIAL_DROP_TABS	These views list tables that have partially completed DROP COLUMN operations. These operations could be incomplete because the operation was interrupted by the user or a system failure.

Example: Displaying Column Information

Column information, such as name, data type, length, precision, scale, and default data values can be listed using one of the views ending with the _COLUMNS suffix. For example, the following query lists all of the default column values for the emp and dept tables:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, LAST_ANALYZED
    FROM DBA_TAB_COLUMNS
    WHERE OWNER = 'HR'
    ORDER BY TABLE_NAME;
```

The following is the output from the query:

```
TABLE_NAME           COLUMN_NAME          DATA_TYPE  DATA_LENGTH LAST_ANALYZED
-------------------- -------------------- ---------- ------------ -------------
COUNTRIES            COUNTRY_ID           CHAR                  2 05-FEB-03
COUNTRIES            COUNTRY_NAME         VARCHAR2             40 05-FEB-03
COUNTRIES            REGION_ID            NUMBER               22 05-FEB-03
DEPARTMENTS          DEPARTMENT_ID        NUMBER               22 05-FEB-03
DEPARTMENTS          DEPARTMENT_NAME      VARCHAR2             30 05-FEB-03
DEPARTMENTS          MANAGER_ID           NUMBER               22 05-FEB-03
DEPARTMENTS          LOCATION_ID          NUMBER               22 05-FEB-03
EMPLOYEES            EMPLOYEE_ID          NUMBER               22 05-FEB-03
EMPLOYEES            FIRST_NAME           VARCHAR2             20 05-FEB-03
EMPLOYEES            LAST_NAME            VARCHAR2             25 05-FEB-03
EMPLOYEES            EMAIL                VARCHAR2             25 05-FEB-03
.
.
.
LOCATIONS            COUNTRY_ID           CHAR                  2 05-FEB-03
REGIONS              REGION_ID            NUMBER               22 05-FEB-03
REGIONS              REGION_NAME          VARCHAR2             25 05-FEB-03

51 rows selected.
```

See Also:

- *Oracle Database Object-Relational Developer's Guide* for information about object tables
- *Oracle Database SQL Tuning Guide* for information about histograms and generating statistics for tables
- "About Analyzing Tables, Indexes, and Clusters"

20 Managing Indexes

Indexes can provide faster data access. You can create, alter, monitor, and drop indexes.

- About Indexes
  Indexes are optional structures associated with tables and clusters that allow SQL queries to execute more quickly against a table.
- Guidelines for Managing Indexes
  You can follow guidelines for managing indexes.
- Creating Indexes
  You can create several different types of indexes. You can create indexes explicitly, and you can create indexes associated with constraints.
- Altering Indexes
  You can alter an index by completing tasks such as changing its storage characteristics, rebuilding it, making it unusable, or making it visible or invisible.
- Monitoring Space Use of Indexes
  If key values in an index are inserted, updated, and deleted frequently, then the index can lose its acquired space efficiency over time.
- Dropping Indexes
  You can drop an index with the `DROP INDEX` statement.
- Managing Automatic Indexes
  You can use the automatic indexing feature to configure and use automatic indexes in an Oracle database to improve database performance.
- Indexes Data Dictionary Views
  You can query a set of data dictionary views for information about indexes.

20.1 About Indexes

Indexes are optional structures associated with tables and clusters that allow SQL queries to execute more quickly against a table.

Just as the index in this manual helps you locate information faster than if there were no index, an Oracle Database index provides a faster access path to table data. You can use indexes without rewriting any queries. Your results are the same, but you see them more quickly.

Oracle Database provides several indexing schemes that provide complementary performance functionality. These are:

- B-tree indexes: the default and the most common
- B-tree cluster indexes: defined specifically for cluster
- Hash cluster indexes: defined specifically for a hash cluster
- Global and local indexes: relate to partitioned tables and indexes
- Reverse key indexes: most useful for Oracle Real Application Clusters applications
- Bitmap indexes: compact; work best for columns with a small set of values

- Function-based indexes: contain the precomputed value of a function/expression
- Domain indexes: specific to an application or cartridge.

Indexes are logically and physically independent of the data in the associated table. Being independent structures, they require storage space. You can create or drop an index without affecting the base tables, database applications, or other indexes. The database automatically maintains indexes when you insert, update, and delete rows of the associated table. If you drop an index, all applications continue to work. However, access to previously indexed data might be slower.

See Also:

- *Oracle Database Concepts* for an overview of indexes
- Managing Space for Schema Objects

20.2 Guidelines for Managing Indexes

You can follow guidelines for managing indexes.

- Create Indexes After Inserting Table Data
  Data is often inserted or loaded into a table using either the SQL*Loader or an import utility. It is more efficient to create an index for a table after inserting or loading the data. If you create one or more indexes before loading data, then the database must update every index as each row is inserted.
- Index the Correct Tables and Columns
  Follow guidelines about tables and columns that are suitable for indexing.
- Order Index Columns for Performance
  The order of columns in the `CREATE INDEX` statement can affect query performance. In general, specify the most frequently used columns first.
- Limit the Number of Indexes for Each Table
  A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified.
- Drop Indexes That Are No Longer Required
  It is best practice to drop indexes that are no longer required.
- Indexes and Deferred Segment Creation
  Index segment creation is deferred when the associated table defers segment creation. This is because index segment creation reflects the behavior of the table with which it is associated.
- Estimate Index Size and Set Storage Parameters
  Estimating the size of an index before creating one can facilitate better disk space planning and management.
- Specify the Tablespace for Each Index
  Indexes can be created in any tablespace. An index can be created in the same or different tablespace as the table it indexes.

- Consider Parallelizing Index Creation
  You can parallelize index creation, much the same as you can parallelize table creation. Because multiple processes work together to create the index, the database can create the index more quickly than if a single server process created the index sequentially.
- Consider Creating Indexes with NOLOGGING
  You can create an index and generate minimal redo log records by specifying `NOLOGGING` in the `CREATE INDEX` statement.
- Understand When to Use Unusable or Invisible Indexes
  Use unusable or invisible indexes when you want to improve the performance of bulk loads, test the effects of removing an index before dropping it, or otherwise suspend the use of an index by the optimizer.
- Understand When to Create Multiple Indexes on the Same Set of Columns
  You can create multiple indexes on the same set of columns when the indexes are different in some way. For example, you can create a B-tree index and a bitmap index on the same set of columns.
- Consider Costs and Benefits of Coalescing or Rebuilding Indexes
  Improper sizing or increased growth can produce index fragmentation. To eliminate or reduce fragmentation, you can rebuild or coalesce the index. But before you perform either task weigh the costs and benefits of each option and choose the one that works best for your situation.
- Consider Cost Before Disabling or Dropping Constraints
  Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether to disable or drop a `UNIQUE` or `PRIMARY KEY` constraint.
- Consider Using the In-Memory Column Store to Reduce the Number of Indexes
  The In-Memory Column Store is an optional portion of the system global area (SGA) that stores copies of tables, table partitions, and other database objects that is optimized for rapid scans. In the In-Memory Column Store, table data is stored by column rather than row in the SGA.

See Also:

- *Oracle Database Concepts* for conceptual information about indexes and indexing, including descriptions of the various indexing schemes offered by Oracle
- *Oracle Database SQL Tuning Guide* and *Oracle Database Data Warehousing Guide* for information about bitmap indexes
- *Oracle Database Data Cartridge Developer's Guide* for information about defining domain-specific operators and indexing schemes and integrating them into the Oracle Database server

20.2.1 Create Indexes After Inserting Table Data

Data is often inserted or loaded into a table using either the SQL*Loader or an import utility. It is more efficient to create an index for a table after inserting or loading the data. If you create one or more indexes before loading data, then the database must update every index as each row is inserted.

Creating an index on a table that already has data requires sort space. Some sort space comes from memory allocated for the index creator. The amount for each user is determined by the initialization parameter `SORT_AREA_SIZE`. The database also swaps sort information to and from temporary segments that are only allocated during the index creation in the user's temporary tablespace.

Under certain conditions, data can be loaded into a table with SQL*Loader direct-path load, and an index can be created as data is loaded.

See Also:

Oracle Database Utilities for information about using SQL*Loader for direct-path load

20.2.2 Index the Correct Tables and Columns

Follow guidelines about tables and columns that are suitable for indexing.

Use the following guidelines for determining when to create an index:

- Create an index if you frequently want to retrieve less than 15% of the rows in a large table. The percentage varies greatly according to the relative speed of a table scan and how the row data is distributed in relation to the index key. The faster the table scan, the lower the percentage; the more clustered the row data, the higher the percentage.
- To improve performance on joins of multiple tables, index columns used for joins.

  Note:

  Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.

- If a query is taking too long, then check the table size. If it has changed significantly, the existing indexes (if any) may need to be reviewed.

Columns That Are Suitable for Indexing

Some columns are strong candidates for indexing. Columns with one or more of the following characteristics are candidates for indexing:

- Values are relatively unique in the column.
- There is a wide range of values (good for regular indexes).
- There is a small range of values (good for bitmap indexes).
- The column contains many nulls, but queries often select all rows having a value. In this case, use the following phrase:

  ```
  WHERE COL_X > -9.99 * power(10,125)
  ```

  Using the preceding phrase is preferable to:

  ```
  WHERE COL_X IS NOT NULL
  ```

This is because the first uses an index on `COL_X` (assuming that `COL_X` is a numeric column).

Columns That Are Not Suitable for Indexing

Columns with the following characteristics are less suitable for indexing:

- There are many nulls in the column, and you do not search on the not null values.

`LONG` and `LONG RAW` columns cannot be indexed.

Virtual Columns

You can create unique or non-unique indexes on virtual columns. A table index defined on a virtual column is equivalent to a function-based index on the table.

> **See Also:**
>
> "Creating a Function-Based Index"

20.2.3 Order Index Columns for Performance

The order of columns in the `CREATE INDEX` statement can affect query performance. In general, specify the most frequently used columns first.

If you create a single index across columns to speed up queries that access, for example, `col1`, `col2`, and `col3`; then queries that access just `col1`, or that access just `col1` and `col2`, are also speeded up. But a query that accessed just `col2`, just `col3`, or just `col2` and `col3` does not use the index.

> **Note:**
>
> In some cases, such as when the leading column has very low cardinality, the database may use a skip scan of this type of index. See *Oracle Database Concepts* for more information about index skip scan.

20.2.4 Limit the Number of Indexes for Each Table

A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified.

Specifically, when rows are inserted or deleted, all indexes on the table must be updated as well. Also, when a column is updated, all indexes that contain the column must be updated.

Thus, there is a trade-off between the speed of retrieving data from a table and the speed of updating the table. For example, if a table is primarily read-only, then having more indexes can be useful; but if a table is heavily updated, then having fewer indexes could be preferable.

20.2.5 Drop Indexes That Are No Longer Required

It is best practice to drop indexes that are no longer required.

Consider dropping an index if:

- It does not speed up queries. The table could be very small, or there could be many rows in the table but very few index entries.
- The queries in your applications do not use the index.
- The index must be dropped before being rebuilt.

> **See Also:**
>
> "Monitoring Index Usage"

20.2.6 Indexes and Deferred Segment Creation

Index segment creation is deferred when the associated table defers segment creation. This is because index segment creation reflects the behavior of the table with which it is associated.

> **See Also:**
>
> "Understand Deferred Segment Creation" for further information

20.2.7 Estimate Index Size and Set Storage Parameters

Estimating the size of an index before creating one can facilitate better disk space planning and management.

You can use the combined estimated size of indexes, along with estimates for tables, the undo tablespace, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.

Use the estimated size of an individual index to better manage the disk space that the index uses. When an index is created, you can set appropriate storage parameters and improve I/O performance of applications that use the index. For example, assume that you estimate the maximum size of an index before creating it. If you then set the storage parameters when you create the index, then fewer extents are allocated for the table data segment, and all of the index data is stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this index.

The maximum size of a single index entry is dependent on the block size of the database.

Storage parameters of an index segment created for the index used to enforce a primary key or unique key constraint can be set in either of the following ways:

- In the `ENABLE` ... `USING INDEX` clause of the `CREATE TABLE` or `ALTER TABLE` statement
- In the `STORAGE` clause of the `ALTER INDEX` statement

See Also:

- *Oracle Database Reference* for more information about the limits related to index size
- *Oracle Database SQL Language Reference* for information about creating an index on an extended data type column

20.2.8 Specify the Tablespace for Each Index

Indexes can be created in any tablespace. An index can be created in the same or different tablespace as the table it indexes.

If you use the same tablespace for a table and its index, then it can be more convenient to perform database maintenance (such as tablespace or file backup) or to ensure application availability. All the related data is always online together.

Using different tablespaces (on different disks) for a table and its index produces better performance than storing the table and index in the same tablespace. Disk contention is reduced. But, if you use different tablespaces for a table and its index, and one tablespace is offline (containing either data or index), then the statements referencing that table are not guaranteed to work.

20.2.9 Consider Parallelizing Index Creation

You can parallelize index creation, much the same as you can parallelize table creation. Because multiple processes work together to create the index, the database can create the index more quickly than if a single server process created the index sequentially.

When creating an index in parallel, storage parameters are used separately by each query server process. Therefore, an index created with an `INITIAL` value of 5M and a parallel degree of 12 consumes at least 60M of storage during index creation.

See Also:

Oracle Database VLDB and Partitioning Guide for information about using parallel execution

20.2.10 Consider Creating Indexes with NOLOGGING

You can create an index and generate minimal redo log records by specifying `NOLOGGING` in the `CREATE INDEX` statement.

Note:

Because indexes created using `NOLOGGING` are not archived, perform a backup after you create the index.

Creating an index with NOLOGGING has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the index is decreased.
- Performance improves for parallel creation of large indexes.

In general, the relative performance improvement is greater for larger indexes created without LOGGING than for smaller ones. Creating small indexes without LOGGING has little effect on the time it takes to create an index. However, for larger indexes the performance improvement can be significant, especially when you are also parallelizing the index creation.

20.2.11 Understand When to Use Unusable or Invisible Indexes

Use unusable or invisible indexes when you want to improve the performance of bulk loads, test the effects of removing an index before dropping it, or otherwise suspend the use of an index by the optimizer.

Unusable indexes

An **unusable index** is ignored by the optimizer and is not maintained by DML. One reason to make an index unusable is to improve bulk load performance. (Bulk loads go more quickly if the database does not need to maintain indexes when inserting rows.) Instead of dropping the index and later re-creating it, which requires you to recall the exact parameters of the CREATE INDEX statement, you can make the index unusable, and then rebuild it.

You can create an index in the unusable state, or you can mark an existing index or index partition unusable. In some cases the database may mark an index unusable, such as when a failure occurs while building the index. When one partition of a partitioned index is marked unusable, the other partitions of the index remain valid.

An unusable index or index partition must be rebuilt, or dropped and re-created, before it can be used. Truncating a table makes an unusable index valid.

When you make an existing index unusable, its index segment is dropped.

The functionality of unusable indexes depends on the setting of the SKIP_UNUSABLE_INDEXES initialization parameter. When SKIP_UNUSABLE_INDEXES is TRUE (the default), then:

- DML statements against the table proceed, but unusable indexes are not maintained.
- DML statements terminate with an error if there are any unusable indexes that are used to enforce the UNIQUE constraint.
- For nonpartitioned indexes, the optimizer does not consider any unusable indexes when creating an access plan for SELECT statements. The only exception is when an index is explicitly specified with the INDEX() hint.
- For a partitioned index where one or more of the partitions is unusable, the optimizer can use table expansion. With table expansion, the optimizer transforms the query into a UNION ALL statement, with some subqueries accessing indexed partitions and other subqueries accessing partitions with unusable indexes. The optimizer can choose the most efficient access method available for a partition. See *Oracle Database SQL Tuning Guide* for more information about table expansion.

When SKIP_UNUSABLE_INDEXES is FALSE, then:

- If any unusable indexes or index partitions are present, then any DML statements that would cause those indexes or index partitions to be updated are terminated with an error.

- For `SELECT` statements, if an unusable index or unusable index partition is present, but the optimizer does not choose to use it for the access plan, then the statement proceeds. However, if the optimizer does choose to use the unusable index or unusable index partition, then the statement terminates with an error.

Invisible Indexes

You can create invisible indexes or make an existing index invisible. An **invisible index** is ignored by the optimizer unless you explicitly set the `OPTIMIZER_USE_INVISIBLE_INDEXES` initialization parameter to `TRUE` at the session or system level. Unlike unusable indexes, an invisible index is maintained during DML statements. Although you can make a partitioned index invisible, you cannot make an individual index partition invisible while leaving the other partitions visible.

Using invisible indexes, you can do the following:

- Test the removal of an index before dropping it.
- Use temporary index structures for certain operations or modules of an application without affecting the overall application.
- Add an index to a set of columns on which an index already exists.

See Also:

- "Creating an Unusable Index"
- "Creating an Invisible Index"
- "Making an Index Unusable"
- "Making an Index Invisible or Visible"

20.2.12 Understand When to Create Multiple Indexes on the Same Set of Columns

You can create multiple indexes on the same set of columns when the indexes are different in some way. For example, you can create a B-tree index and a bitmap index on the same set of columns.

When you have multiple indexes on the same set of columns, only one of these indexes can be visible at a time, and any other indexes must be invisible.

You might create different indexes on the same set of columns because they provide the flexibility to meet your requirements. You can also create multiple indexes on the same set of columns to perform application migrations without dropping an existing index and recreating it with different attributes.

Different types of indexes are useful in different scenarios. For example, B-tree indexes are often used in online transaction processing (OLTP) systems with many concurrent transactions, while bitmap indexes are often used in data warehousing systems that are mostly used for queries. Similarly, locally and globally partitioned indexes are useful in different scenarios. Locally partitioned indexes are easy to manage because partition maintenance operations automatically apply to them. Globally partitioned indexes are useful when you want the partitioning scheme of an index to be different from its table's partitioning scheme.

You can create multiple indexes on the same set of columns when at least one of the following index characteristics is different:

- The indexes are of different types.

  See "About Indexes" and *Oracle Database Concepts* for information about the different types of indexes.

  However, the following exceptions apply:

  – You cannot create a B-tree index and a B-tree cluster index on the same set of columns.

  – You cannot create a B-tree index and an index-organized table on the same set of columns.
- The indexes use different partitioning.

  Partitioning can be different in any of the following ways:

  – Indexes that are not partitioned and indexes that are partitioned

  – Indexes that are locally partitioned and indexes that are globally partitioned

  – Indexes that differ in partitioning type (range or hash)
- The indexes have different uniqueness properties.

  You can create both a unique and a non-unique index on the same set of columns.

See Also:

- "Creating Multiple Indexes on the Same Set of Columns"
- "Understand When to Use Unusable or Invisible Indexes"

20.2.13 Consider Costs and Benefits of Coalescing or Rebuilding Indexes

Improper sizing or increased growth can produce index fragmentation. To eliminate or reduce fragmentation, you can rebuild or coalesce the index. But before you perform either task weigh the costs and benefits of each option and choose the one that works best for your situation.

Table 20-1 is a comparison of the costs and benefits associated with rebuilding and coalescing indexes.

Table 20-1 Costs and Benefits of Coalescing or Rebuilding Indexes

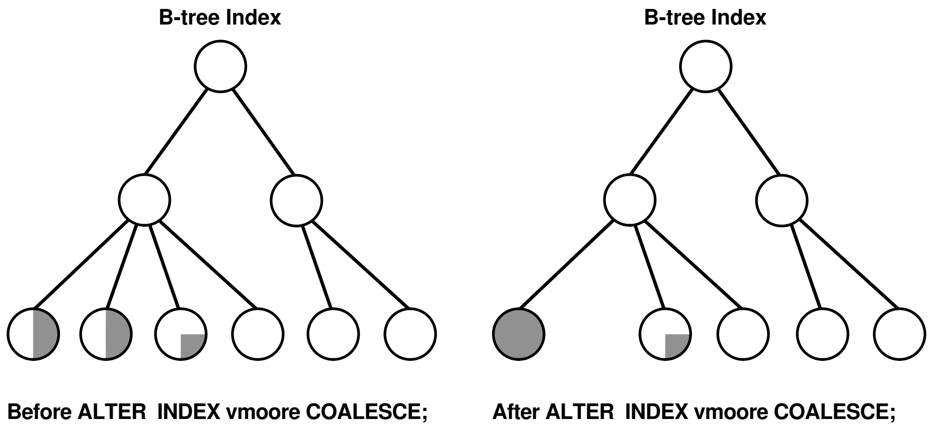
Rebuild Index	Coalesce Index
Quickly moves index to another tablespace	Cannot move index to another tablespace
Higher costs: requires more disk space	Lower costs: does not require more disk space
Creates new tree, shrinks height if applicable	Coalesces leaf blocks within same branch of tree
Enables you to quickly change storage and tablespace parameters without having to drop the original index	Quickly frees up index leaf blocks for use

In situations where you have B-tree index leaf blocks that can be freed up for reuse, you can merge those leaf blocks using the following statement:

```
ALTER INDEX vmoore COALESCE;
```

Figure 20-1 illustrates the effect of an `ALTER INDEX COALESCE` on the index `vmoore`. Before performing the operation, the first two leaf blocks are 50% full. Therefore, you have an opportunity to reduce fragmentation and completely fill the first block, while freeing up the second.

Figure 20-1 Coalescing Indexes


B-tree Index
B-tree Index
Before ALTER INDEX vmoore COALESCE;
After ALTER INDEX vmoore COALESCE;

20.2.14 Consider Cost Before Disabling or Dropping Constraints

Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether to disable or drop a `UNIQUE` or `PRIMARY KEY` constraint.

If the associated index for a `UNIQUE` key or `PRIMARY KEY` constraint is extremely large, then you can save time by leaving the constraint enabled rather than dropping and re-creating the large index. You also have the option of explicitly specifying that you want to keep or drop the index when dropping or disabling a `UNIQUE` or `PRIMARY KEY` constraint.

See Also:

"Managing Integrity Constraints"

20.2.15 Consider Using the In-Memory Column Store to Reduce the Number of Indexes

The In-Memory Column Store is an optional portion of the system global area (SGA) that stores copies of tables, table partitions, and other database objects that is optimized for rapid scans. In the In-Memory Column Store, table data is stored by column rather than row in the SGA.

Note:

This feature is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

For tables used in OLTP or data warehousing environments, multiple indexes typically are created to improve the performance of analytic and reporting queries. These indexes can impede the performance of data manipulation language (DML) statements. When a table is stored in the In-Memory Column Store, indexes used for analytic or reporting queries can be greatly reduced or eliminated without affecting query performance. Eliminating these indexes can improve the performance of transactions and data loading operations.

See Also:

"Improving Query Performance with Oracle Database In-Memory"

20.3 Creating Indexes

You can create several different types of indexes. You can create indexes explicitly, and you can create indexes associated with constraints.

Live SQL:

To view and run examples related to creating indexes on Oracle Live SQL, go to *Oracle Live SQL: Creating Indexes*.

- Prerequisites for Creating Indexes
  Prerequisites must be met before you can create indexes.
- Creating an Index Explicitly
  You can create indexes explicitly (outside of integrity constraints) using the SQL statement `CREATE INDEX`.
- Creating a Unique Index Explicitly
  Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Non-unique indexes do not impose this restriction on the column values.
- Creating an Index Associated with a Constraint
  You can create an index associated with a constraint when you issue the `CREATE TABLE` or `ALTER TABLE` SQL statement.
- Creating a Large Index
  When creating an extremely large index, consider allocating a larger temporary tablespace for the index creation.
- Creating an Index Online
  You can create and rebuild indexes online. Therefore, you can update base tables at the same time you are building or rebuilding indexes on that table.

- Creating a Function-Based Index
  Function-based indexes facilitate queries that qualify a value returned by a function or expression. The value of the function or expression is precomputed and stored in the index.
- Creating a Compressed Index
  As your database grows in size, consider using index compression to save disk space.
- Creating an Unusable Index
  When you create an index in the `UNUSABLE` state, it is ignored by the optimizer and is not maintained by DML. An unusable index must be rebuilt, or dropped and re-created, before it can be used.
- Creating an Invisible Index
  An invisible index is an index that is ignored by the optimizer unless you explicitly set the `OPTIMIZER_USE_INVISIBLE_INDEXES` initialization parameter to `TRUE` at the session or system level.
- Creating Multiple Indexes on the Same Set of Columns
  You can create multiple indexes on the same set of columns when the indexes are different in some way.

20.3.1 Prerequisites for Creating Indexes

Prerequisites must be met before you can create indexes.

To create an index in your own schema, *at least one* of the following prerequisites must be met:

- The table or cluster to be indexed is in your own schema.
- You have `INDEX` privilege on the table to be indexed.
- You have `CREATE ANY INDEX` system privilege.

To create an index in another schema, *all* of the following prerequisites must be met:

- You have `CREATE ANY INDEX` system privilege.
- The owner of the other schema has a quota for the tablespaces to contain the index or index partitions, or `UNLIMITED TABLESPACE` system privilege.

20.3.2 Creating an Index Explicitly

You can create indexes explicitly (outside of integrity constraints) using the SQL statement `CREATE INDEX`.

The following statement creates an index named `emp_ename` for the `ename` column of the `emp` table:

```
CREATE INDEX emp_ename ON emp(ename)
      TABLESPACE users
      STORAGE (INITIAL 20K
      NEXT 20k);
```

Notice that several storage settings and a tablespace are explicitly specified for the index. If you do not specify storage options (such as `INITIAL` and `NEXT`) for an index, then the default storage options of the default or specified tablespace are automatically used.

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating Indexes*.

See Also:

Oracle Database SQL Language Reference for syntax and restrictions on the use of the `CREATE INDEX` statement

20.3.3 Creating a Unique Index Explicitly

Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Non-unique indexes do not impose this restriction on the column values.

Use the `CREATE UNIQUE INDEX` statement to create a unique index. The following example creates a unique index:

```
CREATE UNIQUE INDEX dept_unique_index ON dept (dname)
      TABLESPACE indx;
```

Alternatively, you can define `UNIQUE` integrity constraints on the desired columns. The database enforces `UNIQUE` integrity constraints by automatically defining a unique index on the unique key. This is discussed in the following section. However, it is advisable that any index that exists for query performance, including unique indexes, be created explicitly.

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating Indexes*.

See Also:

Oracle Database SQL Tuning Guide for more information about creating an index for performance

20.3.4 Creating an Index Associated with a Constraint

You can create an index associated with a constraint when you issue the `CREATE TABLE` or `ALTER TABLE` SQL statement.

- About Creating an Index Associated with a Constraint
  Oracle Database enforces a UNIQUE key or PRIMARY KEY integrity constraint on a table by creating a unique index on the unique key or primary key.
- Specifying Storage Options for an Index Associated with a Constraint
  You can set the storage options for the indexes associated with UNIQUE and PRIMARY KEY constraints using the USING INDEX clause.
- Specifying the Index Associated with a Constraint
  You can specify details about the indexes associated with constraints.

20.3.4.1 About Creating an Index Associated with a Constraint

Oracle Database enforces a UNIQUE key or PRIMARY KEY integrity constraint on a table by creating a unique index on the unique key or primary key.

This index is automatically created by the database when the constraint is enabled. No action is required by you when you issue the CREATE TABLE or ALTER TABLE statement to create the index, but you can optionally specify a USING INDEX clause to exercise control over its creation. This includes both when a constraint is defined and enabled, and when a defined but disabled constraint is enabled.

To enable a UNIQUE or PRIMARY KEY constraint, thus creating an associated index, the owner of the table must have a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege. The index associated with a constraint always takes the name of the constraint, unless you optionally specify otherwise.

Note:

An efficient procedure for enabling a constraint that can make use of parallelism is described in "Efficient Use of Integrity Constraints: A Procedure".

20.3.4.2 Specifying Storage Options for an Index Associated with a Constraint

You can set the storage options for the indexes associated with UNIQUE and PRIMARY KEY constraints using the USING INDEX clause.

The following CREATE TABLE statement enables a PRIMARY KEY constraint and specifies the storage options of the associated index:

```
CREATE TABLE emp (
     empno NUMBER(5) PRIMARY KEY, age INTEGER)
     ENABLE PRIMARY KEY USING INDEX
     TABLESPACE users;
```

20.3.4.3 Specifying the Index Associated with a Constraint

You can specify details about the indexes associated with constraints.

If you require more explicit control over the indexes associated with UNIQUE and PRIMARY KEY constraints, the database lets you:

- Specify an existing index that the database is to use to enforce the constraint
- Specify a CREATE INDEX statement that the database is to use to create the index and enforce the constraint

These options are specified using the `USING INDEX` clause. The following statements present some examples.

Example 1:

```
CREATE TABLE a (
     a1 INT PRIMARY KEY USING INDEX (create index ai on a (a1)));
```

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating Indexes*.

Example 2:

```
CREATE TABLE b(
     b1 INT,
     b2 INT,
     CONSTRAINT bu1 UNIQUE (b1, b2)
                    USING INDEX (create unique index bi on b(b1, b2)),
     CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

Example 3:

```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

If a single statement creates an index with one constraint and also uses that index for another constraint, the system will attempt to rearrange the clauses to create the index before reusing it.

See Also:

"Managing Integrity Constraints"

20.3.5 Creating a Large Index

When creating an extremely large index, consider allocating a larger temporary tablespace for the index creation.

To do so, complete the following steps:

1. Create a new temporary tablespace using the `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE` statement.
2. Use the `TEMPORARY TABLESPACE` option of the `ALTER USER` statement to make this your new temporary tablespace.
3. Create the index using the `CREATE INDEX` statement.
4. Drop this tablespace using the `DROP TABLESPACE` statement. Then use the `ALTER USER` statement to reset your temporary tablespace to your original temporary tablespace.

Using this procedure can avoid the problem of expanding your usual, and usually shared, temporary tablespace to an unreasonably large size that might affect future performance.

20.3.6 Creating an Index Online

You can create and rebuild indexes online. Therefore, you can update base tables at the same time you are building or rebuilding indexes on that table.

You can perform DML operations while the index build is taking place, but DDL operations are not allowed. Parallel DML is not supported when creating or rebuilding an index online.

The following statements illustrate online index build operations:

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;
```

Note:

Keep in mind that the time that it takes on online index build to complete is proportional to the size of the table and the number of concurrently executing DML statements. Therefore, it is best to start online index builds when DML activity is low.

Live SQL:

View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating Indexes*.

See Also:

"Rebuilding an Existing Index"

20.3.7 Creating a Function-Based Index

Function-based indexes facilitate queries that qualify a value returned by a function or expression. The value of the function or expression is precomputed and stored in the index.

In addition to the prerequisites for creating a conventional index, if the index is based on user-defined functions, then those functions must be marked `DETERMINISTIC`. Also, a function-based index is executed with the credentials of the owner of the function, so you must have the `EXECUTE` object privilege on the function.

Note:

CREATE INDEX stores the timestamp of the most recent function used in the function-based index. This timestamp is updated when the index is validated. When performing tablespace point-in-time recovery of a function-based index, if the timestamp on the most recent function used in the index is newer than the timestamp stored in the index, then the index is marked invalid. You must use the ANALYZE INDEX...VALIDATE STRUCTURE statement to validate this index.

To illustrate a function-based index, consider the following statement that defines a function-based index (area_index) defined on the function area(geo):

```
CREATE INDEX area_index ON rivers (area(geo));
```

In the following SQL statement, when area(geo) is referenced in the WHERE clause, the optimizer considers using the index area_index.

```
SELECT id, geo, area(geo), desc
     FROM rivers
     WHERE Area(geo) >5000;
```

Because a function-based index depends upon any function it is using, it can be invalidated when a function changes. If the function is valid, then you can use an ALTER INDEX...ENABLE statement to enable a function-based index that has been disabled. The ALTER INDEX...DISABLE statement lets you disable the use of a function-based index. Consider doing this if you are working on the body of the function.

Note:

An alternative to creating a function-based index is to add a virtual column to the target table and index the virtual column. See "About Tables" for more information.

See Also:

- *Oracle Database Concepts* for more information about function-based indexes
- *Oracle Database Development Guide* for information about using function-based indexes in applications and examples of their use

20.3.8 Creating a Compressed Index

As your database grows in size, consider using index compression to save disk space.

- Creating an Index Using Prefix Compression
  Creating an index using prefix compression (also known as key compression) eliminates repeated occurrences of key column prefix values. Prefix compression is most useful for non-unique indexes with a large number of duplicates on the leading columns.

- Creating an Index Using Advanced Index Compression
  Advanced index compression works well on all supported indexes, including those that are not good candidates for prefix compression. Creating an index using advanced index compression can reduce the size of all unique and non-unique indexes and improves the compression ratio significantly, while still providing efficient access to the indexes.

20.3.8.1 Creating an Index Using Prefix Compression

Creating an index using prefix compression (also known as key compression) eliminates repeated occurrences of key column prefix values. Prefix compression is most useful for non-unique indexes with a large number of duplicates on the leading columns.

Prefix compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to substantial savings in space, allowing you to store more keys for each index block while improving performance.

Prefix compression can be useful in the following situations:

- You have a non-unique index where `ROWID` is appended to make the key unique. If you use prefix compression here, then the duplicate key is stored as a prefix entry on the index block without the `ROWID`. The remaining rows become suffix entries consisting of only the `ROWID`.
- You have a unique multicolumn index.

You enable prefix compression using the `COMPRESS` clause. The prefix length (as the number of key columns) can also be specified to identify how the key columns are broken into a prefix and suffix entry. For example, the following statement compresses duplicate occurrences of a key in the index leaf block:

```
CREATE INDEX  hr.emp_ename ON emp(ename)
   TABLESPACE users
   COMPRESS 1;
```

You can also specify the `COMPRESS` clause during rebuild. For example, during rebuild, you can disable compression as follows:

```
ALTER INDEX hr.emp_ename REBUILD NOCOMPRESS;
```

The `COMPRESSION` column in the `ALL_INDEXES` view and `ALL_PART_INDEXES` views shows whether an index is compressed, and, if it is compressed, the type of compression enabled for the index.

> **Live SQL:**
>
> View and run a related example on Oracle Live SQL at *Oracle Live SQL: Creating Indexes*.

See Also:

- *Oracle Database SQL Language Reference*
- *Oracle Database Concepts* for a more detailed discussion of prefix compression

20.3.8.2 Creating an Index Using Advanced Index Compression

Advanced index compression works well on all supported indexes, including those that are not good candidates for prefix compression. Creating an index using advanced index compression can reduce the size of all unique and non-unique indexes and improves the compression ratio significantly, while still providing efficient access to the indexes.

For a partitioned index, you can specify the compression type on a partition by partition basis. You can also specify advanced index compression on index partitions even when the parent index is not compressed.

Advanced index compression works at the block level to provide the best compression for each block.

You can enable advanced index compression using the `COMPRESS ADVANCED` clause and can specify the following compression levels:

- `LOW`: This level provides lower compression ratio at minimal CPU overhead. Before enabling `COMPRESS ADVANCED LOW`, the database must be at 12.1.0 or higher compatibility level.
- `HIGH`: This level, the default, provides higher compression ratio at some CPU overhead. Before enabling `COMPRESS ADVANCED HIGH`, the database must be at 12.2.0 or higher compatibility level.

  When a `CREATE INDEX` DDL statement is executed, a block is filled with rows. At high compression level, when the block is full, it is compressed with advanced index compression if enough space is saved to insert the next row. When a block becomes full, the block might be recompressed using advanced index compression to avoid splitting the block if enough space is saved to insert the incoming key.

The `COMPRESSION` column in the `ALL_INDEXES` view shows whether an index is compressed, and, if it is compressed, the type of compression enabled for the index. The possible values for the `COMPRESSION` column are `ADVANCED HIGH`, `ADVANCED LOW`, `DISABLED`, or `ENABLED`. The `COMPRESSION` column in the `ALL_IND_PARTITIONS` and `ALL_IND_SUBPARTITIONS` views indicates whether index compression is `ENABLED` or `DISABLED` for the partition or subpartition.

Note:

- Advanced index compression is not supported for bitmap indexes or index-organized tables.
- When low level advanced index compression is enabled, advanced index compression cannot be specified on a single column unique index. This restriction does not apply when high level advanced index compression is enabled.

Example 20-1 Enabling Low Level Advanced Index Compression During Index Creation

For example, the following statement enables low level advanced index compression during the creation of the `hr.emp_mndp_ix` index:

```
CREATE INDEX hr.emp_mndp_ix ON hr.employees(manager_id, department_id)
   COMPRESS ADVANCED LOW;
```

Example 20-2 Enabling High Level Advanced Index Compression During Index Rebuild

You can also specify the `COMPRESS ADVANCED` clause during an index rebuild. For example, during rebuild, you can enable high level advanced index compression for the `hr.emp_manager_ix` index as follows:

```
ALTER INDEX hr.emp_manager_ix REBUILD COMPRESS ADVANCED HIGH;
```

20.3.9 Creating an Unusable Index

When you create an index in the `UNUSABLE` state, it is ignored by the optimizer and is not maintained by DML. An unusable index must be rebuilt, or dropped and re-created, before it can be used.

If the index is partitioned, then all index partitions are marked `UNUSABLE`.

The database does not create an index segment when creating an unusable index.

The following procedure illustrates how to create unusable indexes and query the database for details about the index.

To create an unusable index:

1. If necessary, create the table to be indexed.

   For example, create a hash-partitioned table called `hr.employees_part` as follows:

   ```
   sh@PROD> CONNECT hr
   Enter password: **
   Connected.

   hr@PROD> CREATE TABLE employees_part
     2    PARTITION BY HASH (employee_id) PARTITIONS 2
     3    AS SELECT * FROM employees;

   Table created.

   hr@PROD> SELECT COUNT(*) FROM employees_part;

     COUNT(*)
   ----------
          107
   ```

2. Create an index with the keyword `UNUSABLE`.

   The following example creates a locally partitioned index on `employees_part`, naming the index partitions `p1_i_emp_ename` and `p2_i_emp_ename`, and making `p1_i_emp_ename` unusable:

   ```
   hr@PROD> CREATE INDEX i_emp_ename ON employees_part (employee_id)
     2    LOCAL (PARTITION p1_i_emp_ename UNUSABLE, PARTITION p2_i_emp_ename);

   Index created.
   ```

3. (Optional) Verify that the index is unusable by querying the data dictionary.

   The following example queries the status of index `i_emp_ename` and its two partitions, showing that only partition `p2_i_emp_ename` is unusable:

```
hr@PROD> SELECT INDEX_NAME AS "INDEX OR PARTITION NAME", STATUS
  2  FROM   USER_INDEXES
  3  WHERE  INDEX_NAME = 'I_EMP_ENAME'
  4  UNION ALL
  5  SELECT PARTITION_NAME AS "INDEX OR PARTITION NAME", STATUS
  6  FROM   USER_IND_PARTITIONS
  7  WHERE  PARTITION_NAME LIKE '%I_EMP_ENAME%';

INDEX OR PARTITION NAME        STATUS
------------------------------ --------
I_EMP_ENAME                    N/A
P1_I_EMP_ENAME                 UNUSABLE
P2_I_EMP_ENAME                 USABLE
```

4. (Optional) Query the data dictionary to determine whether storage exists for the partitions.

   For example, the following query shows that only index partition `p2_i_emp_ename` occupies a segment. Because you created `p1_i_emp_ename` as unusable, the database did not allocate a segment for it.

```
hr@PROD> COL PARTITION_NAME FORMAT a14
hr@PROD> COL SEG_CREATED FORMAT a11
hr@PROD> SELECT p.PARTITION_NAME, p.STATUS AS "PART_STATUS",
  2         p.SEGMENT_CREATED AS "SEG_CREATED",
  3  FROM   USER_IND_PARTITIONS p, USER_SEGMENTS s
  4  WHERE  s.SEGMENT_NAME = 'I_EMP_ENAME';

PARTITION_NAME PART_STA SEG_CREATED
-------------- -------- -----------
P2_I_EMP_ENAME USABLE   YES
P1_I_EMP_ENAME UNUSABLE NO
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes"
- "Making an Index Unusable"
- *Oracle Database SQL Language Reference* for more information on creating unusable indexes, including restrictions.

20.3.10 Creating an Invisible Index

An invisible index is an index that is ignored by the optimizer unless you explicitly set the `OPTIMIZER_USE_INVISIBLE_INDEXES` initialization parameter to `TRUE` at the session or system level.

To create an invisible index:

- Use the `CREATE INDEX` statement with the `INVISIBLE` keyword.

  The following statement creates an invisible index named `emp_ename` for the `ename` column of the `emp` table:

```
CREATE INDEX emp_ename ON emp(ename)
      TABLESPACE users
      STORAGE (INITIAL 20K
      NEXT 20k)
      INVISIBLE;
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes"
- "Making an Index Invisible or Visible"
- *Oracle Database SQL Language Reference* for more information on creating invisible indexes

20.3.11 Creating Multiple Indexes on the Same Set of Columns

You can create multiple indexes on the same set of columns when the indexes are different in some way.

To create multiple indexes on the same set of columns, the following prerequisites must be met:

- The prerequisites for required privileges in "Creating Indexes".
- Only one index on the same set of columns can be visible at any point in time.

  If you are creating a visible index, then any existing indexes on the set of columns must be invisible..

  Alternatively, you can create an invisible index on the set of columns.

For example, the following steps create a B-tree index and a bitmap index on the same set of columns in the `oe.orders` table:

1. Create a B-tree index on the `customer_id` and `sales_rep_id` columns in the `oe.orders` table:

   ```
   CREATE INDEX oe.ord_customer_ix1 ON oe.orders (customer_id, sales_rep_id);
   ```

   The `oe.ord_customer_ix1` index is visible by default.
2. Alter the index created in Step 1 to make it invisible:

   ```
   ALTER INDEX oe.ord_customer_ix1 INVISIBLE;
   ```

   Alternatively, you can add the `INVISIBLE` clause in Step 1 to avoid this step.
3. Create a bitmap index on the `customer_id` and `sales_rep_id` columns in the `oe.orders` table:

   ```
   CREATE BITMAP INDEX oe.ord_customer_ix2 ON oe.orders (customer_id, sales_rep_id);
   ```

   The `oe.ord_customer_ix2` index is visible by default.

   If the `oe.ord_customer_ix1` index created in Step 1 is visible, then the `CREATE BITMAP INDEX` statement in this step returns an error.

See Also:

- "Understand When to Create Multiple Indexes on the Same Set of Columns"
- "Understand When to Use Unusable or Invisible Indexes"
- "Creating an Invisible Index"

20.4 Altering Indexes

You can alter an index by completing tasks such as changing its storage characteristics, rebuilding it, making it unusable, or making it visible or invisible.

- About Altering Indexes
  To alter an index, your schema must contain the index, or you must have the `ALTER ANY INDEX` system privilege.
- Altering Storage Characteristics of an Index
  Alter the storage parameters of any index, including those created by the database to enforce primary and unique key integrity constraints, using the `ALTER INDEX` statement.
- Rebuilding an Existing Index
  When you rebuild an index, you use an existing index as the data source. Creating an index in this manner enables you to change storage characteristics or move to a new tablespace. Rebuilding an index based on an existing data source removes intra-block fragmentation.
- Making an Index Unusable
  When you make an index unusable, it is ignored by the optimizer and is not maintained by DML. When you make one partition of a partitioned index unusable, the other partitions of the index remain valid.
- Making an Index Invisible or Visible
  Making an index invisible is an alternative to making it unusable or dropping it.
- Renaming an Index
  You can rename an index using an `ALTER INDEX` statement with the `RENAME` clause.
- Monitoring Index Usage
  Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

20.4.1 About Altering Indexes

To alter an index, your schema must contain the index, or you must have the `ALTER ANY INDEX` system privilege.

With the `ALTER INDEX` statement, you can:

- Rebuild or coalesce an existing index
- Deallocate unused space or allocate a new extent
- Specify parallel execution (or not) and alter the degree of parallelism
- Alter storage parameters or physical attributes
- Specify `LOGGING` or `NOLOGGING`

- Enable or disable prefix compression
- Enable or disable advanced compression
- Mark the index unusable
- Make the index invisible
- Rename the index
- Start or stop the monitoring of index usage

You cannot alter index column structure.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for details on the `ALTER INDEX` statement

20.4.2 Altering Storage Characteristics of an Index

Alter the storage parameters of any index, including those created by the database to enforce primary and unique key integrity constraints, using the `ALTER INDEX` statement.

For example, the following statement alters the `emp_ename` index:

```
ALTER INDEX emp_ename
     STORAGE (NEXT 40);
```

The parameters `INITIAL` and `MINEXTENTS` cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the index.

For indexes that implement integrity constraints, you can adjust storage parameters by issuing an `ALTER TABLE` statement that includes the `USING INDEX` subclause of the `ENABLE` clause. For example, the following statement changes the storage options of the index created on table `emp` to enforce the primary key constraint:

```
ALTER TABLE emp
     ENABLE PRIMARY KEY USING INDEX;
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for syntax and restrictions on the use of the `ALTER INDEX` statement

20.4.3 Rebuilding an Existing Index

When you rebuild an index, you use an existing index as the data source. Creating an index in this manner enables you to change storage characteristics or move to a new tablespace. Rebuilding an index based on an existing data source removes intra-block fragmentation.

Compared to dropping the index and using the `CREATE INDEX` statement, rebuilding an existing index offers better performance. Before rebuilding an existing index, compare the costs and benefits associated with rebuilding to those associated with coalescing indexes as described in "Consider Costs and Benefits of Coalescing or Rebuilding Indexes".

The following statement rebuilds the existing index `emp_name`:

```
ALTER INDEX emp_name REBUILD;
```

The `REBUILD` clause must immediately follow the index name, and precede any other options. It cannot be used with the `DEALLOCATE UNUSED` clause.

You have the option of rebuilding the index online. Rebuilding online enables you to update base tables at the same time that you are rebuilding. The following statement rebuilds the `emp_name` index online:

```
ALTER INDEX emp_name REBUILD ONLINE;
```

To rebuild an index in a different user's schema online, the `ALTER ANY INDEX` system privileges is required.

Note:

Online index rebuilding has stricter limitations on the maximum key length that can be handled, compared to other methods of rebuilding an index. If an ORA-1450 (maximum key length exceeded) error occurs when rebuilding online, try rebuilding offline, coalescing, or dropping and recreating the index.

If you do not have the space required to rebuild an index, you can choose instead to coalesce the index. Coalescing an index is an online operation.

See Also:

- "Creating an Index Online"
- "Monitoring Space Use of Indexes"

20.4.4 Making an Index Unusable

When you make an index unusable, it is ignored by the optimizer and is not maintained by DML. When you make one partition of a partitioned index unusable, the other partitions of the index remain valid.

You must rebuild or drop and re-create an unusable index or index partition before using it.

The following procedure illustrates how to make an index and index partition unusable, and how to query the object status.

To make an index unusable:

1. Query the data dictionary to determine whether an existing index or index partition is usable or unusable.

   For example, issue the following query (output truncated to save space):

   ```
   hr@PROD> SELECT INDEX_NAME AS "INDEX OR PART NAME", STATUS, SEGMENT_CREATED
     2  FROM   USER_INDEXES
     3  UNION ALL
     4  SELECT PARTITION_NAME AS "INDEX OR PART NAME", STATUS, SEGMENT_CREATED
     5  FROM   USER_IND_PARTITIONS;

   INDEX OR PART NAME             STATUS   SEG
   ------------------------------ -------- ---
   I_EMP_ENAME                    N/A      N/A
   JHIST_EMP_ID_ST_DATE_PK        VALID    YES
   JHIST_JOB_IX                   VALID    YES
   JHIST_EMPLOYEE_IX              VALID    YES
   JHIST_DEPARTMENT_IX            VALID    YES
   EMP_EMAIL_UK                   VALID    NO
   .
   .
   .
   COUNTRY_C_ID_PK                VALID    YES
   REG_ID_PK                      VALID    YES
   P2_I_EMP_ENAME                 USABLE   YES
   P1_I_EMP_ENAME                 UNUSABLE NO

   22 rows selected.
   ```

   The preceding output shows that only index partition `p1_i_emp_ename` is unusable.

2. Make an index or index partition unusable by specifying the `UNUSABLE` keyword.

   The following example makes index `emp_email_uk` unusable:

   ```
   hr@PROD> ALTER INDEX emp_email_uk UNUSABLE;

   Index altered.
   ```

   The following example makes index partition `p2_i_emp_ename` unusable:

   ```
   hr@PROD> ALTER INDEX i_emp_ename MODIFY PARTITION p2_i_emp_ename UNUSABLE;

   Index altered.
   ```

3. (Optional) Query the data dictionary to verify the status change.

   For example, issue the following query (output truncated to save space):

   ```
   hr@PROD> SELECT INDEX_NAME AS "INDEX OR PARTITION NAME", STATUS,
     2  SEGMENT_CREATED
     3  FROM   USER_INDEXES
     4  UNION ALL
     5  SELECT PARTITION_NAME AS "INDEX OR PARTITION NAME", STATUS,
     6  SEGMENT_CREATED
     7  FROM   USER_IND_PARTITIONS;

   INDEX OR PARTITION NAME        STATUS   SEG
   ------------------------------ -------- ---
   ```

```
I_EMP_ENAME                    N/A      N/A
JHIST_EMP_ID_ST_DATE_PK        VALID    YES
JHIST_JOB_IX                   VALID    YES
JHIST_EMPLOYEE_IX              VALID    YES
JHIST_DEPARTMENT_IX            VALID    YES
EMP_EMAIL_UK                   UNUSABLE NO
.
.
.
COUNTRY_C_ID_PK                VALID    YES
REG_ID_PK                      VALID    YES
P2_I_EMP_ENAME                 UNUSABLE NO
P1_I_EMP_ENAME                 UNUSABLE NO

22 rows selected.
```

A query of space consumed by the `i_emp_ename` and `emp_email_uk` segments shows that the segments no longer exist:

```
hr@PROD> SELECT SEGMENT_NAME, BYTES
  2  FROM   USER_SEGMENTS
  3  WHERE  SEGMENT_NAME IN ('I_EMP_ENAME', 'EMP_EMAIL_UK');

no rows selected
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes"
- "Creating an Unusable Index"
- *Oracle Database SQL Language Reference* for more information about the `UNUSABLE` keyword, including restrictions

20.4.5 Making an Index Invisible or Visible

Making an index invisible is an alternative to making it unusable or dropping it.

An invisible index is ignored by the optimizer unless you explicitly set the `OPTIMIZER_USE_INVISIBLE_INDEXES` initialization parameter to `TRUE` at the session or system level. You cannot make an individual index partition invisible. Attempting to do so produces an error.

To make an index invisible:

- Submit the following SQL statement:

```
ALTER INDEX index INVISIBLE;
```

To make an invisible index visible again:

- Submit the following SQL statement:

```
ALTER INDEX index VISIBLE;
```

Note:

If there are multiple indexes on the same set of columns, then only one of these indexes can be visible at any point in time. If you try to make an index on a set of columns visible, and another index on the same set of columns is visible, then an error is returned.

To determine whether an index is visible or invisible:

- Query the dictionary views `USER_INDEXES`, `ALL_INDEXES`, or `DBA_INDEXES`.

  For example, to determine if the index `ind1` is invisible, issue the following query:

  ```
  SELECT INDEX_NAME, VISIBILITY FROM USER_INDEXES
     WHERE INDEX_NAME = 'IND1';

  INDEX_NAME   VISIBILITY
  ----------   ----------
  IND1         VISIBLE
  ```

See Also:

- "Understand When to Use Unusable or Invisible Indexes"
- "Creating an Invisible Index"
- "Creating Multiple Indexes on the Same Set of Columns"

20.4.6 Renaming an Index

You can rename an index using an `ALTER INDEX` statement with the `RENAME` clause.

To rename an index, issue this statement:

```
ALTER INDEX index_name RENAME TO new_name;
```

20.4.7 Monitoring Index Usage

Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

To start monitoring the usage of an index, issue this statement:

```
ALTER INDEX index MONITORING USAGE;
```

Later, issue the following statement to stop the monitoring:

```
ALTER INDEX index NOMONITORING USAGE;
```

The view `USER_OBJECT_USAGE` can be queried for the index being monitored to see if the index has been used. The view contains a `USED` column whose value is `YES` or `NO`, depending upon if the index has been used within the time period being monitored. The view also contains the start and stop times of the monitoring period, and a `MONITORING` column (`YES`/`NO`) to indicate if usage monitoring is currently active.

Each time that you specify `MONITORING USAGE`, the `USER_OBJECT_USAGE` view is reset for the specified index. The previous usage information is cleared or reset, and a new start time is recorded. When you specify `NOMONITORING USAGE`, no further monitoring is performed, and the end time is recorded for the monitoring period. Until the next `ALTER INDEX...MONITORING USAGE` statement is issued, the view information is left unchanged.

20.5 Monitoring Space Use of Indexes

If key values in an index are inserted, updated, and deleted frequently, then the index can lose its acquired space efficiency over time.

Monitor index efficiency of space usage at regular intervals by first analyzing the index structure, using the `ANALYZE INDEX...VALIDATE STRUCTURE` statement, and then querying the `INDEX_STATS` view:

```
SELECT PCT_USED FROM INDEX_STATS WHERE NAME = 'index';
```

The percentage of index space usage varies according to how often index keys are inserted, updated, or deleted. Develop a history of average efficiency of space usage for an index by performing the following sequence of operations several times:

- Analyzing statistics
- Validating the index
- Checking `PCT_USED`
- Dropping and rebuilding (or coalescing) the index

When you find that index space usage drops below its average, you can condense the index space by dropping the index and rebuilding it, or coalescing it.

See Also:

"About Analyzing Tables, Indexes, and Clusters"

20.6 Dropping Indexes

You can drop an index with the `DROP INDEX` statement.

To drop an index, the index must be contained in your schema, or you must have the `DROP ANY INDEX` system privilege.

Some reasons for dropping an index include:

- The index is no longer required.
- The index is not providing anticipated performance improvements for queries issued against the associated table. For example, the table might be very small, or there might be many rows in the table but very few index entries.
- Applications do not use the index to query the data.
- The index has become invalid and must be dropped before being rebuilt.

- The index has become too fragmented and must be dropped before being rebuilt.

When you drop an index, all extents of the index segment are returned to the containing tablespace and become available for other objects in the tablespace.

How you drop an index depends on whether you created the index explicitly with a `CREATE INDEX` statement, or implicitly by defining a key constraint on a table. If you created the index explicitly with the `CREATE INDEX` statement, then you can drop the index with the `DROP INDEX` statement. The following statement drops the `emp_ename` index:

```
DROP INDEX emp_ename;
```

You cannot drop only the index associated with an enabled `UNIQUE` key or `PRIMARY KEY` constraint. To drop a constraints associated index, you must disable or drop the constraint itself.

Note:

If a table is dropped, all associated indexes are dropped automatically.

See Also:

- *Oracle Database SQL Language Reference* for syntax and restrictions on the use of the `DROP INDEX` statement
- "Managing Integrity Constraints"
- "Making an Index Invisible or Visible" for an alternative to dropping indexes

20.7 Managing Automatic Indexes

You can use the automatic indexing feature to configure and use automatic indexes in an Oracle database to improve database performance.

- About Automatic Indexing
  The automatic indexing feature automates the index management tasks in an Oracle database. Automatic indexing automatically creates and drops indexes in a database based on the changes in application workload, thus improving database performance. The automatically managed indexes are known as *auto indexes*.
- How Automatic Indexing Works
  This section describes how automatic indexing works.
- Configuring Automatic Indexing in an Oracle Database
  You can configure automatic indexing in an Oracle database using the `DBMS_AUTO_INDEX.CONFIGURE` procedure.
- Generating Automatic Indexing Reports
  You can generate reports related to automatic indexing operations in an Oracle database using the `REPORT_ACTIVITY` and `REPORT_LAST_ACTIVITY` functions of the `DBMS_AUTO_INDEX` package.

- Views Containing the Automatic Indexing Information
  You can query a set of data dictionary views for getting information about the auto indexes in an Oracle database.

20.7.1 About Automatic Indexing

The automatic indexing feature automates the index management tasks in an Oracle database. Automatic indexing automatically creates and drops indexes in a database based on the changes in application workload, thus improving database performance. The automatically managed indexes are known as *auto indexes*.

Index structures are an essential feature to database performance. Indexes are critical for OLTP applications, which use large data sets and run millions of SQL statements a day. Indexes are also critical for data warehousing applications, which typically query a relatively small amount of data from very large tables. If you do not update the indexes whenever there are changes in the application workload, the existing indexes can cause the database performance to deteriorate considerably.

Automatic indexing improves database performance by managing indexes automatically and dynamically in an Oracle database based on changes in the application workload.

Automatic indexing provides the following functionality:

- Runs the automatic indexing process in the background periodically at a predefined time interval.
- Analyzes application workload, and accordingly creates new indexes and drops the existing underperforming indexes to improve database performance.
- Rebuilds the indexes that are marked *unusable* due to table partitioning maintenance operations, such as `ALTER TABLE MOVE`.
- Provides PL/SQL APIs for configuring automatic indexing in a database and generating reports related to automatic indexing operations.

Note:

- Auto indexes are local B-tree indexes.
- Auto indexes can be created for partitioned as well as non-partitioned tables.
- Auto indexes cannot be created for temporary tables.
- Automatic indexing uses the SQL performance analyzer framework internally to measure SQL statement performance. Automatic indexing will not function if Real Application Testing is explicitly excluded when linking the SQL executable. Real Application Testing is excluded if the `RAT_OFF` parameter is supplied to the `make` command. Explicitly excluding Real Application Testing and using automatic indexing will generate error `ORA-00438: Real Application Testing Option not installed`.

20.7.2 How Automatic Indexing Works

This section describes how automatic indexing works.

The automatic indexing process runs in the background every 15 minutes and performs the following operations:

1. Automatic index candidates are identified based on the usage of table columns in SQL statements. Ensure that table statistics are up to date. Tables without statistics are not considered for automatic indexing. Tables with stale statistics are not considered for automatic indexing.
2. Index candidates are initially created invisible and unusable. They are not visible to the application workload. Invisible automatic indexes cannot be used by SQL statements in the application workload.

   Automatic indexes can be single-column or multi-column. They are considered for the following:

   - Table columns (including virtual columns)
   - Partitioned and non-partitioned tables
   - Selected expressions (for example, JSON expressions)
3. A sample of workload SQL statements is tested against the candidate indexes. During this verification phase, some or all candidate indexes will be built and made valid so that the performance effect on SQL statements can be measured. All candidate indexes remain invisible during the verification step.

   If the performance of SQL statements is not improved by using the candidate indexes, they remain invisible.
4. Candidate valid indexes found to improve SQL performance will be made visible and available to the application workload. Candidate indexes that do not improve SQL performance will revert to invisible and be unusable after a short delay.

   During the verification stage, if an index is found to be beneficial, but an individual SQL statement suffers a performance regression, a SQL plan baseline is created to prevent the regression when the index is made visible.
5. Unusable and unused valid indexes are deleted by the automatic indexing process.

   Note:

   By default, the unused automatic indexes are deleted after 373 days. The period for retaining the unused automatic indexes in a database can be configured using the `DBMS_AUTO_INDEX.CONFIGURE` procedure.

See Also:

"Configuring Automatic Indexing in an Oracle Database"

20.7.3 Configuring Automatic Indexing in an Oracle Database

You can configure automatic indexing in an Oracle database using the `DBMS_AUTO_INDEX.CONFIGURE` procedure.

The following examples describe some of the configuration settings that can be specified using the `DBMS_AUTO_INDEX.CONFIGURE` procedure:

Enabling and disabling automatic indexing in a database

You can use the `AUTO_INDEX_MODE` configuration setting to enable or disable automatic indexing in a database.

The following statement enables automatic indexing in a database and creates any new auto indexes as *visible* indexes, so that they can be used in SQL statements:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_MODE','IMPLEMENT');
```

The following statement enables automatic indexing in a database, but creates any new auto indexes as *invisible* indexes, so that they cannot be used in SQL statements:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_MODE','REPORT ONLY');
```

The following statement disables automatic indexing in a database so that no new auto indexes are created (existing auto indexes remain enabled):

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_MODE','OFF');
```

Specifying schemas that can use auto indexes

You can use the `AUTO_INDEX_SCHEMA` configuration setting to specify schemas that can use auto indexes.

> **Note:**
>
> When automatic indexing is enabled in a database, all the schemas in the database can use auto indexes by default.

The following statements add the `SH` and `HR` schemas to the *exclusion list*, so that the `SH` and `HR` schemas cannot use auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_SCHEMA', 'SH', FALSE);
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_SCHEMA', 'HR', FALSE);
```

The following statement removes the `HR` schema from the exclusion list, so that the `HR` schema can use auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_SCHEMA', 'HR', NULL);
```

The following statement removes all the schema from the exclusion list, so that all the schemas in the database can use auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_SCHEMA', NULL, TRUE);
```

Specifying tables that can use auto indexes

You can use the `AUTO_INDEX_TABLE` configuration setting to specify tables that can use auto indexes. When you enable automatic indexing for a schema, all the tables in that schema can use auto indexes. However, if there is a conflict between the schema level and table level setting, the table level setting takes precedence.

The following statement includes the `PRODUCTS` table in the `SH` schema for automatic indexing:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_TABLE','SH.PRODUCTS',TRUE);
```

The following statements add the `SALES` and `PRODUCTS` tables in the `SH` schema to the exclusion list, so that these tables cannot use auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_TABLE', 'SH.SALES', FALSE);
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_TABLE', 'SH.PRODUCTS', FALSE);
```

The following statement removes the `SH.SALES` table from the exclusion list, so that the table can use auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_TABLE', 'SH.SALES', NULL);
```

The following statement removes all the tables from the exclusion list, so that all the tables in the database can use auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_TABLE', NULL, TRUE);
```

The following statement checks the current configuration setting:

```
SELECT parameter_name, parameter_value FROM dba_auto_index_config WHERE
parameter_name = 'AUTO_INDEX_TABLE';
```

Specifying a retention period for unused auto indexes

You can use the `AUTO_INDEX_RETENTION_FOR_AUTO` configuration setting to specify a period for retaining unused auto indexes in a database. The unused auto indexes are deleted after the specified retention period.

Note:

By default, the unused auto indexes are deleted after 373 days.

The following statement sets the retention period for unused auto indexes to 90 days.

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_RETENTION_FOR_AUTO', '90');
```

The following statement resets the retention period for auto indexes to the default value of 373 days.

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_RETENTION_FOR_AUTO', NULL);
```

Specifying a retention period for unused non-auto indexes

You can use the `AUTO_INDEX_RETENTION_FOR_MANUAL` configuration setting to specify a period for retaining unused non-auto indexes (manually created indexes) in a database. The unused non-auto indexes are deleted after the specified retention period.

> **Note:**
>
> By default, the unused non-auto indexes are never deleted by the automatic indexing process.

The following statement sets the retention period for unused non-auto indexes to 60 days.

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_RETENTION_FOR_MANUAL', '60');
```

The following statement sets the retention period for unused non-auto indexes to `NULL` so that they are never deleted by the automatic indexing process.

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_RETENTION_FOR_MANUAL', NULL);
```

Specifying a retention period for automatic indexing logs

You can use the `AUTO_INDEX_REPORT_RETENTION` configuration setting to specify a period for retaining automatic indexing logs in a database. The automatic indexing logs are deleted after the specified retention period.

> **Note:**
>
> By default, the automatic indexing logs are deleted after 373 days.

The following statement sets the retention period for automatic indexing logs to 60 days.

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_REPORT_RETENTION', '60');
```

The following statement resets the retention period for automatic indexing logs to the default value 373 days.

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_REPORT_RETENTION', NULL);
```

> **Note:**
>
> Automatic indexing reports are generated based on the automatic indexing logs. Therefore, automatic indexing reports cannot be generated for a period that is more than the retention period of the automatic indexing logs specified using the `AUTO_INDEX_REPORT_RETENTION` configuration setting.

Specifying a tablespace to store auto indexes

You can use the `AUTO_INDEX_DEFAULT_TABLESPACE` configuration setting to specify a tablespace to store auto indexes. Note that you cannot specify an Oracle-owned tablespace (such as `SYSAUX`) as the default tablespace.

> **Note:**
>
> By default, the permanent tablespace specified during the database creation is used for storing auto indexes.

The following statement specifies the tablespace of `TBS_AUTO` to store auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_DEFAULT_TABLESPACE', 'TBS_AUTO');
```

Specifying percentage of tablespace to allocate for auto indexes

You can use the `AUTO_INDEX_SPACE_BUDGET` configuration setting to specify percentage of tablespace to allocate for auto indexes. You can specify this configuration setting only when the tablespace used for storing auto indexes is the default permanent tablespace specified during the database creation, that is, when no value is specified for the `AUTO_INDEX_DEFAULT_TABLESPACE` configuration setting.

The following statement allocates 5 percent of the tablespace for auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_SPACE_BUDGET', '5');
```

Configuring advanced index compression for auto indexes

You can use the `AUTO_INDEX_COMPRESSION` configuration setting to specify whether advanced index compression must be used with auto indexes. Advanced index compression is part of the Oracle Advanced Compression option.

The following example enables advanced index compression when creating auto indexes:

```
EXEC DBMS_AUTO_INDEX.CONFIGURE('AUTO_INDEX_COMPRESSION','ON');
```

> **See Also:**
>
> *Oracle Database Licensing Information User Manual* for information about the Oracle Advanced Compression option.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for a complete list of configuration settings related to automatic indexing that can be specified using the `DBMS_AUTO_INDEX.CONFIGURE` procedure

20.7.4 Generating Automatic Indexing Reports

You can generate reports related to automatic indexing operations in an Oracle database using the `REPORT_ACTIVITY` and `REPORT_LAST_ACTIVITY` functions of the `DBMS_AUTO_INDEX` package.

See Also:

Oracle Database PL/SQL Packages and Types Reference for the syntax of the `REPORT_ACTIVITY` and `REPORT_LAST_ACTIVITY` functions of the `DBMS_AUTO_INDEX` package.

Generating a report of automatic indexing operations for a specific period

The following example generates a report containing *typical* information about the automatic indexing operations for the last 24 hours. The report is generated in the plain text format by default.

```
declare
  report clob := null;
begin
  report := DBMS_AUTO_INDEX.REPORT_ACTIVITY();
end;
```

The following example generates a report containing *basic* information about the automatic indexing operations for the month of November 2018. The report is generated in the HTML format and includes only the summary of automatic indexing operations.

```
declare
  report clob := null;
begin
  report := DBMS_AUTO_INDEX.REPORT_ACTIVITY(
              activity_start => TO_TIMESTAMP('2018-11-01', 'YYYY-MM-DD'),
              activity_end   => TO_TIMESTAMP('2018-12-01', 'YYYY-MM-DD'),
              type           => 'HTML',
              section        => 'SUMMARY',
              level          => 'BASIC');
end;
```

Generating a report of the last automatic indexing operation

The following example generates a report containing *typical* information about the last automatic indexing operation. The report is generated in the plain text format by default.

```
declare
  report clob := null;
begin
  report := DBMS_AUTO_INDEX.REPORT_LAST_ACTIVITY();
end;
```

The following example generates a report containing *basic* information about the last automatic indexing operation. The report includes the summary, index details, and error information of the last automatic indexing operation. The report is generated in the HTML format.

```
declare
  report clob := null;
begin
  report := DBMS_AUTO_INDEX.REPORT_LAST_ACTIVITY(
                type    => 'HTML',
                section => 'SUMMARY +INDEX_DETAILS +ERRORS',
                level   => 'BASIC');
end;
```

20.7.5 Views Containing the Automatic Indexing Information

You can query a set of data dictionary views for getting information about the auto indexes in an Oracle database.

The following views show information about the automatic indexing configuration settings and the auto indexes created in an Oracle database:

View	Description
`DBA_AUTO_INDEX_CONFIG`	Shows the current configuration settings for automatic indexing.
`DBA_INDEXES` `ALL_INDEXES` `USER_INDEXES`	The `AUTO` column in these views indicates whether an index is an auto index (`YES`) or not (`NO`).

See Also:

Oracle Database Reference for complete descriptions of these views

20.8 Indexes Data Dictionary Views

You can query a set of data dictionary views for information about indexes.

The following views show information about indexes:

View	Description
`DBA_INDEXES` `ALL_INDEXES` `USER_INDEXES`	`DBA` view shows information about indexes on all tables in the database. `ALL` view shows information about indexes on all tables accessible to the user. `USER` view is restricted to indexes owned by the user. Some columns in these views contain statistics that are generated by the `DBMS_STATS` package or the `ANALYZE` statement.

View	Description
`DBA_IND_COLUMNS ALL_IND_COLUMNS USER_IND_COLUMNS`	These views show information about the columns of indexes on tables. Some columns in these views contain statistics that are generated by the `DBMS_STATS` package or `ANALYZE` statement.
`DBA_IND_PARTITIONS ALL_IND_PARTITIONSALL_IND_PARTITIONS USER_IND_PARTITIONS`	These views show the following information about each index partition: the partitioning details, the storage parameters for the partition, and various partition statistics generated by the `DBMS_STATS` package.
`DBA_IND_EXPRESSIONS ALL_IND_EXPRESSIONS USER_IND_EXPRESSIONS`	These views show information about the expressions of function-based indexes on tables.
`DBA_IND_STATISTICS ALL_IND_STATISTICS USER_IND_STATISTICS`	These views show information about the optimizer statistics for indexes.
`INDEX_STATS INDEX_HISTOGRAM`	These views show the information about the last `ANALYZE INDEX...VALIDATE STRUCTURE` statement.
`USER_OBJECT_USAGE`	This view shows the index usage information produced by the `ALTER INDEX...MONITORING USAGE` statement.

See Also:

Oracle Database Reference for the complete descriptions of these views

21 Managing Clusters

Using clusters can improve performance and reduce disk space requirements.

21.1 About Clusters

A **cluster** provides an optional method of storing table data. A cluster is made up of a group of tables that share the same data blocks. The tables are grouped together because they share common columns and are often used together.

For example, the `emp` and `dept` table share the `deptno` column. When you cluster the `emp` and `dept` tables (see Figure 21-1), Oracle Database physically stores all rows for each department from both the `emp` and `dept` tables in the same data blocks.

Because clusters store related rows of different tables together in the same data blocks, properly used clusters offer two primary benefits:

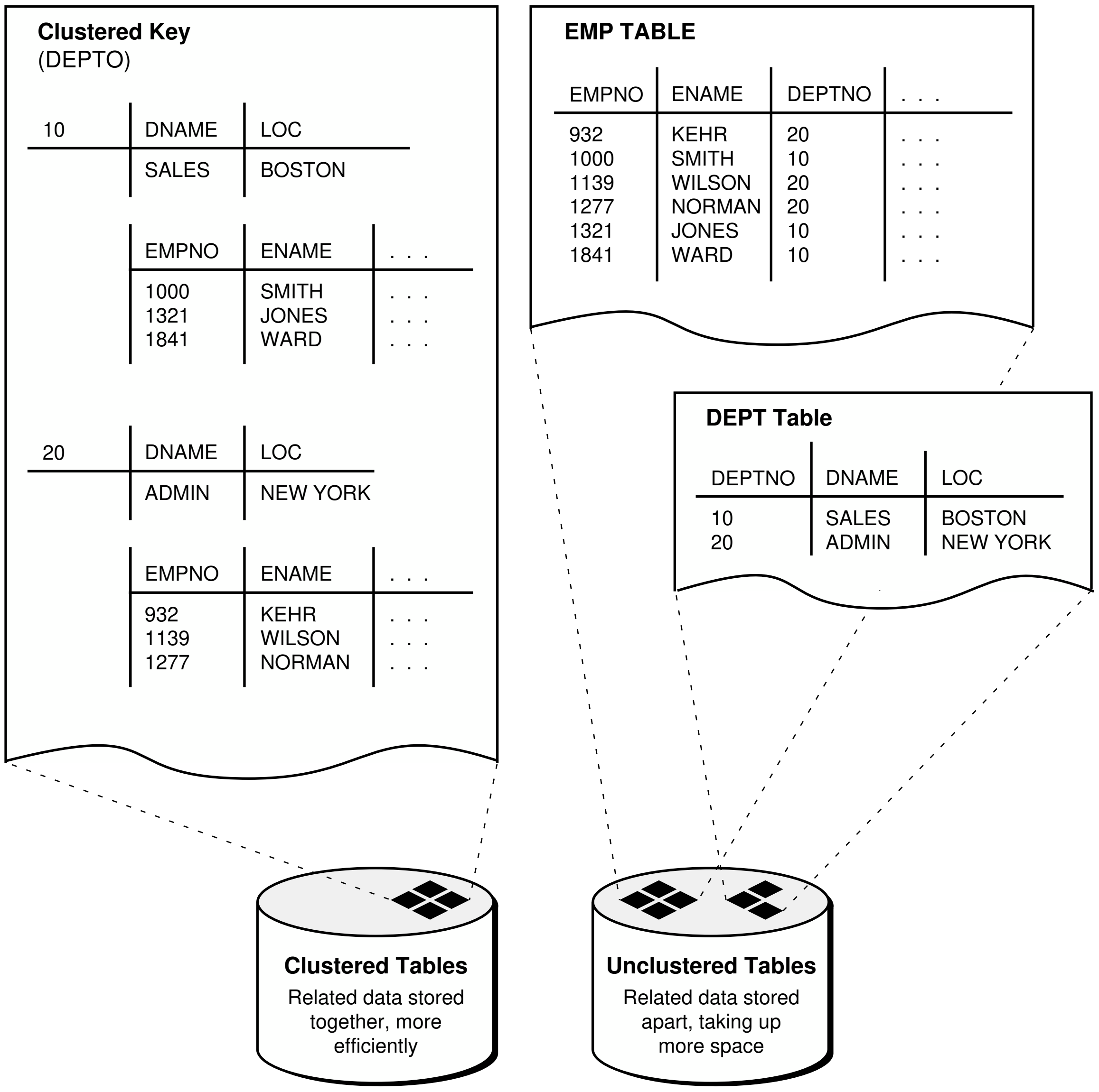
- Disk I/O is reduced and access time improves for joins of clustered tables.
- The **cluster key** is the column, or group of columns, that the clustered tables have in common. You specify the columns of the cluster key when creating the cluster. You subsequently specify the same columns when creating every table added to the cluster. Each cluster key value is stored only once each in the cluster and the cluster index, no matter how many rows of different tables contain the value.

  Therefore, less storage might be required to store related table and index data in a cluster than is necessary in non-clustered table format. For example, in Figure 21-1, notice how each cluster key (each `deptno`) is stored just once for many rows that contain the same value in both the `emp` and `dept` tables.

After creating a cluster, you can create tables in the cluster. However, before any rows can be inserted into the clustered tables, a cluster index must be created. Using clusters does not affect the creation of additional indexes on the clustered tables; they can be created and dropped as usual.

You should not use clusters for tables that are frequently accessed individually.

Figure 21-1 Clustered Table Data



See Also:

- Managing Hash Clusters for a description of another type of cluster: a hash cluster
- Managing Space for Schema Objects is recommended reading before attempting tasks described in this chapter

21.2 Guidelines for Managing Clusters

You can follow guidelines for managing clusters.

- Choose Appropriate Tables for the Cluster
  Use clusters for tables that are primarily queries and frequently queried together.
- Choose Appropriate Columns for the Cluster Key
  Choose cluster key columns carefully. If multiple columns are used in queries that join the tables, make the cluster key a composite key. In general, the characteristics that indicate a good cluster index are the same as those for any index.
- Specify the Space Required by an Average Cluster Key and Its Associated Rows
  The `CREATE CLUSTER` statement has an optional clause, `SIZE`, which is the estimated number of bytes required by an average cluster key and its associated rows.
- Specify the Location of Each Cluster and Cluster Index Rows
  Always specify the `TABLESPACE` clause in a `CREATE CLUSTER`/`INDEX` statement to identify the tablespace to store the new cluster or index.
- Estimate Cluster Size and Set Storage Parameters
  Before creating a cluster, estimate the cluster size and set the storage parameters for the data segments of a cluster.

See Also:

- *Oracle Database Concepts* for more information about clusters
- *Oracle Database SQL Tuning Guide* for guidelines on when to use clusters

21.2.1 Choose Appropriate Tables for the Cluster

Use clusters for tables that are primarily queries and frequently queried together.

Use clusters for tables for which the following conditions are true:

- The tables are primarily queried--that is, tables that are *not* predominantly inserted into or updated.
- Records from the tables are frequently queried together or joined.

21.2.2 Choose Appropriate Columns for the Cluster Key

Choose cluster key columns carefully. If multiple columns are used in queries that join the tables, make the cluster key a composite key. In general, the characteristics that indicate a good cluster index are the same as those for any index.

For information about characteristics of a good index, see "Guidelines for Managing Indexes".

A good cluster key has enough unique values so that the group of rows corresponding to each key value fills approximately one data block. Having too few rows for each cluster key value can waste space and result in negligible performance gains. Cluster keys that are so specific that only a few rows share a common value can cause wasted space in blocks, unless a small `SIZE` was specified at cluster creation time (see "Specify the Space Required by an Average Cluster Key and Its Associated Rows ").

Too many rows for each cluster key value can cause extra searching to find rows for that key. Cluster keys on values that are too general (for example, `male` and `female`) result in excessive searching and can result in worse performance than with no clustering.

A cluster index cannot be unique or include a column defined as `long`.

21.2.3 Specify the Space Required by an Average Cluster Key and Its Associated Rows

The `CREATE CLUSTER` statement has an optional clause, `SIZE`, which is the estimated number of bytes required by an average cluster key and its associated rows.

The database uses the `SIZE` parameter when performing the following tasks:

- Estimating the number of cluster keys (and associated rows) that can fit in a clustered data block
- Limiting the number of cluster keys placed in a clustered data block. This maximizes the storage efficiency of keys within a cluster.

`SIZE` does not limit the space that can be used by a given cluster key. For example, if `SIZE` is set such that two cluster keys can fit in one data block, any amount of the available data block space can still be used by either of the cluster keys.

By default, the database stores only one cluster key and its associated rows in each data block of the cluster data segment. Although block size can vary from one operating system to the next, the rule of one key for each block is maintained as clustered tables are imported to other databases on other systems.

If all the rows for a given cluster key value cannot fit in one block, the blocks are chained together to speed access to all the values with the given key. The cluster index points to the beginning of the chain of blocks, each of which contains the cluster key value and associated rows. If the cluster `SIZE` is such that multiple keys fit in a block, then blocks can belong to multiple chains.

21.2.4 Specify the Location of Each Cluster and Cluster Index Rows

Always specify the `TABLESPACE` clause in a `CREATE CLUSTER`/`INDEX` statement to identify the tablespace to store the new cluster or index.

If you have the proper privileges and tablespace quota, you can create a new cluster and the associated cluster index in any tablespace that is currently online.

The cluster and its cluster index can be created in different tablespaces. In fact, creating a cluster and its index in different tablespaces that are stored on different storage devices allows table data and index data to be retrieved simultaneously with minimal disk contention.

21.2.5 Estimate Cluster Size and Set Storage Parameters

Before creating a cluster, estimate the cluster size and set the storage parameters for the data segments of a cluster.

The following are benefits of estimating cluster size before creating the cluster:

- You can use the combined estimated size of clusters, along with estimates for indexes and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.
- You can use the estimated size of an individual cluster to better manage the disk space that the cluster will use. When a cluster is created, you can set appropriate storage parameters and improve I/O performance of applications that use the cluster.

Set the storage parameters for the data segments of a cluster using the `STORAGE` clause of the `CREATE CLUSTER` or `ALTER CLUSTER` statement, rather than the individual `CREATE` or `ALTER` statements that put tables into the cluster. Storage parameters specified when creating or altering a clustered table are ignored. The storage parameters set for the cluster override the table storage parameters.

21.3 Creating Clusters and Objects That Use Them

You create a cluster using the `CREATE CLUSTER` statement. You create clustered table using the `CREATE TABLE` statement with the `CLUSTER` clause. You create a cluster index using the `CREATE INDEX` statement with the `CLUSTER` clause.

- Creating Clusters
  You create a cluster using the `CREATE CLUSTER` statement.
- Creating Clustered Tables
  You create a table in a cluster using the `CREATE TABLE` statement with the `CLUSTER` clause.
- Creating Cluster Indexes
  A cluster index must be created before any rows can be inserted into any clustered table.

21.3.1 Creating Clusters

You create a cluster using the `CREATE CLUSTER` statement.

To create a cluster in your schema, you must have the `CREATE CLUSTER` system privilege and a quota for the tablespace intended to contain the cluster or the `UNLIMITED TABLESPACE` system privilege.

To create a cluster in another user's schema you must have the `CREATE ANY CLUSTER` system privilege, and the owner must have a quota for the tablespace intended to contain the cluster or the `UNLIMITED TABLESPACE` system privilege.

The following statement creates a cluster named `emp_dept`, which stores the `emp` and `dept` tables, clustered by the `deptno` column:

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
   SIZE 600
   TABLESPACE users
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      PCTINCREASE 33);
```

If no `INDEX` keyword is specified, as is true in this example, an index cluster is created by default. You can also create a `HASH` cluster, when hash parameters (`HASHKEYS`, `HASH IS`, or `SINGLE TABLE HASHKEYS`) are specified. Hash clusters are described in Managing Hash Clusters.

21.3.2 Creating Clustered Tables

You create a table in a cluster using the `CREATE TABLE` statement with the `CLUSTER` clause.

To create a table in a cluster, you must have either the `CREATE TABLE` or `CREATE ANY TABLE` system privilege. You do not need a tablespace quota or the `UNLIMITED TABLESPACE` system privilege to create a table in a cluster.

For example, the `emp` and `dept` tables can be created in the `emp_dept` cluster using the following statements:

```
CREATE TABLE emp (
   empno NUMBER(5) PRIMARY KEY,
   ename VARCHAR2(15) NOT NULL,
   . . .
   deptno NUMBER(3) REFERENCES dept)
   CLUSTER emp_dept (deptno);

CREATE TABLE dept (
   deptno NUMBER(3) PRIMARY KEY, . . . )
   CLUSTER emp_dept (deptno);
```

Note:

You can specify the schema for a clustered table in the `CREATE TABLE` statement. A clustered table can be in a different schema than the schema containing the cluster. Also, the names of the columns are not required to match, but their structure must match.

See Also:

Oracle Database SQL Language Reference for syntax of the `CREATE TABLE` statement for creating cluster tables

21.3.3 Creating Cluster Indexes

A cluster index must be created before any rows can be inserted into any clustered table.

To create a cluster index, one of the following conditions must be true:

- Your schema contains the cluster.
- You have the `CREATE ANY INDEX` system privilege.

In either case, you must also have either a quota for the tablespace intended to contain the cluster index, or the `UNLIMITED TABLESPACE` system privilege.

The following statement creates a cluster index for the `emp_dept` cluster:

```
CREATE INDEX emp_dept_index
   ON CLUSTER emp_dept
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33);
```

The cluster index clause (`ON CLUSTER`) identifies the cluster, `emp_dept`, for which the cluster index is being created. The statement also explicitly specifies several storage settings for the cluster and cluster index.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for syntax of the `CREATE INDEX` statement for creating cluster indexes

21.4 Altering Clusters and Objects That Use Them

You can alter a cluster to change its physical attributes, size, and default degree of parallelism.

- Altering Clusters
  You alter a cluster using the `ALTER CLUSTER` statement.
- Altering Clustered Tables
  You can alter clustered tables using the `ALTER TABLE` statement, but some parameters of a clustered table cannot be set with the `ALTER TABLE` statement.
- Altering Cluster Indexes
  You alter cluster indexes exactly as you do other indexes.

21.4.1 Altering Clusters

You alter a cluster using the `ALTER CLUSTER` statement.

To alter a cluster, your schema must contain the cluster or you must have the `ALTER ANY CLUSTER` system privilege. You can alter an existing cluster to change the following settings:

- Physical attributes (`INITRANS` and storage characteristics)

- The average amount of space required to store all the rows for a cluster key value (`SIZE`)
- The default degree of parallelism

Additionally, you can explicitly allocate a new extent for the cluster, or deallocate any unused extents at the end of the cluster. The database dynamically allocates additional extents for the data segment of a cluster as required. In some circumstances, however, you might want to explicitly allocate an additional extent for a cluster. For example, when using Real Application Clusters, you can allocate an extent of a cluster explicitly for a specific instance. You allocate a new extent for a cluster using the `ALTER CLUSTER` statement with the `ALLOCATE EXTENT` clause.

When you alter the cluster size parameter (`SIZE`) of a cluster, the new settings apply to all data blocks used by the cluster, including blocks already allocated and blocks subsequently allocated for the cluster. Blocks already allocated for the table are reorganized when necessary (not immediately).

When you alter the transaction entry setting `INITRANS` of a cluster, the new setting for `INITRANS` applies only to data blocks subsequently allocated for the cluster.

The storage parameters `INITIAL` and `MINEXTENTS` cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the cluster.

To alter a cluster, use the `ALTER CLUSTER` statement.

See Also:

Oracle Database SQL Language Reference for syntax of the `ALTER CLUSTER` statement

21.4.2 Altering Clustered Tables

You can alter clustered tables using the `ALTER TABLE` statement, but some parameters of a clustered table cannot be set with the `ALTER TABLE` statement.

However, any data block space parameters, transaction entry parameters, or storage parameters you set in an `ALTER TABLE` statement for a clustered table generate an error message (`ORA-01771, illegal option for a clustered table`). The database uses the parameters of the cluster for all clustered tables. Therefore, you can use the `ALTER TABLE` statement only to add or modify columns, drop non-cluster-key columns, or add, drop, enable, or disable integrity constraints or triggers for a clustered table. For information about altering tables, see "Altering Tables".

See Also:

Oracle Database SQL Language Reference for syntax of the `ALTER TABLE` statement

21.4.3 Altering Cluster Indexes

You alter cluster indexes exactly as you do other indexes.

See "Altering Indexes".

Note:

When estimating the size of cluster indexes, remember that the index is on each cluster key, not the actual rows. Therefore, each key appears only once in the index.

21.5 Dropping Clusters and Objects That Use Them

You drop a cluster using the `DROP CLUSTER` statement. You drop a clustered table using the `DROP TABLE` statement. You drop a cluster index using the `DROP INDEX` statement.

- Dropping Clusters
  You can drop a cluster using the `DROP CLUSTER` statement.
- Dropping Clustered Tables
  Clustered tables can be dropped individually without affecting the cluster, other clustered tables, or the cluster index. A clustered table is dropped just as a nonclustered table is dropped, with the `DROP TABLE` statement.
- Dropping Cluster Indexes
  A cluster index can be dropped without affecting the cluster or its clustered tables. However, clustered tables cannot be used if there is no cluster index; you must re-create the cluster index to allow access to the cluster.

21.5.1 Dropping Clusters

You can drop a cluster using the `DROP CLUSTER` statement.

A cluster can be dropped if the tables within the cluster are no longer needed. When a cluster is dropped, so are the tables within the cluster and the corresponding cluster index. All extents belonging to both the cluster data segment and the index segment of the cluster index are returned to the containing tablespace and become available for other segments within the tablespace.

To drop a cluster that contains no tables, and its cluster index, use the `DROP CLUSTER` statement. For example, the following statement drops the empty cluster named `emp_dept`:

```
DROP CLUSTER emp_dept;
```

If the cluster contains one or more clustered tables and you intend to drop the tables as well, add the `INCLUDING TABLES` clause of the `DROP CLUSTER` statement, as follows:

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

If the `INCLUDING TABLES` clause is not included and the cluster contains tables, an error is returned.

If one or more tables in a cluster contain primary or unique keys that are referenced by `FOREIGN KEY` constraints of tables outside the cluster, the cluster cannot be dropped unless the dependent `FOREIGN KEY` constraints are also dropped. This can be easily done using the `CASCADE CONSTRAINTS` clause of the `DROP CLUSTER` statement, as shown in the following example:

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

The database returns an error if you do not use the `CASCADE CONSTRAINTS` clause and constraints exist.

See Also:

Oracle Database SQL Language Reference for syntax of the `DROP CLUSTER` statement

21.5.2 Dropping Clustered Tables

Clustered tables can be dropped individually without affecting the cluster, other clustered tables, or the cluster index. A clustered table is dropped just as a nonclustered table is dropped, with the `DROP TABLE` statement.

To drop a cluster, your schema must contain the cluster or you must have the `DROP ANY CLUSTER` system privilege. You do not need additional privileges to drop a cluster that contains tables, even if the clustered tables are not owned by the owner of the cluster.

See "Dropping Table Columns ".

Note:

When you drop a single table from a cluster, the database deletes each row of the table individually. To maximize efficiency when you intend to drop an entire cluster, drop the cluster including all tables by using the `DROP CLUSTER` statement with the `INCLUDING TABLES` clause. Drop an individual table from a cluster (using the `DROP TABLE` statement) only if you want the rest of the cluster to remain.

21.5.3 Dropping Cluster Indexes

A cluster index can be dropped without affecting the cluster or its clustered tables. However, clustered tables cannot be used if there is no cluster index; you must re-create the cluster index to allow access to the cluster.

Cluster indexes are sometimes dropped as part of the procedure to rebuild a fragmented cluster index.

Note:

Hash cluster indexes cannot be dropped.

See Also:

"Dropping Indexes"

21.6 Clusters Data Dictionary Views

You can query a set of data dictionary views for information about clusters.

The following views display information about clusters:

View	Description

`DBA_CLUSTERS`
`ALL_CLUSTERS`
`USER_CLUSTERS`
	`DBA` view describes all clusters in the database. `ALL` view describes all clusters accessible to the user. `USER` view is restricted to clusters owned by the user. Some columns in these views contain statistics that are generated by the `DBMS_STATS` package or `ANALYZE` statement.

`DBA_CLU_COLUMNS` `USER_CLU_COLUMNS`	These views map table columns to cluster columns

See Also:

Oracle Database Reference for complete descriptions of these views

22 Managing Hash Clusters

Hash clusters can improve the performance of data retrieval.

- About Hash Clusters
  Storing a table in a hash cluster is an optional way to improve the performance of data retrieval. A hash cluster provides an alternative to a non-clustered table with an index or an index cluster.
- When to Use Hash Clusters
  You can decide when to use hash clusters by contrasting situations where hashing is most useful against situations where there is no advantage. If you find your decision is to use indexing rather than hashing, then you should consider whether to store a table individually or as part of a cluster.
- Creating Different Types of Hash Clusters
  You can use the `CREATE CLUSTER` statement with the `HASHKEYS` clause to create different types of hash clusters.
- Altering Hash Clusters
  You can alter a hash cluster with the `ALTER CLUSTER` statement.
- Dropping Hash Clusters
  You can drop a hash cluster using the `DROP CLUSTER` statement.
- Hash Clusters Data Dictionary Views
  You can query a set of data dictionary views for information about hash clusters.

22.1 About Hash Clusters

Storing a table in a hash cluster is an optional way to improve the performance of data retrieval. A hash cluster provides an alternative to a non-clustered table with an index or an index cluster.

With an indexed table or index cluster, Oracle Database locates the rows in a table using key values that the database stores in a separate index. To use hashing, you create a hash cluster and load tables into it. The database physically stores the rows of a table in a hash cluster and retrieves them according to the results of a **hash function**.

Oracle Database uses a hash function to generate a distribution of numeric values, called **hash values**, that are based on specific cluster key values. The key of a hash cluster, like the key of an index cluster, can be a single column or composite key (multiple column key). To find or store a row in a hash cluster, the database applies the hash function to the cluster key value of the row. The resulting hash value corresponds to a data block in the cluster, which the database then reads or writes on behalf of the issued statement.

To find or store a row in an indexed table or cluster, a minimum of two (there are usually more) I/Os must be performed:

- One or more I/Os to find or store the key value in the index
- Another I/O to read or write the row in the table or cluster

In contrast, the database uses a hash function to locate a row in a hash cluster; no I/O is required. As a result, a minimum of one I/O operation is necessary to read or write a row in a hash cluster.

See Also:

Managing Space for Schema Objects is recommended reading before attempting tasks described in this chapter.

22.2 When to Use Hash Clusters

You can decide when to use hash clusters by contrasting situations where hashing is most useful against situations where there is no advantage. If you find your decision is to use indexing rather than hashing, then you should consider whether to store a table individually or as part of a cluster.

Note:

Even if you decide to use hashing, a table can still have separate indexes on any columns, including the cluster key.

- Situations Where Hashing Is Useful
  Hashing is useful when most queries are equality queries on the cluster key and the tables in the hash cluster are primarily static in size.
- Situations Where Hashing Is Not Advantageous
  Hashing is not advantageous in certain situations.

22.2.1 Situations Where Hashing Is Useful

Hashing is useful when most queries are equality queries on the cluster key and the tables in the hash cluster are primarily static in size.

Hashing is useful when you have the following conditions:

- Most queries are equality queries on the cluster key:

  ```
  SELECT ... WHERE cluster_key = ...;
  ```

  In such cases, the cluster key in the equality condition is hashed, and the corresponding hash key is usually found with a single read. In comparison, for an indexed table the key value must first be found in the index (usually several reads), and then the row is read from the table (another read).
- The tables in the hash cluster are primarily static in size so that you can determine the number of rows and amount of space required for the tables in the cluster. If tables in a hash cluster require more space than the initial allocation for the cluster, performance degradation can be substantial because overflow blocks are required.

22.2.2 Situations Where Hashing Is Not Advantageous

Hashing is not advantageous in certain situations.

Hashing is not advantageous in the following situations:

- Most queries on the table retrieve rows over a range of cluster key values. For example, in full table scans or queries such as the following, a hash function cannot be used to determine the location of specific hash keys. Instead, the equivalent of a full table scan must be done to fetch the rows for the query.

  ```
  SELECT . . . WHERE cluster_key < . . . ;
  ```

  With an index, key values are ordered in the index, so cluster key values that satisfy the `WHERE` clause of a query can be found with relatively few I/Os.
- The table is not static, but instead is continually growing. If a table grows without limit, the space required over the life of the table (its cluster) cannot be predetermined.
- Applications frequently perform full-table scans on the table and the table is sparsely populated. A full-table scan in this situation takes longer under hashing.
- You cannot afford to preallocate the space that the hash cluster will eventually need.

22.3 Creating Different Types of Hash Clusters

You can use the `CREATE CLUSTER` statement with the `HASHKEYS` clause to create different types of hash clusters.

- Creating Hash Clusters
  You create a hash cluster using a `CREATE CLUSTER` statement, but you specify a `HASHKEYS` clause.
- Creating a Sorted Hash Cluster
  A **sorted hash cluster** stores the rows corresponding to each value of the hash function in such a way that the database can efficiently return them in sorted order. For applications that always consume data in sorted order, sorted hash clusters can retrieve data faster by minimizing logical I/Os.
- Creating Single-Table Hash Clusters
  You can create a **single-table hash cluster**, which provides fast access to rows in a table. However, this table must be the only table in the hash cluster.
- Controlling Space Use Within a Hash Cluster
  When creating a hash cluster, it is important to choose the cluster key correctly and set the `HASH IS`, `SIZE`, and `HASHKEYS` parameters so that performance and space use are optimal. The following guidelines describe how to set these parameters.
- Estimating Size Required by Hash Clusters
  As with index clusters, it is important to estimate the storage required for the data in a hash cluster.

22.3.1 Creating Hash Clusters

You create a hash cluster using a CREATE CLUSTER statement, but you specify a HASHKEYS clause.

The following statement creates a cluster named trial_cluster, clustered by the trialno column (the cluster key):

```
CREATE CLUSTER trial_cluster ( trialno NUMBER(5,0) )
    TABLESPACE users
    STORAGE ( INITIAL 250K
              NEXT 50K
              MINEXTENTS 1
              MAXEXTENTS 3
              PCTINCREASE 0 )
    HASH IS trialno
    HASHKEYS 150;
```

The following statement creates the trial table in the trial_cluster hash cluster:

```
CREATE TABLE trial (
    trialno NUMBER(5,0) PRIMARY KEY,
    ... )
    CLUSTER trial_cluster (trialno);
```

As with index clusters, the key of a hash cluster can be a single column or a composite key (multiple column key). In the preceding example, the key is the trialno column.

The HASHKEYS value, in this case 150, specifies and limits the number of unique hash values that the hash function can generate. The database rounds the number specified to the nearest prime number.

If no HASH IS clause is specified, then the database uses an internal hash function. If the cluster key is already a unique identifier that is uniformly distributed over its range, then you can bypass the internal hash function and specify the cluster key as the hash value, as in the preceding example. You can also use the HASH IS clause to specify a user-defined hash function.

You cannot create a cluster index on a hash cluster, and you need not create an index on a hash cluster key.

See Also:

Managing Clusters for additional information about creating tables in a cluster, guidelines for setting parameters of the CREATE CLUSTER statement common to index and hash clusters, and the privileges required to create any cluster

22.3.2 Creating a Sorted Hash Cluster

A **sorted hash cluster** stores the rows corresponding to each value of the hash function in such a way that the database can efficiently return them in sorted order. For applications that always consume data in sorted order, sorted hash clusters can retrieve data faster by minimizing logical I/Os.

Assume that a telecommunications company stores detailed call records for a fixed number of originating telephone numbers through a telecommunications switch. From each originating telephone number there can be an unlimited number of calls.

The application stores calls records as calls are made. Each call has a detailed call record identified by a timestamp. For example, the application stores a call record with timestamp 0, then a call record with timestamp 1, and so on.

When generating bills for each originating phone number, the application processes them in first-in, first-out (FIFO) order. The following table shows sample details for three originating phone numbers:

telephone_number	call_timestamp
6505551212	0, 1, 2, 3, 4, ...
6505551213	0, 1, 2, 3, 4, ...
6505551214	0, 1, 2, 3, 4, ...

In the following SQL statements, the `telephone_number` column is the hash key. The hash cluster is sorted on the `call_timestamp` and `call_duration` columns. The example uses the same names for the clustering and sorting columns in the table definition as in the cluster definition, but this is not required. The number of hash keys is based on 10-digit telephone numbers.

```
CREATE CLUSTER call_detail_cluster (
   telephone_number NUMBER,
   call_timestamp   NUMBER SORT,
   call_duration    NUMBER SORT )
  HASHKEYS 10000
  HASH IS telephone_number
  SIZE 256;

CREATE TABLE call_detail (
   telephone_number     NUMBER,
   call_timestamp       NUMBER   SORT,
   call_duration        NUMBER   SORT,
   other_info           VARCHAR2(30) )
  CLUSTER call_detail_cluster (
   telephone_number, call_timestamp, call_duration );
```

Example 22-1   Data Inserted in Sequential Order

Suppose that you seed the `call_detail` table with the rows in FIFO order as shown in this example.

```
INSERT INTO call_detail VALUES (6505551212, 0, 9, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 1, 17, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 2, 5, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 3, 90, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 0, 35, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 1, 6, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 2, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 3, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 0, 15, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 1, 20, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 2, 1, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 3, 25, 'misc info');
COMMIT;
```

Example 22-2 Querying call_detail

In this example, you `SET AUTOTRACE ON`, and then query the `call_detail` table for the call details for the phone number `6505551212`.

```
SQL> SET AUTOTRACE ON;
SQL> SELECT * FROM call_detail WHERE telephone_number = 6505551212;

TELEPHONE_NUMBER CALL_TIMESTAMP CALL_DURATION OTHER_INFO
---------------- -------------- ------------- ------------------------------
      6505551212              0             9 misc info
      6505551212              1            17 misc info
      6505551212              2             5 misc info
      6505551212              3            90 misc info

Execution Plan
----------------------------------------------------------
Plan hash value: 2118876266

----------------------------------------------------------------------
| Id  | Operation         | Name        | Rows  | Bytes | Cost (%CPU)|
----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |     1 |    56 |     0   (0)|
|*  1 |  TABLE ACCESS HASH| CALL_DETAIL |     1 |    56 |            |
----------------------------------------------------------------------
```

The query retrieves the rows ordered by timestamp even though no sort appears in the query plan.

Suppose you then delete the existing rows and insert the same rows out of sequence:

```
DELETE FROM call_detail;
INSERT INTO call_detail VALUES (6505551213, 3, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 0, 15, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 0, 9, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 1, 20, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 2, 1, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 1, 6, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 2, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 3, 25, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 1, 17, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 2, 5, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 3, 90, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 0, 35, 'misc info');
COMMIT;
```

If you rerun the same query of `call_detail`, the database again retrieves the rows in sorted order even though no `ORDER BY` clause is specified. No `SORT ORDER BY` operation appears in the query plan because the database performs an internal sort.

Now assume that you create a nonclustered table `call_detail_nonclustered` and then load it with the same sample values in Example 22-1. To retrieve the data in sorted order, you must use an `ORDER BY` clause as follows:

```
SQL> SELECT * FROM call_detail_nonclustered WHERE telephone_number = 6505551212
  2  ORDER BY call_timestamp, call_duration;

TELEPHONE_NUMBER CALL_TIMESTAMP CALL_DURATION OTHER_INFO
---------------- -------------- ------------- ------------------------------
```

```
      6505551212            0            9 misc info
      6505551212            1           17 misc info
      6505551212            2            5 misc info
      6505551212            3           90 misc info

Execution Plan
----------------------------------------------------------
Plan hash value: 2555750302

----------------------------------------------------------------------------------
|Id| Operation          | Name                     |Rows|Bytes|Cost (%CPU)|Time  |
----------------------------------------------------------------------------------
| 0| SELECT STATEMENT   |                          | 4  | 224 | 4 (25)| 00:00:01 |
| 1|  SORT ORDER BY     |                          | 4  | 224 | 4 (25)| 00:00:01 |
|*2|   TABLE ACCESS FULL| CALL_DETAIL_NONCLUSTERED | 4  | 224 | 3  (0)| 00:00:01 |
----------------------------------------------------------------------------------
```

The preceding plan shows that in the nonclustered case the sort is more expensive than in the clustered case. The rows, bytes, cost, and time are all greater in the case of the table that is not stored in a sorted hash cluster.

22.3.3 Creating Single-Table Hash Clusters

You can create a **single-table hash cluster**, which provides fast access to rows in a table. However, this table must be the only table in the hash cluster.

Essentially, there must be a one-to-one mapping between hash keys and data rows. The following statement creates a single-table hash cluster named `peanut` with the cluster key `variety`:

```
CREATE CLUSTER peanut (variety NUMBER)
   SIZE 512 SINGLE TABLE HASHKEYS 500;
```

The database rounds the `HASHKEYS` value up to the nearest prime number, so this cluster has a maximum of 503 hash key values, each of size 512 bytes. The `SINGLE TABLE` clause is valid only for hash clusters. `HASHKEYS` must also be specified.

See Also:

Oracle Database SQL Language Reference for the syntax of the `CREATE CLUSTER` statement

22.3.4 Controlling Space Use Within a Hash Cluster

When creating a hash cluster, it is important to choose the cluster key correctly and set the `HASH IS`, `SIZE`, and `HASHKEYS` parameters so that performance and space use are optimal. The following guidelines describe how to set these parameters.

- Choosing the Key
  Choosing the correct cluster key is dependent on the most common types of queries issued against the clustered tables.
- Setting HASH IS
  Specify the `HASH IS` parameter only if the cluster key is a single column of the `NUMBER` data type, and contains uniformly distributed integers.

- Setting SIZE
  SIZE should be set to the average amount of space required to hold all rows for any given hash key.
- Setting HASHKEYS
  Specify the HASHKEYS clause to create a hash cluster and specify the number of hash values for the hash cluster.
- Controlling Space in Hash Clusters
  Examples illustrate how to correctly choose the cluster key and set the HASH IS, SIZE, and HASHKEYS parameters. For all examples, assume that the data block size is 2K and that on average, 1950 bytes of each block is available data space (block size minus overhead).

22.3.4.1 Choosing the Key

Choosing the correct cluster key is dependent on the most common types of queries issued against the clustered tables.

For example, consider the `emp` table in a hash cluster. If queries often select rows by employee number, the `empno` column should be the cluster key. If queries often select rows by department number, the `deptno` column should be the cluster key. For hash clusters that contain a single table, the cluster key is typically the entire primary key of the contained table.

The key of a hash cluster, like that of an index cluster, can be a single column or a composite key (multiple column key). A hash cluster with a composite key must use the internal hash function of the database.

22.3.4.2 Setting HASH IS

Specify the `HASH IS` parameter only if the cluster key is a single column of the `NUMBER` data type, and contains uniformly distributed integers.

If these conditions apply, you can distribute rows in the cluster so that each unique cluster key value hashes, with no collisions (two cluster key values having the same hash value), to a unique hash value. If these conditions do not apply, omit this clause so that you use the internal hash function.

22.3.4.3 Setting SIZE

`SIZE` should be set to the average amount of space required to hold all rows for any given hash key.

Therefore, to properly determine `SIZE`, you must be aware of the characteristics of your data:

- If the hash cluster is to contain only a single table and the hash key values of the rows in that table are unique (one row for each value), `SIZE` can be set to the average row size in the cluster.
- If the hash cluster is to contain multiple tables, `SIZE` can be set to the average amount of space required to hold all rows associated with a representative hash value.

Further, once you have determined a (preliminary) value for `SIZE`, consider the following. If the `SIZE` value is small (more than four hash keys can be assigned for each data block) you can use this value for `SIZE` in the `CREATE CLUSTER` statement. However, if the value of `SIZE` is large (four or fewer hash keys can be assigned for each data block), then you should also consider the expected frequency of collisions and whether performance of data retrieval or efficiency of space usage is more important to you.

- If the hash cluster does not use the internal hash function (if you specified `HASH IS`) and you expect few or no collisions, you can use your preliminary value of `SIZE`. No collisions occur and space is used as efficiently as possible.
- If you expect frequent collisions on inserts, the likelihood of overflow blocks being allocated to store rows is high. To reduce the possibility of overflow blocks and maximize performance when collisions are frequent, you should adjust `SIZE` as shown in the following chart.

Available Space for each Block / Calculated SIZE	Setting for SIZE
1	`SIZE`
2	`SIZE` + 15%
3	`SIZE` + 12%
4	`SIZE` + 8%
>4	`SIZE`

Overestimating the value of `SIZE` increases the amount of unused space in the cluster. If space efficiency is more important than the performance of data retrieval, disregard the adjustments shown in the preceding table and use the original value for `SIZE`.

22.3.4.4 Setting HASHKEYS

Specify the `HASHKEYS` clause to create a hash cluster and specify the number of hash values for the hash cluster.

For maximum distribution of rows in a hash cluster, the database rounds the `HASHKEYS` value up to the nearest prime number.

22.3.4.5 Controlling Space in Hash Clusters

Examples illustrate how to correctly choose the cluster key and set the `HASH IS`, `SIZE`, and `HASHKEYS` parameters. For all examples, assume that the data block size is 2K and that on average, 1950 bytes of each block is available data space (block size minus overhead).

- Controlling Space in Hash Clusters: Example 1
  An example illustrates controlling space in hash clusters.
- Controlling Space in Hash Clusters: Example 2
  An example illustrates controlling space in hash clusters.

22.3.4.5.1 Controlling Space in Hash Clusters: Example 1

An example illustrates controlling space in hash clusters.

You decide to load the `emp` table into a hash cluster. Most queries retrieve employee records by their employee number. You estimate that the maximum number of rows in the `emp` table at any given time is 10000 and that the average row size is 55 bytes.

In this case, `empno` should be the cluster key. Because this column contains integers that are unique, the internal hash function can be bypassed. `SIZE` can be set to the average row size, 55 bytes. Note that 34 hash keys are assigned for each data block. `HASHKEYS` can be set to the number of rows in the table, 10000. The database rounds this value up to the next highest prime number: 10007.

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10000;
```

22.3.4.5.2 Controlling Space in Hash Clusters: Example 2

An example illustrates controlling space in hash clusters.

In this example, conditions are similar to the example in "Controlling Space in Hash Clusters: Example 1 ". In this case, however, rows are usually retrieved by department number. At most, there are 1000 departments with an average of 10 employees for each department. Department numbers increment by 10 (0, 10, 20, 30, . . .).

In this case, `deptno` should be the cluster key. Since this column contains integers that are uniformly distributed, the internal hash function can be bypassed. A preliminary value of `SIZE` (the average amount of space required to hold all rows for each department) is 55 bytes * 10, or 550 bytes. Using this value for `SIZE`, only three hash keys can be assigned for each data block. If you expect some collisions and want maximum performance of data retrieval, slightly alter your estimated `SIZE` to prevent collisions from requiring overflow blocks. By adjusting `SIZE` by 12%, to 620 bytes (see "Setting SIZE"), there is more space for rows from expected collisions.

`HASHKEYS` can be set to the number of unique department numbers, 1000. The database rounds this value up to the next highest prime number: 1009.

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
. . .
SIZE 620
HASH IS deptno HASHKEYS 1000;
```

22.3.5 Estimating Size Required by Hash Clusters

As with index clusters, it is important to estimate the storage required for the data in a hash cluster.

Oracle Database guarantees that the initial allocation of space is sufficient to store the hash table according to the settings `SIZE` and `HASHKEYS`. If settings for the storage parameters `INITIAL`, `NEXT`, and `MINEXTENTS` do not account for the hash table size, incremental (additional) extents are allocated until at least `SIZE*HASHKEYS` is reached. For example, assume that the data block size is 2K, the available data space for each block is approximately 1900 bytes (data block size minus overhead), and that the `STORAGE` and `HASH` parameters are specified in the `CREATE CLUSTER` statement as follows:

```
STORAGE (INITIAL 100K
    NEXT 150K
    MINEXTENTS 1
    PCTINCREASE 0)
SIZE 1500
HASHKEYS 100
```

In this example, only one hash key can be assigned for each data block. Therefore, the initial space required for the hash cluster is at least 100*2K or 200K. The settings for the storage parameters do not account for this requirement. Therefore, an initial extent of 100K and a second extent of 150K are allocated to the hash cluster.

Alternatively, assume the `HASH` parameters are specified as follows:

```
SIZE 500 HASHKEYS 100
```

In this case, three hash keys are assigned to each data block. Therefore, the initial space required for the hash cluster is at least 34*2K or 68K. The initial settings for the storage parameters are sufficient for this requirement (an initial extent of 100K is allocated to the hash cluster).

22.4 Altering Hash Clusters

You can alter a hash cluster with the `ALTER CLUSTER` statement.

For example, the following `ALTER CLUSTER` statement alters the `emp_dept` cluster:

```
ALTER CLUSTER emp_dept . . . ;
```

The implications for altering a hash cluster are identical to those for altering an index cluster, described in "Altering Clusters". However, the `SIZE`, `HASHKEYS`, and `HASH IS` parameters cannot be specified in an `ALTER CLUSTER` statement. To change these parameters, you must re-create the cluster, then copy the data from the original cluster.

22.5 Dropping Hash Clusters

You can drop a hash cluster using the `DROP CLUSTER` statement.

For example, the following `DROP CLUSTER` statement drops the `emp_dept` cluster:

```
DROP CLUSTER emp_dept;
```

A table in a hash cluster is dropped using the `DROP TABLE` statement. The implications of dropping hash clusters and tables in hash clusters are the same as those for dropping index clusters.

See Also:

"Dropping Clusters"

22.6 Hash Clusters Data Dictionary Views

You can query a set of data dictionary views for information about hash clusters.

The following views display information about hash clusters:

View	Description

`DBA_CLUSTERS`
`ALL_CLUSTERS`
`USER_CLUSTERS`
	`DBA` view describes all clusters (including hash clusters) in the database. `ALL` view describes all clusters accessible to the user. `USER` view is restricted to clusters owned by the user. Some columns in these views contain statistics that are generated by the `DBMS_STATS` package or `ANALYZE` statement.

`DBA_CLU_COLUMNS` `USER_CLU_COLUMNS`	These views map table columns to cluster columns.

View	Description

`DBA_CLUSTER_HASH_EXPRESSIONS`
`ALL_CLUSTER_HASH_EXPRESSIONS`
`USER_CLUSTER_HASH_EXPRESSIONS`
	These views list hash functions for hash clusters.

23 Managing Views, Sequences, and Synonyms

You can create and manage views, sequences, and synonyms with Oracle Database.

- Managing Views
  You can perform tasks such as creating views, replacing views, altering views, and dropping views.
- Managing Sequences
  You can perform tasks such as creating sequences, altering sequences, using sequences, and dropping sequences.
- Managing Synonyms
  You can perform tasks such as creating synonyms, using synonyms, and dropping synonyms.
- Views, Synonyms, and Sequences Data Dictionary Views
  You can query data dictionary views for information about views, synonyms, and sequences.

23.1 Managing Views

You can perform tasks such as creating views, replacing views, altering views, and dropping views.

Live SQL:

To view and run examples related to managing views on Oracle Live SQL, go to *Oracle Live SQL: Creating, Replacing, and Dropping a View*.

- About Views
  A **view** is a logical representation of a table or combination of tables. In essence, a view is a stored query.
- Creating Views and Join Views
  You can create views using the `CREATE VIEW` statement. Each view is defined by a query that references tables, materialized views, or other views. You can also create join views that specify multiple base tables or views in the `FROM` clause.
- Replacing Views
  You can replace a view by dropping it and re-creating it or by issuing a `CREATE VIEW` statement that contains the `OR REPLACE` clause.
- Using Views in Queries
  You can query a view. You can also perform data manipulation language (DML) operations on views, with some restrictions.
- DML Statements and Join Views
  Restrictions apply when issuing DML statements on join views.

- Altering Views
  You use the `ALTER VIEW` statement only to explicitly recompile a view that is invalid.
- Dropping Views
  You can drop a view with the `DROP VIEW` statement.

23.1.1 About Views

A **view** is a logical representation of a table or combination of tables. In essence, a view is a stored query.

A view derives its data from the tables on which it is based. These tables are called **base tables**. Base tables might in turn be actual tables or might be views themselves. All operations performed on a view actually affect the base table of the view. You can use views in almost the same way as tables. You can query, update, insert into, and delete from views, just as you can standard tables.

Views can provide a different representation (such as subsets or supersets) of the data that resides within other tables and views. Views are very powerful because they allow you to tailor the presentation of data to different types of users.

Note:

One special type of view is the editioning view, which is used only to support online upgrade of applications using edition-based redefinition. The remainder of this section on managing views describes all views except editioning views. See *Oracle Database Development Guide* for a discussion of editioning views and edition-based redefinition.

See Also:

Oracle Database Concepts for an overview of views

23.1.2 Creating Views and Join Views

You can create views using the `CREATE VIEW` statement. Each view is defined by a query that references tables, materialized views, or other views. You can also create join views that specify multiple base tables or views in the `FROM` clause.

- Creating Views
  You can create a view with the `CREATE VIEW` statement.
- Creating Join Views
  You can also create views that specify multiple base tables or views in the `FROM` clause of a `CREATE VIEW` statement. These are called **join views**.
- Expansion of Defining Queries at View Creation Time
  When a view is created, Oracle Database expands any wildcard (*) in a top-level view query into a column list. The resulting query is stored in the data dictionary; any subqueries are left intact.

- Creating Views with Errors
  If there are no syntax errors in a `CREATE VIEW` statement, then the database can create the view even if the defining query of the view cannot be executed. In this case, the view is considered "created with errors."

23.1.2.1 Creating Views

You can create a view with the `CREATE VIEW` statement.

To create a view, you must meet the following requirements:

- To create a view in your schema, you must have the `CREATE VIEW` privilege. To create a view in another user's schema, you must have the `CREATE ANY VIEW` system privilege. You can acquire these privileges explicitly or through a role.
- The owner of the view (whether it is you or another user) must have been explicitly granted privileges to access all objects referenced in the view definition. The owner *cannot* have obtained these privileges through roles. Also, the functionality of the view depends on the privileges of the view owner. For example, if the owner of the view has only the `INSERT` privilege for Scott's `emp` table, then the view can be used only to insert new rows into the `emp` table, not to `SELECT`, `UPDATE`, or `DELETE` rows.
- If the owner of the view intends to grant access to the view to other users, the owner must have received the object privileges to the base objects with the `GRANT OPTION` or the system privileges with the `ADMIN OPTION`.

You can create views using the `CREATE VIEW` statement. Each view is defined by a query that references tables, materialized views, or other views. As with all subqueries, the query that defines a view cannot contain the `FOR UPDATE` clause.

The following statement creates a view on a subset of data in the `hr.departments` table:

```
CREATE VIEW departments_hq AS
      SELECT department_id, department_name, location_id
      FROM hr.departments
      WHERE location_id = 1700
    WITH CHECK OPTION CONSTRAINT departments_hq_cnst;
```

The query that defines the `departments_hq` view references only rows in location 1700. Furthermore, the `CHECK OPTION` creates the view with the constraint (named `departments_hq_cnst`) so that `INSERT` and `UPDATE` statements issued against the view cannot result in rows that the view cannot select. For example, the following `INSERT` statement successfully inserts a row into the `departments` table with the `departments_hq` view, which contains all rows with location 1700:

```
INSERT INTO departments_hq VALUES (300, 'NETWORKING', 1700);
```

However, the following `INSERT` statement returns an error because it attempts to insert a row for location 2700, which cannot be selected using the `departments_hq` view:

```
INSERT INTO departments_hq VALUES (301, 'TRANSPORTATION', 2700);
```

The view could have been constructed specifying the `WITH READ ONLY` clause, which prevents any updates, inserts, or deletes from being done to the base table through the view. If no `WITH` clause is specified, the view, with some restrictions, is inherently updatable.

You can also create views with invisible columns. For example, the following statements creates the `departments_hq_man` view and makes the `manager_id` column invisible:

```
CREATE VIEW departments_hq_man
         (department_id, department_name, manager_id INVISIBLE, location_id)
      AS SELECT department_id, department_name, manager_id, location_id
      FROM hr.departments
      WHERE location_id = 1700
    WITH CHECK OPTION CONSTRAINT departments_hq_man_cnst;
```

See Also:

- *Oracle Database SQL Language Reference* for syntax and semantics of the `CREATE VIEW` statement
- "Understand Invisible Columns"

23.1.2.2 Creating Join Views

You can also create views that specify multiple base tables or views in the `FROM` clause of a `CREATE VIEW` statement. These are called **join views**.

The following statement creates the `division1_staff` view that joins data from the `emp` and `dept` tables:

```
CREATE VIEW division1_staff AS
      SELECT ename, empno, job, dname
      FROM emp, dept
      WHERE emp.deptno IN (10, 30)
         AND emp.deptno = dept.deptno;
```

An **updatable join view** is a join view where `UPDATE`, `INSERT`, and `DELETE` operations are allowed. See "Updating a Join View" for further discussion.

23.1.2.3 Expansion of Defining Queries at View Creation Time

When a view is created, Oracle Database expands any wildcard (*) in a top-level view query into a column list. The resulting query is stored in the data dictionary; any subqueries are left intact.

The column names in an expanded column list are enclosed in quotation marks to account for the possibility that the columns of the base object were originally entered with quotes and require them for the query to be syntactically correct.

As an example, assume that the `dept` view is created as follows:

```
CREATE VIEW dept AS SELECT * FROM scott.dept;
```

The database stores the defining query of the `dept` view as:

```
SELECT "DEPTNO", "DNAME", "LOC" FROM scott.dept;
```

Views created with errors do not have wildcards expanded. However, if the view is eventually compiled without errors, wildcards in the defining query are expanded.

23.1.2.4 Creating Views with Errors

If there are no syntax errors in a `CREATE VIEW` statement, then the database can create the view even if the defining query of the view cannot be executed. In this case, the view is considered "created with errors."

For example, when a view is created that refers to a nonexistent table or an invalid column of an existing table, or when the view owner does not have the required privileges, the view can be created anyway and entered into the data dictionary. However, the view is not yet usable.

To create a view with errors, you must include the `FORCE` clause of the `CREATE VIEW` statement.

```
CREATE FORCE VIEW AS ...;
```

By default, views with errors are created as `INVALID`. When you try to create such a view, the database returns a message indicating the view was created with errors. If conditions later change so that the query of an invalid view can be executed, the view can be recompiled and be made valid (usable). For information changing conditions and their impact on views, see "Managing Object Dependencies".

23.1.3 Replacing Views

You can replace a view by dropping it and re-creating it or by issuing a `CREATE VIEW` statement that contains the `OR REPLACE` clause.

To replace a view, you must have all of the privileges required to drop and create a view. If the definition of a view must change, the view must be replaced; you cannot use an `ALTER VIEW` statement to change the definition of a view. You can replace views in the following ways:

- You can drop and re-create the view.

  Note:

  When a view is dropped, all grants of corresponding object privileges are revoked from roles and users. After the view is re-created, privileges must be regranted.

- You can redefine the view with a `CREATE VIEW` statement that contains the `OR REPLACE` clause. The `OR REPLACE` clause replaces the current definition of a view and preserves the current security authorizations. For example, assume that you created the `sales_staff` view as shown earlier, and, in addition, you granted several object privileges to roles and other users. However, now you must redefine the `sales_staff` view to change the department number specified in the `WHERE` clause. You can replace the current version of the `sales_staff` view with the following statement:

  ```
  CREATE OR REPLACE VIEW sales_staff AS
       SELECT empno, ename, deptno
       FROM emp
       WHERE deptno = 30
       WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
  ```

Before replacing a view, consider the following effects:

- Replacing a view replaces the view definition in the data dictionary. All underlying objects referenced by the view are not affected.

- If a constraint in the CHECK OPTION was previously defined but not included in the new view definition, the constraint is dropped.
- All views dependent on a replaced view become invalid (not usable). In addition, dependent PL/SQL program units may become invalid, depending on what was changed in the new version of the view. For example, if only the WHERE clause of the view changes, dependent PL/SQL program units remain valid. However, if any changes are made to the number of view columns or to the view column names or data types, dependent PL/SQL program units are invalidated. See "Managing Object Dependencies" for more information on how the database manages such dependencies.

23.1.4 Using Views in Queries

You can query a view. You can also perform data manipulation language (DML) operations on views, with some restrictions.

To issue a query or an INSERT, UPDATE, or DELETE statement against a view, you must have the SELECT, READ, INSERT, UPDATE, or DELETE object privilege for the view, respectively, either explicitly or through a role.

Views can be queried in the same manner as tables. For example, to query the Division1_staff view, enter a valid SELECT statement that references the view:

```
SELECT * FROM Division1_staff;

ENAME        EMPNO       JOB             DNAME
------------------------------------------------------
CLARK         7782       MANAGER         ACCOUNTING
KING          7839       PRESIDENT       ACCOUNTING
MILLER        7934       CLERK           ACCOUNTING
ALLEN         7499       SALESMAN        SALES
WARD          7521       SALESMAN        SALES
JAMES         7900       CLERK           SALES
TURNER        7844       SALESMAN        SALES
MARTIN        7654       SALESMAN        SALES
BLAKE         7698       MANAGER         SALES
```

With some restrictions, rows can be inserted into, updated in, or deleted from a base table using a view. The following statement inserts a new row into the emp table using the sales_staff view:

```
INSERT INTO sales_staff
    VALUES (7954, 'OSTER', 30);
```

Restrictions on DML operations for views use the following criteria in the order listed:

1. If a view is defined by a query that contains SET or DISTINCT operators, a GROUP BY clause, or a group function, then rows cannot be inserted into, updated in, or deleted from the base tables using the view.
2. If a view is defined with WITH CHECK OPTION, a row cannot be inserted into, or updated in, the base table (using the view), if the view cannot select the row from the base table.
3. If a NOT NULL column that does not have a DEFAULT clause is omitted from the view, then a row cannot be inserted into the base table using the view.
4. If the view was created by using an expression, such as DECODE(deptno, 10, "SALES", ...), then rows cannot be inserted into or updated in the base table using the view.

The constraint created by WITH CHECK OPTION of the sales_staff view only allows rows that have a department number of 30 to be inserted into, or updated in, the emp table. Alternatively, assume that the sales_staff view is defined by the following statement (that is, excluding the deptno column):

```
CREATE VIEW sales_staff AS
    SELECT empno, ename
    FROM emp
    WHERE deptno = 10
    WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

Considering this view definition, you can update the empno or ename fields of existing records, but you cannot insert rows into the emp table through the sales_staff view because the view does not let you alter the deptno field. If you had defined a DEFAULT value of 10 on the deptno field, then you could perform inserts.

When a user attempts to reference an invalid view, the database returns an error message to the user:

```
ORA-04063: view 'view_name' has errors
```

This error message is returned when a view exists but is unusable due to errors in its query (whether it had errors when originally created or it was created successfully but became unusable later because underlying objects were altered or dropped).

23.1.5 DML Statements and Join Views

Restrictions apply when issuing DML statements on join views.

- Updating a Join View
  An updatable join view (also referred to as a **modifiable join view**) is a view that contains multiple tables in the top-level FROM clause of the SELECT statement, and is not restricted by the WITH READ ONLY clause.
- Key-Preserved Tables
  A table is key-preserved if every key of the table can also be a key of the result of the join that is based on the table. So, a key-preserved table has its keys preserved through a join.
- Rules for DML Statements and Join Views
  The general rule is that any UPDATE, DELETE, or INSERT statement on a join view can modify only one underlying base table.
- Updating Views That Involve Outer Joins
  Views that involve outer joins are modifiable in some cases.
- Using the UPDATABLE_ COLUMNS Views
  A set of views can assist you in identifying inherently updatable join views.

23.1.5.1 Updating a Join View

An updatable join view (also referred to as a **modifiable join view**) is a view that contains multiple tables in the top-level FROM clause of the SELECT statement, and is not restricted by the WITH READ ONLY clause.

The rules for updatable join views are shown in the following table. Views that meet these criteria are said to be inherently updatable.

Rule	Description
General Rule	Any INSERT, UPDATE, or DELETE operation on a join view can modify only one underlying base table at a time.
UPDATE Rule	Rows from a join view can be updated if the join column keys in the base tables are unique. That is, the WHERE clause in the UPDATE statement must be deterministic. If the base tables are not key-preserved, you must ensure that the join column keys are unique. If the view is defined with the WITH CHECK OPTION clause, then all join columns and all columns of repeated tables are not updatable.
DELETE Rule	Rows from a join view can be deleted as long as there is exactly one key-preserved table in the join. The key preserved table can be repeated in the FROM clause. If the view is defined with the WITH CHECK OPTION clause and the key preserved table is repeated, then the rows cannot be deleted from the view.
INSERT Rule	An INSERT statement must not explicitly or implicitly refer to the columns of a **non-key-preserved table**. If the join view is defined with the WITH CHECK OPTION clause, INSERT statements are not permitted.

There are data dictionary views that indicate whether the columns in a join view are inherently updatable. See "Using the UPDATABLE_ COLUMNS Views" for descriptions of these views.

Note:

There are some additional restrictions and conditions that can affect whether a join view is inherently updatable. Specifics are listed in the description of the CREATE VIEW statement in the *Oracle Database SQL Language Reference.*

If a view is not inherently updatable, it can be made updatable by creating an INSTEAD OF trigger on it. See *Oracle Database PL/SQL Language Reference* for information about triggers.

Additionally, if a view is a join on other nested views, then the other nested views must be mergeable into the top level view. For a discussion of mergeable and unmergeable views, and more generally, how the optimizer optimizes statements that reference views, see the *Oracle Database SQL Tuning Guide.*

Examples illustrating the rules for inherently updatable join views, and a discussion of key-preserved tables, are presented in following sections. The examples in these sections work only if you explicitly define the primary and foreign keys in the tables, or define unique indexes. The following statements create the appropriately constrained table definitions for emp and dept.

```
CREATE TABLE dept (
      deptno        NUMBER(4) PRIMARY KEY,
      dname         VARCHAR2(14),
      loc           VARCHAR2(13));

CREATE TABLE emp (
      empno        NUMBER(4) PRIMARY KEY,
      ename        VARCHAR2(10),
      job          VARCHAR2(9),
      mgr          NUMBER(4),
      sal          NUMBER(7,2),
      comm         NUMBER(7,2),
      deptno       NUMBER(2),
      FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO));
```

You could also omit the primary and foreign key constraints listed in the preceding example, and create a `UNIQUE INDEX` on `dept (deptno)` to make the following examples work.

The following statement created the `emp_dept` join view which is referenced in the examples:

```
CREATE VIEW emp_dept AS
      SELECT emp.empno, emp.ename, emp.deptno, emp.sal, dept.dname, dept.loc
      FROM emp, dept
      WHERE emp.deptno = dept.deptno
         AND dept.loc IN ('DALLAS', 'NEW YORK', 'BOSTON');
```

23.1.5.2 Key-Preserved Tables

A table is key-preserved if every key of the table can also be a key of the result of the join that is based on the table. So, a key-preserved table has its keys preserved through a join.

Note:

It is not necessary that the key or keys of a table be selected for it to be key preserved. It is sufficient that if the key or keys were selected, then they would also be keys of the result of the join.

The concept of a key-preserved table is fundamental to understanding the restrictions on modifying join views. Each row in a key-preserved table appears at most only once in a join view based on this table. If a table `T` is joined to a table `S` using the condition `T.col1 = S.col3`, then `T` is key-preserved if the join key is `S.col3` are unique. The data in a table is not relevant when determining whether a table is key-preserved. Instead, the constraints on the table determine if a table is key-preserved. Key-preserved tables are joined with a source table using the primary key or unique key of the source table.

For example, in the `emp_dept` view, because `emp` is joined with the primary key of `dept`, `emp` is a key-preserved table. If, in the `emp` table, there was at most one employee in each department, then `deptno` would be unique in the result of a join of `emp` and `dept`, but `dept` would still not be a key-preserved table.

If you select all rows from the `emp_dept` view, the results are:

```
EMPNO      ENAME      DEPTNO  DNAME          LOC
---------- ---------- ------- -------------- -----------
      7782 CLARK           10 ACCOUNTING     NEW YORK
      7839 KING            10 ACCOUNTING     NEW YORK
      7934 MILLER          10 ACCOUNTING     NEW YORK
      7369 SMITH           20 RESEARCH       DALLAS
      7876 ADAMS           20 RESEARCH       DALLAS
      7902 FORD            20 RESEARCH       DALLAS
      7788 SCOTT           20 RESEARCH       DALLAS
      7566 JONES           20 RESEARCH       DALLAS
8 rows selected.
```

In this view, `emp` is a key-preserved table, because `empno` is a key of the `emp` table, and also a key of the result of the join. `dept` is *not* a key-preserved table, because although `deptno` is a key of the `dept` table, it is not a key of the join.

23.1.5.3 Rules for DML Statements and Join Views

The general rule is that any `UPDATE`, `DELETE`, or `INSERT` statement on a join view can modify only one underlying base table.

- UPDATE Statements and Join Views
  Examples illustrate `UPDATE` statements that can modify join views.
- DELETE Statements and Join Views
  For most join views, a delete is successful only if there is *one and only one* key-preserved table in the join. The key-preserved table can be repeated in the `FROM` clause.
- INSERT Statements and Join Views
  Examples illustrate `INSERT` statements that can modify join views.

23.1.5.3.1 UPDATE Statements and Join Views

Examples illustrate `UPDATE` statements that can modify join views.

Starting with Oracle Database Release 21c, it is not mandatory for all updatable columns in a join view to map to columns of a key-preserved table. The columns in a non-key-preserved table can be updated if the `UPDATE` operation only updates columns from a single table and the update is deterministic, meaning that it updates each row only once.

The following example shows an `UPDATE` statement that successfully modifies the `emp_dept` view:

```
UPDATE emp_dept
     SET    sal = sal * 1.10
     WHERE  deptno = 10;
```

The following `UPDATE` statement successfully modifies the `LOC` column in the `DEPT` table (the non-key-preserved table) because it updates only one row in the `EMP` table:

```
UPDATE emp_dept
     SET    loc = 'BOSTON'
     WHERE  ename = 'SMITH';
```

The following `UPDATE` statement results in an `ORA-30926` error because the update operation is non-deterministic:

```
UPDATE emp_dept
     SET    loc = 'BOSTON'
     WHERE  ename = 'S%';
```

A row from the `dept` table is joined to multiple rows from `emp` table (with `ename` = '`SCOTT`' and `ename` = 'SMITH'). Therefore, an attempt is made to modify the same row multiple times. To make the `UPDATE` is deterministic, ensure that a row from the `dept` table is only joined to one row from `emp` table.

In general, all updatable columns of a join view must map to columns of a key-preserved table. If the view is defined using the `WITH CHECK OPTION` clause, then all join columns and all columns taken from tables that are referenced more than once in the view are not modifiable.

So, for example, if the `emp_dept` view were defined using `WITH CHECK OPTION`, the following `UPDATE` statement would fail:

```
UPDATE emp_dept
     SET    deptno = 10
     WHERE  ename = 'SMITH';
```

The statement fails because it is trying to update a join column.

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `UPDATE` statement

23.1.5.3.2 DELETE Statements and Join Views

For most join views, a delete is successful only if there is *one and only one* key-preserved table in the join. The key-preserved table can be repeated in the `FROM` clause.

The following `DELETE` statement works on the `emp_dept` view:

```
DELETE FROM emp_dept
     WHERE ename = 'SMITH';
```

This `DELETE` statement on the `emp_dept` view is legal because it can be translated to a `DELETE` operation on the base `emp` table, and because the `emp` table is the only key-preserved table in the join.

In the following view, a `DELETE` operation is permitted, because although there are two key-preserved tables, they are the same table. That is, the key-preserved table is repeated. In this case, the delete statement operates on the first table in the `FROM` clause (`e1`, in this example):

```
CREATE VIEW emp_emp AS
     SELECT e1.ename, e2.empno, e2.deptno
     FROM emp e1, emp e2
     WHERE e1.empno = e2.empno;
```

If a view is defined using the `WITH CHECK OPTION` clause and the key-preserved table is repeated, rows cannot be deleted from such a view.

```
CREATE VIEW emp_mgr AS
     SELECT e1.ename, e2.ename mname
     FROM emp e1, emp e2
     WHERE e1.mgr = e2.empno
     WITH CHECK OPTION;
```

Note:

- If the DELETE statement uses the same column in its WHERE clause that was used to create the view as a join condition, then the delete operation can be successful when there are different key-preserved tables in the join. In this case, the DELETE statement operates on the first table in the FROM clause, and the tables in the FROM clause can be different from the tables in the WHERE clause.
- The DELETE statement is successful, even if it does not use the WHERE clause.
- The DELETE statement is successful, even if it uses a different column in its WHERE clause than the one that was used to create the view as a join condition.
- The DELETE statement operates on the second table in the FROM clause in all the cases, because no primary key is defined on the second table.
- If a primary key is defined on the second table, then the DELETE statement operates on the first table in the FROM clause.

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the DELETE statement

23.1.5.3.3 INSERT Statements and Join Views

Examples illustrate INSERT statements that can modify join views.

The following INSERT statement on the emp_dept view succeeds:

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 40);
```

This statement works because only one key-preserved base table is being modified (emp), and 40 is a valid deptno in the dept table (thus satisfying the FOREIGN KEY integrity constraint on the emp table).

An INSERT statement, such as the following, would fail for the same reason that such an UPDATE on the base emp table would fail: the FOREIGN KEY integrity constraint on the emp table is violated (because there is no deptno 77).

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 77);
```

The following INSERT statement would fail with an error (ORA-01776 cannot modify more than one base table through a join view):

```
INSERT INTO emp_dept (empno, ename, loc)
   VALUES (9010, 'KURODA', 'BOSTON');
```

An INSERT cannot implicitly or explicitly refer to columns of a non-key-preserved table. If the join view is defined using the WITH CHECK OPTION clause, then you cannot perform an INSERT to it.

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `INSERT` statement

23.1.5.4 Updating Views That Involve Outer Joins

Views that involve outer joins are modifiable in some cases.

For example:

```
CREATE VIEW emp_dept_oj1 AS
    SELECT empno, ename, e.deptno, dname, loc
    FROM emp e, dept d
    WHERE e.deptno = d.deptno (+);
```

The statement:

```
SELECT * FROM emp_dept_oj1;
```

Results in:

```
EMPNO   ENAME      DEPTNO  DNAME          LOC
------- ---------- ------- -------------- -------------
7369    SMITH      40      OPERATIONS     BOSTON
7499    ALLEN      30      SALES          CHICAGO
7566    JONES      20      RESEARCH       DALLAS
7654    MARTIN     30      SALES          CHICAGO
7698    BLAKE      30      SALES          CHICAGO
7782    CLARK      10      ACCOUNTING     NEW YORK
7788    SCOTT      20      RESEARCH       DALLAS
7839    KING       10      ACCOUNTING     NEW YORK
7844    TURNER     30      SALES          CHICAGO
7876    ADAMS      20      RESEARCH       DALLAS
7900    JAMES      30      SALES          CHICAGO
7902    FORD       20      RESEARCH       DALLAS
7934    MILLER     10      ACCOUNTING     NEW YORK
7521    WARD       30      SALES          CHICAGO
14 rows selected.
```

Columns in the base `emp` table of `emp_dept_oj1` are modifiable through the view, because `emp` is a key-preserved table in the join.

The following view also contains an outer join:

```
CREATE VIEW emp_dept_oj2 AS
SELECT e.empno, e.ename, e.deptno, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno (+) = d.deptno;
```

The following statement:

```
SELECT * FROM emp_dept_oj2;
```

Results in:

```
EMPNO      ENAME      DEPTNO    DNAME          LOC
---------- ---------- --------- -------------- ----
```

```
7782       CLARK      10        ACCOUNTING     NEW YORK
7839       KING       10        ACCOUNTING     NEW YORK
7934       MILLER     10        ACCOUNTING     NEW YORK
7369       SMITH      20        RESEARCH       DALLAS
7876       ADAMS      20        RESEARCH       DALLAS
7902       FORD       20        RESEARCH       DALLAS
7788       SCOTT      20        RESEARCH       DALLAS
7566       JONES      20        RESEARCH       DALLAS
7499       ALLEN      30        SALES          CHICAGO
7698       BLAKE      30        SALES          CHICAGO
7654       MARTIN     30        SALES          CHICAGO
7900       JAMES      30        SALES          CHICAGO
7844       TURNER     30        SALES          CHICAGO
7521       WARD       30        SALES          CHICAGO
                                OPERATIONS     BOSTON
15 rows selected.
```

In this view, `emp` is no longer a key-preserved table, because the `empno` column in the result of the join can have nulls (the last row in the preceding `SELECT` statement). So, `UPDATE`, `DELETE`, and `INSERT` operations cannot be performed on this view.

In the case of views containing an outer join on other nested views, a table is key preserved if the view or views containing the table are merged into their outer views, all the way to the top. A view which is being outer-joined is currently merged only if it is "simple." For example:

```
SELECT col1, col2, ... FROM T;
```

The select list of the view has no expressions.

If you are in doubt whether a view is modifiable, then you can select from the `USER_UPDATABLE_COLUMNS` view to see if it is. For example:

```
SELECT owner, table_name, column_name, updatable FROM USER_UPDATABLE_COLUMNS
     WHERE TABLE_NAME = 'EMP_DEPT_VIEW';
```

This returns output similar to the following:

```
OWNER       TABLE_NAME      COLUMN_NAM      UPD
----------  ----------      ----------      ---
SCOTT       EMP_DEPT_V      EMPNO           NO
SCOTT       EMP_DEPT_V      ENAME           NO
SCOTT       EMP_DEPT_V      DEPTNO          NO
SCOTT       EMP_DEPT_V      DNAME           NO
SCOTT       EMP_DEPT_V      LOC             NO
5 rows selected.
```

23.1.5.5 Using the UPDATABLE_ COLUMNS Views

A set of views can assist you in identifying inherently updatable join views.

View	Description
`DBA_UPDATABLE_COLUMNS`	Shows all columns in all tables and views that are modifiable.
`ALL_UPDATABLE_COLUMNS`	Shows all columns in all tables and views accessible to the user that are modifiable.
`USER_UPDATABLE_COLUMNS`	Shows all columns in all tables and views in the user's schema that are modifiable.

The updatable columns in view `emp_dept` are shown below.

```
SELECT COLUMN_NAME, UPDATABLE
      FROM USER_UPDATABLE_COLUMNS
      WHERE TABLE_NAME = 'EMP_DEPT';

COLUMN_NAME                    UPD
------------------------------ ---
EMPNO                          YES
ENAME                          YES
DEPTNO                         YES
SAL                            YES
DNAME                          NO
LOC                            NO

6 rows selected.
```

See Also:

Oracle Database Reference for complete descriptions of the updatable column views

23.1.6 Altering Views

You use the `ALTER VIEW` statement only to explicitly recompile a view that is invalid.

To change the definition of a view, see "Replacing Views".

The `ALTER VIEW` statement lets you locate recompilation errors before run time. To ensure that the alteration does not affect the view or other objects that depend on it, you can explicitly recompile a view after altering one of its base tables.

To use the `ALTER VIEW` statement, the view must be in your schema, or you must have the `ALTER ANY TABLE` system privilege.

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `ALTER VIEW` statement

23.1.7 Dropping Views

You can drop a view with the `DROP VIEW` statement.

You can drop any view contained in your schema. To drop a view in another user's schema, you must have the `DROP ANY VIEW` system privilege. Drop a view using the `DROP VIEW` statement. For example, the following statement drops the `emp_dept` view:

```
DROP VIEW emp_dept;
```

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `DROP VIEW` statement

23.2 Managing Sequences

You can perform tasks such as creating sequences, altering sequences, using sequences, and dropping sequences.

- About Sequences
  Sequences are database objects from which multiple users can generate unique integers. The sequence generator generates sequential numbers, which can be used to generate unique primary keys automatically, and to coordinate keys across multiple rows or tables.
- Creating Sequences
  Create a sequence using the `CREATE SEQUENCE` statement.
- Altering Sequences
  Alter a sequence using the `ALTER SEQUENCE` statement.
- Using Sequences
  A sequence can be accessed and incremented by multiple users.
- Dropping Sequences
  If a sequence is no longer required, you can drop the sequence using the `DROP SEQUENCE` statement.

23.2.1 About Sequences

Sequences are database objects from which multiple users can generate unique integers. The sequence generator generates sequential numbers, which can be used to generate unique primary keys automatically, and to coordinate keys across multiple rows or tables.

Without sequences, sequential values can only be produced programmatically. A new primary key value can be obtained by selecting the most recently produced value and incrementing it. This method requires a lock during the transaction and causes multiple users to wait for the next value of the primary key; this waiting is known as **serialization**. If developers have such constructs in applications, then you should encourage the developers to replace them with access to sequences. Sequences eliminate serialization and improve the concurrency of an application.

See Also:

Oracle Database Concepts for an overview of sequences

23.2.2 Creating Sequences

Create a sequence using the `CREATE SEQUENCE` statement.

To create a sequence in your schema, you must have the `CREATE SEQUENCE` system privilege. To create a sequence in another user's schema, you must have the `CREATE ANY SEQUENCE` privilege.

For example, the following statement creates a sequence used to generate employee numbers for the `empno` column of the `emp` table:

```
CREATE SEQUENCE emp_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;
```

Notice that several parameters can be specified to control the function of sequences. You can use these parameters to indicate whether the sequence is ascending or descending, the starting point of the sequence, the minimum and maximum values, and the interval between sequence values. The `NOCYCLE` option indicates that the sequence cannot generate more values after reaching its maximum or minimum value.

The `CACHE` clause preallocates a set of sequence numbers and keeps them in memory so that sequence numbers can be accessed faster. When the last of the sequence numbers in the cache has been used, the database reads another set of numbers into the cache.

The database might skip sequence numbers if you choose to cache a set of sequence numbers. For example, when an instance abnormally shuts down (for example, when an instance failure occurs or a `SHUTDOWN ABORT` statement is issued), sequence numbers that have been cached but not used are lost. Also, sequence numbers that have been used but not saved are lost as well. The database might also skip cached sequence numbers after an export and import. See *Oracle Database Utilities* for details.

See Also:

- *Oracle Database SQL Language Reference* for the `CREATE SEQUENCE` statement syntax
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about using sequences in an Oracle Real Application Clusters environment

23.2.3 Altering Sequences

Alter a sequence using the `ALTER SEQUENCE` statement.

To alter a sequence, your schema must contain the sequence, you must have the `ALTER` object privilege on the sequence, or you must have the `ALTER ANY SEQUENCE` system privilege. You can alter a sequence to change any of the parameters that define how it generates sequence numbers. To change the starting point of a sequence, you can either drop the sequence and then re-create it, or use the `RESTART` clause to restart the sequence. For an ascending sequence, the `RESTART` clause resets `NEXTVAL` to `MINVALUE`. For a descending sequence, `NEXTVAL` is reset to `MAXVALUE`.

The following example alters the `emp_sequence` sequence:

```
ALTER SEQUENCE emp_sequence
    INCREMENT BY 10
    MAXVALUE 10000
    CYCLE
    CACHE 20;
```

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `ALTER SEQUENCE` statement

23.2.4 Using Sequences

A sequence can be accessed and incremented by multiple users.

To use a sequence, your schema must contain the sequence or you must have been granted the `SELECT` object privilege for another user's sequence. Once a sequence is defined, it can be accessed and incremented by multiple users (who have `SELECT` object privilege for the sequence containing the sequence) with no waiting. The database does not wait for a transaction that has incremented a sequence to complete before that sequence can be incremented again.

The examples outlined in the following sections show how sequences can be used in master/detail table relationships. Assume an order entry system is partially comprised of two tables, `orders_tab` (master table) and `line_items_tab` (detail table), that hold information about customer orders. A sequence named `order_seq` is defined by the following statement:

```
CREATE SEQUENCE Order_seq
    START WITH 1
    INCREMENT BY 1
    NOMAXVALUE
    NOCYCLE
    CACHE 20;
```

- Referencing a Sequence
  A sequence is referenced in SQL statements with the `NEXTVAL` and `CURRVAL` pseudocolumns; each new sequence number is generated by a reference to the sequence pseudocolumn `NEXTVAL`, while the current sequence number can be repeatedly referenced using the pseudo-column `CURRVAL`.
- Caching Sequence Numbers
  Caching sequence numbers can improve access time.
- Making a Sequence Scalable
  A sequence can be made *scalable* by specifying the `SCALE` clause in the `CREATE SEQUENCE` or `ALTER SEQUENCE` statement.

23.2.4.1 Referencing a Sequence

A sequence is referenced in SQL statements with the `NEXTVAL` and `CURRVAL` pseudocolumns; each new sequence number is generated by a reference to the sequence pseudocolumn `NEXTVAL`, while the current sequence number can be repeatedly referenced using the pseudo-column `CURRVAL`.

`NEXTVAL` and `CURRVAL` are not reserved words or keywords and can be used as pseudocolumn names in SQL statements such as `SELECT`, `INSERT`, or `UPDATE`.

- Generating Sequence Numbers with NEXTVAL
  To generate and use a sequence number, reference *seq_name*.`NEXTVAL` in a SQL statement.
- Using Sequence Numbers with CURRVAL
  To use or refer to the current sequence value of your session, reference *seq_name*.`CURRVAL` in a SQL statement.
- Uses and Restrictions of NEXTVAL and CURRVAL
  `CURRVAL` and `NEXTVAL` can be used in specific places, and restrictions apply to their use.

23.2.4.1.1 Generating Sequence Numbers with NEXTVAL

To generate and use a sequence number, reference *seq_name*.`NEXTVAL` in a SQL statement.

For example, assume a customer places an order. The sequence number can be referenced in a values list. For example:

```
INSERT INTO Orders_tab (Orderno, Custno)
    VALUES (Order_seq.NEXTVAL, 1032);
```

Or, the sequence number can be referenced in the `SET` clause of an `UPDATE` statement. For example:

```
UPDATE Orders_tab
    SET Orderno = Order_seq.NEXTVAL
    WHERE Orderno = 10112;
```

The sequence number can also be referenced outermost `SELECT` of a query or subquery. For example:

```
SELECT Order_seq.NEXTVAL FROM dual;
```

As defined, the first reference to `order_seq.NEXTVAL` returns the value 1. Each subsequent statement that references `order_seq.NEXTVAL` generates the next sequence number (2, 3, 4,. . .). The pseudo-column `NEXTVAL` can be used to generate as many new sequence numbers as necessary. However, only a single sequence number can be generated for each row. In other words, if `NEXTVAL` is referenced more than once in a single statement, then the first reference generates the next number, and all subsequent references in the statement return the same number.

Once a sequence number is generated, the sequence number is available only to the session that generated the number. Independent of transactions committing or rolling back, other users referencing `order_seq.NEXTVAL` obtain unique values. If two users are accessing the same sequence concurrently, then the sequence numbers each user receives might have gaps because sequence numbers are also being generated by the other user.

23.2.4.1.2 Using Sequence Numbers with CURRVAL

To use or refer to the current sequence value of your session, reference *seq_name*.`CURRVAL` in a SQL statement.

`CURRVAL` can only be used if *seq_name*.`NEXTVAL` has been referenced in the current user session (in the current or a previous transaction). `CURRVAL` can be referenced as many times as necessary, including multiple times within the same statement. The next sequence number is not generated until `NEXTVAL` is referenced. Continuing with the previous example, you would finish placing the customer's order by inserting the line items for the order:

```
INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 20321, 3);

INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 29374, 1);
```

Assuming the `INSERT` statement given in the previous section generated a new sequence number of 347, both rows inserted by the statements in this section insert rows with order numbers of 347.

23.2.4.1.3 Uses and Restrictions of NEXTVAL and CURRVAL

`CURRVAL` and `NEXTVAL` can be used in specific places, and restrictions apply to their use.

`CURRVAL` and `NEXTVAL` can be used in the following places:

- `VALUES` clause of `INSERT` statements
- The `SELECT` list of a `SELECT` statement
- A view query or materialized view query

  However, the use of `CURRVAL` and `NEXTVAL` in a materialized view query makes the materialized view complex. Therefore, it cannot be fast refreshed.
- The `SET` clause of an `UPDATE` statement

`CURRVAL` and `NEXTVAL` cannot be used in these places:

- A subquery
- A `SELECT` statement with the `DISTINCT` operator
- A `SELECT` statement with a `GROUP BY` or `ORDER BY` clause
- A `SELECT` statement that is combined with another `SELECT` statement with the `UNION`, `INTERSECT`, or `MINUS` set operator
- The `WHERE` clause of a `SELECT` statement
- The condition of a `CHECK` constraint

23.2.4.2 Caching Sequence Numbers

Caching sequence numbers can improve access time.

- About Caching Sequence Numbers
  Sequence numbers can be kept in the sequence cache in the System Global Area (SGA). Sequence numbers can be accessed more quickly in the sequence cache than they can be read from disk.

- About Automatic Sizing of the Sequence Cache
  Automatic resizing of the sequence cache improves performance significantly for fast insert workloads that use sequences.
- The Number of Entries in the Sequence Cache
  When an application accesses a sequence in the sequence cache, the sequence numbers are read quickly. However, if an application accesses a sequence that is not in the cache, then the sequence must be read from disk to the cache before the sequence numbers are used.
- The Number of Values in Each Sequence Cache Entry
  When a sequence is read into the sequence cache, sequence values are generated and stored in a cache entry. These values can then be accessed quickly.

23.2.4.2.1 About Caching Sequence Numbers

Sequence numbers can be kept in the sequence cache in the System Global Area (SGA). Sequence numbers can be accessed more quickly in the sequence cache than they can be read from disk.

The sequence cache consists of entries. Each entry can hold many sequence numbers for a single sequence.

Follow these guidelines for fast access to all sequence numbers:

- Be sure the sequence cache can hold all the sequences used concurrently by your applications.
- Increase the number of values for each sequence held in the sequence cache.

23.2.4.2.2 About Automatic Sizing of the Sequence Cache

Automatic resizing of the sequence cache improves performance significantly for fast insert workloads that use sequences.

The automatic sequence cache size on each instance is dynamically computed based on the rate of usage of sequence numbers. Each instance caches the maximum of the manually configured sequence cache size and the projected cache size requirement for the next 10 seconds. Based on the sequence usage, the sequence cache size can shrink or grow. The minimum size to which the cache can shrink is the manually configured cache size. To prevent the sequence cache size from growing indefinitely, the cache size and each increment in the cache size is capped.

For cycle sequences, the upper bound for the automatic sequence cache size is the size of one cycle.

23.2.4.2.3 The Number of Entries in the Sequence Cache

When an application accesses a sequence in the sequence cache, the sequence numbers are read quickly. However, if an application accesses a sequence that is not in the cache, then the sequence must be read from disk to the cache before the sequence numbers are used.

If your applications use many sequences concurrently, then your sequence cache might not be large enough to hold all the sequences. In this case, access to sequence numbers might often require disk reads. For fast access to all sequences, be sure your cache has enough entries to hold all the sequences used concurrently by your applications.

23.2.4.2.4 The Number of Values in Each Sequence Cache Entry

When a sequence is read into the sequence cache, sequence values are generated and stored in a cache entry. These values can then be accessed quickly.

The number of sequence values stored in the cache is determined by the `CACHE` parameter in the `CREATE SEQUENCE` statement. The default value for this parameter is 20.

This `CREATE SEQUENCE` statement creates the `seq2` sequence so that 50 values of the sequence are stored in the `SEQUENCE` cache:

```
CREATE SEQUENCE seq2
    CACHE 50;
```

The first 50 values of `seq2` can then be read from the cache. When the 51st value is accessed, the next 50 values will be read from disk.

Choosing a high value for `CACHE` lets you access more successive sequence numbers with fewer reads from disk to the sequence cache. However, if there is an instance failure, then all sequence values in the cache are lost. Cached sequence numbers also could be skipped after an export and import if transactions continue to access the sequence numbers while the export is running.

If you use the `NOCACHE` option in the `CREATE SEQUENCE` statement, then the values of the sequence are not stored in the sequence cache. In this case, every access to the sequence requires a disk read. Such disk reads slow access to the sequence. This `CREATE SEQUENCE` statement creates the `SEQ3` sequence so that its values are never stored in the cache:

```
CREATE SEQUENCE seq3
    NOCACHE;
```

23.2.4.3 Making a Sequence Scalable

A sequence can be made *scalable* by specifying the `SCALE` clause in the `CREATE SEQUENCE` or `ALTER SEQUENCE` statement.

A scalable sequence is particularly efficient when used to generate unordered primary or unique keys for data ingestion workloads having high level of concurrency. Single Oracle database instances as well as Oracle RAC databases benefit from this feature. Scalable sequences significantly reduce the sequence and index block contention and provide better data load scalability compared to the solution of configuring a very large sequence cache using the `CACHE` clause of `CREATE SEQUENCE` or `ALTER SEQUENCE` statement.

Note:

In addition to using a scalable sequence, you can also partition the data to increase the performance of a data load operation.

The following is the syntax for defining a scalable sequence:

```
CREATE | ALTER SEQUENCE sequence_name
   ...
   SCALE [EXTEND | NOEXTEND] | NOSCALE
   ...
```

When the `SCALE` clause is specified, a 6 digit numeric *scalable sequence offset number* is prefixed to the digits of the sequence:

```
   scalable sequence number = 6 digit scalable sequence offset number ||
normal sequence number
```

where,

- `||` is the concatenation operator.
- 6 digit scalable sequence offset number = 3 digit *instance* offset number `||` 3 digit *session* offset number.

  The 3 digit *instance* offset number is generated as [`(instance id % 100) + 100`]. The 3 digit *session* offset number is generated as [`session id % 1000`].

Additionally, you can also specify `EXTEND` or `NOEXTEND` option for the `SCALE` clause:

- `EXTEND` option

  When the `EXTEND` option is specified for the `SCALE` clause, the scalable sequence values are of the length [`X` digits + `Y` digits], where `X` is the number of digits in the scalable sequence offset number (default is 6 digits), and `Y` is the number of digits specified in the `MAXVALUE` clause.

  For example, for an ascending scalable sequence with `MINVALUE` of 1, `MAXVALUE` of 100 (3 digits), and `EXTEND` option specified, the scalable sequence values will be of 9 digits (6 digit scalable sequence offset number + 3 digit `MAXVALUE`) and will be of the form:

  ```
  6 digit scalable sequence offset number || 001
  6 digit scalable sequence offset number || 002
  6 digit scalable sequence offset number || 003
  ...
  6 digit scalable sequence offset number || 100
  ```

- `NOEXTEND` option

  When the `NOEXTEND` option is specified for the `SCALE` clause, which is the default option, the number of scalable sequence digits cannot exceed the number of digits specified in the `MAXVALUE` clause.

  For example, for an ascending scalable sequence with `MINVALUE` of 1, `MAXVALUE` of 1000000 (7 digits), and `NOEXTEND` option specified, the scalable sequence values will be of 7 digits, because `MAXVALUE` of 1000000 contains 7 digits, and will be of the form:

  ```
  6 digit scalable sequence offset number || 1
  6 digit scalable sequence offset number || 2
  6 digit scalable sequence offset number || 3
  ...
  6 digit scalable sequence offset number || 9
  ```

  Note that the `NEXTVAL` operation on this scalable sequence after the sequence value of `[6 digit scalable sequence offset number || 9]` will report the following error message, because the next scalable sequence value is `[6 digit scalable sequence offset` `number || 10]`, which contains 8 digits and is greater than `MAXVALUE` of 1000000 that contains 7 digits:

```
ORA-64603: NEXTVAL cannot be instantiated for SQ. Widen the sequence by 1
digits or alter sequence with SCALE EXTEND.
```

> **Note:**
>
> The `NOEXTEND` option is useful for integration with the existing applications where sequences are used to populate fixed width columns.

To convert an existing scalable sequence to a non-scalable sequence, use the `NOSCALE` clause in the `ALTER SEQUENCE` statement.

> **Note:**
>
> Oracle recommends that you should not specify ordering for a scalable sequence, because scalable sequence numbers are globally unordered.

To know whether a sequence is scalable or whether a scalable sequence is extendable, check the values of the following columns of the `DBA_SEQUENCES`, `USER_SEQUENCES`, and `ALL_SEQUENCES` views.

Table 23-1 Columns Related to Scalable Sequences in the DBA_SEQUENCES, USER_SEQUENCES, and ALL_SEQUENCES Views

Column Name	Description
`SCALE_FLAG`	Indicates whether the sequence is a scalable sequence: • `Y` • `N`
`EXTEND_FLAG`	Indicates whether the scalable sequence is *extendable*, that is, whether the `EXTEND` option is applied for the scalable sequence, so that sequence values can extend beyond the value specified for `MAXVALUE`: • `Y` • `N`

23.2.5 Dropping Sequences

If a sequence is no longer required, you can drop the sequence using the `DROP SEQUENCE` statement.

You can drop any sequence in your schema. To drop a sequence in another schema, you must have the `DROP ANY SEQUENCE` system privilege. For example, the following statement drops the `order_seq` sequence:

```
DROP SEQUENCE order_seq;
```

When a sequence is dropped, its definition is removed from the data dictionary. Any synonyms for the sequence remain, but return an error when referenced.

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `DROP SEQUENCE` statement

23.3 Managing Synonyms

You can perform tasks such as creating synonyms, using synonyms, and dropping synonyms.

- About Synonyms
  A synonym is an alias for a schema object.
- Creating Synonyms
  Create a synonym using the `CREATE SYNONYM` statement.
- Using Synonyms in DML Statements
  A synonym can be referenced in a DML statement the same way that the underlying object of the synonym can be referenced.
- Dropping Synonyms
  Drop a synonym that is no longer required using `DROP SYNONYM` statement. To drop a private synonym, omit the `PUBLIC` keyword. To drop a public synonym, include the `PUBLIC` keyword.

23.3.1 About Synonyms

A synonym is an alias for a schema object.

Synonyms can provide a level of security by masking the name and owner of an object and by providing location transparency for remote objects of a distributed database. Also, they are convenient to use and reduce the complexity of SQL statements for database users.

Synonyms allow underlying objects to be renamed or moved, where only the synonym must be redefined and applications based on the synonym continue to function without modification.

You can create both public and private synonyms. A **public** synonym is owned by the special user group named `PUBLIC` and is accessible to every user in a database. A **private** synonym is contained in the schema of a specific user and available only to the user and to grantees for the underlying object.

Synonyms themselves are not securable. When you grant object privileges on a synonym, you are really granting privileges on the underlying object, and the synonym is acting only as an alias for the object in the `GRANT` statement.

See Also:

Oracle Database Concepts for a more complete description of synonyms

23.3.2 Creating Synonyms

Create a synonym using the `CREATE SYNONYM` statement.

To create a private synonym in your own schema, you must have the `CREATE SYNONYM` privilege. To create a private synonym in another user's schema, you must have the `CREATE ANY SYNONYM` privilege. To create a public synonym, you must have the `CREATE PUBLIC SYNONYM` system privilege.

When you create a synonym, the underlying schema object need not exist, nor do you need privileges to access the object for the `CREATE SYNONYM` statement to succeed. The following statement creates a public synonym named `public_emp` on the `emp` table contained in the schema of `jward`:

```
CREATE PUBLIC SYNONYM public_emp FOR jward.emp
```

When you create a synonym for a remote procedure or function, you must qualify the remote object with its schema name. Alternatively, you can create a local public synonym on the database where the remote object resides, in which case the database link must be included in all subsequent calls to the procedure or function.

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the `CREATE SYNONYM` statement

23.3.3 Using Synonyms in DML Statements

A synonym can be referenced in a DML statement the same way that the underlying object of the synonym can be referenced.

You can successfully use any private synonym contained in your schema or any public synonym, assuming that you have the necessary privileges to access the underlying object, either explicitly, from an enabled role, or from `PUBLIC`. You can also reference any private synonym contained in another schema if you have been granted the necessary object privileges for the underlying object.

You can reference another user's synonym using only the object privileges that you have been granted. For example, if you have only the `SELECT` privilege on the `jward.emp` table, and the synonym `jward.employee` is created for `jward.emp`, you can query the `jward.employee` synonym, but you cannot insert rows using the `jward.employee` synonym.

For example, if a synonym named `emp`loyee refers to a table or view, then the following statement is valid:

```
INSERT INTO employee (empno, ename, job)
    VALUES (emp_sequence.NEXTVAL, 'SMITH', 'CLERK');
```

If the synonym named `fire_emp` refers to a standalone procedure or package procedure, then you could execute it with the command

```
EXECUTE Fire_emp(7344);
```

23.3.4 Dropping Synonyms

Drop a synonym that is no longer required using DROP SYNONYM statement. To drop a private synonym, omit the PUBLIC keyword. To drop a public synonym, include the PUBLIC keyword.

You can drop any private synonym in your own schema. To drop a private synonym in another user's schema, you must have the DROP ANY SYNONYM system privilege. To drop a public synonym, you must have the DROP PUBLIC SYNONYM system privilege.

For example, the following statement drops the private synonym named emp:

```
DROP SYNONYM emp;
```

The following statement drops the public synonym named public_emp:

```
DROP PUBLIC SYNONYM public_emp;
```

When you drop a synonym, its definition is removed from the data dictionary. All objects that reference a dropped synonym remain. However, they become invalid (not usable). For more information about how dropping synonyms can affect other schema objects, see "Managing Object Dependencies".

See Also:

Oracle Database SQL Language Reference for syntax and additional information about the DROP SYNONYM statement

23.4 Views, Synonyms, and Sequences Data Dictionary Views

You can query data dictionary views for information about views, synonyms, and sequences.

The following views display information about views, synonyms, and sequences:

View	Description
DBA_VIEWS ALL_VIEWS USER_VIEWS	DBA view describes all views in the database. ALL view is restricted to views accessible to the current user. USER view is restricted to views owned by the current user.
DBA_SYNONYMS ALL_SYNONYMS USER_SYNONYMS	These views describe synonyms.
DBA_SEQUENCES ALL_SEQUENCES USER_SEQUENCES	These views describe sequences.
DBA_UPDATABLE_COLUMNS ALL_UPDATABLE_COLUMNS USER_UPDATABLE_COLUMNS	These views describe all columns in join views that are updatable.

24
Repairing Corrupted Data

You can detect and correct data block corruption.

Note:

If you are not familiar with the `DBMS_REPAIR` package, then it is recommended that you work with an Oracle Support Services analyst when performing any of the repair procedures included in this package.

- Options for Repairing Data Block Corruption
  Oracle Database provides different methods for detecting and correcting data block corruption.
- About the DBMS_REPAIR Package
  The `DBMS_REPAIR` package contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes.
- Using the DBMS_REPAIR Package
  You can use the `DBMS_REPAIR` package to address data block corruption:
- DBMS_REPAIR Examples
  Examples illustrate how to use the `DBMS_REPAIR` package.

24.1 Options for Repairing Data Block Corruption

Oracle Database provides different methods for detecting and correcting data block corruption.

One method of correction is to drop and re-create an object after the corruption is detected. However, this is not always possible or desirable. If data block corruption is limited to a subset of rows, then another option is to rebuild the table by selecting all data except for the corrupt rows.

Another way to manage data block corruption is to use the `DBMS_REPAIR` package. You can use `DBMS_REPAIR` to detect and repair corrupt blocks in tables and indexes. You can continue to use objects while you attempt to rebuild or repair them.

You can also use the Recovery Manager (RMAN) command `RECOVER BLOCK` to recover a corrupt data block or set of data blocks.

Note:

Any corruption that involves the loss of data requires analysis and understanding of how that data fits into the overall database system. Depending on the nature of the repair, you might lose data, and logical inconsistencies can be introduced. You must determine whether the repair approach provided by this package is the appropriate tool for each specific corruption problem.

See Also:

Oracle Database Backup and Recovery Reference for more information about the `RECOVER BLOCK` RMAN command

24.2 About the DBMS_REPAIR Package

The `DBMS_REPAIR` package contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes.

- DBMS_REPAIR Procedures
  Procedures in the `DBMS_REPAIR` package enable you to detect and repair corrupt blocks.
- Limitations and Restrictions for DBMS_REPAIR Procedures
  Some limitations and restrictions apply to `DBMS_REPAIR` procedures.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information on the syntax, restrictions, and exceptions for the `DBMS_REPAIR` procedures

24.2.1 DBMS_REPAIR Procedures

Procedures in the `DBMS_REPAIR` package enable you to detect and repair corrupt blocks.

The following table lists the procedures included in the `DBMS_REPAIR` package.

Procedure Name	Description
`ADMIN_TABLES`	Provides administrative functions (create, drop, purge) for repair or orphan key tables. **Note:** These tables are always created in the `SYS` schema.
`CHECK_OBJECT`	Detects and reports corruptions in a table or index
`DUMP_ORPHAN_KEYS`	Reports on index entries that point to rows in corrupt data blocks
`FIX_CORRUPT_BLOCKS`	Marks blocks as software corrupt that have been previously identified as corrupt by the `CHECK_OBJECT` procedure
`REBUILD_FREELISTS`	Rebuilds the free lists of the object
`SEGMENT_FIX_STATUS`	Provides the capability to fix the corrupted state of a bitmap entry when segment space management is `AUTO`
`SKIP_CORRUPT_BLOCKS`	When used, ignores blocks marked corrupt during table and index scans. If not used, you get error `ORA-01578` when encountering blocks marked corrupt.

These procedures are further described, with examples of their use, in "DBMS_REPAIR Examples".

24.2.2 Limitations and Restrictions for DBMS_REPAIR Procedures

Some limitations and restrictions apply to `DBMS_REPAIR` procedures.

`DBMS_REPAIR` procedures have the following limitations:

- Tables with LOB data types, nested tables, and varrays are supported, but the out-of-line columns are ignored.
- Clusters are supported in the `SKIP_CORRUPT_BLOCKS` and `REBUILD_FREELISTS` procedures, but not in the `CHECK_OBJECT` procedure.
- Index-organized tables and LOB indexes are not supported.
- Global temporary tables are not supported.
- The `DUMP_ORPHAN_KEYS` procedure does not operate on bitmap indexes or function-based indexes.
- The `DUMP_ORPHAN_KEYS` procedure processes keys that are no more than 3,950 bytes long.

24.3 Using the DBMS_REPAIR Package

You can use the `DBMS_REPAIR` package to address data block corruption:

- Task 1: Detect and Report Corruptions
  The first task is the detection and reporting of corruptions. Reporting not only indicates what is wrong with a block, but also identifies the associated repair directive.
- Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR
  Before using `DBMS_REPAIR` you must weigh the benefits of its use in relation to the liabilities. You should also examine other options available for addressing corrupt objects.
- Task 3: Make Objects Usable
  `DBMS_REPAIR` makes the object usable by ignoring corruptions during table and index scans.
- Task 4: Repair Corruptions and Rebuild Lost Data
  After making an object usable, perform the following repair activities.

24.3.1 Task 1: Detect and Report Corruptions

The first task is the detection and reporting of corruptions. Reporting not only indicates what is wrong with a block, but also identifies the associated repair directive.

- About Detecting and Reporting Corruptions
  There are several ways to detect corruptions.
- DBMS_REPAIR: Using the CHECK_OBJECT and ADMIN_TABLES Procedures
  The `CHECK_OBJECT` procedure checks and reports block corruptions for a specified object. The `ADMIN_TABLES` procedure creates a repair table that facilitates correcting corruptions.
- DB_VERIFY: Performing an Offline Database Check
  Use `DB_VERIFY` as an offline diagnostic utility when you encounter data corruption.
- ANALYZE: Reporting Corruption
  The `ANALYZE TABLE...VALIDATE STRUCTURE` statement validates the structure of the analyzed object. If the database encounters corruption in the structure of the object, then an error message is returned. In this case, drop and re-create the object.

- DB_BLOCK_CHECKING Initialization Parameter
  You can enable database block checking by setting the `DB_BLOCK_CHECKING` initialization parameter to `TRUE`.

24.3.1.1 About Detecting and Reporting Corruptions

There are several ways to detect corruptions.

Table 24-1 describes the different detection methodologies.

Table 24-1 Comparison of Corruption Detection Methods

Detection Method	Description
`DBMS_REPAIR` PL/SQL package	Performs block checking for a specified table, partition, or index. It populates a repair table with results.
`DB_VERIFY` utility	Performs block checking on an offline database
`ANALYZE TABLE` SQL statement	Used with the `VALIDATE STRUCTURE` option, the `ANALYZE TABLE` statement verifies the integrity of the structure of an index, table, or cluster; checks or verifies that tables and indexes are synchronized.
`DB_BLOCK_CHECKING` initialization parameter	When `DB_BLOCK_CHECKING=TRUE`, corrupt blocks are identified before they are marked corrupt. Checks are performed when changes are made to a block.

24.3.1.2 DBMS_REPAIR: Using the CHECK_OBJECT and ADMIN_TABLES Procedures

The `CHECK_OBJECT` procedure checks and reports block corruptions for a specified object. The `ADMIN_TABLES` procedure creates a repair table that facilitates correcting corruptions.

The `CHECK_OBJECT` procedure is similar to the `ANALYZE...VALIDATE STRUCTURE` statement for indexes and tables, block checking is performed for index and data blocks.

Not only does `CHECK_OBJECT` report corruptions, but it also identifies any fixes that would occur if `FIX_CORRUPT_BLOCKS` is subsequently run on the object. This information is made available by populating a repair table, which must first be created by the `ADMIN_TABLES` procedure.

After you run the `CHECK_OBJECT` procedure, a simple query on the repair table shows the corruptions and repair directives for the object. With this information, you can assess how best to address the reported problems.

24.3.1.3 DB_VERIFY: Performing an Offline Database Check

Use `DB_VERIFY` as an offline diagnostic utility when you encounter data corruption.

See Also:

Oracle Database Utilities for more information about `DB_VERIFY`

24.3.1.4 ANALYZE: Reporting Corruption

The `ANALYZE TABLE...VALIDATE STRUCTURE` statement validates the structure of the analyzed object. If the database encounters corruption in the structure of the object, then an error message is returned. In this case, drop and re-create the object.

You can use the `CASCADE` clause of the `ANALYZE TABLE` statement to check the structure of the table and all of its indexes in one operation. Because this operation can consume significant resources, there is a FAST option that performs a lightweight check. See "Validating Tables, Indexes, Clusters, and Materialized Views" for details.

See Also:

- *Oracle Database SQL Language Reference* for more information about the `ANALYZE` statement

24.3.1.5 DB_BLOCK_CHECKING Initialization Parameter

You can enable database block checking by setting the `DB_BLOCK_CHECKING` initialization parameter to `TRUE`.

This checks data and index blocks for internal consistency whenever they are modified. `DB_BLOCK_CHECKING` is a dynamic parameter, modifiable by the `ALTER SYSTEM SET` statement. Block checking is always enabled for the system tablespace.

Caution:

Before enabling block checking with this parameter, Oracle recommends that you detect and repair any logical corruptions in the database. Otherwise, a block that contain logical corruption will be marked as "soft corrupt" after block checking is enabled and the block is modified by a DML statement. This will result in `ORA-1578` errors and the block will be unreadable.

See Also:

Oracle Database Reference for more information about the `DB_BLOCK_CHECKING` initialization parameter.

Oracle Database Backup and Recovery User’s Guide for more information about detecting and repairing logical corruptions.

24.3.2 Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR

Before using `DBMS_REPAIR` you must weigh the benefits of its use in relation to the liabilities. You should also examine other options available for addressing corrupt objects.

Begin by answering the following questions:

- What is the extent of the corruption?

  To determine if there are corruptions and repair actions, execute the `CHECK_OBJECT` procedure and query the repair table.
- What other options are available for addressing block corruptions? Consider the following:
  - If the data is available from another source, then drop, re-create, and repopulate the object.
  - Issue the `CREATE TABLE...AS SELECT` statement from the corrupt table to create a new one.
  - Ignore the corruption by excluding corrupt rows from `SELECT` statements.
  - Perform media recovery.
- What logical corruptions or side effects are introduced when you use `DBMS_REPAIR` to make an object usable? Can these be addressed? What is the effort required to do so?

  You might not have access to rows in blocks marked corrupt. However, a block can be marked corrupt even if there are rows that you can validly access.

  It is also possible that referential integrity constraints are broken when blocks are marked corrupt. If this occurs, then disable and reenable the constraint; any inconsistencies are reported. After fixing all problems, you should be able to reenable the constraint.

  Logical corruption can occur when there are triggers defined on the table. For example, if rows are reinserted, should insert triggers be fired or not? You can address these issues only if you understand triggers and their use in your installation.

  If indexes and tables are not synchronized, then execute the `DUMP_ORPHAN_KEYS` procedure to obtain information from the keys that might be useful in rebuilding corrupted data. Then issue the `ALTER INDEX...REBUILD ONLINE` statement to synchronize the table with its indexes.
- If repair involves loss of data, can this data be retrieved?

  You can retrieve data from the index when a data block is marked corrupt. The `DUMP_ORPHAN_KEYS` procedure can help you retrieve this information.

24.3.3 Task 3: Make Objects Usable

`DBMS_REPAIR` makes the object usable by ignoring corruptions during table and index scans.

- Corruption Repair: Using the FIX_CORRUPT_BLOCKS and SKIP_CORRUPT_BLOCKS Procedures
  You can make a corrupt object usable by establishing an environment that skips corruptions that remain outside the scope of `DBMS_REPAIR` capabilities.
- Implications When Skipping Corrupt Blocks
  When skipping corrupt blocks, a query can return different results in some situations.

24.3.3.1 Corruption Repair: Using the FIX_CORRUPT_BLOCKS and SKIP_CORRUPT_BLOCKS Procedures

You can make a corrupt object usable by establishing an environment that skips corruptions that remain outside the scope of `DBMS_REPAIR` capabilities.

If corruptions involve a loss of data, such as a bad row in a data block, then all such blocks are marked corrupt by the `FIX_CORRUPT_BLOCKS` procedure. Then you can run the `SKIP_CORRUPT_BLOCKS` procedure, which skips blocks that are marked as corrupt. When the `SKIP_FLAG` parameter in the procedure is set, table and index scans skip all blocks marked corrupt. This applies to both media and software corrupt blocks.

24.3.3.2 Implications When Skipping Corrupt Blocks

When skipping corrupt blocks, a query can return different results in some situations.

If an index and table are not synchronized, then a `SET TRANSACTION READ ONLY` transaction can be inconsistent in situations where one query probes only the index, and a subsequent query probes both the index and the table. If the table block is marked corrupt, then the two queries return different results, thereby breaking the rules of a read-only transaction. One way to approach this is not to skip corruptions in a `SET TRANSACTION READ ONLY` transaction.

A similar issue occurs when selecting rows that are chained. A query of the same row may or may not access the corruption, producing different results.

24.3.4 Task 4: Repair Corruptions and Rebuild Lost Data

After making an object usable, perform the following repair activities.

- Recover Data Using the DUMP_ORPHAN_KEYS Procedures
  The `DUMP_ORPHAN_KEYS` procedure reports on index entries that point to rows in corrupt data blocks. All such index entries are inserted into an orphan key table that stores the key and rowid of the corruption.
- Fix Segment Bitmaps Using the SEGMENT_FIX_STATUS Procedure
  Use the `SEGMENT_FIX_STATUS` procedure if free space in segments is being managed by using bitmaps (`SEGMENT SPACE MANAGEMENT AUTO`).

24.3.4.1 Recover Data Using the DUMP_ORPHAN_KEYS Procedures

The `DUMP_ORPHAN_KEYS` procedure reports on index entries that point to rows in corrupt data blocks. All such index entries are inserted into an orphan key table that stores the key and rowid of the corruption.

After the index entry information has been retrieved, you can rebuild the index using the `ALTER INDEX...REBUILD ONLINE` statement.

24.3.4.2 Fix Segment Bitmaps Using the SEGMENT_FIX_STATUS Procedure

Use the `SEGMENT_FIX_STATUS` procedure if free space in segments is being managed by using bitmaps (`SEGMENT SPACE MANAGEMENT AUTO`).

This procedure recalculates the state of a bitmap entry based on the current contents of the corresponding block. Alternatively, you can specify that a bitmap entry be set to a specific value. Usually the state is recalculated correctly and there is no need to force a setting.

24.4 DBMS_REPAIR Examples

Examples illustrate how to use the `DBMS_REPAIR` package.

- Examples: Building a Repair Table or Orphan Key Table
  A repair table provides information about the corruptions. An orphan key table provides information about index entries that point to corrupt rows.
- Example: Detecting Corruption
  An example illustrates detecting corruption with the `CHECK_OBJECT` procedure.
- Example: Fixing Corrupt Blocks
  An example illustrates fixing corrupt blocks with the `FIX_CORRUPT_BLOCKS` procedure.
- Example: Finding Index Entries Pointing to Corrupt Data Blocks
  An example illustrates finding index entries pointing to corrupt data blocks using the `DUMP_ORPHAN_KEYS` procedure.
- Example: Skipping Corrupt Blocks
  An example illustrates skipping corrupt blocks using the `SKIP_CORRUPT_BLOCKS` procedure.

24.4.1 Examples: Building a Repair Table or Orphan Key Table

A repair table provides information about the corruptions. An orphan key table provides information about index entries that point to corrupt rows.

- About Repair Tables or Orphan Key Tables
  The `ADMIN_TABLES` procedure is used to create, purge, or drop a repair table or an orphan key table.
- Example: Creating a Repair Table
  An example illustrates creating a repair table using the `ADMIN_TABLES` procedure.
- Example: Creating an Orphan Key Table
  An example illustrates creating an orphan key table using the `ADMIN_TABLES` procedure.

24.4.1.1 About Repair Tables or Orphan Key Tables

The `ADMIN_TABLES` procedure is used to create, purge, or drop a repair table or an orphan key table.

A repair table provides information about the corruptions that were found by the `CHECK_OBJECT` procedure and how these will be addressed if the `FIX_CORRUPT_BLOCKS` procedure is run. Further, it is used to drive the execution of the `FIX_CORRUPT_BLOCKS` procedure.

An orphan key table is used when the `DUMP_ORPHAN_KEYS` procedure is executed and it discovers index entries that point to corrupt rows. The `DUMP_ORPHAN_KEYS` procedure populates the orphan key table by logging its activity and providing the index information in a usable manner.

24.4.1.2 Example: Creating a Repair Table

An example illustrates creating a repair table using the ADMIN_TABLES procedure.

The following example creates a repair table for the users tablespace.

```
BEGIN
  DBMS_REPAIR.ADMIN_TABLES (
     TABLE_NAME => 'REPAIR_TABLE',
     TABLE_TYPE => dbms_repair.repair_table,
     ACTION     => dbms_repair.create_action,
     TABLESPACE => 'USERS');
END;
/
```

For each repair or orphan key table, a view is also created that eliminates any rows that pertain to objects that no longer exist. The name of the view corresponds to the name of the repair or orphan key table and is prefixed by DBA_ (for example, DBA_REPAIR_TABLE or DBA_ORPHAN_KEY_TABLE).

The following query describes the repair table that was created for the users tablespace.

```
DESC REPAIR_TABLE

 Name                         Null?    Type
 ---------------------------- -------- --------------
 OBJECT_ID                    NOT NULL NUMBER
 TABLESPACE_ID                NOT NULL NUMBER
 RELATIVE_FILE_ID             NOT NULL NUMBER
 BLOCK_ID                     NOT NULL NUMBER
 CORRUPT_TYPE                 NOT NULL NUMBER
 SCHEMA_NAME                  NOT NULL VARCHAR2(128)
 OBJECT_NAME                  NOT NULL VARCHAR2(128)
 BASEOBJECT_NAME                       VARCHAR2(128)
 PARTITION_NAME                        VARCHAR2(128)
 CORRUPT_DESCRIPTION                   VARCHAR2(2000)
 REPAIR_DESCRIPTION                    VARCHAR2(200)
 MARKED_CORRUPT               NOT NULL VARCHAR2(10)
 CHECK_TIMESTAMP              NOT NULL DATE
 FIX_TIMESTAMP                         DATE
 REFORMAT_TIMESTAMP                    DATE
```

24.4.1.3 Example: Creating an Orphan Key Table

An example illustrates creating an orphan key table using the ADMIN_TABLES procedure.

This example illustrates the creation of an orphan key table for the users tablespace.

```
BEGIN
  DBMS_REPAIR.ADMIN_TABLES (
     TABLE_NAME => 'ORPHAN_KEY_TABLE',
     TABLE_TYPE => dbms_repair.orphan_table,
     ACTION     => dbms_repair.create_action,
     TABLESPACE => 'USERS');
END;
/
```

The orphan key table is described in the following query:

```
DESC ORPHAN_KEY_TABLE

 Name                            Null?    Type
 ---------------------------- -------- -----------------
 SCHEMA_NAME                     NOT NULL VARCHAR2(128)
 INDEX_NAME                      NOT NULL VARCHAR2(128)
 IPART_NAME                               VARCHAR2(128)
 INDEX_ID                        NOT NULL NUMBER
 TABLE_NAME                      NOT NULL VARCHAR2(128)
 PART_NAME                                VARCHAR2(128)
 TABLE_ID                        NOT NULL NUMBER
 KEYROWID                        NOT NULL ROWID
 KEY                             NOT NULL ROWID
 DUMP_TIMESTAMP                  NOT NULL DATE
```

24.4.2 Example: Detecting Corruption

An example illustrates detecting corruption with the `CHECK_OBJECT` procedure.

The `CHECK_OBJECT` procedure checks the specified object, and populates the repair table with information about corruptions and repair directives. You can optionally specify a range, partition name, or subpartition name when you want to check a portion of an object.

Validation consists of checking all blocks in the object that have not previously been marked corrupt. For each block, the transaction and data layer portions are checked for self consistency. During `CHECK_OBJECT`, if a block is encountered that has a corrupt buffer cache header, then that block is skipped.

The following is an example of executing the `CHECK_OBJECT` procedure for the `scott.dept` table.

```
SET SERVEROUTPUT ON
DECLARE num_corrupt INT;
BEGIN
 num_corrupt := 0;
 DBMS_REPAIR.CHECK_OBJECT (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     CORRUPT_COUNT =>  num_corrupt);
 DBMS_OUTPUT.PUT_LINE('number corrupt: ' || TO_CHAR (num_corrupt));
END;
/
```

SQL*Plus outputs the following line, indicating one corruption:

```
number corrupt: 1
```

Querying the repair table produces information describing the corruption and suggesting a repair action.

```
SELECT OBJECT_NAME, BLOCK_ID, CORRUPT_TYPE, MARKED_CORRUPT,
       CORRUPT_DESCRIPTION, REPAIR_DESCRIPTION
     FROM REPAIR_TABLE;

OBJECT_NAME                       BLOCK_ID CORRUPT_TYPE MARKED_COR
------------------------------ ---------- ------------ ----------
CORRUPT_DESCRIPTION
------------------------------------------------------------------------------
REPAIR_DESCRIPTION
------------------------------------------------------------------------------
```

```
DEPT                                    3            1 FALSE
kdbchk: row locked by non-existent transaction
        table=0   slot=0
        lockid=32   ktbbhitc=1
mark block software corrupt
```

The corrupted block has not yet been marked corrupt, so this is the time to extract any meaningful data. After the block is marked corrupt, the entire block must be skipped.

24.4.3 Example: Fixing Corrupt Blocks

An example illustrates fixing corrupt blocks with the `FIX_CORRUPT_BLOCKS` procedure.

Use the `FIX_CORRUPT_BLOCKS` procedure to fix the corrupt blocks in specified objects based on information in the repair table that was generated by the `CHECK_OBJECT` procedure. Before changing a block, the block is checked to ensure that the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is performed, the associated row in the repair table is updated with a timestamp.

This example fixes the corrupt block in table `scott.dept` that was reported by the `CHECK_OBJECT` procedure.

```
SET SERVEROUTPUT ON
DECLARE num_fix INT;
BEGIN
 num_fix := 0;
 DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME=> 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object,
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     FIX_COUNT=> num_fix);
 DBMS_OUTPUT.PUT_LINE('num fix: ' || TO_CHAR(num_fix));
END;
/
```

SQL*Plus outputs the following line:

```
num fix: 1
```

The following query confirms that the repair was done.

```
SELECT OBJECT_NAME, BLOCK_ID, MARKED_CORRUPT
     FROM REPAIR_TABLE;

OBJECT_NAME                       BLOCK_ID MARKED_COR
------------------------------ ---------- ----------
DEPT                                    3 TRUE
```

24.4.4 Example: Finding Index Entries Pointing to Corrupt Data Blocks

An example illustrates finding index entries pointing to corrupt data blocks using the `DUMP_ORPHAN_KEYS` procedure.

The `DUMP_ORPHAN_KEYS` procedure reports on index entries that point to rows in corrupt data blocks. For each index entry, a row is inserted into the specified orphan key table. The orphan key table must have been previously created.

This information can be useful for rebuilding lost rows in the table and for diagnostic purposes.

Note:

This should be run for every index associated with a table identified in the repair table.

In this example, `pk_dept` is an index on the `scott.dept` table. It is scanned to determine if there are any index entries pointing to rows in the corrupt data block.

```
SET SERVEROUTPUT ON
DECLARE num_orphans INT;
BEGIN
 num_orphans := 0;
 DBMS_REPAIR.DUMP_ORPHAN_KEYS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'PK_DEPT',
     OBJECT_TYPE => dbms_repair.index_object,
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     ORPHAN_TABLE_NAME=> 'ORPHAN_KEY_TABLE',
     KEY_COUNT => num_orphans);
 DBMS_OUTPUT.PUT_LINE('orphan key count: ' || TO_CHAR(num_orphans));
END;
/
```

The following output indicates that there are three orphan keys:

```
orphan key count: 3
```

Index entries in the orphan key table implies that the index should be rebuilt. This guarantees that a table probe and an index probe return the same result set.

24.4.5 Example: Skipping Corrupt Blocks

An example illustrates skipping corrupt blocks using the `SKIP_CORRUPT_BLOCKS` procedure.

The `SKIP_CORRUPT_BLOCKS` procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object. When the object is a table, skipping applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

The following example enables the skipping of software corrupt blocks for the `scott.dept` table:

```
BEGIN
  DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object,
     FLAGS => dbms_repair.skip_flag);
END;
/
```

Querying `scott`'s tables using the `DBA_TABLES` view shows that `SKIP_CORRUPT` is enabled for table `scott.dept`.

```
SELECT OWNER, TABLE_NAME, SKIP_CORRUPT FROM DBA_TABLES
    WHERE OWNER = 'SCOTT';

OWNER                          TABLE_NAME                     SKIP_COR
```

```
------------------------------ ------------------------------ --------
SCOTT                          ACCOUNT                        DISABLED
SCOTT                          BONUS                          DISABLED
SCOTT                          DEPT                           ENABLED
SCOTT                          DOCINDEX                       DISABLED
SCOTT                          EMP                            DISABLED
SCOTT                          RECEIPT                        DISABLED
SCOTT                          SALGRADE                       DISABLED
SCOTT                          SCOTT_EMP                      DISABLED
SCOTT                          SYS_IOT_OVER_12255             DISABLED
SCOTT                          WORK_AREA                      DISABLED

10 rows selected.
```

Part IV

Database Resource Management and Task Scheduling

You can manage automated database maintenance tasks, database resources, and task scheduling.

- Managing Automated Database Maintenance Tasks
  Oracle Database has automated several common maintenance tasks typically performed by database administrators. These automated maintenance tasks are performed when the system load is expected to be light. You can enable and disable individual maintenance tasks, and can configure when these tasks run and what resource allocations they are allotted.
- Managing Resources with Oracle Database Resource Manager
  Oracle Database Resource Manager (Resource Manager) enables you to manage resource allocation for a database.
- Oracle Scheduler Concepts
  You can schedule tasks with Oracle Scheduler.
- Scheduling Jobs with Oracle Scheduler
  You can create, run, and manage jobs with Oracle Scheduler.
- Administering Oracle Scheduler
  You can configure, manage, monitor, and troubleshoot Oracle Scheduler.

25 Managing Automated Database Maintenance Tasks

Oracle Database has automated several common maintenance tasks typically performed by database administrators. These automated maintenance tasks are performed when the system load is expected to be light. You can enable and disable individual maintenance tasks, and can configure when these tasks run and what resource allocations they are allotted.

Note:

This chapter explains how to administer automated maintenance tasks using PL/SQL packages. An easier way is to use the graphical interface of Oracle Enterprise Manager Cloud Control (Cloud Control).

To manage automatic maintenance tasks with Cloud Control:

1. Access the Database Home Page.
2. From the Administration menu, select **Oracle Scheduler**, then **Automated Maintenance Tasks**.
3. On the Automated Maintenance Tasks page, click **Configure**.

- About Automated Maintenance Tasks
  Automated maintenance tasks are tasks that are started automatically at regular intervals to perform maintenance operations on the database. An example is a task that gathers statistics on schema objects for the query optimizer.
- About Maintenance Windows
  A **maintenance window** is a contiguous time interval during which automated maintenance tasks are run. Maintenance windows are Oracle Scheduler windows that belong to the window group named `MAINTENANCE_WINDOW_GROUP`.
- Configuring Automated Maintenance Tasks
  To enable or disable specific maintenance tasks in any subset of maintenance windows, you can use the `DBMS_AUTO_TASK_ADMIN` PL/SQL package.
- Configuring Maintenance Windows
  You may want to adjust the predefined maintenance windows to a time suitable to your database environment or create a new maintenance window. You can customize maintenance windows using the `DBMS_SCHEDULER` PL/SQL package.
- Configuring Resource Allocations for Automated Maintenance Tasks
  You can reduce or increase resource allocation to the automated maintenance tasks.
- Automated Maintenance Tasks Reference
  Oracle Database has predefined maintenance windows. It also has data dictionary views that you can query for information about automated maintenance.

25.1 About Automated Maintenance Tasks

Automated maintenance tasks are tasks that are started automatically at regular intervals to perform maintenance operations on the database. An example is a task that gathers statistics on schema objects for the query optimizer.

Automated maintenance tasks run in *maintenance windows*, which are predefined time intervals that are intended to occur during a period of low system load. You can customize maintenance windows based on the resource usage patterns of your database, or disable certain default windows from running. You can also create your own maintenance windows.

Oracle Database has these predefined automated maintenance tasks:

- **Automatic Optimizer Statistics Collection—**Collects optimizer statistics for all schema objects in the database for which there are no statistics or only stale statistics. The statistics gathered by this task are used by the SQL query optimizer to improve the performance of SQL execution.

  See Also:

  Oracle Database SQL Tuning Guide for more information on automatic statistics collection

- **Optimizer Statistics Advisor—**Analyzes how statistics are being gathered and suggests changes that can be made to fine tune statistics collection.

  See Also:

  Oracle Database SQL Tuning Guide

- **Automatic Segment Advisor—** Identifies segments that have space available for reclamation, and makes recommendations on how to defragment those segments.

  You can also run the Segment Advisor manually to obtain more up-to-the-minute recommendations or to obtain recommendations on segments that the Automatic Segment Advisor did not examine for possible space reclamation.

  See Also:

  "Using the Segment Advisor" for more information.

- **Automatic SQL Tuning Advisor—**Examines the performance of high-load SQL statements, and makes recommendations on how to tune those statements. You can configure this advisor to automatically implement SQL profile recommendations.

  See Also:

  Oracle Database SQL Tuning Guide for more information on SQL Tuning Advisor

- **SQL Plan Management (SPM) Evolve Advisor—**Evolves plans that have recently been added to the SQL plan baseline. The advisor simplifies plan evolution by eliminating the requirement to do it manually.

See Also:

Oracle Database SQL Tuning Guide for more information on SPM Evolve Advisor

By default, all of these automated maintenance tasks are configured to run in all maintenance windows.

25.2 About Maintenance Windows

A **maintenance window** is a contiguous time interval during which automated maintenance tasks are run. Maintenance windows are Oracle Scheduler windows that belong to the window group named `MAINTENANCE_WINDOW_GROUP`.

A Scheduler window can be a simple repeating interval (such as "between midnight and 6 a.m., every Saturday"), or a more complex interval (such as "between midnight and 6 a.m., on the last workday of every month, excluding company holidays").

When a maintenance window opens, Oracle Database creates an Oracle Scheduler job for each maintenance task that is scheduled to run in that window. Each job is assigned a job name that is generated at run time. All automated maintenance task job names begin with `ORA$AT`. For example, the job for the Automatic Segment Advisor might be called `ORA$AT_SA_SPC_SY_26`. When an automated maintenance task job finishes, it is deleted from the Oracle Scheduler job system. However, the job can still be found in the Scheduler job history.

Note:

To view job history, you must log in as the `SYS` user.

In the case of a very long maintenance window, all automated maintenance tasks except Automatic SQL Tuning Advisor are restarted every four hours. This feature ensures that maintenance tasks are run regularly, regardless of window size.

The framework of automated maintenance tasks relies on maintenance windows being defined in the database. Table 25-1 lists the maintenance windows that are automatically defined with each new Oracle Database installation.

See Also:

- "About Jobs and Supporting Scheduler Objects" for more information on windows and groups.

25.3 Configuring Automated Maintenance Tasks

To enable or disable specific maintenance tasks in any subset of maintenance windows, you can use the DBMS_AUTO_TASK_ADMIN PL/SQL package.

- Enabling and Disabling Maintenance Tasks for all Maintenance Windows
  With a single operation, you can disable or enable a particular automated maintenance task for all maintenance windows.
- Enabling and Disabling Maintenance Tasks for Specific Maintenance Windows
  By default, all maintenance tasks run in all predefined maintenance windows. You can disable a maintenance task for a specific window.

25.3.1 Enabling and Disabling Maintenance Tasks for all Maintenance Windows

With a single operation, you can disable or enable a particular automated maintenance task for all maintenance windows.

You can disable a particular automated maintenance task for all maintenance windows with a single operation. You do so by calling the DISABLE procedure of the DBMS_AUTO_TASK_ADMIN PL/SQL package without supplying the window_name argument. For example, you can completely disable the Automatic SQL Tuning Advisor task as follows:

```
BEGIN
  dbms_auto_task_admin.disable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => NULL);
END;
/
```

To enable this maintenance task again, use the ENABLE procedure, as follows:

```
BEGIN
  dbms_auto_task_admin.enable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => NULL);
END;
/
```

The task names to use for the client_name argument are listed in the DBA_AUTOTASK_CLIENT database dictionary view.

To enable or disable all automated maintenance tasks for all windows, call the ENABLE or DISABLE procedure with no arguments.

```
EXECUTE DBMS_AUTO_TASK_ADMIN.DISABLE;
```

See Also:

- "Automated Maintenance Tasks Database Dictionary Views"
- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_AUTO_TASK_ADMIN` PL/SQL package.

25.3.2 Enabling and Disabling Maintenance Tasks for Specific Maintenance Windows

By default, all maintenance tasks run in all predefined maintenance windows. You can disable a maintenance task for a specific window.

The following example disables the Automatic SQL Tuning Advisor from running in the window `MONDAY_WINDOW`:

```
BEGIN
  dbms_auto_task_admin.disable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => 'MONDAY_WINDOW');
END;
/
```

25.4 Configuring Maintenance Windows

You may want to adjust the predefined maintenance windows to a time suitable to your database environment or create a new maintenance window. You can customize maintenance windows using the `DBMS_SCHEDULER` PL/SQL package.

- Modifying a Maintenance Window
  The `DBMS_SCHEDULER` PL/SQL package includes a `SET_ATTRIBUTE` procedure for modifying the attributes of a window.
- Creating a New Maintenance Window
  To create a new maintenance window, you must create an Oracle Scheduler window object and then add it to the window group `MAINTENANCE_WINDOW_GROUP`.
- Removing a Maintenance Window
  To remove an existing maintenance window, remove it from the `MAINTENANCE_WINDOW_GROUP` window group.

25.4.1 Modifying a Maintenance Window

The `DBMS_SCHEDULER` PL/SQL package includes a `SET_ATTRIBUTE` procedure for modifying the attributes of a window.

For example, the following script changes the duration of the maintenance window `SATURDAY_WINDOW` to 4 hours:

```
BEGIN
  dbms_scheduler.disable(
    name  => 'SATURDAY_WINDOW');
  dbms_scheduler.set_attribute(
    name      => 'SATURDAY_WINDOW',
    attribute => 'DURATION',
    value     => numtodsinterval(4, 'hour'));
  dbms_scheduler.enable(
    name => 'SATURDAY_WINDOW');
END;
/
```

Note that you must use the `DBMS_SCHEDULER.DISABLE` subprogram to disable the window before making changes to it, and then re-enable the window with `DBMS_SCHEDULER.ENABLE` when you are finished. If you change a window when it is currently open, the change does not take effect until the next time the window opens.

See Also:

"Managing Job Scheduling and Job Priorities with Windows" for more information about modifying windows.

25.4.2 Creating a New Maintenance Window

To create a new maintenance window, you must create an Oracle Scheduler window object and then add it to the window group `MAINTENANCE_WINDOW_GROUP`.

You use the `DBMS_SCHEDULER.CREATE_WINDOW` package procedure to create the window, and the `DBMS_SCHEDULER.ADD_GROUP_MEMBER` procedure to add the new window to the window group.

The following example creates a maintenance window named `EARLY_MORNING_WINDOW`. This window runs for one hour daily between 5 a.m. and 6 a.m.

```
BEGIN
  DBMS_SCHEDULER.CREATE_WINDOW(
    window_name     => 'EARLY_MORNING_WINDOW',
    duration        =>  NUMTODSINTERVAL(1, 'hour'),
    resource_plan   => 'DEFAULT_MAINTENANCE_PLAN',
    repeat_interval => 'FREQ=DAILY;BYHOUR=5;BYMINUTE=0;BYSECOND=0');
  DBMS_SCHEDULER.ADD_GROUP_MEMBER(
    group_name  => 'MAINTENANCE_WINDOW_GROUP',
    member      => 'EARLY_MORNING_WINDOW');
END;
/
```

See Also:

- "Creating Windows"
- *Oracle Database PL/SQL Packages and Types Reference* for information on the `DBMS_SCHEDULER` package

25.4.3 Removing a Maintenance Window

To remove an existing maintenance window, remove it from the MAINTENANCE_WINDOW_GROUP window group.

The window continues to exist but no longer runs automated maintenance tasks. Any other Oracle Scheduler jobs assigned to this window continue to run as usual.

The following example removes EARLY_MORNING_WINDOW from the window group:

```
BEGIN
  DBMS_SCHEDULER.REMOVE_GROUP_MEMBER(
    group_name  => 'MAINTENANCE_WINDOW_GROUP',
    member      => 'EARLY_MORNING_WINDOW');
END;
/
```

See Also:

- "Removing a Member from a Window Group"
- "Dropping Windows"
- *Oracle Database PL/SQL Packages and Types Reference* for information on the DBMS_SCHEDULER package

25.5 Configuring Resource Allocations for Automated Maintenance Tasks

You can reduce or increase resource allocation to the automated maintenance tasks.

- About Resource Allocations for Automated Maintenance Tasks
  By default, all predefined maintenance windows use the resource plan DEFAULT_MAINTENANCE_PLAN. Automated maintenance tasks run under its subplan ORA$AUTOTASK. This subplan divides its portion of total resource allocation equally among the maintenance tasks.
- Changing Resource Allocations for Automated Maintenance Tasks
  To change the resource allocation for automated maintenance tasks within a maintenance window, you must change the percentage of resources allocated to the subplan ORA$AUTOTASK in the resource plan for that window.

See Also:

Managing Resources with Oracle Database Resource Manager

25.5.1 About Resource Allocations for Automated Maintenance Tasks

By default, all predefined maintenance windows use the resource plan `DEFAULT_MAINTENANCE_PLAN`. Automated maintenance tasks run under its subplan `ORA$AUTOTASK`. This subplan divides its portion of total resource allocation equally among the maintenance tasks.

`DEFAULT_MAINTENANCE_PLAN` defines the following resource allocations:

Consumer Group/subplan	Level 1	Maximum Utilization Limit
`ORA$AUTOTASK`	5%	90
`OTHER_GROUPS`	20%	-
`SYS_GROUP`	75%	-

In this plan, any sessions in the `SYS_GROUP` consumer group get priority. (Sessions in this group are sessions created by user accounts `SYS` and `SYSTEM`.) Any resource allocation that is unused by sessions in `SYS_GROUP` is then shared by sessions belonging to the other consumer groups and subplans in the plan. Of that allocation, 5% goes to maintenance tasks and 20% goes to user sessions. The maximum utilization limit for `ORA$AUTOTASK` is 90. Therefore, even if the CPU is idle, this group/plan cannot be allocated more than 90% of the CPU resources.

To reduce or increase resource allocation to the automated maintenance tasks, you make adjustments to `DEFAULT_MAINTENANCE_PLAN`. See "Changing Resource Allocations for Automated Maintenance Tasks" for more information.

Note that as with any resource plan, the portion of an allocation that is not used by a consumer group or subplan is available for other consumer groups or subplans. Note also that the Database Resource Manager does not begin to limit resource allocations according to resource plans until 100% of CPU is being used.

Note:

Although `DEFAULT_MAINTENANCE_PLAN` is the default, you can assign any resource plan to any maintenance window. If you do change a maintenance window resource plan, ensure that you include the subplan `ORA$AUTOTASK` in the new plan.

See Also:

Managing Resources with Oracle Database Resource Manager for more information on resource plans.

25.5.2 Changing Resource Allocations for Automated Maintenance Tasks

To change the resource allocation for automated maintenance tasks within a maintenance window, you must change the percentage of resources allocated to the subplan `ORA$AUTOTASK` in the resource plan for that window.

(By default, the resource plan for each predefined maintenance window is `DEFAULT_MAINTENANCE_PLAN`.) You must also adjust the resource allocation for one or more other subplans or consumer groups in the window's resource plan such that the resource allocation at the top level of the plan adds up to 100%. For information on changing resource allocations, see Managing Resources with Oracle Database Resource Manager.

25.6 Automated Maintenance Tasks Reference

Oracle Database has predefined maintenance windows. It also has data dictionary views that you can query for information about automated maintenance.

- Predefined Maintenance Windows
  By default there are seven predefined maintenance windows, each one representing a day of the week.
- Automated Maintenance Tasks Database Dictionary Views
  You can query a set of data dictionary views for information about automated maintenance tasks.

25.6.1 Predefined Maintenance Windows

By default there are seven predefined maintenance windows, each one representing a day of the week.

The weekend maintenance windows, `SATURDAY_WINDOW` and `SUNDAY_WINDOW`, are longer in duration than the weekday maintenance windows. The window group `MAINTENANCE_WINDOW_GROUP` consists of these seven windows. The list of predefined maintenance windows is given in Table 25-1.

Table 25-1 Predefined Maintenance Windows

Window Name	Description
`MONDAY_WINDOW`	Starts at 10 p.m. on Monday and ends at 2 a.m.
`TUESDAY_WINDOW`	Starts at 10 p.m. on Tuesday and ends at 2 a.m.
`WEDNESDAY_WINDOW`	Starts at 10 p.m. on Wednesday and ends at 2 a.m.
`THURSDAY_WINDOW`	Starts at 10 p.m. on Thursday and ends at 2 a.m.
`FRIDAY_WINDOW`	Starts at 10 p.m. on Friday and ends at 2 a.m.
`SATURDAY_WINDOW`	Starts at 6 a.m. on Saturday and is 20 hours long.
`SUNDAY_WINDOW`	Starts at 6 a.m. on Sunday and is 20 hours long.

25.6.2 Automated Maintenance Tasks Database Dictionary Views

You can query a set of data dictionary views for information about automated maintenance tasks.

Table 25-2 displays information about database dictionary views for automated maintenance tasks:

Table 25-2 Automated Maintenance Tasks Database Dictionary Views

View Name	Description
`DBA_AUTOTASK_CLIENT_JOB`	Contains information about currently running Scheduler jobs created for automated maintenance tasks. It provides information about some objects targeted by those jobs, as well as some additional statistics from previous instantiations of the same task. Some of this additional data is taken from generic Scheduler views.
`DBA_AUTOTASK_CLIENT`	Provides statistical data for each automated maintenance task over 7-day and 30-day periods.
`DBA_AUTOTASK_JOB_HISTORY`	Lists the history of automated maintenance task job runs. Jobs are added to this view after they finish executing.
`DBA_AUTOTASK_WINDOW_CLIENTS`	Lists the windows that belong to `MAINTENANCE_WINDOW_GROUP`, along with the Enabled or Disabled status for the window for each maintenance task. Primarily used by Cloud Control.
`DBA_AUTOTASK_CLIENT_HISTORY`	Provides per-window history of job execution counts for each automated maintenance task. This information is viewable in the Job History page of Cloud Control.

See Also:

"Resource Manager Data Dictionary Views" for column descriptions for views.

26 Managing Resources with Oracle Database Resource Manager

Oracle Database Resource Manager (Resource Manager) enables you to manage resource allocation for a database.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

> **Note:**
>
> This chapter discusses using PL/SQL package procedures to administer the Resource Manager. An easier way to administer the Resource Manager is with the graphical user interface of Oracle Enterprise Manager Cloud Control (Cloud Control). For instructions about administering Resource Manager with Cloud Control, see the Cloud Control online help.
>
> To use Resource Manager with Cloud Control:
>
> 1. Access the Database Home Page.
> 2. From the Administration menu, select **Resource Manager**.

- Managing Resource Plans
  Resource Manager allocates resources to pluggable databases (PDBs) in a multitenant container database (CDB).
- Putting It All Together: Oracle Database Resource Manager Examples
  Examples illustrate how to allocate resources with Resource Manager.
- Managing Multiple Database Instances on a Single Server
  Oracle Database provides a method for managing CPU allocations on a multi-CPU server running multiple database instances. This method is called instance caging. Instance caging and Oracle Database Resource Manager (the Resource Manager) work together to support desired levels of service across multiple instances.
- Maintaining Consumer Groups, Plans, and Directives
  You can maintain consumer groups, resource plans, and resource plan directives for Oracle Database Resource Manager (the Resource Manager). You perform maintenance tasks using the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- Viewing Database Resource Manager Configuration and Status
  You can use several static data dictionary views and dynamic performance views to view the current configuration and status of Oracle Database Resource Manager (the Resource Manager).
- Interacting with Operating-System Resource Control
  Many operating systems provide tools for resource management. These tools often contain "workload manager" or "resource manager" in their names, and are intended to allow multiple applications to share the resources of a single server, using an administrator-defined policy. Examples are Hewlett Packard's Process Resource Manager or Solaris Containers, Zones, and Resource Pools.
- Oracle Database Resource Manager Reference
  Resource Manager includes predefined resource plans, consumer groups, and consumer groups mapping rules. You can query data dictionary views for information about your Resource Manager configuration.

26.1 About Oracle Database Resource Manager

Oracle Database Resource Manager (the Resource Manager) enables you to manage multiple workloads within a database that are contending for system and database resources.

Note:

The Resource Manager manages activity in the CDB root automatically.

- CDB and PDB Resource Management
  Using Oracle Resource Manager (Resource Manager), you can create CDB resource plans and set initialization parameters to allocate resources to PDBs.
- Purpose of Resource Management
  When database resource allocation decisions are left to the operating system, workload management can be problematic. The Resource Manager helps solve these problems.
- Consumer Groups, Plans, and Plan Directives
  Resource Manager includes several elements that you can manage.

- User Interface for PDB Resource Management
  You can manage PDB resources using `DBMS_RESOURCE_MANAGER` and initialization parameters.

26.1.1 CDB and PDB Resource Management

Using Oracle Resource Manager (Resource Manager), you can create CDB resource plans and set initialization parameters to allocate resources to PDBs.

In a CDB, multiple workloads within multiple PDBs can complete for system and CDB resources. Resource Manager can manage resources on two levels: CDB and PDB.

CDB Resource Plans

A CDB resource plan allocates resources to its PDBs according to its set of resource plan directives (directives). A parent-child relationship exists between a CDB resource plan and its directives. Each resource plan directive references either a set of PDBs or an individual PDB.

A performance profile specifies shares of system resources for a set of PDBs. PDB performance profiles enable you to manage resources for large numbers of PDBs by specifying Resource Manager directives for profiles instead of individual PDBs.

The directives control allocation of CPU and parallel execution servers. A directive can control the allocation of resources to PDBs based on the share value that you specify for each PDB or PDB performance profile. A higher share value results in more guaranteed resources. For PDBs and PDB performance profiles, you can also set utilization limits for CPU and parallel servers.

You can create a CDB resource plan by using the `CREATE_CDB_PLAN` procedure in the `DBMS_RESOURCE_MANAGER` PL/SQL package, and set a CDB resource plan using the `RESOURCE_MANAGER_PLAN` parameter. You create directives for a CDB resource plan by using the `CREATE_CDB_PLAN_DIRECTIVE` procedure.

PDB Resource Plans

A CDB resource plan allocates a portion of the system resources to a PDB. A PDB resource plan determines how this portion is allocated within the PDB.

You can create a PDB resource plan by using the `CREATE_PLAN` procedure in the `DBMS_RESOURCE_MANAGER` PL/SQL package, and set a PDB resource plan using the `RESOURCE_MANAGER_PLAN` parameter. You create directives for a PDB resource plan by using the `CREATE_PLAN_DIRECTIVE` procedure.

PDB-Level Memory Controls

In a CDB, PDBs may contend for SGA or PGA memory. When you set the following initialization parameters with the PDB as the current container, the parameters limit the memory usage of the current PDB.

Examples of important parameters include:

- `SGA_TARGET` specifies the maximum SGA that the PDB can use at any time.
- `PGA_AGGREGATE_LIMIT` sets the maximum PGA that the PDB can use at any time.

The only shared memory sizing parameter that should be set for a PDB is `SGA_TARGET`, which specifies the maximum SGA that the PDB can use at any time.

PDB-Level I/O Controls

Intensive disk I/O can cause poor performance. Several factors can result in excess disk I/O, such as poorly designed SQL or index and table scans in high-volume transactions. If one PDB generates excessive disk I/O, then it can degrade the performance of other PDBs in the same CDB.

On non-Engineered Systems, use one or both of the following initialization parameters to limit the I/O generated by a particular PDB:

- `MAX_IOPS` limits the number of I/O operations for each second.
- `MAX_MBPS` limits the MB/s for I/O operations.

For Engineered Systems, manage PDB I/Os with I/O Resource Management.

See Also:

- Managing Resource Plans
- *Oracle Database Reference* to learn more about `DB_CACHE_SIZE` and other initialization parameters
- *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_RESOURCE_MANAGER` package
- *Oracle Exadata Storage Server Software User's Guide* to learn more about I/O Resource Management

26.1.2 Purpose of Resource Management

When database resource allocation decisions are left to the operating system, workload management can be problematic. The Resource Manager helps solve these problems.

- Purpose of Resource Management for a CDB
  Resource Manager allows a CDB to have more control over how hardware resources are allocated.
- Purpose of Resource Management for PDBs
  In a CDB, workloads within multiple PDBs can compete for system and CDB resources. Resource plans solve this problem.

26.1.2.1 Purpose of Resource Management for a CDB

Resource Manager allows a CDB to have more control over how hardware resources are allocated.

Resource Management Problems for a CDB

When database resource allocation decisions are left to the operating system, you may encounter the following problems with workload management:

- Excessive overhead

  Excessive overhead results from operating system context switching between Oracle Database server processes when the number of server processes is high.

- Inefficient scheduling

  The operating system deschedules database servers while they hold latches, which is inefficient.
- Inappropriate allocation of resources

  The operating system distributes resources equally among all active processes and cannot prioritize one task over another.
- Inability to manage database-specific resources, such as parallel execution servers and active sessions

The Resource Manager Solution

The Resource Manager helps to overcome these problems by allowing a CDB more control over how hardware resources are allocated. In an environment with multiple concurrent user sessions that run jobs with differing priorities, all sessions should not be treated equally. The Resource Manager enables you to classify sessions into groups based on session attributes, and to then allocate resources to those groups in a way that optimizes hardware utilization for your application environment.

With the Resource Manager, you can:

- Guarantee certain sessions a minimum amount of CPU regardless of the load on the system and the number of users.
- Distribute available CPU by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage can be given to ROLAP (relational online analytical processing) applications than to batch jobs.
- Limit the degree of parallelism of any operation performed by members of a group of users.
- Manage the order of parallel statements in the parallel statement queue. Parallel statements from a critical application can be enqueued ahead of parallel statements from a low priority group of users.
- Limit the number of parallel execution servers that a group of users can use. This ensures that all the available parallel execution servers are not allocated to only one group of users.
- Create an active session pool. An *active session pool* consists of a specified maximum number of user sessions allowed to be concurrently active within a group of users. Additional sessions beyond the maximum are queued for execution, but you can specify a timeout period, after which queued jobs will terminate. The active session pool limits the total number of sessions actively competing for resources, thereby enabling active sessions to make faster progress.
- Monitor resources

  Automatically record statistics about resource usage. You can examine these statistics using real-time SQL monitoring and the Resource Manager dynamic performance views (the `V$RSRC_*` views). See "Monitoring Oracle Database Resource Manager" for information about using real-time SQL monitoring and the Resource Manager dynamic performance views.
- Limit the amount of PGA memory used by each session that belongs to a group of users.
- Manage runaway sessions or calls in the following ways:
  - By detecting when a session or call consumes more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time, and then automatically either terminating the session or call, or switching to a consumer group with a lower resource allocation or a limit on the percentage of CPU that the group can use. The SQL statements that are

terminated due to their excessive consumption of system resources are *quarantined*, that is, they are not allowed to run again by generating compilation errors during their subsequent runs.

A logical I/O, also known as a buffer I/O, refers to reads and writes of buffers in the buffer cache. When a requested buffer is not found in memory, the database performs a physical I/O to copy the buffer from either disk or the flash cache into memory, and then a logical I/O to read the cached buffer.

  – By recording detailed information about SQL statements that consume more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time with real-time SQL monitoring.
  – By using the Automatic Workload Repository (AWR) to analyze a persistent record of SQL statements that consume more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time.
  – By logging information about a runaway session without taking any other action related to the session.

- Prevent the execution of operations that the optimizer estimates will run for a longer time than a specified limit.
- Limit the amount of time that a session can be idle. This can be further defined to mean only sessions that are blocking other sessions.
- Allow a database to use different resource plans, based on changing workload requirements. You can dynamically change the resource plan, for example, from a daytime resource plan to a nighttime resource plan, without having to shut down and restart the instance. You can also schedule a resource plan change with Oracle Scheduler. See Oracle Scheduler Concepts for more information.

26.1.2.2 Purpose of Resource Management for PDBs

In a CDB, workloads within multiple PDBs can compete for system and CDB resources. Resource plans solve this problem.

Resource Management Problems for PDBs

When multiple PDBs in a CDB are contending for resources, you may encounter the following problems with workload management:

- Inappropriate allocation of resources among PDBs

  The operating system distributes resources equally among all active processes and cannot prioritize one task over another. Therefore, one or more PDBs might use an inordinate amount of the system resources, leaving the other PDBs starved for resources.
- Inappropriate allocation of resources within a single PDB

  One or more sessions connected to a single PDB might use an inordinate amount of the system resources, leaving other sessions connected to the same PDB starved for resources.
- Inconsistent performance of PDBs

  A single PDB might perform inconsistently when other PDBs are competing for more system resources or less system resources at various times.
- Lack of resource usage data for PDBs

  Resource usage data is critical for monitoring and tuning PDBs. Operating system monitoring tools are typically not useful because there are multiple PDBs running on the system.

The Resource Management Solution in the Multitenant Environment

Resource Manager helps to overcome these problems by enabling you to prioritize and limit the resource usage of specific PDBs. With the Resource Manager, you can:

- Specify that different PDBs should receive different shares of the system resources so that more resources are allocated to the more important PDBs
- Limit the CPU usage of a particular PDB
- Limit the number of parallel execution servers that a particular PDB can use
- Limit the memory usage of a particular PDB
- Specify the amount of memory guaranteed for a particular PDB
- Specify the maximum amount of memory a particular PDB can use
- Use PDB performance profiles for different sets of PDB

  A performance profile for a set of PDBs can specify shares of system resources, CPU usage, and number of parallel execution servers. PDB performance profiles enable you to manage resources for large numbers of PDBs by specifying Resource Manager directives for profiles instead of individual PDBs.
- Limit the resource usage of different sessions connected to a single PDB
- Limit the I/O generated by specific PDBs
- Monitor the resource usage of PDBs

26.1.3 Consumer Groups, Plans, and Plan Directives

Resource Manager includes several elements that you can manage.

- About the Elements of Resource Manager
  The elements of the Resource Manager include resource consumer groups, resource plans, and resource plan directives.
- About Resource Consumer Groups
  A resource consumer group (consumer group) is a collection of user sessions that are grouped together based on their processing needs.
- About Resource Plan Directives
  The Resource Manager allocates resources to consumer groups according to the set of resource plan directives (directives) that belong to the currently active resource plan.
- About Resource Plans
  A resource plan is a container for directives that specify how resources are allocated to resource consumer groups.
- About Subplans
  Instead of referencing a consumer group, a resource plan directive (directive) can reference another resource plan. In this case, the plan is referred to as a subplan.

26.1.3.1 About the Elements of Resource Manager

The elements of the Resource Manager include resource consumer groups, resource plans, and resource plan directives.

Element	Description
Resource consumer group	A group of sessions that are grouped together based on resource requirements. The Resource Manager allocates resources to resource consumer groups, not to individual sessions.
Resource plan	A container for directives that specify how resources are allocated to resource consumer groups. You specify how the database allocates resources by activating a specific resource plan.
Resource plan directive	Associates a resource consumer group with a particular plan and specifies how resources are to be allocated to that resource consumer group.

You use the `DBMS_RESOURCE_MANAGER` PL/SQL package to create and maintain these elements. The elements are stored in tables in the data dictionary. You can view information about them with data dictionary views.

See Also:

"Resource Manager Data Dictionary Views"

26.1.3.2 About Resource Consumer Groups

A resource consumer group (consumer group) is a collection of user sessions that are grouped together based on their processing needs.

When a session is created, it is automatically mapped to a consumer group based on mapping rules that you set up. As a database administrator (DBA), you can manually switch a session to a different consumer group. Similarly, an application can run a PL/SQL package procedure that switches its session to a particular consumer group.

Because the Resource Manager allocates resources (such as CPU) only to consumer groups, when a session becomes a member of a consumer group, its resource allocation is determined by the allocation for the consumer group.

There are special consumer groups that are always present in the data dictionary. They cannot be modified or deleted. They are:

- `SYS_GROUP`

  This is the initial consumer group for all sessions created by user accounts `SYS` or `SYSTEM`. This initial consumer group can be overridden by session-to-consumer group mapping rules.

- `OTHER_GROUPS`

  This consumer group contains all sessions that have not been assigned to a consumer group. Every resource plan must contain a directive to `OTHER_GROUPS`.

There can be no more than 28 resource consumer groups in any active plan.

- Consumer Groups for PDBs
  In a CDB, background and administrative tasks map to the Resource Manager consumer groups that run them optimally.

See Also:

- Table 26-18
- "Specifying Session-to-Consumer Group Mapping Rules"

26.1.3.2.1 Consumer Groups for PDBs

In a CDB, background and administrative tasks map to the Resource Manager consumer groups that run them optimally.

Resource Manager uses the following rules to map a task to a consumer group:

- A task is mapped to a consumer group in the container that starts the task.

  If a task starts in the CDB root, then the task maps to a consumer group in the CDB root. If the task starts in a PDB, then the task maps to a consumer group in the PDB.
- Many maintenance and administrative tasks automatically map to a consumer group.

  For example, automated maintenance tasks map to `ORA$AUTOTASK`. In certain cases, the tasks map to a consumer group, but the mapping is modifiable. Such tasks include RMAN backup, RMAN image copy, Oracle Data Pump, and In-Memory population.

Note:

Oracle Database Administrator's Guide to learn more about the mapping rules for predefined consumer groups

26.1.3.3 About Resource Plan Directives

The Resource Manager allocates resources to consumer groups according to the set of resource plan directives (directives) that belong to the currently active resource plan.

There is a parent-child relationship between a resource plan and its resource plan directives. Each directive references one consumer group, and no two directives for the currently active plan can reference the same consumer group.

A directive has several ways in which it can limit resource allocation for a consumer group. For example, it can control how much CPU the consumer group gets as a percentage of total CPU, and it can limit the total number of sessions that can be active in the consumer group.

CDB resource plan allocates resources to its PDBs according to its set of resource plan directives (directives). A parent-child relationship exists between a CDB resource plan and its resource plan directives. Each directive references either a set of PDBs in a performance profile, or a single PDB. You can specify directives for both individual PDBs and for PDB performance profiles in the same CDB. No two directives for the currently active plan can reference the same PDB or the same PDB performance profile.

- Resources Managed by the Resource Manager
  Resource plan directives specify how resources are allocated to resource consumer groups or subplans. Each directive can specify several different methods for allocating resources to its consumer group or subplan.

- Resource Plan Directives for PDBs
  Directives control allocation of CPU and parallel execution servers.
- Performance Profiles for PDBs
  A **PDB performance profile** configures resource plan directives for a set of PDBs that have the same priorities and resource controls.

26.1.3.3.1 Resources Managed by the Resource Manager

Resource plan directives specify how resources are allocated to resource consumer groups or subplans. Each directive can specify several different methods for allocating resources to its consumer group or subplan.

- CPU
  To manage CPU resources, Resource Manager allocates resources among consumer groups and redistributes CPU resources that were allocated but were not used. You can also set a limit on the amount of CPU resources that can be allocated to a particular consumer group.
- Exadata I/O
  Management attributes enable you to specify CPU resource allocation for Exadata I/O.
- Parallel Execution Servers
  Resource Manager can manage usage of the available parallel execution servers for a database.
- Program Global Area (PGA)
  To manage PGA resources, Resource Manager can limit the amount of PGA memory that can be allocated to each session in a particular consumer group.
- Runaway Queries
  Runaway sessions and calls can adversely impact overall performance if they are not managed properly. Resource Manager can take action when a session or call consumes more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time. Resource Manager can either switch the session or call to a consumer group that is allocated a small amount of CPU or terminate the session or call.
- Active Session Pool with Queuing
  You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum defines the **active session pool**.
- Undo Pool
  You can specify an undo pool for each consumer group. An undo pool controls the total amount of undo for uncommitted transactions that can be generated by a consumer group.
- Idle Time Limit
  You can specify an amount of time that a session can be idle, after which it is terminated.

26.1.3.3.1.1 CPU

To manage CPU resources, Resource Manager allocates resources among consumer groups and redistributes CPU resources that were allocated but were not used. You can also set a limit on the amount of CPU resources that can be allocated to a particular consumer group.

- Management Attributes
  Management attributes enable you to specify how CPU resources are to be allocated among consumer groups and subplans.

- Utilization Limit
  Use the `UTILIZATION_LIMIT` attribute to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any redistribution of CPU within a plan.

26.1.3.3.1.1.1 Management Attributes

Management attributes enable you to specify how CPU resources are to be allocated among consumer groups and subplans.

Multiple levels of CPU resource allocation (up to eight levels) provide a means of prioritizing CPU usage within a plan. Consumer groups and subplans at level 2 get resources that were not allocated at level 1 or that were allocated at level 1 but were not completely consumed by a consumer group or subplan at level 1. Similarly, resource consumers at level 3 are allocated resources only when some allocation remains from levels 1 and 2. The same rules apply to levels 4 through 8. Multiple levels not only provide a way of prioritizing, but they provide a way of explicitly specifying how all primary and leftover resources are to be used.

Use the management attributes `MGMT_P`*`n`*, where *n* is an integer between 1 and 8, to specify multiple levels of CPU resource allocation. For example, use the `MGMT_P1` directive attribute to specify CPU resource allocation at level 1 and `MGMT_P2` directive attribute to specify resource allocation at level 2.

Use management attributes with parallel statement directive attributes, such as Degree of Parallelism Limit and Parallel Server Limit, to control parallel statement queuing. When parallel statement queuing is used, management attributes are used to determine which consumer group is allowed to issue the next parallel statement. For example, if you set the `MGMT_P1` directive attribute for a consumer group to 80, that group has an 80% chance of issuing the next parallel statement.

See Also:

Oracle Database VLDB and Partitioning Guide for information about parallel statement queuing

Table 26-1 illustrates a simple resource plan with three levels.

Table 26-1 A Simple Three-Level Resource Plan

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation
`HIGH_GROUP`	80%		
`LOW_GROUP`		50%	
`MAINT_SUBPLAN`		50%	
`OTHER_GROUPS`			100%

High priority applications run within `HIGH_GROUP`, which is allocated 80% of CPU. Because `HIGH_GROUP` is at level one, it gets priority for CPU utilization, but only up to 80% of CPU. This leaves a remaining 20% of CPU to be shared 50-50 by `LOW_GROUP` and the `MAINT_SUPLAN` at level 2. Any unused allocation from levels 1 and 2 are then available to `OTHER_GROUPS` at level 3. Because `OTHER_GROUPS` has no sibling consumer groups or subplans at its level, 100% is specified.

Within a particular level, CPU allocations are not fixed. If there is not sufficient load in a particular consumer group or subplan, residual CPU can be allocated to remaining consumer groups or subplans. Thus, when there is only one level, unused allocation by any consumer group or subplan can be redistributed to other "sibling" consumer groups or subplans. If there are multiple levels, then the unused allocation is distributed to the consumer groups or subplans at the next level. If the last level has unused allocations, these allocations can be redistributed to all other levels in proportion to their designated allocations.

As an example of redistribution of unused allocations from one level to another, if during a particular period, `HIGH_GROUP` consumes only 25% of CPU, then 75% is available to be shared by `LOW_GROUP` and `MAINT_SUBPLAN`. Any unused portion of the 75% at level 2 is then made available to `OTHER_GROUPS` at level 3. However, if `OTHER_GROUPS` has no session activity at level 3, then the 75% at level 2 can be redistributed to all other consumer groups and subplans in the plan proportionally.

26.1.3.3.1.1.2 Utilization Limit

Use the `UTILIZATION_LIMIT` attribute to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any redistribution of CPU within a plan.

In the previous scenario, suppose that due to inactivity elsewhere, `LOW_GROUP` acquires 90% of CPU. Suppose that you do not want to allow `LOW_GROUP` to use 90% of the server because you do not want non-critical sessions to inundate the CPUs. The `UTILIZATION_LIMIT` attribute of resource plan directives can prevent this situation.

Setting the `UTILIZATION_LIMIT` attribute is optional. If you omit this attribute for a consumer group, there is no limit on the amount of CPU that the consumer group can use. Therefore, if all the other applications are idle, a consumer group that does not have `UTILIZATION_LIMIT` set can be allocated 100% of the CPU resources.

You can also use the `UTILIZATION_LIMIT` attribute as the sole means of limiting CPU utilization for consumer groups, without specifying level limits.

Table 26-2 shows a variation of the previous plan. In this plan, using `UTILIZATION_LIMIT`, CPU utilization is capped at 75% for `LOW_GROUP`, 50% for `MAINT_SUBPLAN`, and 75% for `OTHER_GROUPS`. (Note that the sum of all utilization limits can exceed 100%. Each limit is applied independently.)

Table 26-2 A Three-Level Resource Plan with Utilization Limits

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation	Utilization Limit
`HIGH_GROUP`	80%			
`LOW_GROUP`		50%		75%
`MAINT_SUBPLAN`		50%		50%
`OTHER_GROUPS`			100%	75%

In the example described in Table 26-2, if `HIGH_GROUP` is using only 10% of the CPU at a given time, then the remaining 90% is available to `LOW_GROUP` and the consumer groups in `MAINT_SUBPLAN` at level 2. If `LOW_GROUP` uses only 20% of the CPU, then 70% can be allocated to `MAINT_SUBPLAN`. However, `MAINT_SUBPLAN` has a `UTILIZATION_LIMIT` of 50%. Therefore, even though more CPU resources are available, the server cannot allocate more than 50% of the CPU to the consumer groups that belong to the subplan `MAINT_SUBPLAN`.

You can set `UTILIZATION_LIMIT` for both a subplan and the consumer groups that the subplan contains. In such cases, the limit for a consumer group is computed using the limits specified for the subplan and that consumer group. For example, the MAINT_SUBPLAN contains the consumer groups MAINT_GROUP1 and MAINT_GROUP2. MAINT_GROUP1 has UTILIZATION_LIMIT set to 40%. However, the limit for MAINT_SUBPLAN is set to 50%. Therefore, the limit for consumer group MAINT_GROUP1 is computed as 40% of 50%, or 20%. For an example of how to compute UTILIZATION_LIMIT for a consumer group when limits are specified for both the consumer group and the subplan to which the group belongs, see "Example 4 - Specifying a Utilization Limit for Consumer Groups and Subplans".

See Also:

- "Creating Resource Plan Directives "
- "Putting It All Together: Oracle Database Resource Manager Examples"

26.1.3.3.1.2 Exadata I/O

Management attributes enable you to specify CPU resource allocation for Exadata I/O.

See Also:

The Exadata documentation for information about using management attributes for Exadata I/O

26.1.3.3.1.3 Parallel Execution Servers

Resource Manager can manage usage of the available parallel execution servers for a database.

- Degree of Parallelism Limit
  You can limit the maximum degree of parallelism for any operation within a consumer group. Use the PARALLEL_DEGREE_LIMIT_P1 directive attribute to specify the degree of parallelism for a consumer group.
- Parallel Server Limit
  Use the PARALLEL_SERVER_LIMIT directive attribute to specify the maximum percentage of the parallel execution server pool that a particular consumer group can use. The number of parallel execution servers used by a particular consumer group is counted as the sum of the parallel execution servers used by all sessions in that consumer group.
- Parallel Queue Timeout
  The PARALLEL_QUEUE_TIMEOUT directive attribute enables you to specify the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.

26.1.3.3.1.3.1 Degree of Parallelism Limit

You can limit the maximum degree of parallelism for any operation within a consumer group. Use the PARALLEL_DEGREE_LIMIT_P1 directive attribute to specify the degree of parallelism for a consumer group.

The degree of parallelism limit applies to one operation within a consumer group; it does not limit the total degree of parallelism across all operations within the consumer group. However, you can combine both the PARALLEL_DEGREE_LIMIT_P1 and the PARALLEL_SERVER_LIMIT directive attributes to achieve the desired control. For more information about the PARALLEL_SERVER_LIMIT attribute, see "Parallel Server Limit".

See Also:

Oracle Database VLDB and Partitioning Guide for more information about degree of parallelism in producer/consumer operations

26.1.3.3.1.3.2 Parallel Server Limit

Use the PARALLEL_SERVER_LIMIT directive attribute to specify the maximum percentage of the parallel execution server pool that a particular consumer group can use. The number of parallel execution servers used by a particular consumer group is counted as the sum of the parallel execution servers used by all sessions in that consumer group.

It is possible for a single consumer group to launch enough parallel statements to use all of the available parallel execution servers. If this happens when a high-priority parallel statement from a different consumer group is run, then no parallel execution servers are available to allocate to this group. You can avoid such a scenario by limiting the number of parallel execution servers that can be used by a particular consumer group. You can also set the directive PARALLEL_STMT_CRITICAL to BYPASS_QUEUE for the high-priority consumer group so that parallel statements from the consumer group bypass the parallel statement queue.

For example, assume that the total number of parallel execution servers is 32, as set by the PARALLEL_SERVERS_TARGET initialization parameter, and the PARALLEL_SERVER_LIMIT directive attribute for the consumer group MY_GROUP is set to 50%. This consumer group can use a maximum of 50% of 32, or 16 parallel execution servers.

If your resource plan has management attributes (MGMT_P1, MGMT_P2, and so on), then a separate parallel statement queue is managed as a First In First Out (FIFO) queue for each management attribute.

If your resource plan does not have any management attributes, then a single parallel statement queue is managed as a FIFO queue.

In the case of an Oracle Real Application Clusters (Oracle RAC) environment, the target number of parallel execution servers is the sum of (PARALLEL_SERVER_LIMIT * PARALLEL_SERVERS_TARGET / 100) across all Oracle RAC instances. If a consumer group is using the number of parallel execution servers computed above or more, then it has exceeded its limit, and its parallel statements will be queued.

If a consumer group does not have any parallel statements running within an Oracle RAC database, then the first parallel statement is allowed to exceed the limit specified by PARALLEL_SERVER_LIMIT.

Note:

In an Oracle Real Application Clusters (Oracle RAC) environment, the PARALLEL_SERVER_LIMIT attribute applies to the entire cluster and not to a single instance.

- Managing Parallel Statement Queuing Using Parallel Server Limit
  The PARALLEL_SERVER_LIMIT attribute enables you to specify when parallel statements from a consumer group can be queued. Oracle Database maintains a separate parallel statement queue for each consumer group.

See Also:

- "Creating Resource Plan Directives "
- "Managing Parallel Statement Queuing Using Parallel Server Limit"
- *Oracle Database VLDB and Partitioning Guide* for information about parallel statement queuing

26.1.3.3.1.3.2.1 Managing Parallel Statement Queuing Using Parallel Server Limit

The PARALLEL_SERVER_LIMIT attribute enables you to specify when parallel statements from a consumer group can be queued. Oracle Database maintains a separate parallel statement queue for each consumer group.

A parallel statement from a consumer group is not run and instead is added to the parallel statement queue of that consumer group if the following conditions are met:

- PARALLEL_DEGREE_POLICY is set to AUTO.

  Setting this initialization parameter to AUTO enables automatic degree of parallelism (Auto DOP), parallel statement queuing, and in-memory parallel execution.

  Note that parallel statements which have PARALLEL_DEGREE_POLICY set to MANUAL or LIMITED are executed immediately and are not added to the parallel statement queue.
- The number of active parallel execution servers across all consumer groups exceeds the PARALLEL_SERVERS_TARGET initialization parameter setting. This condition applies regardless of whether you specify PARALLEL_SERVER_LIMIT. If PARALLEL_SERVER_LIMIT is not specified, then it defaults to 100%.
- The sum of the number of active parallel execution servers for the consumer group and the degree of parallelism of the parallel statement exceeds the target number of active parallel execution servers.

  The target number of active parallel execution servers is computed as follows:

  PARALLEL_SERVER_LIMIT/100 * PARALLEL_SERVERS_TARGET

Note:

Although parallel execution server usage is monitored for all sessions, the parallel execution server directive attributes you set affect only sessions for which parallel statement queuing is enabled (PARALLEL_DEGREE_POLICY is set to AUTO). If a session has the PARALLEL_DEGREE_POLICY set to MANUAL, parallel statements from this session are not queued. However, any parallel execution servers used by such sessions are included in the count that is used to determine the limit for PARALLEL_SERVER_LIMIT. Even if this limit is exceeded, parallel statements from this session are not queued.

See Also:

"Parallel Server Limit"

26.1.3.3.1.3.3 Parallel Queue Timeout

The `PARALLEL_QUEUE_TIMEOUT` directive attribute enables you to specify the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.

When you use parallel statement queuing, if the database does not have sufficient resources to execute a parallel statement, the statement is queued until the required resources become available. However, there is a chance that a parallel statement may be waiting in the parallel statement queue for longer than is desired. You can prevent such scenarios by specifying the maximum time a parallel statement can wait in the parallel statement queue.

The `PARALLEL_QUEUE_TIMEOUT` attribute can be set for each consumer group. This attribute is applicable even if you do not specify other management attributes (`MGMT_P1`, `MGMT_P2`, and so on) in your resource plan.

See Also:

Oracle Database VLDB and Partitioning Guide for more information about parallel statement queuing

Note:

Because the parallel statement queue is clusterwide, all directives related to the parallel statement queue are also clusterwide.

You can control how a timed out parallel statement is handled by setting the `PQ_TIMEOUT_ACTION` attribute for each consumer group. You can set this attribute to the following values:

- `CANCEL` - The statement execution ends with the error `ORA-07454`. This is the default action for a timed out parallel statement.
- `RUN` - The statement runs immediately. If there are not sufficient parallel servers to run the statement immediately, then the statement is downgraded to run at a lower degree of parallelism.

See Also:

"Example of Managing Parallel Statements Using Directive Attributes" for more information about the combined use of all the parallel execution server directive attributes

26.1.3.3.1.4 Program Global Area (PGA)

To manage PGA resources, Resource Manager can limit the amount of PGA memory that can be allocated to each session in a particular consumer group.

To limit the PGA resources for each session in a consumer group, set the `session_pga_limit` parameter in the package procedure `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE`. The value of this parameter is the maximum amount of PGA memory, in megabytes, allowed for each session in the consumer group. If a session exceeds the limit set for its consumer group, then error ORA-10260 is raised. This limit includes parallel query slaves and job queue processes.

For example, poorly written PL/SQL code can consume an unbounded amount of PGA. You can use the `session_pga_limit` parameter to limit sessions that run PL/SQL code to ensure that those sessions do not use an inordinate amount of PGA resource.

The following table illustrates a simple resource plan with PGA limits.

Table 26-3 A Simple Resource Plan with PGA Limits

Consumer Group	session_pga_limit Value
`HIGH_GROUP`	20
`LOW_GROUP`	10
`MAINT_SUBPLAN`	Null (unlimited)
`OTHER_GROUPS`	Null (unlimited)

In this resource plan, high priority applications run within `HIGH_GROUP`, and each session in that group is limited to 20 MB of PGA resource. The sessions used by the lower priority applications within `LOW_GROUP` are limited to 10 MB of PGA resource. The sessions used for maintenance jobs within `MAINT_SUPLAN` and other sessions within `OTHER_GROUPS` can use unlimited PGA resource.

Note:

You can limit the PGA usage of a whole instance with the `PGA_AGGREGATE_LIMIT` initialization parameter.

26.1.3.3.1.5 Runaway Queries

Runaway sessions and calls can adversely impact overall performance if they are not managed properly. Resource Manager can take action when a session or call consumes more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time. Resource Manager can either switch the session or call to a consumer group that is allocated a small amount of CPU or terminate the session or call.

Note:

Starting with Oracle Database 12*c* Release 2 (12.2), Resource Manager can also limit the amount of PGA memory that can be allocated to each session in a particular consumer group.

- Automatic Consumer Group Switching
  You can control resource allocation by specifying criteria that, if met, causes the automatic switching of a session to a specified consumer group.
- Canceling SQL and Terminating Sessions
  You can use the Resource Manager to cancel long-running SQL queries or to terminate long-running sessions based on their amount of consumption of system resources, such as CPU and I/O.
- Execution Time Limit
  You can specify a maximum execution time allowed for an operation.

See Also:

"Program Global Area (PGA)"

26.1.3.3.1.5.1 Automatic Consumer Group Switching

You can control resource allocation by specifying criteria that, if met, causes the automatic switching of a session to a specified consumer group.

Typically, this method is used to switch a session from a high-priority consumer group—one that receives a high proportion of system resources—to a lower priority consumer group because that session exceeded the expected resource consumption for a typical session in the group.

See "Specifying Automatic Switching by Setting Resource Limits" for more information.

26.1.3.3.1.5.2 Canceling SQL and Terminating Sessions

You can use the Resource Manager to cancel long-running SQL queries or to terminate long-running sessions based on their amount of consumption of system resources, such as CPU and I/O.

The SQL queries canceled by the Resource Manager can be configured for quarantine using the`DBMS_SQLQ` package subprograms, so that those queries are not allowed to run again.

See Also:

- "Specifying Automatic Switching by Setting Resource Limits" for more information about how to configure the Resource Manager to cancel SQL queries or to terminate sessions based on their consumption of system resources
- "Quarantine for Execution Plans for SQL Statements Consuming Excessive System Resources" for more information about how to configure quarantine settings for SQL queries using the `DBMS_SQLQ` package subprograms

26.1.3.3.1.5.3 Execution Time Limit

You can specify a maximum execution time allowed for an operation.

If the database estimates that an operation will run longer than the specified maximum execution time, then the operation is terminated with an error. This error can be trapped and the operation rescheduled.

26.1.3.3.1.6 Active Session Pool with Queuing

You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum defines the **active session pool**.

An **active session** is a session that is actively processing a transaction or SQL statement. Specifically, an active session is either in a transaction, holding a user enqueue, or has an open cursor and has not been idle for over 5 seconds. An active session is considered active even if it is blocked, for example waiting for an I/O request to complete. When the active session pool is full, a session that is trying to process a call is placed into a queue. When an active session completes, the first session in the queue can then be removed from the queue and scheduled for execution. You can also specify a period after which a session in the execution queue times out, causing the call to terminate with an error.

Active session limits should not be used for OLTP workloads. In addition, active session limits should not be used to implement connection pooling or parallel statement queuing.

To manage parallel statements, you must use parallel statement queuing with the `PARALLEL_SERVER_LIMIT` attribute and management attributes (`MGMT_P1`, `MGMT_P2`, and so on).

26.1.3.3.1.7 Undo Pool

You can specify an undo pool for each consumer group. An undo pool controls the total amount of undo for uncommitted transactions that can be generated by a consumer group.

When the total undo generated by a consumer group exceeds its undo limit, the current DML statement generating the undo is terminated. No other members of the consumer group can perform further data manipulation until undo space is freed from the pool.

26.1.3.3.1.8 Idle Time Limit

You can specify an amount of time that a session can be idle, after which it is terminated.

You can also specify a more stringent idle time limit that applies to sessions that are idle and blocking other sessions.

26.1.3.3.2 Resource Plan Directives for PDBs

Directives control allocation of CPU and parallel execution servers.

A directive can control the allocation of resources to PDBs based on the share value that you specify for each PDB or PDB performance profile. A higher share value results in more resources. The settings apply to the set of PDBs that use each profile.

For example, you can specify that `salespdb` is allocated double the resources allocated to `hrpdb` by setting the share value for `salespdb` twice as high as the share value for `hrpdb`. Similarly, you can specify that the `salespdb` performance profile is allocated double the resources allocated to the `hrpdb` performance profile by setting the share value for the `salespdb` performance profile twice as high as the share value for the `hrpdb` performance profile.

You can also specify utilization limits for PDBs and PDB performance profiles. The limit controls allocation to the PDB or performance profile. For example, the limit can control how much CPU `salespdb` gets as a percentage of the total CPU available to the CDB.

You can use both shares and utilization limits together for precise control over the resources allocated to each PDB and PDB performance profile in a CDB.

See Also:

Oracle Database SQL Language Reference for more information about PDB lockdown profiles

26.1.3.3.3 Performance Profiles for PDBs

A **PDB performance profile** configures resource plan directives for a set of PDBs that have the same priorities and resource controls.

For example, you might create a performance profiles called Gold, Silver, and Bronze. Each profile specifies a different set of directives depending on the importance of the type of PDB. Gold PDBs are more mission critical than Silver PDBs, which are more mission critical than Bronze PDBs. A PDB specifies its performance profile with the `DB_PERFORMANCE_PROFILE` initialization parameter.

You can use PDB lockdown profiles to specify PDB initialization parameters that control resources, such as `SGA_TARGET` and `PGA_AGGREGATE_LIMIT`. A lockdown profile prevents the PDB administrator from modifying the settings.

Oracle recommends using matching names for performance profiles and lockdown profiles. To prevent PDB owners from switching profiles, Oracle recommends putting the PDB performance profile in the PDB lockdown profile.

26.1.3.4 About Resource Plans

A resource plan is a container for directives that specify how resources are allocated to resource consumer groups.

In addition to the resource plans that are predefined for each Oracle database, you can create any number of resource plans. However, only one resource plan is active at a time. When a resource plan is active, each of its child resource plan directives controls resource allocation for a different consumer group. Each plan must include a directive that allocates resources to the consumer group named `OTHER_GROUPS`. `OTHER_GROUPS` applies to all sessions that belong to a consumer group that is not part of the currently active plan.

Note:

Although the term "resource plan" (or just "plan") denotes one element of the Resource Manager, in this chapter it is also used to refer to a complete *resource plan schema*, which includes the resource plan element itself, its resource plan directives, and the consumer groups that the directives reference. For example, when this chapter refers to the `DAYTIME` resource plan, it could mean either the resource plan element named `DAYTIME`, or the particular resource allocation schema that the `DAYTIME` resource plan and its directives define. Thus, for brevity, it is acceptable to say, "the `DAYTIME` plan favors interactive applications over batch applications."

- About CDB Resource Plans
  Create CDB resource plans that allocate shares and resource limits for PDBs.
- About PDB Resource Plans
  A PDB resource plan determines how the resources for a specific PDB are allocated to consumer groups within this PDB.
- Example: A Simple Resource Plan
  An example illustrates a simple resource plan.

26.1.3.4.1 About CDB Resource Plans

Create CDB resource plans that allocate shares and resource limits for PDBs.

- Shares for Allocating Resources to PDBs
  To allocate resources among PDBs, assign a share value to each PDB or performance profile. A higher share value results in more guaranteed resources for a PDB or the PDBs that use the performance profile.
- Utilization Limits for PDBs
  A utilization limit restrains the system resource usage of a specific PDB or a specific PDB performance profile.
- The Default Directive for PDBs
  When you do not explicitly define directives for a PDB, the PDB uses the default directive for PDBs.

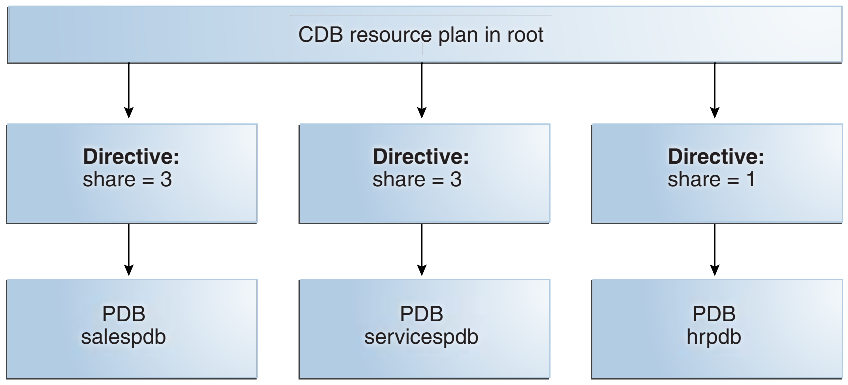
26.1.3.4.1.1 Shares for Allocating Resources to PDBs

To allocate resources among PDBs, assign a share value to each PDB or performance profile. A higher share value results in more guaranteed resources for a PDB or the PDBs that use the performance profile.

Specify a share value for a PDB using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE` procedure and for a PDB performance profile using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE` procedure. In both cases, the `shares` parameter specifies the share value for the PDB. Multiple PDBs can use the same PDB performance profile.

The following figure shows an example of three PDBs with share values specified for them in a CDB resource plan.

Figure 26-1 Shares in a CDB Resource Plan



The preceding figure shows that the total number of shares is seven (3 plus 3 plus 1). The `salespdb` and the `servicespdb` PDB are each guaranteed 3/7 of the resources, while the `hrpdb` PDB is guaranteed 1/7 of the resources. However, any PDB can use more than the guaranteed amount of a resource when no resource contention exists.

The following table shows the resource allocation to the PDBs in the preceding figure based on the share values. The table assumes that loads of the PDBs consume all system resources allocated.

Table 26-4 Resource Allocation for Sample PDBs

Resource	Resource Allocation	See Also
CPU	The `salespdb` and `servicespdb` PDBs can consume the same amount of CPU resources. The `salespdb` and `servicespdb` PDBs are each guaranteed three times more CPU resource than the `hrpdb` PDB.	CPU for more information about this resource
Parallel execution servers	Queued parallel queries from the `salespdb` and `servicespdb` PDBs are selected equally. Queued parallel queries from the `salespdb` and `servicespdb` PDBs are selected three times as often as queued parallel queries from the `hrpdb` PDB.	Degree of Parallelism Limit for more information about this resource

26.1.3.4.1.2 Utilization Limits for PDBs

A utilization limit restrains the system resource usage of a specific PDB or a specific PDB performance profile.

You can specify utilization limits for CPU and parallel execution servers. Utilization limits for a PDB are set by the CDB resource plan.

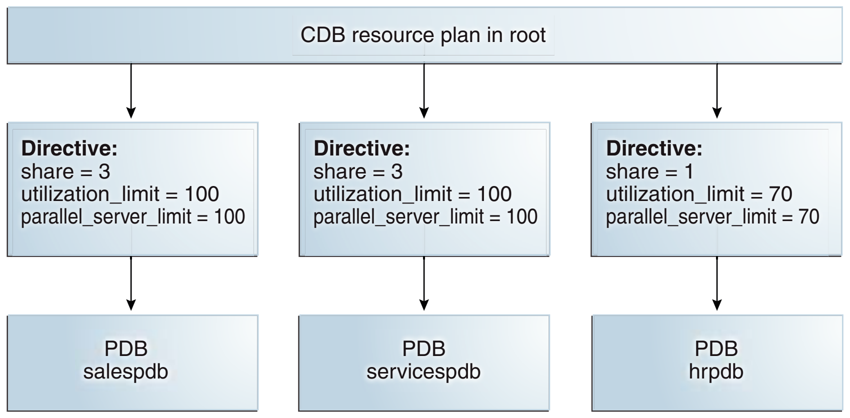
The following table describes utilization limits for PDBs and the Resource Manager action taken when a PDB reaches a utilization limit. For limits specified with a PDB performance profile, the limit applies to every PDB that uses the PDB performance profile. For example, if `pdb1` and `pdb20` have a performance profile `BRONZE`, and if `BRONZE` has a limit set to 10%, then `pdb1` has a 10% limit and `pdb20` has a 10% limit.

Table 26-5 Utilization Limits for PDBs

Resource	Resource Utilization Limit	Resource Manager Action When Limit Is Reached
CPU	The CPU utilization limit for sessions connected to a PDB is set by the `utilization_limit` parameter in subprograms of the `DBMS_RESOURCE_MANAGER` package. The `utilization_limit` parameter specifies the percentage of the system resources that a PDB can use. The value ranges from 0 to 100. You can also limit CPU for a PDB by setting the initialization parameters `CPU_COUNT` (upper limit) and `CPU_MIN_COUNT` (lower limit). For example, if you set `CPU_COUNT` to `8` and `CPU_MIN_COUNT` to `.1` at the PDB level, then the PDB cannot use more than 8 CPU threads at any time and must have at least 1 CPU thread 10% of the time. If both `utilization_limit` and `CPU_COUNT` are specified, then the more restrictive (lower) value is enforced.	Resource Manager throttles the PDB sessions so that the CPU utilization for the PDB does not exceed the utilization limit.
Parallel execution servers	You can limit the number of parallel execution servers in a PDB by means of parallel statement queuing. The limit is a "queuing point" because the database queues parallel queries when the limit is reached. You can set the limit (queuing point) in either of the following ways: • The value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the PDB • The value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the CDB root multiplied by the value of the `parallel_server_limit` directive set for the PDB in the CDB resource manager plan For example, if the `PARALLEL_SERVERS_TARGET` initialization parameter is set to 200 in the CDB root, and if the `parallel_server_limit` directive for a PDB is set to 10%, then utilization limit for the PDB is 20 parallel execution servers (200 * .10). If the limit is set in both preceding ways, then the lower limit of the two is used. See *Oracle Database Reference* for the default value for `PARALLEL_SERVERS_TARGET`. **Note:** Oracle recommends using the `PARALLEL_SERVERS_TARGET` initialization parameter instead of the `parallel_server_limit` directive in a CDB plan.	Resource Manager queues parallel queries when the number of parallel execution servers used by the PDB would exceed the limit. **Note:** In a CDB, parallel statements are queued based on the `PARALLEL_SERVERS_TARGET` settings at both the PDB and CDB level. A statement is queued when the number of parallel servers used by the PDB exceeds the target for the PDB or when the number of parallel servers used by all PDBs exceeds the target for the CDB.

The following figure shows an example of three PDBs with shares and utilization limits specified for them in a CDB resource plan.

Figure 26-2 Shares and Utilization Limits in a CDB Resource Plan



The preceding figure shows that there are no utilization limits on the `salespdb` and `servicespdb` PDBs because `utilization_limit` and `parallel_server_limit` are both set to 100% for them. However, the `hrpdb` PDB is limited to 70% of the applicable system resources because `utilization_limit` and `parallel_server_limit` are both set to 70%.

> **Note:**
>
> This scenario assumes that the `PARALLEL_SERVERS_TARGET` initialization parameter does not specify a lower limit in a PDB. When the `PARALLEL_SERVERS_TARGET` initialization parameter specifies a lower limit for parallel execution servers in a PDB, the lower limit is used.

> **See Also:**
>
> - Parallel Execution Servers
> - *Oracle Database Reference* to learn about `CPU_COUNT`

26.1.3.4.1.3 The Default Directive for PDBs

When you do not explicitly define directives for a PDB, the PDB uses the default directive for PDBs.

The following table shows the attributes of the initial default directive for PDBs.

Table 26-6 Initial Default Directive Attributes for PDBs

Directive Attribute	Value
`shares`	1
`utilization_limit`	100

Table 26-6 (Cont.) Initial Default Directive Attributes for PDBs

Directive Attribute	Value
`parallel_server_limit`	100

When a PDB is plugged into a CDB and no directive is defined for it, the PDB uses the default directive for PDBs.

You can create new directives for the new PDB. You can also change the default directive attribute values for PDBs by using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure in the `DBMS_RESOURCE_MANAGER` package.

When a PDB is unplugged from a CDB, the directive for the PDB is retained. If the same PDB is plugged back into the CDB, then it uses the directive defined for it if the directive was not deleted manually.

Figure 26-3 shows an example of the default directive in a CDB resource plan.

Figure 26-3 Default Directive in a CDB Resource Plan

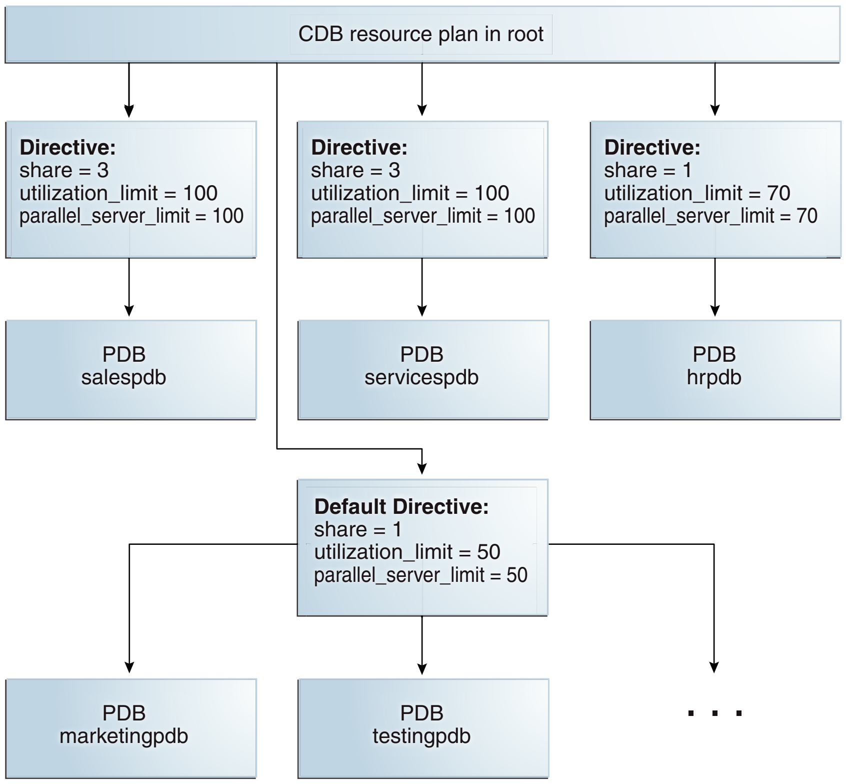


Figure 26-3 shows that the default PDB directive specifies that the `share` is 1, the `utilization_limit` is 50%, and the `parallel_server_limit` is 50%. Any PDB that is part of the CDB and does not have directives defined for it uses the default PDB directive. Figure 26-3 shows the PDBs `marketingpdb` and `testingpdb` using the default PDB directive. Therefore, `marketingpdb` and `testingpdb` each get 1 share and utilization limits of 50.

See Also:

- "Creating New CDB Resource Plan Directives for a PDB"
- "Updating the Default Directive for PDBs in a CDB Resource Plan"
- "Parallel Server Limit"
- *Oracle Multitenant Administrator's Guide* for information about creating and removing PDBs and application containers
- *Oracle Multitenant Administrator's Guide* for information about unplugging a PDB from a CDB

26.1.3.4.2 About PDB Resource Plans

A PDB resource plan determines how the resources for a specific PDB are allocated to consumer groups within this PDB.

A PDB resource plan differs from a CDB resource plan, which determines the amount of resources allocated to each PDB. The following restrictions apply to PDB resource plans:

- A PDB resource plan cannot have subplans.
- A PDB resource plan cannot have a multiple-level scheduling policy.

If you create a PDB by upgrading a non-CDB from a previous release, and if the non-CDB contains resource plans, then these resource plans might not conform to the preceding restrictions. In this case, Oracle Database automatically transforms these resource plans into equivalent PDB resource plans that meet these requirements. The original resource plans and directives are recorded in the `DBA_RSRC_PLANS` and `DBA_RSRC_PLAN_DIRECTIVES` views with the `LEGACY` status.

- CDB Resource Plan Requirements When Creating PDB Resource Plans
  When you create PDB resource plans, the CDB resource plan must meet certain requirements.
- PDB Resource Plan: Example
  A one-to-many relationship exists between CDB resource plans and PDB resource plans.

See Also:

- "About CDB Resource Plans"
- Resources Managed by the Resource Manager

26.1.3.4.2.1 CDB Resource Plan Requirements When Creating PDB Resource Plans

When you create PDB resource plans, the CDB resource plan must meet certain requirements.

Create directives for a CDB resource plan by using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE` procedure. Create directives for a PDB resource plan using the `CREATE_PLAN_DIRECTIVE` procedure in the same package. When you create one or more PDB resource plans and there is no CDB resource plan, the CDB uses the `DEFAULT_CDB_PLAN` that is supplied with Oracle Database.

When the CDB resource plan is set to `DEFAULT_CDB_PLAN` or `DEFAULT_MAINTENANCE_PLAN`, the share value and utilization limit for each PDB is determined as follows:

`share` = `CPU_MIN_COUNT` of the PDB

`utilization_limit` = `CPU_COUNT` of the PDB/`CPU_COUNT` of the CDB

The following table describes the requirements for the CDB resource plan and the results when the requirements are not met. The parameter values described in the "CDB Resource Plan Requirements" column are for the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The parameter values described in the "Results When Requirements Are Not Met" column are for the `CREATE_PLAN_DIRECTIVE` procedure.

Table 26-7   CDB Resource Plan Requirements for PDB Resource Plans

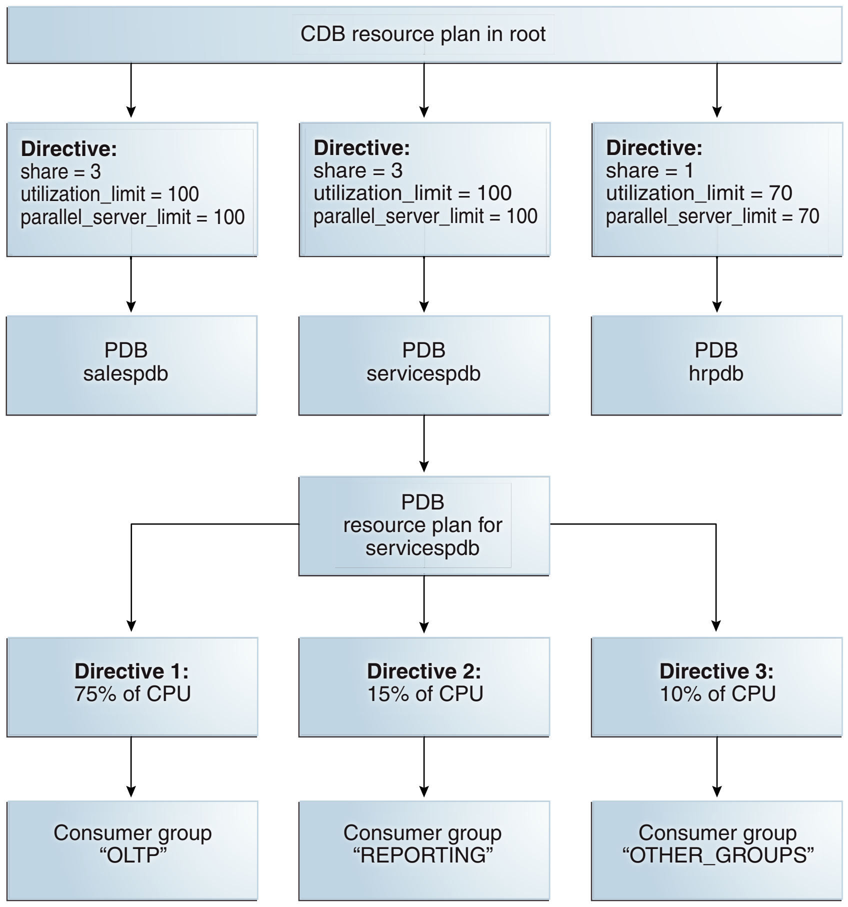
Resource	CDB Resource Plan Requirements	Results When Requirements Are Not Met
CPU	One of the following requirements must be met: • A share value must be specified for the PDB using the `shares` parameter. • A utilization limit for CPU below 100 must be specified for the PDB using the `utilization_limit` parameter. These values can be set in a directive for the specific PDB or in a default directive.	The CPU allocation policy of the PDB resource plan is not enforced. The CPU limit specified by the `utilization_limit` parameter in the PDB resource plan is not enforced.
Parallel execution servers	One of the following requirements must be met: • A share value must be specified for the PDB using the `shares` parameter. • A parallel server limit below 100 must be specified for the PDB using the `parallel_server_limit` parameter. These values can be set in a directive for the specific PDB or in a default directive.	The parallel execution server allocation policy of the PDB resource plan is not enforced. The parallel server limit specified by `parallel_server_limit` in the PDB resource plan is not enforced. However, you can set the `PARALLEL_SERVERS_TARGET` initialization parameter in a PDB to enforce the parallel limit.

26.1.3.4.2.2 PDB Resource Plan: Example

A one-to-many relationship exists between CDB resource plans and PDB resource plans.

The following figure shows an example of a CDB resource plan and a PDB resource plan.

Figure 26-4 A CDB Resource Plan and a PDB Resource Plan


CDB resource plan in root
Directive: share = 3 utilization_limit = 100 parallel_server_limit = 100
Directive: share = 3 utilization_limit = 100 parallel_server_limit = 100
Directive: share = 1 utilization_limit = 70 parallel_server_limit = 70
PDB salespdb
PDB servicespdb
PDB hrpdb
PDB resource plan for servicespdb
Directive 1: 75% of CPU
Directive 2: 15% of CPU
Directive 3: 10% of CPU
Consumer group “OLTP”
Consumer group “REPORTING”
Consumer group “OTHER_GROUPS”

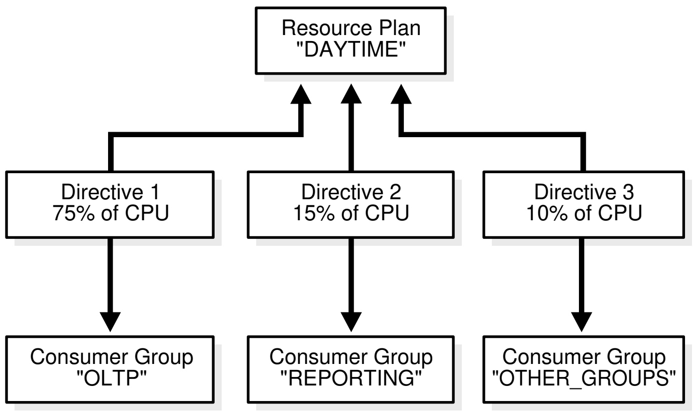
The preceding figure shows some of the directives in a PDB resource plan for the `servicespdb` PDB. Other PDBs in the CDB can also have PDB resource plans.

26.1.3.4.3 Example: A Simple Resource Plan

An example illustrates a simple resource plan.

Figure 26-5 shows a simple resource plan for an organization that runs online transaction processing (OLTP) applications and reporting applications simultaneously during the daytime. The currently active plan, `DAYTIME`, allocates CPU resources among three resource consumer groups. Specifically, `OLTP` is allotted 75% of the CPU time, `REPORTS` is allotted 15%, and `OTHER_GROUPS` receives the remaining 10%. Any group can use more resources than it is guaranteed if there is no resource contention. For example, `OLTP` is guaranteed 75% of the CPU, but if there is no resource contention, it can use up to 100% of the CPU.

Figure 26-5 A Simple Resource Plan



Oracle Database provides a procedure (`CREATE_SIMPLE_PLAN`) that enables you to quickly create a simple resource plan. This procedure is discussed in "Creating a Simple Resource Plan ".

> **Note:**
>
> The currently active resource plan does not enforce allocations until CPU usage is at 100%. If the CPU usage is below 100%, the database is not CPU-bound and hence there is no need to enforce allocations to ensure that all sessions get their designated resource allocation.
>
> In addition, when allocations are enforced, unused allocation by any consumer group can be used by other consumer groups. In the previous example, if the `OLTP` group does not use all of its allocation, the Resource Manager permits the `REPORTS` group or `OTHER_GROUPS` group to use the unused allocation.

26.1.3.5 About Subplans

Instead of referencing a consumer group, a resource plan directive (directive) can reference another resource plan. In this case, the plan is referred to as a subplan.

The subplan itself has directives that allocate resources to consumer groups and other subplans. The resource allocation scheme then works like this: The *top* resource plan (the currently active plan) divides resources among consumer groups and subplans. Each subplan allocates its portion of the total resource allocation among its consumer groups and subplans. You can create hierarchical plans with any number of subplans.

You create a resource subplan in the same way that you create a resource plan. To create a plan that is to be used only as a subplan, you use the `SUB_PLAN` argument in the package procedure `DBMS_RESOURCE_MANAGER.CREATE_PLAN`.

In any top level plan, you can reference a subplan only once. A subplan is not required to have a directive to `OTHER_GROUPS` and cannot be set as a resource plan.

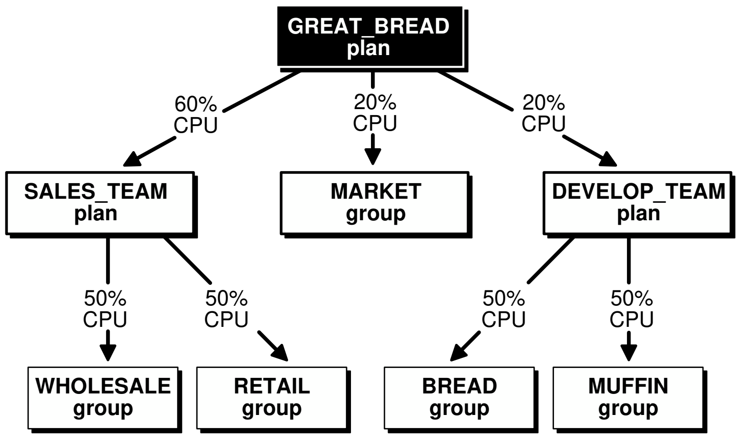
- Example: A Resource Plan with Subplans
  An example illustrates a resource plan with subplans.

26.1.3.5.1 Example: A Resource Plan with Subplans

An example illustrates a resource plan with subplans.

In this example, the Great Bread Company allocates the CPU resource as shown in Figure 26-6. The figure illustrates a top plan (GREAT_BREAD) and all of its descendents. For simplicity, the requirement to include the OTHER_GROUPS consumer group is ignored, and resource plan directives are not shown, even though they are part of the plan. Rather, the CPU percentages that the directives allocate are shown along the connecting lines between plans, subplans, and consumer groups.

Figure 26-6 A Resource Plan With Subplans



The GREAT_BREAD plan allocates resources as follows:

- 20% of CPU resources to the consumer group MARKET
- 60% of CPU resources to subplan SALES_TEAM, which in turn divides its share equally between the WHOLESALE and RETAIL consumer groups
- 20% of CPU resources to subplan DEVELOP_TEAM, which in turn divides its resources equally between the BREAD and MUFFIN consumer groups

It is possible for a subplan or consumer group to have multiple parents. An example would be if the MARKET group were included in the SALES_TEAM subplan. However, a plan cannot contain any loops. For example, the SALES_TEAM subplan cannot have a directive that references the GREAT_BREAD plan.

See Also:

"Putting It All Together: Oracle Database Resource Manager Examples" for an example of a more complex resource plan.

26.1.4 User Interface for PDB Resource Management

You can manage PDB resources using DBMS_RESOURCE_MANAGER and initialization parameters.

- About Resource Manager Administration Privileges
  You must have the required privileges to administer the Resource Manager.
- DBMS_RESOURCE_MANAGER for CDBs and PDBs
  The `DBMS_RESOURCE_MANAGER` package maintains plans, consumer groups, and plan directives for CDBs and PDBs.
- Initialization Parameters for PDB-Level Resources
  Use initialization parameters to control CPU, memory, sessions, and I/O in a PDB.

26.1.4.1 About Resource Manager Administration Privileges

You must have the required privileges to administer the Resource Manager.

You must have the system privilege `ADMINISTER_RESOURCE_MANAGER` to administer the Resource Manager. This privilege (with the `ADMIN` option) is granted to database administrators through the `DBA` role.

Being an administrator for the Resource Manager enables you to execute all of the procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package.

You may, as an administrator with the `ADMIN` option, choose to grant the administrative privilege to other users or roles. To do so, use the `DBMS_RESOURCE_MANAGER_PRIVS` PL/SQL package. The relevant package procedures are listed in the following table.

Procedure	Description
`GRANT_SYSTEM_PRIVILEGE`	Grants the `ADMINISTER_RESOURCE_MANAGER` system privilege to a user or role.
`REVOKE_SYSTEM_PRIVILEGE`	Revokes the `ADMINISTER_RESOURCE_MANAGER` system privilege from a user or role.

The following PL/SQL block grants the administrative privilege to user `HR`, but does not grant `HR` the `ADMIN` option. Therefore, `HR` can execute all of the procedures in the `DBMS_RESOURCE_MANAGER` package, but `HR` cannot use the `GRANT_SYSTEM_PRIVILEGE` procedure to grant the administrative privilege to others.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE(
   GRANTEE_NAME   => 'HR',
   PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER',
   ADMIN_OPTION   => FALSE);
END;
/
```

You can revoke this privilege using the `REVOKE_SYSTEM_PRVILEGE` procedure.

Note:

The `ADMINISTER_RESOURCE_MANAGER` system privilege can only be granted or revoked using the `DBMS_RESOURCE_MANAGER_PRIVS` package. It cannot be granted or revoked through the SQL `GRANT` or `REVOKE` statements.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference.* for information about the `DBMS_RESOURCE_MANAGER` package
- *Oracle Database PL/SQL Packages and Types Reference.* for information about the `DBMS_RESOURCE_MANAGER_PRIVS` package
- *Oracle Database Security Guide* for information about the `ADMIN` option

26.1.4.2 DBMS_RESOURCE_MANAGER for CDBs and PDBs

The `DBMS_RESOURCE_MANAGER` package maintains plans, consumer groups, and plan directives for CDBs and PDBs.

The following table describes the program units related to managing resources in PDBs.

Table 26-8 DBMS_RESOURCE_MANAGER Program Units

PL/SQL Program Unit	Description
`CREATE_CDB_PLAN_DIRECTIVE`	This procedure creates the plan directives of the CDB resource plan. Plan directives specify the resource allocation policy for PDBs.
`CREATE_CDB_PROFILE_DIRECTIVE`	This procedure creates the performance profile directives of the CDB resource plan. The directives specify the resource allocation policy for PDBs that use the performance profile.
`CREATE_CDB_PLAN`	This procedure creates entries that define CDB resource plans.
`UPDATE_CDB_DEFAULT_DIRECTIVE`	This procedure updates the plan directives of the CDB resource plan.
`UPDATE_CDB_PLAN`	This procedure updates the CDB resource plan.

Note:

Oracle Database PL/SQL Packages and Types Reference to learn more about `DBMS_RESOURCE_MANAGER`

26.1.4.3 Initialization Parameters for PDB-Level Resources

Use initialization parameters to control CPU, memory, sessions, and I/O in a PDB.

- CPU-Related Initialization Parameters for PDBs
  The `CPU_COUNT` initialization parameter specifies the number of CPU threads available to a PDB.
- Memory-Related Initialization Parameters for PDBs
  Several initialization parameters control the SGA and PGA usage of a PDB.
- Session-Related Initialization Parameters for PDBs
  Several initialization parameters control how sessions consume resources in a PDB.

- I/O-Related Initialization Parameters for PDBs
  The MAX_IOPS and MAX_MBPS initialization parameters limit the disk I/O generated by a PDB.

26.1.4.3.1 CPU-Related Initialization Parameters for PDBs

The CPU_COUNT initialization parameter specifies the number of CPU threads available to a PDB.

The term **CPU thread** refers to a thread of execution on a CPU core. For example, a server might have 4 CPU sockets. Each CPU in a socket might have 2 cores, making a total of 8 cores. If each CPU core were multithreaded, with 2 threads of execution in each core, then the server would have a total of 16 CPU threads.

If CPU Resource Management is enabled for the CDB, then the PDB-level CPU_MIN_COUNT and CPU_COUNT initialization parameters manage CPU resources for the PDB. CPU Resource Management is enabled, for example, when the DEFAULT_CDB_PLAN is set at the CDB level. The CPU Resource Manager *cages* (restricts) the CPU for the PDB according to the lesser of CPU_COUNT and the PDB-level utilization_limit directive, if it exists. If the CDB resource plan has no shares or utilization limits set in non-default directives, then the CPU Resource Manager uses the PDB-level CPU_MIN_COUNT to set the PDB shares in its CDB resource plan.

Note:

CPU_COUNT and CPU_MIN_COUNT do not specify where the threads must be obtained, that is, on which specific CPU or core.

Table 26-9 Initialization Parameters That Control CPU Usage in PDBs

Initialization Parameter	Description	Default Value at PDB Level
CPU_COUNT	Specifies the maximum number of CPU threads that the PDB can use at one time. If set to a nonzero value, then Oracle Database uses this count rather than the actual number of CPUs, thus disabling dynamic CPU reconfiguration. CPU_COUNT works the same way as the utilization_limit directive in the CDB plan. However, the CPU_COUNT limit is expressed in terms of number of CPU threads rather than utilization percentage. If both the utilization_limit and CPU_COUNT are specified, then the lower limit is enforced. **Note:** When the PDB is plugged into a new container, the CPU_COUNT setting remains with the plugged-in PDB.	CPU_COUNT of the CDB

Table 26-9 (Cont.) Initialization Parameters That Control CPU Usage in PDBs

Initialization Parameter	Description	Default Value at PDB Level
`CPU_MIN_COUNT`	Specifies the minimum number of CPU threads for the PDB. Valid values are `0.1` to the `CPU_COUNT` setting. The value must be a multiple of `0.05`. When less than `1`, the value specifies the minimum percentage of time that the PDB requires a thread. For example, a setting of `0.1` means that over a span of 10 seconds, the PDB requires a CPU thread at least 1 second. The CDB is oversubscribed when the sum of the PDB-level `CPU_MIN_COUNT` settings across all PDBs that are open on a database instance exceeds the CDB-level `CPU_MIN_COUNT` setting. For example, if a single-instance CDB containing 100 PDBs has a `CPU_MIN_COUNT` of `8`, and if each PDB has a `CPU_MIN_COUNT` setting of `.1`, then the CDB is oversubscribed. The minimum CPU is only guaranteed when the CDB is not oversubscribed.	`CPU_COUNT`

26.1.4.3.2 Memory-Related Initialization Parameters for PDBs

Several initialization parameters control the SGA and PGA usage of a PDB.

When the PDB is the current container, the initialization parameters in the following table control the memory usage of the current PDB. When one or more of these parameters is set for a PDB, ensure that the CDB and the other PDBs have sufficient memory for their operations.

The initialization parameters in the following table control the memory usage of PDBs only if the following conditions are met:

- The `NONCDB_COMPATIBLE` initialization parameter is set to `FALSE` in the CDB root.
- The `MEMORY_TARGET` initialization parameter is not set or is set to `0` (zero) in the CDB root.

Table 26-10   Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
`DB_CACHE_SIZE`	Sets the minimum guaranteed buffer cache size for the PDB. This parameter is optional at the PDB level. If the `SGA_TARGET` initialization parameter is not set, and if the `DB_CACHE_SIZE` initialization parameter is set at the CDB level, then the following requirements must be met: • The value of `DB_CACHE_SIZE` set in a PDB must be less than or equal to 50% of the `DB_CACHE_SIZE` value at the CDB level. • The sum of the `DB_CACHE_SIZE` values across all the PDBs in the CDB must be less than or equal to 50% of the `DB_CACHE_SIZE` value at the CDB level. If `SGA_TARGET` is set at the CDB level, then the following requirements must be met to avoid an error: • The values of `DB_CACHE_SIZE` plus `SHARED_POOL_SIZE` in a PDB must be less than or equal to 50% of the PDB's `SGA_TARGET` value. • The values of `DB_CACHE_SIZE` plus `SHARED_POOL_SIZE` in a PDB must be less than or equal to 50% of the `SGA_TARGET` value at the CDB level. • The sum of `DB_CACHE_SIZE` plus `SHARED_POOL_SIZE` across all the PDBs in a CDB must be less than or equal to 50% of the `SGA_TARGET` value at the CDB level.	None

Table 26-10 (Cont.) Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
`SHARED_POOL_SIZE`	Sets the minimum guaranteed shared pool size for the PDB. If the `SGA_TARGET` initialization parameter is not set, but the `SHARED_POOL_SIZE` initialization parameter is set at the CDB level, then the following requirements must be met: • The value of `SHARED_POOL_SIZE` set in a PDB must be less than or equal to 50% of the `SHARED_POOL_SIZE` value at the CDB level. • The sum of the `SHARED_POOL_SIZE` values across all the PDBs in the CDB must be less than or equal to 50% of the `SHARED_POOL_SIZE` value at the CDB level. When the `SGA_TARGET` initialization parameter is set to a nonzero value at the CDB level, the following requirements must be met to avoid an error: • The values of `DB_CACHE_SIZE` plus `SHARED_POOL_SIZE` in a PDB must be less than or equal to 50% of the PDB's `SGA_TARGET` value. • The values of `DB_CACHE_SIZE` plus `SHARED_POOL_SIZE` in a PDB must be less than or equal to 50% of the `SGA_TARGET` value at the CDB level. • The sum of `DB_CACHE_SIZE` plus `SHARED_POOL_SIZE` across all the PDBs in a CDB must be less than or equal to 50% of the `SGA_TARGET` value at the CDB level.	None When this parameter is not set at the PDB level, the PDB has no limit for the amount of shared pool it can use, other than the CDB's shared pool size.
`SGA_MIN_SIZE`	Sets the minimum SGA size for the PDB. The setting must meet the following requirements: • `SGA_TARGET` initialization parameter must *not* be set or must be set to `0` (zero) in the CDB root. Otherwise, setting `SGA_MIN_SIZE` in a PDB has no effect. • It must be less than or equal to 50% of the setting for the `SGA_TARGET` in the CDB root. • It must be less than or equal to 50% of the setting for the `SGA_TARGET` in the PDB. • The sum of the `SGA_MIN_SIZE` settings for all PDBs must be less than or equal to 50% of the setting for the `SGA_TARGET` in the CDB root. Note that setting `SGA_MIN_SIZE` at the CDB level has no effect.	`0`
`SGA_TARGET`	Sets the maximum SGA size for the PDB. The PDB enforces the PDB-level `SGA_TARGET` setting only if the `SGA_TARGET` initialization parameter is set to a nonzero value in the CDB root. The `SGA_TARGET` setting in the PDB must be less than or equal to the `SGA_TARGET` setting in the CDB root.	`SGA_TARGET` at the CDB level multiplied by ratio of PDB-level `CPU_COUNT` / CDB-level `CPU_COUNT`

Table 26-10 (Cont.) Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
`PGA_AGGREGATE_LIMIT`	Sets the maximum PGA size for the PDB. If you set `PGA_AGGREGATE_LIMIT` manually, then the value must meet the following requirements: • It must be less than or equal to the setting for the `PGA_AGGREGATE_LIMIT` in the CDB root. • It must be greater than or equal to two times the setting for the `PGA_AGGREGATE_TARGET` in the PDB.	`PGA_AGGREGATE_LIMIT` at the CDB level multiplied by ratio of PDB-level `CPU_COUNT` / CDB-level `CPU_COUNT`
`PGA_AGGREGATE_TARGET`	Sets the target aggregate PGA size for the PDB. If you set `PGA_AGGREGATE_TARGET` manually, then the value must meet the following requirements: • It must be less than or equal to the `PGA_AGGREGATE_TARGET` value set at the CDB level. • It must be less than or equal to 50% of the `PGA_AGGREGATE_LIMIT` initialization parameter value set at the CDB level. • It must be less than or equal to 50% of the `PGA_AGGREGATE_LIMIT` value set in the PDB.	`PGA_AGGREGATE_TARGET` at the CDB level multiplied by ratio of PDB-level `CPU_COUNT` / CDB-level `CPU_COUNT`

Example 26-1 Setting the Maximum Aggregate PGA Memory Available for a PDB

With the PDB as the current container, run the following SQL statement to set the `PGA_AGGREGATE_LIMIT` initialization parameter both in memory and in the SPFILE to 90 MB:

```
ALTER SYSTEM SET PGA_AGGREGATE_LIMIT = 90M SCOPE = BOTH;
```

Example 26-2 Setting the Minimum SGA Size for a PDB

With the PDB as the current container, run the following SQL statement to set the `SGA_MIN_SIZE` initialization parameter both in memory and in the SPFILE to 500 MB:

```
ALTER SYSTEM SET SGA_MIN_SIZE = 500M SCOPE = BOTH;
```

26.1.4.3.3 Session-Related Initialization Parameters for PDBs

Several initialization parameters control how sessions consume resources in a PDB.

Table 26-11 Initialization Parameters That Control the Session Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
`SESSIONS`	Sets the maximum of number of sessions that a PDB can use. If the PDB tries to use more sessions than configured by its `SESSIONS` parameter, then an `ORA-00018` error message is generated. For PDBs, the `SESSIONS` parameter does not count recursive sessions and therefore does not require the 10% adjustment. The `SESSIONS` parameter for a PDB can only be modified by the PDB. It cannot be set higher than the `SESSIONS` value set at the CDB level.	`SESSIONS` at the CDB level multiplied by ratio of PDB-level `CPU_COUNT` / CDB-level `CPU_COUNT`
`MAX_IDLE_TIME`	Specifies the maximum number of minutes that a session can be idle. After the maximum is reached, Oracle Database automatically terminates the session.	`0` (not set)
`MAX_IDLE_BLOCKER_TIME`	Sets the number of minutes that a session can be idle before it is a candidate for termination. With this parameter, an idle session is terminated if it is blocking another session. Oracle Database considers a session blocked in any of the following situations: • The session is holding a lock needed by another session. • The session is a parallel operation and its consumer group, PDB, or CDB has either reached its maximum parallel server limit or has queued parallel operations. • The PDB or database instance for the session is about to hit its sessions or processes limit. Unlike `MAX_IDLE_TIME`, `MAX_IDLE_BLOCKER_TIME` terminates resources only when they are needed.	`0` (not set)

26.1.4.3.4 I/O-Related Initialization Parameters for PDBs

The `MAX_IOPS` and `MAX_MBPS` initialization parameters limit the disk I/O generated by a PDB.

A large amount of disk I/O can cause poor performance. Several factors can result in excess disk I/O, such as poorly designed SQL or index and table scans in high-volume transactions. If one PDB is generating heavy disk I/O, then it can degrade the performance of other PDBs.

Use one or both of the following initialization parameters to limit the I/O generated by a specific PDB:

- The `MAX_IOPS` initialization parameter limits the number of I/O operations for each second.
- The `MAX_IOPS` initialization parameter limits the megabytes for I/O operations for each second.

If you set both preceding initialization parameters for a single PDB, then Oracle Database enforces both limits. The `MAX_IOPS` and `MAX_IOPS` limits are *not* enforced for Oracle Exadata, which uses I/O Resource Management (IORM) to manage I/Os between PDBs.

If these initialization parameters are set with the CDB root as the current container, then the values become the default values for all containers in the CDB. If they are set with an application root as the current container, then the values become the default values for all application PDBs in the application container. When they are set with a PDB or application

PDB as the current container, then the settings take precedence over the default settings in the CDB root or the application root.

The default for both initialization parameters is 0 (zero). If these initialization parameters are set to 0 (zero) in a PDB, and the CDB root is set to 0, then there is no I/O limit for the PDB. If these initialization parameters are set to 0 (zero) in an application PDB, and its application root is set to 0, then there is no I/O limit for the application PDB.

Critical I/O operations, such as ones for the control file and password file, are exempted from the limit and continue to run even if the limit is reached. However, all I/O operations, including critical I/O operations, are counted when the number of I/O operations and the megabytes for I/O operations are calculated.

You can use the `DBA_HIST_RSRC_PDB_METRIC` view to calculate a reasonable I/O limit for a PDB. Consider the values in the following columns when calculating a limit: `IOPS`, `IOMBPS`, `IOPS_THROTTLE_EXEMPT`, and `IOMBPS_THROTTLE_EXEMPT`. The `rsmgr:io rate limit` wait event indicates that a limit was reached.

Example 26-3 Limiting the I/O Generated by a PDB

With the PDB as the current container, run the following SQL statement to set the `MAX_IOPS` initialization parameter both in memory and in the SPFILE to a limit of 1,000 I/O operations for each second:

```
ALTER SYSTEM SET MAX_IOPS = 1000 SCOPE = BOTH;
```

Example 26-4 Limiting the Megabytes of I/O Generated by a PDB

With the PDB as the current container, run the following SQL statement to set the `MAX_MBPS` initialization parameter both in memory and in the SPFILE to a limit of 200 MB of I/O for each second:

```
ALTER SYSTEM SET MAX_MBPS = 200 SCOPE = BOTH;
```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about modifying a PDB at the system level
- *Oracle Database Reference* for more information about the `MAX_IOPS` initialization parameter
- *Oracle Database Reference* for more information about the `MAX_MBPS` initialization parameter

26.2 Enabling Oracle Database Resource Manager and Switching Plans

You enable Oracle Database Resource Manager (the Resource Manager) by setting the `RESOURCE_MANAGER_PLAN` initialization parameter. This parameter specifies the top plan, identifying the plan to be used for the current instance. If no plan is specified with this parameter, the Resource Manager is not enabled.

By default the Resource Manager is not enabled, except in the following situations:

- During preconfigured maintenance windows, described later in this section.
- When Oracle Database In-Memory is enabled by setting the `INMEMORY_SIZE` initialization parameter to a value greater than 0.

The following statement in a text initialization parameter file activates the Resource Manager upon database startup and sets the top plan as `mydb_plan`.

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

You can also activate or deactivate the Resource Manager, or change the current top plan, using the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure or the `ALTER SYSTEM` statement.

The following SQL statement sets the top plan to `mydb_plan`, and activates the Resource Manager if it is not already active:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'mydb_plan';
```

An error message is returned if the specified plan does not exist in the data dictionary.

Automatic Enabling of the Resource Manager by Oracle Scheduler Windows

The Resource Manager automatically activates if an Oracle Scheduler window that specifies a resource plan opens. When the Scheduler window closes, the resource plan associated with the window is disabled, and the resource plan that was running before the Scheduler window opened is reenabled. (If no resource plan was enabled before the window opened, then the Resource Manager is disabled.) In an Oracle Real Application Clusters environment, a Scheduler window applies to all instances, so the window's resource plan is enabled on every instance.

Note that by default a set of automated maintenance tasks run during **maintenance windows**, which are predefined Scheduler windows that are members of the `MAINTENANCE_WINDOW_GROUP` window group and which specify the `DEFAULT_MAINTENANCE_PLAN` resource plan. Thus, the Resource Manager activates by default during maintenance windows. You can modify these maintenance windows to use a different resource plan, if desired.

Note:

If you change the plan associated with maintenance windows, then ensure that you include the subplan `ORA$AUTOTASK` in the new plan.

See Also:

- "Windows"
- Managing Automated Database Maintenance Tasks

Disabling Plan Switches by Oracle Scheduler Windows

In some cases, the automatic change of Resource Manager plans at Scheduler window boundaries may be undesirable. For example, if you have an important task to finish, and if you set the Resource Manager plan to give your task priority, then you expect that the plan will remain the same until you change it. However, because a Scheduler window could activate after you have set your plan, the Resource Manager plan might change while your task is running.

To prevent this situation, you can set the `RESOURCE_MANAGER_PLAN` initialization parameter to the name of the plan that you want for the system and prepend "`FORCE:`" to the name, as shown in the following SQL statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'FORCE:mydb_plan';
```

Using the prefix `FORCE:` indicates that the current resource plan can be changed only when the database administrator changes the value of the `RESOURCE_MANAGER_PLAN` initialization parameter. This restriction can be lifted by rerunning the command without preceding the plan name with "`FORCE:`".

The `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure has a similar capability.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information on `DBMS_RESOURCE_MANAGER.SWITCH_PLAN`.

Disabling the Resource Manager

To disable the Resource Manager, complete the following steps:

1. Issue the following SQL statement:

   ```
   ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
   ```

2. Disassociate the Resource Manager from all Oracle Scheduler windows.

   To do so, for any Scheduler window that references a resource plan in its `resource_plan` attribute, use the `DBMS_SCHEDULER.SET_ATTRIBUTE` procedure to set `resource_plan` to the empty string (''). Qualify the window name with the `SYS` schema name if you are not logged in as user `SYS`. You can view Scheduler windows with the `DBA_SCHEDULER_WINDOWS` data dictionary view. See "Altering Windows" and *Oracle Database PL/SQL Packages and Types Reference* for more information.

   Note:

   By default, all maintenance windows reference the `DEFAULT_MAINTENANCE_PLAN` resource plan. To completely disable the Resource Manager, you must alter all maintenance windows to remove this plan. However, use caution, because resource consumption by automated maintenance tasks will no longer be regulated, which may adversely affect the performance of your other sessions. See Managing Automated Database Maintenance Tasks for more information on maintenance windows.

26.3 Assigning Sessions to Resource Consumer Groups

There are automatic and manual methods that database administrators, users, and applications can use to assign sessions to resource consumer groups. When a session is assigned to a resource consumer group, Oracle Database Resource Manager (the Resource Manager) can manage resource allocation for it.

> **Note:**
>
> Sessions that are not assigned to a consumer group are placed in the consumer group `OTHER_GROUPS`.

- Overview of Assigning Sessions to Resource Consumer Groups
  Before you enable the Resource Manager, you must specify how user sessions are assigned to resource consumer groups.
- Assigning an Initial Resource Consumer Group
  The initial consumer group of a session is determined by the mapping rules that you configure.
- Specifying Session-to-Consumer Group Mapping Rules
  You can create and prioritize session-to-consumer group mapping rules.
- Switching Resource Consumer Groups
  You can switch the resource consumer group of a session.
- Specifying Automatic Consumer Group Switching
  You can configure the Resource Manager to automatically switch a session to another consumer group when a certain condition is met.
- Granting and Revoking the Switch Privilege
  A user or application must have the switch privilege to switch a session to a specified resource consumer group.

26.3.1 Overview of Assigning Sessions to Resource Consumer Groups

Before you enable the Resource Manager, you must specify how user sessions are assigned to resource consumer groups.

You do this by creating *mapping rules* that enable the Resource Manager to automatically assign each session to a consumer group upon session startup, based upon session attributes. After a session is assigned to its initial consumer group and is running, you can call a procedure to manually switch the session to a different consumer group. You would typically do this if the session is using excessive resources and must be moved to a consumer group that is more limited in its resource allocation. You can also grant the *switch privilege* to users and to applications so that they can switch their sessions from one consumer group to another.

The database can also automatically switch a session from one consumer group to another (typically lower priority) consumer group when there are changes in session attributes or when a session exceeds designated resource consumption limits.

26.3.2 Assigning an Initial Resource Consumer Group

The initial consumer group of a session is determined by the mapping rules that you configure.

For information on how to configure mapping rules, see "Specifying Session-to-Consumer Group Mapping Rules".

26.3.3 Specifying Session-to-Consumer Group Mapping Rules

You can create and prioritize session-to-consumer group mapping rules.

- About Session-to-Consumer Group Mapping Rules
  You can specify the initial consumer group for a session and dynamically switch the session to a different consumer group if the session attributes change.
- Creating Consumer Group Mapping Rules
  You use the `SET_CONSUMER_GROUP_MAPPING` procedure to map a session attribute/value pair to a consumer group.
- Modifying and Deleting Consumer Group Mapping Rules
  To modify a consumer group mapping rule, run the `SET_CONSUMER_GROUP_MAPPING` procedure against the desired attribute/value pair, specifying a new consumer group.
- Creating Mapping Rule Priorities
  To resolve conflicting mapping rules, you can establish a priority ordering of the session attributes from most important to least important.

26.3.3.1 About Session-to-Consumer Group Mapping Rules

You can specify the initial consumer group for a session and dynamically switch the session to a different consumer group if the session attributes change.

By creating session-to-consumer group mapping rules, you can:

- Specify the initial consumer group for a session based on session attributes.
- Enable the Resource Manager to dynamically switch a running session to another consumer group based on changing session attributes.

The mapping rules are based on session attributes such as the user name, the service that the session used to connect to the database, or the name of the client program.

To resolve conflicts among mapping rules, the Resource Manager orders the rules by priority. For example, suppose user `SCOTT` connects to the database with the `SALES` service. If one mapping rule states that user `SCOTT` starts in the `MED_PRIORITY` consumer group, and another states that sessions that connect with the `SALES` service start in the `HIGH_PRIORITY` consumer group, mapping rule priorities resolve this conflict.

There are two types of session attributes upon which mapping rules are based: login attributes and run-time attributes. The login attributes are meaningful only at session login time, when the Resource Manager determines the initial consumer group of the session. Run-time attributes apply any time during and after session login. You can reassign a logged in session to another consumer group by changing any of its run-time attributes.

You use the `SET_CONSUMER_GROUP_MAPPING` and `SET_CONSUMER_GROUP_MAPPING_PRI` procedures to configure the automatic assignment of sessions to consumer groups. You must use a pending area for these procedures. (You must create the pending area, run the procedures, optionally validate the pending area, and then submit the pending area. For examples of using the pending area, see "Creating a Complex Resource Plan".)

A session is automatically switched to a consumer group through mapping rules at distinct points in time:

- When the session first logs in, the mapping rules are evaluated to determine the initial group of the session.
- If a session attribute is dynamically changed to a new value (which is only possible for run-time attributes), then the mapping rules are reevaluated, and the session might be switched to another consumer group.

Predefined Consumer Group Mapping Rules

Each Oracle database comes with a set of predefined consumer group mapping rules:

- As described in "About Resource Consumer Groups", all sessions created by user accounts `SYS` or `SYSTEM` are initially mapped to the `SYS_GROUP` consumer group.
- Sessions performing a data load with Data Pump or performing backup or copy operations with RMAN are automatically mapped to the predefined consumer groups designated in Table 26-19.

You can use the `DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING` procedure to modify or delete any of these predefined mapping rules.

See Also:

- "Assigning an Initial Resource Consumer Group"
- "Specifying Automatic Switching with Mapping Rules"

26.3.3.2 Creating Consumer Group Mapping Rules

You use the `SET_CONSUMER_GROUP_MAPPING` procedure to map a session attribute/value pair to a consumer group.

The parameters for this procedure are the following:

Parameter	Description
`ATTRIBUTE`	The session attribute type, specified as a package constant
`VALUE`	The value of the attribute
`CONSUMER_GROUP`	The consumer group to map to for this attribute/value pair

`ATTRIBUTE` can be one of the following:

Attribute	Type	Description
`ORACLE_USER`	Login	The Oracle Database user name
`SERVICE_NAME`	Login	The database service name used by the client to establish a connection

Attribute	Type	Description
CLIENT_OS_USER	Login	The operating system user name of the client that is logging in
CLIENT_PROGRAM	Login	The name of the client program used to log in to the server
CLIENT_MACHINE	Login	The name of the computer from which the client is making the connection
CLIENT_ID	Login	The client identifier for the session The client identifier session attribute is set by the DBMS_SESSION.SET_IDENTIFIER procedure.
MODULE_NAME	Run-time	The module name in the currently running application as set by the DBMS_APPLICATION_INFO.SET_MODULE procedure or the equivalent OCI attribute setting
MODULE_NAME_ACTION	Run-time	A combination of the current module and the action being performed as set by either of the following procedures or their equivalent OCI attribute setting: • DBMS_APPLICATION_INFO.SET_MODULE • DBMS_APPLICATION_INFO.SET_ACTION The attribute is specified as the module name followed by a period (.), followed by the action name (*module_name.action_name*).
SERVICE_MODULE	Run-time	A combination of service and module names in this form: *service_name.module_name*
SERVICE_MODULE_ACTION	Run-time	A combination of service name, module name, and action name, in this form: *service_name.module_name.action_name*
ORACLE_FUNCTION	Run-time	An RMAN or Data Pump operation. Valid values are DATALOAD, BACKUP, and COPY. There are predefined mappings for each of these values. If your session is performing any of these functions, it is automatically mapped to a predefined consumer group. See Table 26-19 for details.

For example, the following PL/SQL block causes user SCOTT to map to the DEV_GROUP consumer group every time that he logs in:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.ORACLE_USER, 'SCOTT', 'DEV_GROUP');
END;
/
```

Again, you must create a pending area before running the `SET_CONSUMER_GROUP_MAPPING` procedure.

You can use wildcards for the value of most attributes in the `value` parameter in the `SET_CONSUMER_GROUP_MAPPING` procedure. To specify values with wildcards, use the same semantics as the SQL `LIKE` operator. Specifically, wildcards use the following semantics:

- `%` for a multicharacter wildcard
- _ for a single character wildcard
- \ to escape the wildcards

Wildcards can only be used if the attribute is one of the following:

- `CLIENT_OS_USER`
- `CLIENT_PROGRAM`
- `CLIENT_MACHINE`
- `MODULE_NAME`
- `MODULE_NAME_ACTION`
- `SERVICE_MODULE`
- `SERVICE_MODULE_ACTION`

26.3.3.3 Modifying and Deleting Consumer Group Mapping Rules

To modify a consumer group mapping rule, run the `SET_CONSUMER_GROUP_MAPPING` procedure against the desired attribute/value pair, specifying a new consumer group.

To delete a rule, run the `SET_CONSUMER_GROUP_MAPPING` procedure against the desired attribute/value pair and specify a `NULL` consumer group.

26.3.3.4 Creating Mapping Rule Priorities

To resolve conflicting mapping rules, you can establish a priority ordering of the session attributes from most important to least important.

You use the `SET_CONSUMER_GROUP_MAPPING_PRI` procedure to set the priority of each attribute to a unique integer from 1 (most important) to 12 (least important). The following example illustrates this setting of priorities:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI(
    EXPLICIT => 1,
    SERVICE_MODULE_ACTION => 2,
    SERVICE_MODULE => 3,
    MODULE_NAME_ACTION => 4,
    MODULE_NAME => 5,
    SERVICE_NAME => 6,
    ORACLE_USER => 7,
    CLIENT_PROGRAM => 8,
    CLIENT_OS_USER => 9,
    CLIENT_MACHINE => 10,
    CLIENT_ID => 11);
END;
/
```

In this example, the priority of the database user name is set to 7 (less important), while the priority of the module name is set to 5 (more important).

Note:

SET_CONSUMER_GROUP_MAPPING_PRI requires that you include the pseudo-attribute EXPLICIT as an argument. It must be set to 1. It indicates that explicit consumer group switches have the highest priority. You explicitly switch consumer groups with these package procedures, which are described in detail in *Oracle Database PL/SQL Packages and Types Reference*:

- DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP
- DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS
- DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER

To illustrate how mapping rule priorities work, continuing with the previous example, assume that in addition to the mapping of user SCOTT to the DEV_GROUP consumer group, there is also a module name mapping rule as follows:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.MODULE_NAME, 'EOD_REPORTS', 'LOW_PRIORITY');
END;
/
```

Now if the application in user SCOTT's session sets its module name to EOD_REPORTS, the session is reassigned to the LOW_PRIORITY consumer group, because module name mapping has a higher priority than database user mapping.

You can query the view DBA_RSRC_MAPPING_PRIORITY to see the current priority ordering of session attributes.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about setting the module name with the DBMS_APPLICATION_INFO.SET_MODULE procedure
- "Granting and Revoking the Switch Privilege"

26.3.4 Switching Resource Consumer Groups

You can switch the resource consumer group of a session.

- Manually Switching Resource Consumer Groups
  You can change the resource consumer group of running sessions.

- Enabling Users or Applications to Manually Switch Consumer Groups
  You can grant a user the switch privilege so that he can switch his current consumer group using the `SWITCH_CURRENT_CONSUMER_GROUP` procedure in the `DBMS_SESSION` package.

26.3.4.1 Manually Switching Resource Consumer Groups

You can change the resource consumer group of running sessions.

- About Manually Switching Resource Consumer Groups
  The `DBMS_RESOURCE_MANAGER` PL/SQL package provides two procedures that enable you to change the resource consumer group of running sessions.
- Switching a Single Session
  The `SWITCH_CONSUMER_GROUP_FOR_SESS` procedure causes the specified session to immediately be moved into the specified resource consumer group. In effect, this procedure can raise or lower priority of the session.
- Switching All Sessions for a User
  The `SWITCH_CONSUMER_GROUP_FOR_USER` procedure changes the resource consumer group for all sessions pertaining to the specified user name.

26.3.4.1.1 About Manually Switching Resource Consumer Groups

The `DBMS_RESOURCE_MANAGER` PL/SQL package provides two procedures that enable you to change the resource consumer group of running sessions.

Both of these procedures can also change the consumer group of any parallel execution server sessions associated with the coordinator session. The changes made by these procedures pertain to current sessions only; they are not persistent. They also do not change the initial consumer groups for users.

Instead of terminating a session of a user who is using excessive CPU, you can change that user's consumer group to one that is allocated fewer resources.

26.3.4.1.2 Switching a Single Session

The `SWITCH_CONSUMER_GROUP_FOR_SESS` procedure causes the specified session to immediately be moved into the specified resource consumer group. In effect, this procedure can raise or lower priority of the session.

The `SWITCH_CONSUMER_GROUP_FOR_SESS` procedure is Oracle Real Application Clusters (Oracle RAC) instance specific. You must connect to the pluggable database in the same Oracle RAC instance where the session to be switched is running and then run this procedure.

The following PL/SQL block switches a specific session to a new consumer group. The session identifier (`SID`) is 17, the session serial number (`SERIAL#`) is 12345, and the new consumer group is the `HIGH_PRIORITY` consumer group.

```
BEGIN
  DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345',
   'HIGH_PRIORITY');
END;
/
```

The `SID`, session serial number, and current resource consumer group for a session are viewable using the `V$SESSION` view.

See Also:

Oracle Database Reference for details about the V$SESSION view.

26.3.4.1.3 Switching All Sessions for a User

The SWITCH_CONSUMER_GROUP_FOR_USER procedure changes the resource consumer group for all sessions pertaining to the specified user name.

The following PL/SQL block switches all sessions that belong to user HR to the LOW_GROUP consumer group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('HR',
    'LOW_GROUP');
END;
/
```

26.3.4.2 Enabling Users or Applications to Manually Switch Consumer Groups

You can grant a user the switch privilege so that he can switch his current consumer group using the SWITCH_CURRENT_CONSUMER_GROUP procedure in the DBMS_SESSION package.

A user can run this procedure from an interactive session, for example from SQL*Plus, or an application can call this procedure to switch its session, effectively dynamically changing its priority.

The SWITCH_CURRENT_CONSUMER_GROUP procedure enables users to switch to only those consumer groups for which they have the switch privilege. If the caller is another procedure, then this procedure enables users to switch to a consumer group for which the owner of that procedure has switch privileges.

The parameters for this procedure are the following:

Parameter	Description
NEW_CONSUMER_GROUP	The consumer group to which the user is switching.
OLD_CONSUMER_GROUP	Returns the name of the consumer group from which the user switched. Can be used to switch back later.
INITIAL_GROUP_ON_ERROR	Controls behavior if a switching error occurs. If TRUE, in the event of an error, the user is switched to the initial consumer group. If FALSE, raises an error.

The following SQL*Plus session illustrates switching to a new consumer group. By printing the value of the output parameter old_group, the example illustrates how the old consumer group name is saved.

```
SET serveroutput on
DECLARE
    old_group varchar2(30);
BEGIN
  DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('BATCH_GROUP', old_group, FALSE);
  DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
```

```
END;
/
```

The following line is output:

```
OLD GROUP = OLTP_GROUP
```

Note that the Resource Manager considers a switch to have taken place even if the `SWITCH_CURRENT_CONSUMER_GROUP` procedure is called to switch the session to the consumer group that it is already in.

Note:

The Resource Manager also works in environments where a generic database user name is used to log on to an application. The `DBMS_SESSION` package can be called to switch the consumer group assignment of a session at session startup, or as particular modules are called.

See Also:

- "Granting and Revoking the Switch Privilege"
- *Oracle Database PL/SQL Packages and Types Reference* for additional examples and more information about the `DBMS_SESSION` package

26.3.5 Specifying Automatic Consumer Group Switching

You can configure the Resource Manager to automatically switch a session to another consumer group when a certain condition is met.

Automatic switching can occur when: a session attribute changes, causing a new mapping rule to take effect, or a session exceeds the CPU, physical I/O, or logical I/O resource consumption limits set by its consumer group, or it exceeds the elapsed time limit set by its consumer group.

- Specifying Automatic Switching with Mapping Rules
  If a session attribute changes while the session is running, then the session-to-consumer group mapping rules are reevaluated. If a new rule takes effect, then the session might be moved to a different consumer group.
- Specifying Automatic Switching by Setting Resource Limits
  You can manage runaway sessions or calls that use CPU, physical I/O, or logical I/O resources beyond a specified limit. A runaway session is a SQL query, while a runaway call is a PL/SQL call.

26.3.5.1 Specifying Automatic Switching with Mapping Rules

If a session attribute changes while the session is running, then the session-to-consumer group mapping rules are reevaluated. If a new rule takes effect, then the session might be moved to a different consumer group.

See "Specifying Session-to-Consumer Group Mapping Rules" for more information.

26.3.5.2 Specifying Automatic Switching by Setting Resource Limits

You can manage runaway sessions or calls that use CPU, physical I/O, or logical I/O resources beyond a specified limit. A runaway session is a SQL query, while a runaway call is a PL/SQL call.

When you create a resource plan directive for a consumer group, you can specify limits for CPU, physical I/O, or logical I/O resource consumption for sessions in that group. You can specify limits for physical I/O and logical I/O separately. You can also specify a limit for elapsed time. If the `SWITCH_FOR_CALL` resource plan directive is set to `FALSE`, then Resource Manager enforces these limits from the start of the session. If the `SWITCH_FOR_CALL` resource plan directive is set to `TRUE`, then Resource Manager enforces these limits from the start of the SQL operation or PL/SQL block.

You can then specify the action that is to be taken if any single session or call exceeds one of these limits. The possible actions are the following:

- The session is dynamically switched to a designated consumer group.

  The target consumer group is typically one that has lower resource allocations.
- The session is terminated.
- The session's current SQL statement is terminated.
- Information about the session is logged, but no other action is taken for the session.

The following are the resource plan directive attributes that are involved in this type of automatic session switching.

- `SWITCH_GROUP`
- `SWITCH_TIME`
- `SWITCH_ESTIMATE`
- `SWITCH_IO_MEGABYTES`
- `SWITCH_IO_REQS`
- `SWITCH_FOR_CALL`
- `SWITCH_IO_LOGICAL`
- `SWITCH_ELAPSED_TIME`

See "Creating Resource Plan Directives " for descriptions of these attributes.

Switches occur for sessions that are running and consuming resources, not waiting for user input or waiting for CPU cycles. After a session is switched, it continues in the target consumer group until it becomes idle, at which point it is switched back to its original consumer group. However, if `SWITCH_FOR_CALL` is set to `TRUE`, then the Resource Manager does not wait until the session is idle to return it to its original resource consumer group. Instead, the session is returned when the current top-level call completes. A **top-level call** in PL/SQL is an entire PL/SQL block treated as one call. A top-level call in SQL is an individual SQL statement.

`SWITCH_FOR_CALL` is useful for three-tier applications where the middle tier server is using session pooling.

A switched session is allowed to continue running even if the active session pool for the new group is full. Under these conditions, a consumer group can have more sessions running than specified by its active session pool.

When `SWITCH_FOR_CALL` is `FALSE`, the Resource Manager views a session as idle if a certain amount of time passes between calls. This time interval is a few seconds and is not configurable.

The following are examples of automatic switching based on resource limits. You must create a pending area before running these examples.

Example 1

The following PL/SQL block creates a resource plan directive for the `OLTP` group that switches any session in that group to the `LOW_GROUP` consumer group if a call in the sessions exceeds 5 seconds of CPU time. This example prevents unexpectedly long queries from consuming too many resources. The switched-to consumer group is typically one with lower resource allocations.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'LOW_GROUP',
   SWITCH_TIME      => 5);
END;
/
```

Example 2

The following PL/SQL block creates a resource plan directive for the `OLTP` group that temporarily switches any session in that group to the `LOW_GROUP` consumer group if the session exceeds 10,000 physical I/O requests or exceeds 2,500 Megabytes of data transferred. The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                => 'DAYTIME',
   GROUP_OR_SUBPLAN    => 'OLTP',
   COMMENT             => 'OLTP group',
   MGMT_P1             => 75,
   SWITCH_GROUP        => 'LOW_GROUP',
   SWITCH_IO_REQS      => 10000,
   SWITCH_IO_MEGABYTES => 2500,
   SWITCH_FOR_CALL     => TRUE);
END;
/
```

Example 3

The following PL/SQL block creates a resource plan directive for the `REPORTING` group that terminates any session that exceeds 60 seconds of CPU time. This example prevents runaway queries from consuming too many resources.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'REPORTING',
   COMMENT          => 'Reporting group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'KILL_SESSION',
   SWITCH_TIME      => 60);
```

```
END;
/
```

In this example, the reserved consumer group name `KILL_SESSION` is specified for `SWITCH_GROUP`. Therefore, the session is terminated when the switch criteria is met. Other reserved consumer group names are `CANCEL_SQL` and `LOG_ONLY`. When `CANCEL_SQL` is specified, the current call is canceled when switch criteria are met, but the session is not terminated. When `LOG_ONLY` is specified, information about the session is recorded in real-time SQL monitoring, but no specific action is taken for the session.

Example 4

The following PL/SQL block creates a resource plan directive for the `OLTP` group that temporarily switches any session in that group to the `LOW_GROUP` consumer group if the session exceeds 100 logical I/O requests. The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                   => 'DAYTIME',
   GROUP_OR_SUBPLAN       => 'OLTP',
   COMMENT                => 'OLTP group',
   MGMT_P1                => 75,
   SWITCH_GROUP           => 'LOW_GROUP',
   SWITCH_IO_LOGICAL      => 100,
   SWITCH_FOR_CALL        => TRUE);
END;
/
```

Example 5

The following PL/SQL block creates a resource plan directive for the `OLTP` group that temporarily switches any session in that group to the `LOW_GROUP` consumer group if a call in a session exceeds five minutes (300 seconds). The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                   => 'DAYTIME',
   GROUP_OR_SUBPLAN       => 'OLTP',
   COMMENT                => 'OLTP group',
   MGMT_P1                => 75,
   SWITCH_GROUP           => 'LOW_GROUP',
   SWITCH_FOR_CALL        => TRUE,
   SWITCH_ELAPSED_TIME => 300);
END;
/
```

See Also:

- "Creating Resource Plan Directives "
- "What Solutions Does the Resource Manager Provide for Workload Management?" for information about logical I/O

26.3.6 Granting and Revoking the Switch Privilege

A user or application must have the switch privilege to switch a session to a specified resource consumer group.

- About Granting and Revoking the Switch Privilege
  Using the `DBMS_RESOURCE_MANAGER_PRIVS` PL/SQL package, you can grant or revoke the switch privilege to a user, role, or `PUBLIC`. The switch privilege enables a user or application to switch a session to a specified resource consumer group.
- Granting the Switch Privilege
  You can grant a user the privilege to switch to a specific consumer group using the `GRANT_SWITCH_CONSUMER_GROUP` procedure.
- Revoking Switch Privileges
  You can revoke a user’s privilege to switch to a specific consumer group using the `REVOKE_SWITCH_CONSUMER_GROUP` procedure.

26.3.6.1 About Granting and Revoking the Switch Privilege

Using the `DBMS_RESOURCE_MANAGER_PRIVS` PL/SQL package, you can grant or revoke the switch privilege to a user, role, or `PUBLIC`. The switch privilege enables a user or application to switch a session to a specified resource consumer group.

The package also enables you to revoke the switch privilege. The relevant package procedures are listed in the following table.

Procedure	Description
`GRANT_SWITCH_CONSUMER_GROUP`	Grants permission to a user, role, or `PUBLIC` to switch to a specified resource consumer group.
`REVOKE_SWITCH_CONSUMER_GROUP`	Revokes permission for a user, role, or `PUBLIC` to switch to a specified resource consumer group.

`OTHER_GROUPS` has switch privileges granted to `PUBLIC`. Therefore, all users are automatically granted the switch privilege for this consumer group.

The following switches do not require explicit switch privilege:

- There is a consumer group mapping specified by the `SET_CONSUMER_GROUP_MAPPING` procedure in the `DBMS_RESOURCE_MANAGER` package, and a session is switching to a different consumer group due to the mapping. See "Creating Consumer Group Mapping Rules".
- There is an automatic consumer group switch when a switch condition is met based on the setting of the `switch_group` parameter of a resource plan directive.

Explicit switch privilege is required for a user to switch a session to a consumer group in all other cases.

See Also:

- "Enabling Users or Applications to Manually Switch Consumer Groups"
- "Specifying Automatic Consumer Group Switching"

26.3.6.2 Granting the Switch Privilege

You can grant a user the privilege to switch to a specific consumer group using the `GRANT_SWITCH_CONSUMER_GROUP` procedure.

The following example grants user `SCOTT` the privilege to switch to consumer group `OLTP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   GRANTEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP',
   GRANT_OPTION   =>  TRUE);
END;
/
```

User `SCOTT` is also granted permission to grant switch privileges for `OLTP` to others.

If you grant permission to a role to switch to a particular resource consumer group, then any user who is granted that role and has enabled that role can switch his session to that consumer group.

If you grant `PUBLIC` the permission to switch to a particular consumer group, then any user can switch to that group.

If the `GRANT_OPTION` argument is `TRUE`, then users granted switch privilege for the consumer group can also grant switch privileges for that consumer group to others.

26.3.6.3 Revoking Switch Privileges

You can revoke a user’s privilege to switch to a specific consumer group using the `REVOKE_SWITCH_CONSUMER_GROUP` procedure.

The following example revokes user `SCOTT`'s privilege to switch to consumer group `OLTP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   REVOKEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP');
END;
/
```

If you revoke a user's switch privileges for a particular consumer group, any subsequent attempts by that user to switch to that consumer group manually will fail. The user's session will then be automatically assigned to `OTHER_GROUPS`.

If you revoke from a role the switch privileges to a consumer group, any users who had switch privileges for the consumer group only through that role are no longer able to switch to that consumer group.

If you revoke switch privileges to a consumer group from `PUBLIC`, any users other than those who are explicitly assigned switch privileges either directly or through a role are no longer able to switch to that consumer group.

26.4 Managing Resource Plans

Resource Manager allocates resources to pluggable databases (PDBs) in a multitenant container database (CDB).

This chapter assumes that you meet the following prerequisites:

- You understand how to configure and manage a CDB.

> **Note:**
>
> - You can complete the tasks in this chapter using SQL*Plus or Oracle SQL Developer.
> - You can also administer the Resource Manager with the graphical user interface of Oracle Enterprise Manager Cloud Control (Cloud Control).
> - For simplicity, this chapter refers to PDBs, application roots, and application PDBs as “PDBs.”

- Managing CDB Resource Plans
  In a CDB, PDBs might have different levels of priority. You can create CDB resource plans to distribute resources to different PDBs based on these priorities.
- Managing PDB Resource Plans
  You can create, enable, and modify resource plans for individual PDBs.
- Creating a Simple Resource Plan
  You can quickly create a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure.
- Creating a Complex Resource Plan
  When your situation calls for a more complex resource plan, you must create the plan, with its directives and consumer groups, in a staging area called the pending area, and then validate the plan before storing it in the data dictionary.

26.4.1 Managing CDB Resource Plans

In a CDB, PDBs might have different levels of priority. You can create CDB resource plans to distribute resources to different PDBs based on these priorities.

- Creating a CDB Resource Plan for Managing PDBs
  To create a CDB resource plan for individual PDBs and define the directives for the plan, use the `DBMS_RESOURCE_MANAGER` package.
- Creating a CDB Resource Plan for Managing PDBs: Scenario
  This scenario illustrates each of the steps involved in creating a CDB resource plan for individual PDBs.

- Creating a CDB Resource Plan with PDB Performance Profiles
  Use the DBMS_RESOURCE_MANAGER package to create a CDB resource plan for PDB performance profiles and define the directives for the plan. Each PDB that uses a profile adopts the CDB resource plan directive.
- Creating a CDB Resource Plan for PDB Performance Profiles: Scenario
  This scenario illustrates the steps involved in creating a CDB resource plan for PDB performance profiles.
- Enabling a CDB Resource Plan
  You enable the Resource Manager for a CDB by setting the RESOURCE_MANAGER_PLAN initialization parameter in the root.
- Modifying a CDB Resource Plan
  Modifying a CDB resource plan includes tasks such as updating the plan, creating, updating, or deleting plan directives for PDBs, and updating default directives.
- Disabling a CDB Resource Plan
  Disable the Resource Manager for a CDB by unsetting the RESOURCE_MANAGER_PLAN initialization parameter in the CDB root.
- Viewing Information About Plans and Directives in a CDB
  You can view information about CDB resource plans, CDB resource plan directives, and predefined resource plans in a CDB.

26.4.1.1 Creating a CDB Resource Plan for Managing PDBs

To create a CDB resource plan for individual PDBs and define the directives for the plan, use the DBMS_RESOURCE_MANAGER package.

The general steps for creating a CDB resource plan for individual PDBs are the following:

1. Create the pending area using the CREATE_PENDING_AREA procedure.
2. Create the CDB resource plan using the CREATE_CDB_PLAN procedure.
3. Create directives for the PDBs using the CREATE_CDB_PLAN_DIRECTIVE procedure.
4. (Optional) Update the default PDB directive using the UPDATE_CDB_DEFAULT_DIRECTIVE procedure.
5. Validate the pending area using the VALIDATE_PENDING_AREA procedure.
6. Submit the pending area using the SUBMIT_PENDING_AREA procedure.

26.4.1.2 Creating a CDB Resource Plan for Managing PDBs: Scenario

This scenario illustrates each of the steps involved in creating a CDB resource plan for individual PDBs.

The scenario assumes that you want to create a CDB resource plan for a CDB named newcdb. The plan includes a directive for each PDB. In this scenario, you also update the default directive and the AutoTask directive.

The directives are defined using various procedures in the DBMS_RESOURCE_MANAGER package. The attributes of each directive are defined using parameters in these procedures. Table 26-12 describes the types of directives in the plan.

Table 26-12 Attributes for PDB Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`shares`	Resource allocation share for CPU and parallel execution server resources.	"Shares for Allocating Resources to PDBs"
`utilization_limit`	Resource utilization limit for CPU.	"Utilization Limits for PDBs"
`parallel_server_limit`	Maximum percentage of parallel execution servers that a PDB can use before queuing parallel statements. When the `parallel_server_limit` directive is specified for a PDB, the limit is the `PARALLEL_SERVERS_TARGET` value of the CDB root multiplied by the value of the `parallel_server_limit` parameter in the `CREATE_CDB_PLAN_DIRECTIVE` procedure. **Note:** Oracle recommends using the `PARALLEL_SERVERS_TARGET` initialization parameter instead of the `parallel_server_limit` directive in a CDB plan.	"Utilization Limits for PDBs"

Table 26-13 describes how the CDB resource plan allocates resources to its PDBs using the directive attributes described in Table 26-12.

Table 26-13 Sample Directives for PDBs in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
`salespdb`	3	Unlimited	Unlimited
`servicespdb`	3	Unlimited	Unlimited
`hrpdb`	1	70	70
Default	1	50	50
AutoTask	1	75	75

The `salespdb` and `servicespdb` PDBs are more important than the other PDBs in the CDB. Therefore, they get a higher share (3), unlimited CPU utilization resource, and unlimited parallel execution server resource.

The default directive applies to PDBs for which specific directives have not been defined. For this scenario, assume that the CDB has several PDBs that use the default directive. This scenario updates the default directive.

In addition, this scenario updates the AutoTask directive. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

To create a CDB resource plan:

1. Create a pending area using the `CREATE_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

2. Create a CDB resource plan named `newcdb_plan` using the `CREATE_CDB_PLAN` procedure:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
       plan    => 'newcdb_plan',
       comment => 'CDB resource plan for newcdb');
   END;
   /
   ```

3. Create the CDB resource plan directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. Each directive specifies how resources are allocated to a specific PDB.

   Table 26-13 describes the directives for the `salespdb`, `servicespdb`, and `hrpdb` PDBs in this scenario. Run the following procedures to create these directives:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
       plan                  => 'newcdb_plan',
       pluggable_database    => 'salespdb',
       shares                => 3,
       utilization_limit     => 100,
       parallel_server_limit => 100);
   END;
   /

   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
       plan                  => 'newcdb_plan',
       pluggable_database    => 'servicespdb',
       shares                => 3,
       utilization_limit     => 100,
       parallel_server_limit => 100);
   END;
   /

   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
       plan                  => 'newcdb_plan',
       pluggable_database    => 'hrpdb',
       shares                => 1,
       utilization_limit     => 70,
       parallel_server_limit => 70);
   END;
   /
   ```

   All other PDBs in this CDB use the default PDB directive.

4. If the current default CDB resource plan directive for PDBs does not meet your requirements, then update the directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.

   The default directive applies to PDBs for which specific directives have not been defined. See "The Default Directive for PDBs" for more information.

   Table 26-13 describes the default directive that PDBs use in this scenario. Run the following procedure to update the default directive:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
       plan                      => 'newcdb_plan',
       new_shares                => 1,
       new_utilization_limit     => 50,
       new_parallel_server_limit => 50);
   END;
   /
   ```

5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

26.4.1.3 Creating a CDB Resource Plan with PDB Performance Profiles

Use the `DBMS_RESOURCE_MANAGER` package to create a CDB resource plan for PDB performance profiles and define the directives for the plan. Each PDB that uses a profile adopts the CDB resource plan directive.

The general steps for creating a CDB resource plan with PDB performance profiles are the following:

1. Create the pending area using the `CREATE_PENDING_AREA` procedure.
2. Create the CDB resource plan using the `CREATE_CDB_PLAN` procedure.
3. Create directives for the PDB performance profiles using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure.
4. (Optional) Update the default PDB directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.
5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.
7. For each PDB that will use a profile, set the `DB_PERFORMANCE_PROFILE` initialization parameter and specify the profile name.

26.4.1.4 Creating a CDB Resource Plan for PDB Performance Profiles: Scenario

This scenario illustrates the steps involved in creating a CDB resource plan for PDB performance profiles.

The scenario assumes that you want to create a CDB resource plan for a CDB named `newcdb`. The plan includes a directive for each PDB performance profile. In this scenario, you also update the default directive and the AutoTask directive.

In the CDB resource plan, you give each profile a name. In each PDB, you set the `DB_PERFORMANCE_PROFILE` initialization parameter to specify which PDB performance profile the PDB uses.

The directives are defined using various procedures in the `DBMS_RESOURCE_MANAGER` package. The attributes of each directive are defined using parameters in these procedures. The following table describes the types of directives in the plan.

Table 26-14 Attributes for PDB Performance Profile Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`shares`	Resource allocation share for CPU and parallel execution server resources.	"Shares for Allocating Resources to PDBs"
`utilization_limit`	Resource utilization limit for CPU.	"Utilization Limits for PDBs"
`parallel_server_limit`	Maximum percentage of parallel execution servers that a PDB can use. When the `parallel_server_limit` directive is specified for a PDB performance profile, the limit is the value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the CDB root multiplied by the value of the `parallel_server_limit` parameter in the `CREATE_CDB_PROFILE_DIRECTIVE` procedure.	"Utilization Limits for PDBs"

The following table describes how the CDB resource plan allocates resources to its PDB performance profiles using the directive attributes described in Table 26-14.

Table 26-15 Sample Directives for PDB Performance Profiles in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
`gold`	3	Unlimited	Unlimited
`silver`	2	40	40
`bronze`	1	20	20
Default	1	10	10
AutoTask	2	60	60

The default directive applies to PDBs for which specific directives have not been defined. For this scenario, assume that the CDB has several PDBs that use the default directive. This scenario updates the default directive.

In addition, this scenario updates the AutoTask directive. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

To create a CDB resource plan for PDB performance profiles:

1. For each PDB that will use a profile, set the `DB_PERFORMANCE_PROFILE` initialization parameter to the name of the profile that the PDB will use.
   a. Run an `ALTER SYSTEM` statement to set the parameter.

      For example, with the PDB as the current container, run the following SQL statement:

      ```
      ALTER SYSTEM SET DB_PERFORMANCE_PROFILE=gold SCOPE=spfile;
      ```

   b. Close the PDB:

      ```
      ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
      ```

   c. Open the PDB:

      ```
      ALTER PLUGGABLE DATABASE OPEN;
      ```

2. Create a pending area using the `CREATE_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Create a CDB resource plan named `newcdb_plan` using the `CREATE_CDB_PLAN` procedure:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
       plan    => 'newcdb_plan',
       comment => 'CDB resource plan for newcdb');
   END;
   /
   ```

4. Create the CDB resource plan directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. Each directive specifies how resources are allocated to a specific PDB.

   Table 26-13 describes the directives for the `gold`, `silver`, and `bronze` profiles in this scenario. Run the following procedures to create these directives:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
       plan                  => 'newcdb_plan',
       profile               => 'gold',
       shares                => 3,
       utilization_limit     => 100,
       parallel_server_limit => 100);
   END;
   /

   BEGIN
   ```

```
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'silver',
    shares                => 2,
    utilization_limit     => 40,
    parallel_server_limit => 40);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'bronze',
    shares                => 1,
    utilization_limit     => 20,
    parallel_server_limit => 20);
END;
/
```

All other PDBs in this CDB use the default PDB directive.

5. If the current default CDB resource plan directive for PDBs does not meet your requirements, then update the directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.

   The default directive applies to PDBs for which specific directives have not been defined.

   Table 26-13 describes the default directive that PDBs use in this scenario. Run the following procedure to update the default directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
    plan                      => 'newcdb_plan',
    new_shares                => 1,
    new_utilization_limit     => 10,
    new_parallel_server_limit => 10);
END;
/
```

6. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

7. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

"The Default Directive for PDBs"

26.4.1.5 Enabling a CDB Resource Plan

You enable the Resource Manager for a CDB by setting the `RESOURCE_MANAGER_PLAN` initialization parameter in the root.

This parameter specifies the top plan, which is the plan to be used for the current CDB instance. If no plan is specified with this parameter, then the Resource Manager is not enabled.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To enable a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Perform one of the following actions:
   - Use an `ALTER SYSTEM` statement to set the `RESOURCE_MANAGER_PLAN` initialization parameter to the CDB resource plan.

     The following example sets the CDB resource plan to `newcdb_plan` using an `ALTER SYSTEM` statement:

     ```
     ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'newcdb_plan';
     ```

   - In a text initialization parameter file, set the `RESOURCE_MANAGER_PLAN` initialization parameter to the CDB resource plan, and restart the CDB.

     The following example sets the CDB resource plan to `newcdb_plan` in an initialization parameter file:

     ```
     RESOURCE_MANAGER_PLAN = 'newcdb_plan'
     ```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about accessing a container in a CDB
- Oracle Scheduler Concepts to learn how to schedule a CDB resource plan change with Oracle Scheduler

26.4.1.6 Modifying a CDB Resource Plan

Modifying a CDB resource plan includes tasks such as updating the plan, creating, updating, or deleting plan directives for PDBs, and updating default directives.

- Updating a CDB Resource Plan
  You can update a CDB resource plan to change its comment using the `UPDATE_CDB_PLAN` procedure.

- Managing CDB Resource Plan Directives for a PDB
  You can create, update, and delete CDB resource plan directives for a PDB.
- Managing CDB Resource Plan Directives for a PDB Performance Profile
  You can create, update, and delete CDB resource plan directives for a PDB performance profile.
- Updating the Default Directive for PDBs in a CDB Resource Plan
  You can update the default directive for PDBs in a CDB resource plan using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure. The default directive applies to PDBs for which specific directives have not been defined.
- Updating the Default Directive for Maintenance Tasks in a CDB Resource Plan
  You can update the AutoTask directive in a CDB resource plan using the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.
- Deleting a CDB Resource Plan
  You can delete a CDB resource plan using the `DELETE_CDB_PLAN` procedure.

26.4.1.6.1 Updating a CDB Resource Plan

You can update a CDB resource plan to change its comment using the `UPDATE_CDB_PLAN` procedure.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_PLAN` procedure, and enter a new comment in the `new_comment` parameter.

   For example, the following procedure changes the comment for the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN(
       plan        => 'newcdb_plan',
       new_comment => 'CDB plan for PDBs in newcdb');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide*

26.4.1.6.2 Managing CDB Resource Plan Directives for a PDB

You can create, update, and delete CDB resource plan directives for a PDB.

- Creating New CDB Resource Plan Directives for a PDB
  When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the new PDB.
- Updating CDB Resource Plan Directives for a PDB
  You can update the CDB resource plan directive for a PDB using the `UPDATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDB.
- Deleting CDB Resource Plan Directives for a PDB
  You can delete the CDB resource plan directive for a PDB using the `DELETE_CDB_PLAN_DIRECTIVE` procedure.

26.4.1.6.2.1 Creating New CDB Resource Plan Directives for a PDB

When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the new PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To create a new CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `CREATE_CDB_PLAN_DIRECTIVE` procedure, and specify the appropriate values for the new PDB.

For example, the following procedure allocates resources to a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                  => 'newcdb_plan',
    pluggable_database    => 'operpdb',
    shares                => 1,
    utilization_limit     => 20,
    parallel_server_limit => 30);
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide*

26.4.1.6.2.2 Updating CDB Resource Plan Directives for a PDB

You can update the CDB resource plan directive for a PDB using the `UPDATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update a CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `UPDATE_CDB_PLAN_DIRECTIVE` procedure, and specify the new resource allocation values for the PDB.

For example, the following procedure updates the resource allocation to a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN_DIRECTIVE(
    plan                      => 'newcdb_plan',
    pluggable_database        => 'operpdb',
    new_shares                => 1,
    new_utilization_limit     => 10,
    new_parallel_server_limit => 20);
END;
/
```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- *Oracle Multitenant Administrator's Guide*
- "About CDB Resource Plans"

26.4.1.6.2.3 Deleting CDB Resource Plan Directives for a PDB

You can delete the CDB resource plan directive for a PDB using the `DELETE_CDB_PLAN_DIRECTIVE` procedure.

You might delete the directive for a PDB if you unplug or drop the PDB. However, you can retain the directive, and if the PDB is plugged into the CDB in the future, the existing directive applies to the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To delete a CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PLAN_DIRECTIVE` procedure, and specify the CDB resource plan and the PDB.

For example, the following procedure deletes the directive for a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE(
    plan               => 'newcdb_plan',
    pluggable_database => 'operpdb');
END;
/
```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- *Oracle Multitenant Administrator's Guide*
- "About CDB Resource Plans"

26.4.1.6.3 Managing CDB Resource Plan Directives for a PDB Performance Profile

You can create, update, and delete CDB resource plan directives for a PDB performance profile.

- Creating New CDB Resource Plan Directives for a PDB Performance Profile
  You can create a CDB resource plan directive for the a new PDB performance profile using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the all PDBs that use the new profile.
- Updating CDB Resource Plan Directives for a PDB Performance Profile
  Update the CDB resource plan directive for a PDB performance profile using the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDBs that use the PDB performance profile.
- Deleting CDB Resource Plan Directives for a PDB Performance Profile
  You can delete the CDB resource plan directive for a PDB performance profile using the `DELETE_CDB_PROFILE_DIRECTIVE` procedure.

26.4.1.6.3.1 Creating New CDB Resource Plan Directives for a PDB Performance Profile

You can create a CDB resource plan directive for the a new PDB performance profile using the CREATE_CDB_PROFILE_DIRECTIVE procedure. The directive specifies how resources are allocated to the all PDBs that use the new profile.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the DBMS_RESOURCE_MANAGER package, you must have ADMINISTER_RESOURCE_MANAGER system privilege.

To create a new CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the CREATE_CDB_PROFILE_DIRECTIVE procedure, and specify the appropriate values for the new PDB performance profile.

   For example, the following procedure allocates resources to a PDB performance profile named copper in the newcdb_plan CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
       plan                  => 'newcdb_plan',
       profile               => 'copper',
       shares                => 1,
       utilization_limit     => 20,
       parallel_server_limit => 30);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

Note:

For a PDB to use the new profile, the PDB must have the DB_PERFORMANCE_PROFILE initialization parameter set to the profile name.

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB

26.4.1.6.3.2 Updating CDB Resource Plan Directives for a PDB Performance Profile

Update the CDB resource plan directive for a PDB performance profile using the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDBs that use the PDB performance profile.

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update a CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure, and specify the new resource allocation values for the PDB performance profile.

   For example, the following procedure updates the resource allocation for a PDB performance profile named `copper` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_PROFILE_DIRECTIVE(
       plan                      => 'newcdb_plan',
       profile                   => 'copper',
       new_shares                => 1,
       new_utilization_limit     => 10,
       new_parallel_server_limit => 20);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB

26.4.1.6.3.3 Deleting CDB Resource Plan Directives for a PDB Performance Profile

You can delete the CDB resource plan directive for a PDB performance profile using the `DELETE_CDB_PROFILE_DIRECTIVE` procedure.

If no PDBs use a performance profile, then you might delete the directive for the profile.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To delete a CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PROFILE_DIRECTIVE` procedure, and specify the CDB resource plan and the PDB performance profile.

   For example, the following procedure deletes the directive for a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE(
       plan    => 'newcdb_plan',
       profile => 'operpdb');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB
- "About CDB Resource Plans"

26.4.1.6.4 Updating the Default Directive for PDBs in a CDB Resource Plan

You can update the default directive for PDBs in a CDB resource plan using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure. The default directive applies to PDBs for which specific directives have not been defined.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update the default directive for PDBs in a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure, and specify the appropriate default resource allocation values.

   For example, the following procedure updates the default directive for PDBs in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
       plan                  => 'newcdb_plan',
       new_shares            => 2,
       new_utilization_limit => 40);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- "The Default Directive for PDBs"
- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB

26.4.1.6.5 Updating the Default Directive for Maintenance Tasks in a CDB Resource Plan

You can update the AutoTask directive in a CDB resource plan using the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update the AutoTask directive for maintenance tasks in a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure, and specify the appropriate AutoTask resource allocation values.

   For example, the following procedure updates the AutoTask directive for maintenance tasks in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_AUTOTASK_DIRECTIVE(
       plan                  => 'newcdb_plan',
       new_shares            => 2,
       new_utilization_limit => 60);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about accessing containers
- "About CDB Resource Plans"

26.4.1.6.6 Deleting a CDB Resource Plan

You can delete a CDB resource plan using the `DELETE_CDB_PLAN` procedure.

The resource plan must be disabled. You might delete a CDB resource plan if the plan is no longer needed. You can enable a different CDB resource plan, or you can disable Resource Manager for the CDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To delete a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PLAN` procedure, and specify the CDB resource plan.

   For example, the following procedure deletes the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN(
       plan => 'newcdb_plan');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide* for information about accessing containers
- "Enabling a CDB Resource Plan"
- "Disabling a CDB Resource Plan"

26.4.1.7 Disabling a CDB Resource Plan

Disable the Resource Manager for a CDB by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the CDB root.

A CDB resource plan that specifies shares or utilization limits for PDBs is required to enable CPU management, both between PDBs and within a PDB. If a resource plan with shares or utilization limits is enabled for a PDB, and if the CDB resource plan is not specified, then the CDB resource plan is set to `DEFAULT_CDB_PLAN`. This setting gives equal shares to all PDBs and specifies no utilization limits. To disable CPU resource management throughout the CDB, set `RESOURCE_MANAGER_PLAN` to `ORA$INTERNAL_CDB_PLAN`.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To disable a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Perform one of the following actions:
   - Use an `ALTER SYSTEM` statement to unset the `RESOURCE_MANAGER_PLAN` initialization parameter for the CDB.

     The following example unsets the `RESOURCE_MANAGER_PLAN` initialization parameter using an `ALTER SYSTEM` statement:

     ```
     ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
     ```

   - In an initialization parameter file, unset the `RESOURCE_MANAGER_PLAN` initialization parameter, and restart the CDB.

     The following example unsets the `RESOURCE_MANAGER_PLAN` initialization parameter in an initialization parameter file:

     ```
     RESOURCE_MANAGER_PLAN =
     ```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about accessing a container
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide* for information about starting up a database

26.4.1.8 Viewing Information About Plans and Directives in a CDB

You can view information about CDB resource plans, CDB resource plan directives, and predefined resource plans in a CDB.

- Viewing CDB Resource Plans
  An example illustrates using the `DBA_CDB_RSRC_PLANS` view to display all CDB resource plans defined in the CDB.
- Viewing CDB Resource Plan Directives
  An example illustrates using the `DBA_CDB_RSRC_PLAN_DIRECTIVES` view to display all directives defined in all CDB resource plans in the CDB.

See Also:

About Resource Manager Views for information about monitoring Oracle Database Resource Manager

26.4.1.8.1 Viewing CDB Resource Plans

An example illustrates using the `DBA_CDB_RSRC_PLANS` view to display all CDB resource plans defined in the CDB.

The `DEFAULT_CDB_PLAN` is supplied with Oracle Database. You can use this default plan if it meets your requirements.

To view CDB resource plans:

1. Start SQL*Plus or SQL Developer, and log in to the CDB root.
2. Run the following query:

```
COLUMN PLAN FORMAT A30
COLUMN STATUS FORMAT A10
COLUMN COMMENTS FORMAT A35

SELECT PLAN, STATUS, COMMENTS
FROM   DBA_CDB_RSRC_PLANS
ORDER BY PLAN;
```

Your output looks similar to the following:

```
PLAN                      STATUS      COMMENTS
------------------------- ----------- ----------------------------
DEFAULT_CDB_PLAN                      Default CDB plan
DEFAULT_MAINTENANCE_PLAN              Default CDB maintenance plan
NEWCDB_PLAN                           CDB plan for PDBs in newcdb
ORA$INTERNAL_CDB_PLAN                 Internal CDB plan
```

Note:

Plans in the pending area have a status of `PENDING`. Plans in the pending area are being edited. Any plan that is not in the pending area has a `NULL` status.

See Also:

"About CDB Resource Plans"

26.4.1.8.2 Viewing CDB Resource Plan Directives

An example illustrates using the `DBA_CDB_RSRC_PLAN_DIRECTIVES` view to display all directives defined in all CDB resource plans in the CDB.

The `DEFAULT_CDB_PLAN` is a default CDB plan that is supplied with Oracle Database. With `DEFAULT_CDB_PLAN`, every PDB has 1 share and a utilization limit of 100. If the CDB resource plan has no CPU directives configured, that is, the `shares` and `utilization_limits` directives are unset, then CPU Resource Manager uses the PDB-level `CPU_MIN_COUNT` and `CPU_COUNT` parameters to manage CPU. Note that `ORA$DEFAULT_PDB_DIRECTIVE` is the default directive for PDBs.

To view CDB resource plan directives:

1. Start SQL*Plus or SQL Developer, and log in to the CDB root.
2. Run the following query:

```
COLUMN PLAN HEADING 'Plan' FORMAT A24
COLUMN PLUGGABLE_DATABASE HEADING 'Pluggable Database' FORMAT A25
COLUMN SHARES HEADING 'Shares' FORMAT 999
COLUMN UTILIZATION_LIMIT HEADING 'Utilization|Limit' FORMAT 999
COLUMN PARALLEL_SERVER_LIMIT HEADING 'Parallel|Server|Limit' FORMAT 999

SELECT PLAN,
       PLUGGABLE_DATABASE,
       SHARES,
       UTILIZATION_LIMIT,
       PARALLEL_SERVER_LIMIT
  FROM DBA_CDB_RSRC_PLAN_DIRECTIVES
  ORDER BY PLAN;
```

Your output looks similar to the following:

```
                                                                          Parallel
                                                              Utilization   Server
Plan                     Pluggable Database        Shares           Limit    Limit
------------------------ ------------------------- ------ ----------- --------
DEFAULT_CDB_PLAN         ORA$DEFAULT_PDB_DIRECTIVE      1         100      100
DEFAULT_CDB_PLAN         ORA$AUTOTASK                              90      100
DEFAULT_MAINTENANCE_PLAN ORA$AUTOTASK                              90      100
DEFAULT_MAINTENANCE_PLAN ORA$DEFAULT_PDB_DIRECTIVE      1         100      100
NEWCDB_PLAN              HRPDB                          1          70       70
NEWCDB_PLAN              SALESPDB                       3         100      100
NEWCDB_PLAN              ORA$DEFAULT_PDB_DIRECTIVE      1          50       50
NEWCDB_PLAN              ORA$AUTOTASK                   1          75       75
NEWCDB_PLAN              SERVICESPDB                    3         100      100
```

The preceding output shows the directives for the `newcdb_plan` created in "Creating a CDB Resource Plan for Managing PDBs: Scenario" and modified in "Modifying a CDB Resource Plan".

See Also:

- "About CDB Resource Plans"
- "The Default Directive for PDBs"

26.4.2 Managing PDB Resource Plans

You can create, enable, and modify resource plans for individual PDBs.

- Creating a PDB Resource Plan
  You create a PDB resource plan by using procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- Enabling a PDB Resource Plan
  Enable a PDB resource plan by setting the `RESOURCE_MANAGER_PLAN` initialization parameter to the plan with an `ALTER SYSTEM` statement when the current container is the PDB.
- Modifying a PDB Resource Plan
  You can use the `DBMS_RESOURCE_MANAGER` package to modify a PDB resource plan.
- Disabling a PDB Resource Plan
  You disable a PDB resource plan by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the PDB.

26.4.2.1 Creating a PDB Resource Plan

You create a PDB resource plan by using procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package.

A CDB resource plan allocates a portion of the system's resources to a PDB. A PDB resource plan determines how this portion is allocated within the PDB.

The following is a summary of the steps required to create a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Create a pending area using the `CREATE_PENDING_AREA` procedure.
3. Create, modify, or delete consumer groups using the `CREATE_CONSUMER_GROUP` procedure.
4. Map sessions to consumer groups using the `SET_CONSUMER_GROUP_MAPPING` procedure.
5. Create the PDB resource plan using the `CREATE_PLAN` procedure.
6. Create PDB resource plan directives using the `CREATE_PLAN_DIRECTIVE` procedure.
7. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
8. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.

Ensure that the current container is a PDB and that the user has the required privileges when you complete these steps. See "Creating a Complex Resource Plan" for detailed information about completing these steps.

You also have the option of creating a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure. See "Creating a Simple Resource Plan " for information about creating a simple resource plan.

> **Note:**
>
> Some restrictions apply to PDB resource plans. See "About PDB Resource Plans" for information.

26.4.2.2 Enabling a PDB Resource Plan

Enable a PDB resource plan by setting the `RESOURCE_MANAGER_PLAN` initialization parameter to the plan with an `ALTER SYSTEM` statement when the current container is the PDB.

If no plan is specified with this parameter, then no PDB resource plan is enabled for the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To enable a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Use an `ALTER SYSTEM` statement to set the `RESOURCE_MANAGER_PLAN` initialization parameter to the PDB resource plan.

You can also schedule a PDB resource plan change with Oracle Scheduler.

Example 26-5 Enabling a PDB Resource Plan

The following example sets the PDB resource plan to `salespdb_plan`.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'salespdb_plan';
```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about accessing a container
- " Oracle Scheduler Concepts" to learn how to schedule a PDB resource plan change with Oracle Scheduler

26.4.2.3 Modifying a PDB Resource Plan

You can use the `DBMS_RESOURCE_MANAGER` package to modify a PDB resource plan.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To modify a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Modify the PDB resource plan by completing one or more of the following tasks:
   - Update a consumer group using the `UPDATE_CONSUMER_GROUP` procedure.
   - Delete a consumer group using the `DELETE_CONSUMER_GROUP` procedure.
   - Update a resource plan using the `UPDATE_PLAN` procedure.
   - Delete a resource plan using the `DELETE_PLAN` procedure.
   - Update a resource plan directive using the `UPDATE_PLAN_DIRECTIVE` procedure.
   - Delete a resource plan directive using the `DELETE_PLAN_DIRECTIVE` procedure.
4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

See Also:

- "About PDB Resource Plans"
- *Oracle Multitenant Administrator's Guide* for information about accessing a container
- "Maintaining Consumer Groups, Plans, and Directives" for instructions about completing the consumer group tasks

26.4.2.4 Disabling a PDB Resource Plan

You disable a PDB resource plan by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To disable a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Use an `ALTER SYSTEM` statement to unset the `RESOURCE_MANAGER_PLAN` initialization parameter for the PDB.

Example 26-6 Disabling a PDB Resource Plan

The following example disables the PDB resource plan.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

See Also:

- "*Oracle Multitenant Administrator's Guide*" for information about modifying a PDB at the system level
- *Oracle Multitenant Administrator's Guide* for information about accessing a container

26.4.3 Creating a Simple Resource Plan

You can quickly create a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure.

This procedure enables you to both create consumer groups and allocate resources to them by executing a single procedure call. Using this procedure, you are not required to invoke the procedures that are described in succeeding sections for creating a pending area, creating each consumer group individually, specifying resource plan directives, and so on.

You specify the following arguments for the `CREATE_SIMPLE_PLAN` procedure:

Parameter	Description
`SIMPLE_PLAN`	Name of the plan
`CONSUMER_GROUP1`	Consumer group name for first group
`GROUP1_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP2`	Consumer group name for second group
`GROUP2_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP3`	Consumer group name for third group
`GROUP3_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP4`	Consumer group name for fourth group
`GROUP4_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP5`	Consumer group name for fifth group
`GROUP5_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP6`	Consumer group name for sixth group
`GROUP6_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP7`	Consumer group name for seventh group
`GROUP7_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP8`	Consumer group name for eighth group
`GROUP8_PERCENT`	CPU resource allocated to this group

You can specify up to eight consumer groups with this procedure. The only resource allocation method supported is CPU. The plan uses the `EMPHASIS` CPU allocation policy (the default) and each consumer group uses the `ROUND_ROBIN` scheduling policy (also the default).

Example: Creating a Simple Plan with the CREATE_SIMPLE_PLAN Procedure

The following PL/SQL block creates a simple resource plan with two user-specified consumer groups:

```
BEGIN
   DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'SIMPLE_PLAN1',
   CONSUMER_GROUP1 => 'MYGROUP1', GROUP1_PERCENT => 80,
   CONSUMER_GROUP2 => 'MYGROUP2', GROUP2_PERCENT => 20);
END;
/
```

After executing the preceding statements, you can display the plan created using the following query:

```
SELECT plan, group_or_subplan, mgmt_p1
FROM   dba_rsrc_plan_directives
WHERE  plan = 'SIMPLE_PLAN1';
```

The plan created in a non-multitenant environment is:

```
PLAN                 GROUP_OR_SUBPLAN       MGMT_P1
-------------------- -------------------- ----------
SIMPLE_PLAN1         MYGROUP1                     80
SIMPLE_PLAN1         MYGROUP2                     20
SIMPLE_PLAN1         SYS_GROUP                    50
SIMPLE_PLAN1         OTHER_GROUPS                  5
```

The plan created in a multitenant environment is:

```
PLAN                 GROUP_OR_SUBPLAN       MGMT_P1
-------------------- -------------------- ----------
SIMPLE_PLAN1         MYGROUP1                     80
SIMPLE_PLAN1         MYGROUP2                     20
SIMPLE_PLAN1         OTHER_GROUPS                  5
```

See Also:

- "Creating a Resource Plan" for more information on the `EMPHASIS` CPU allocation policy
- "Creating Resource Consumer Groups " for more information on the `ROUND_ROBIN` scheduling policy
- "Elements of the Resource Manager"

26.4.4 Creating a Complex Resource Plan

When your situation calls for a more complex resource plan, you must create the plan, with its directives and consumer groups, in a staging area called the pending area, and then validate the plan before storing it in the data dictionary.

The following is a summary of the steps required to create a complex resource plan.

Note:

A complex resource plan is any resource plan that is not created with the `DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN` procedure.

Step 1: Create a pending area.

Step 2: Create, modify, or delete consumer groups.

Step 3: Map sessions to consumer groups.

Step 4: Create the resource plan.

Step 5: Create resource plan directives.

Step 6: Validate the pending area.

Step 7: Submit the pending area.

You use procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package to complete these steps.

- About the Pending Area
  The **pending area** is a staging area where you can create a new resource plan, update an existing plan, or delete a plan without affecting currently running applications.
- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Creating Resource Consumer Groups
  You create a resource consumer group using the `CREATE_CONSUMER_GROUP` procedure.
- Mapping Sessions to Consumer Groups
  You can map sessions to consumer groups using the `SET_CONSUMER_GROUP_MAPPING` procedure.
- Creating a Resource Plan
  You create a resource plan with the `CREATE_PLAN` procedure.
- Creating Resource Plan Directives
  You use the `CREATE_PLAN_DIRECTIVE` procedure to create resource plan directives. Each directive belongs to a plan or subplan and allocates resources to either a consumer group or subplan.
- Validating the Pending Area
  At any time when you are making changes in the pending area, you can call `VALIDATE_PENDING_AREA` to ensure that the pending area is valid so far.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.
- Clearing the Pending Area
  You can clear the pending area at any time using the `CLEAR_PENDING_AREA` procedure.

See Also:

- Predefined Consumer Group Mapping Rules
- *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- "Elements of the Resource Manager"

26.4.4.1 About the Pending Area

The **pending area** is a staging area where you can create a new resource plan, update an existing plan, or delete a plan without affecting currently running applications.

When you create a pending area, the database initializes it and then copies existing plans into the pending area so that they can be updated.

Tip:

After you create the pending area, if you list all plans by querying the `DBA_RSRC_PLANS` data dictionary view, you see two copies of each plan: one with the `PENDING` status, and one without. The plans with the `PENDING` status reflect any changes you made to the plans since creating the pending area. Pending changes can also be viewed for consumer groups using `DBA_RSRC_CONSUMER_GROUPS` and for resource plan directives using `DBA_RSRC_PLAN_DIRECTIVES`. See Resource Manager Data Dictionary Views for more information.

After you make changes in the pending area, you validate the pending area and then submit it. Upon submission, all pending changes are applied to the data dictionary, and the pending area is cleared and deactivated.

If you attempt to create, update, or delete a plan (or create, update, or delete consumer groups or resource plan directives) without first creating the pending area, you receive an error message.

Submitting the pending area does not activate any new plan that you create; it just stores new or updated plan information in the data dictionary. However, if you modify a plan that is currently active, the plan is reactivated with the new plan definition. See "Enabling Oracle Database Resource Manager and Switching Plans" for information about activating a resource plan.

When you create a pending area, no other users can create one until you submit or clear the pending area or log out.

26.4.4.2 Creating a Pending Area

You create a pending area with the `CREATE_PENDING_AREA` procedure.

Example: Creating a pending area:

The following PL/SQL block creates and initializes a pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
END;
/
```

26.4.4.3 Creating Resource Consumer Groups

You create a resource consumer group using the `CREATE_CONSUMER_GROUP` procedure.

You can specify the following parameters:

Parameter	Description
`CONSUMER_GROUP`	Name to assign to the consumer group.
`COMMENT`	Any comment.
`CPU_MTH`	Deprecated. Use `MGMT_MTH`.

Parameter	Description
MGMT_MTH	The resource allocation method for distributing CPU among sessions in the consumer group. The default is 'ROUND-ROBIN', which uses a round-robin scheduler to ensure that sessions are fairly executed. 'RUN-TO-COMPLETION' specifies that long-running sessions are scheduled ahead of other sessions. This setting helps long-running sessions (such as batch processes) complete sooner.

Example: Creating a Resource Consumer Group

The following PL/SQL block creates a consumer group called OLTP with the default (ROUND-ROBIN) method of allocating resources to sessions in the group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   CONSUMER_GROUP => 'OLTP',
   COMMENT        => 'OLTP applications');
END;
/
```

See Also:

- "Updating a Consumer Group"
- "Deleting a Consumer Group"

26.4.4.4 Mapping Sessions to Consumer Groups

You can map sessions to consumer groups using the SET_CONSUMER_GROUP_MAPPING procedure.

You can specify the following parameters:

Parameter	Description
ATTRIBUTE	Session attribute type, specified as a package constant.
VALUE	Value of the attribute.
CONSUMER_GROUP	Name of the consumer group.

Example: Mapping a Session to a Consumer Group

The following PL/SQL block maps the oe user to the OLTP consumer group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING(
    ATTRIBUTE      => DBMS_RESOURCE_MANAGER.ORACLE_USER,
    VALUE          => 'OE',
    CONSUMER_GROUP => 'OLTP');
END;
/
```

See Also:

"Creating Consumer Group Mapping Rules"

26.4.4.5 Creating a Resource Plan

You create a resource plan with the `CREATE_PLAN` procedure.

You can specify the parameters shown in the following table. The first two parameters are required. The remainder are optional.

Parameter	Description
`PLAN`	Name to assign to the plan.
`COMMENT`	Any descriptive comment.
`CPU_MTH`	Deprecated. Use `MGMT_MTH`.
`ACTIVE_SESS_POOL_MTH`	Active session pool resource allocation method. `ACTIVE_SESS_POOL_ABSOLUTE` is the default and only method available.
`PARALLEL_DEGREE_LIMIT_MTH`	Resource allocation method for specifying a limit on the `PARALLEL_DEGREE_LIMIT_ABSOLUTE` is the default and only method available.
`QUEUEING_MTH`	Queuing resource allocation method. Controls the order in which queued inactive sessions are removed from the queue and added to the active session pool. `FIFO_TIMEOUT` is the default and only method available.
`MGMT_MTH`	Resource allocation method for specifying how much CPU each consumer group or subplan gets. `'EMPHASIS'`, the default method, is for single-level or multilevel plans that use percentages to specify how CPU is distributed among consumer groups. `'RATIO'` is for single-level plans that use ratios to specify how CPU is distributed.
`SUB_PLAN`	If `TRUE`, the plan cannot be used as the top plan; it can be used as a subplan only. Default is `FALSE`.

Example: Creating a Resource Plan

The following PL/SQL block creates a resource plan named `DAYTIME`:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
   PLAN    => 'DAYTIME',
   COMMENT => 'More resources for OLTP applications');
END;
/
```

- About the RATIO CPU Allocation Method
  The `RATIO` method is an alternate CPU allocation method intended for simple plans that have only a single level of CPU allocation.

26.4.4.5.1 About the RATIO CPU Allocation Method

The `RATIO` method is an alternate CPU allocation method intended for simple plans that have only a single level of CPU allocation.

Instead of percentages, you specify numbers corresponding to the ratio of CPU that you want to give to each consumer group. To use the `RATIO` method, you set the `MGMT_MTH` argument for the `CREATE_PLAN` procedure to '`RATIO`'. See "Creating Resource Plan Directives " for an example of a plan that uses this method.

See Also:

- "Updating a Plan"
- "Deleting a Plan"

26.4.4.6 Creating Resource Plan Directives

You use the `CREATE_PLAN_DIRECTIVE` procedure to create resource plan directives. Each directive belongs to a plan or subplan and allocates resources to either a consumer group or subplan.

Note:

The set of directives for a resource plan and its subplans can name a particular subplan only once.

You can specify directives for a particular consumer group in a top plan and its subplans. However, Oracle recommends that the set of directives for a resource plan and its subplans name a particular consumer group only once.

You can specify the following parameters:

Parameter	Description
`PLAN`	Name of the resource plan to which the directive belongs.
`GROUP_OR_SUBPLAN`	Name of the consumer group or subplan to which to allocate resources.
`COMMENT`	Any comment.
`CPU_P1`	Deprecated. Use `MGMT_P1`.
`CPU_P2`	Deprecated. Use `MGMT_P2`.
`CPU_P3`	Deprecated. Use `MGMT_P3`.
`CPU_P4`	Deprecated. Use `MGMT_P4`.
`CPU_P5`	Deprecated. Use `MGMT_P5`.
`CPU_P6`	Deprecated. Use `MGMT_P6`.
`CPU_P7`	Deprecated. Use `MGMT_P7`.
`CPU_P8`	Deprecated. Use `MGMT_P8`.
`ACTIVE_SESS_POOL_P1`	Specifies the maximum number of concurrently active sessions for a consumer group. Other sessions await execution in an inactive session queue. Default is `UNLIMITED`.

Parameter	Description
QUEUEING_P1	Specifies time (in seconds) after which a session in an inactive session queue (waiting for execution) times out and the call is terminated. Default is UNLIMITED.
PARALLEL_DEGREE_LIMIT_P1	Specifies a limit on the degree of parallelism for any operation. Default is UNLIMITED.
SWITCH_GROUP	Specifies the consumer group to which a session is switched if switch criteria are met. If the group name is CANCEL_SQL, then the current call is canceled when switch criteria are met. If the group name is CANCEL_SQL, then the SWITCH_FOR_CALL parameter is always set to TRUE, overriding the user-specified setting. If the group name is KILL_SESSION, then the session is terminated when switch criteria are met. If the group name is LOG_ONLY, then information about the session is recorded in real-time SQL monitoring, but no specific action is taken for the session. If NULL, then the session is not switched and no additional logging is performed. The default is NULL. An error is returned if this parameter is set to NULL and any other switch parameter is set to non-NULL. **Note:** The following consumer group names are reserved: CANCEL_SQL, KILL_SESSION, and LOG_ONLY. An error results if you attempt to create a consumer group with one of these names.
SWITCH_TIME	Specifies the time (in CPU seconds) that a call can execute before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_ESTIMATE	If TRUE, the database estimates the execution time of each call, and if estimated execution time exceeds SWITCH_TIME, the session is switched to the SWITCH_GROUP before beginning the call. Default is FALSE. The execution time estimate is obtained from the optimizer. The accuracy of the estimate is dependent on many factors, especially the quality of the optimizer statistics. In general, you should expect statistics to be no more accurate than ± 10 minutes.
MAX_EST_EXEC_TIME	Specifies the maximum execution time (in CPU seconds) allowed for a call. If the optimizer estimates that a call will take longer than MAX_EST_EXEC_TIME, the call is not allowed to proceed and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is UNLIMITED. The accuracy of the estimate is dependent on many factors, especially the quality of the optimizer statistics.
UNDO_POOL	Sets a maximum in kilobytes (K) on the total amount of undo for uncommitted transactions that can be generated by a consumer group. Default is UNLIMITED.
MAX_IDLE_TIME	Indicates the maximum session idle time, in seconds. Default is NULL, which implies unlimited.
MAX_IDLE_BLOCKER_TIME	Indicates the maximum session idle time of a blocking session, in seconds. Default is NULL, which implies unlimited.
SWITCH_TIME_IN_CALL	Deprecated. Use SWITCH_FOR_CALL.

Parameter	Description
MGMT_P1	For a plan with the MGMT_MTH parameter set to EMPHASIS, specifies the CPU percentage to allocate at the first level. For MGMT_MTH set to RATIO, specifies the weight of CPU usage. Default is NULL for all MGMT_P*n* parameters.
MGMT_P2	For EMPHASIS, specifies CPU percentage to allocate at the second level. Not applicable for RATIO.
MGMT_P3	For EMPHASIS, specifies CPU percentage to allocate at the third level. Not applicable for RATIO.
MGMT_P4	For EMPHASIS, specifies CPU percentage to allocate at the fourth level. Not applicable for RATIO.
MGMT_P5	For EMPHASIS, specifies CPU percentage to allocate at the fifth level. Not applicable for RATIO.
MGMT_P6	For EMPHASIS, specifies CPU percentage to allocate at the sixth level. Not applicable for RATIO.
MGMT_P7	For EMPHASIS, specifies CPU percentage to allocate at the seventh level. Not applicable for RATIO.
MGMT_P8	For EMPHASIS, specifies CPU percentage to allocate at the eighth level. Not applicable for RATIO.
SWITCH_IO_MEGABYTES	Specifies the number of megabytes of physical I/O that a session can transfer (read and write) before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_IO_REQS	Specifies the number of physical I/O requests that a session can execute before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_FOR_CALL	If TRUE, a session that was automatically switched to another consumer group (according to SWITCH_TIME, SWITCH_IO_MEGABYTES, or SWITCH_IO_REQS) is returned to its original consumer group when the top level call completes. Default is NULL.
PARALLEL_QUEUE_TIMEOUT	Specifies the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.
PARALLEL_SERVER_LIMIT	Specifies the maximum percentage of the parallel execution server pool that a particular consumer group can use. The number of parallel execution servers used by a particular consumer group is counted as the sum of the parallel execution servers used by all sessions in that consumer group.
UTILIZATION_LIMIT	Specifies the maximum CPU utilization percentage permitted for the consumer group. This value overrides any level allocations for CPU (MGMT_P1 through MGMT_P8), and also imposes a limit on total CPU utilization when unused allocations are redistributed. You can specify this attribute and leave MGMT_P1 through MGMT_P8 NULL.
SWITCH_IO_LOGICAL	Number of logical I/O requests that will trigger the action specified by SWITCH_GROUP. As with other switch directives, if SWITCH_FOR_CALL is TRUE, then the number of logical I/O requests is accumulated from the start of a call. Otherwise, the number of logical I/O requests is accumulated for the length of the session.

Parameter	Description
SWITCH_ELAPSED_TIME	Elapsed time, in seconds, that will trigger the action specified by SWITCH_GROUP. As with other switch directives, if SWITCH_FOR_CALL is TRUE, then the elapsed time is accumulated from the start of a call. Otherwise, the elapsed time is accumulated for the length of the session.
SHARES	Allocates resources among pluggable databases (PDBs) in a multitenant container database (CDB). Also allocates resources among consumer groups in a PDB.
PARALLEL_STMT_CRITICAL	Specifies whether parallel statements from the consumer group are critical. When BYPASS_QUEUE is specified, parallel statements from the consumer group are critical. These statements bypass the parallel queue and are executed immediately. When FALSE or NULL (the default) is specified, parallel statements from the consumer group are not critical. These statements are added to the parallel queue when necessary.
SESSION_PGA_LIMIT	Specifies the maximum amount of PGA memory, in megabytes, that can be allocated to each session in a particular consumer group. If a session exceeds the limit, then its process is terminated with an ORA-10260 error.

Example 1

The following PL/SQL block creates a resource plan directive for plan DAYTIME. (It assumes that the DAYTIME plan and OLTP consumer group are already created in the pending area.)

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75);
END;
/
```

This directive assigns 75% of CPU resources to the OLTP consumer group at level 1.

You can also create the REPORTING consumer group, and then execute the following PL/SQL block:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                     => 'DAYTIME',
   GROUP_OR_SUBPLAN         => 'REPORTING',
   COMMENT                  => 'Reporting group',
   MGMT_P1                  => 15,
   PARALLEL_DEGREE_LIMIT_P1 => 8,
   ACTIVE_SESS_POOL_P1      => 4,
   SESSION_PGA_LIMIT        => 20);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                     => 'DAYTIME',
   GROUP_OR_SUBPLAN         => 'OTHER_GROUPS',
   COMMENT                  => 'This one is required',
   MGMT_P1                  => 10);
```

```
END;
/
```

In this plan, consumer group REPORTING has a maximum degree of parallelism of 8 for any operation, while none of the other consumer groups are limited in their degree of parallelism. In addition, the REPORTING group has a maximum of 4 concurrently active sessions. Each session can use a maximum of 20 MB of PGA memory.

Example 2

This example uses the RATIO method to allocate CPU, which uses ratios instead of percentages. Suppose your application suite offers three service levels to clients: Gold, Silver, and Bronze. You create three consumer groups named GOLD_CG, SILVER_CG, and BRONZE_CG, and you create the following resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    MGMT_MTH         => 'RATIO',
    COMMENT          => 'Plan that supports three service levels');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'GOLD_CG',
    COMMENT          => 'Gold service level customers',
    MGMT_P1          => 10);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'SILVER_CG',
    COMMENT          => 'Silver service level customers',
    MGMT_P1          => 5);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'BRONZE_CG',
    COMMENT          => 'Bronze service level customers',
    MGMT_P1          => 2);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
    (PLAN            => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT          => 'Lowest priority sessions',
    MGMT_P1          => 1);
END;
/
```

The ratio of CPU allocation is 10:5:2:1 for the GOLD_CG, SILVER_CG, BRONZE_CG, and OTHER_GROUPS consumer groups, respectively.

If sessions exist only in the GOLD_CG and SILVER_CG consumer groups, then the ratio of CPU allocation is 10:5 between the two groups.

- Conflicting Resource Plan Directives
  Although this is allowed, Oracle strongly recommends that you avoid referencing the same consumer group from a top plan and any of its subplans.

26.4.4.6.1 Conflicting Resource Plan Directives

Although this is allowed, Oracle strongly recommends that you avoid referencing the same consumer group from a top plan and any of its subplans.

You may have occasion to reference the same consumer group from the top plan and any number of subplans. This results in multiple resource plan directives referring to the same consumer group.

Similarly, when multiple resource plan directives refer to the same consumer group, they have conflicting directives. Although this is allowed, Oracle strongly recommends that you avoid multiple resource plan directives that refer to the same consumer group.

See Also:

- "Updating a Resource Plan Directive"
- "Deleting a Resource Plan Directive"

26.4.4.7 Validating the Pending Area

At any time when you are making changes in the pending area, you can call `VALIDATE_PENDING_AREA` to ensure that the pending area is valid so far.

The following rules must be adhered to, and are checked by the validate procedure:

- No plan can contain any loops. A loop occurs when a subplan contains a directive that references a plan that is above the subplan in the plan hierarchy. For example, a subplan cannot reference the top plan.
- All plans and resource consumer groups referred to by plan directives must exist.
- All plans must have plan directives that point to either plans or resource consumer groups.
- All percentages in any given level must not add up to greater than 100.
- A plan that is currently being used as a top plan by an active instance cannot be deleted.
- The following parameters can appear only in plan directives that refer to resource consumer groups, not other resource plans:
  - `ACTIVE_SESS_POOL_P1`
  - `MAX_EST_EXEC_TIME`
  - `MAX_IDLE_BLOCKER_TIME`
  - `MAX_IDLE_TIME`
  - `PARALLEL_DEGREE_LIMIT_P1`
  - `QUEUEING_P1`
  - `SESSION_PGA_LIMIT`
  - `SWITCH_ESTIMATE`
  - `SWITCH_FOR_CALL`
  - `SWITCH_GROUP`
  - `SWITCH_IO_MEGABYTES`
  - `SWITCH_IO_REQS`
  - `SWITCH_TIME`

  - `UNDO_POOL`
  - `UTILIZATION_LIMIT`
- There can be no more than 28 resource consumer groups in any active plan. Also, at most, a plan can have 28 children.
- Plans and resource consumer groups cannot have the same name.
- There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan. This ensures that a session that is not part of any of the consumer groups included in the currently active plan is allocated resources (as specified by the directive for `OTHER_GROUPS`).

`VALIDATE_PENDING_AREA` raises an error if any of the preceding rules are violated. You can then make changes to fix any problems and call the procedure again.

It is possible to create "orphan" consumer groups that have no plan directives referring to them. This allows the creation of consumer groups that will not currently be used, but might be part of some plan to be implemented in the future.

Example: Validating the Pending Area:

The following PL/SQL block validates the pending area.

```
BEGIN
  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
END;
/
```

See Also:

"About the Pending Area"

26.4.4.8 Submitting the Pending Area

After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

The submit procedure also performs validation, so you do not necessarily need to make separate calls to the validate procedure. However, if you are making major changes to plans, debugging problems is often easier if you incrementally validate your changes. No changes are submitted (made active) until validation is successful on all of the changes in the pending area.

The `SUBMIT_PENDING_AREA` procedure clears (deactivates) the pending area after successfully validating and committing the changes.

Note:

A call to `SUBMIT_PENDING_AREA` might fail even if `VALIDATE_PENDING_AREA` succeeds. This can happen if, for example, a plan being deleted is loaded by an instance after a call to `VALIDATE_PENDING_AREA`, but before a call to `SUBMIT_PENDING_AREA`.

Example: Submitting the Pending Area:

The following PL/SQL block submits the pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

See Also:

"About the Pending Area"

26.4.4.9 Clearing the Pending Area

You can clear the pending area at any time using the `CLEAR_PENDING_AREA` procedure.

This PL/SQL block causes all of your changes to be cleared from the pending area and deactivates the pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
END;
/
```

After calling `CLEAR_PENDING_AREA`, you must call the `CREATE_PENDING_AREA` procedure before you can again attempt to make changes.

See Also:

"About the Pending Area"

26.5 Putting It All Together: Oracle Database Resource Manager Examples

Examples illustrate how to allocate resources with Resource Manager.

- Multilevel Plan Example
  An example illustrates a multilevel plan.
- Examples of Using the Utilization Limit Attribute
  You can use the `UTILIZATION_LIMIT` directive attribute to limit the CPU utilization for applications. One of the most common scenarios in which this attribute can be used is for database consolidation.
- Example of Using Several Resource Allocation Methods
  An example illustrates using several resource allocation methods.
- Example of Managing Parallel Statements Using Directive Attributes
  An example illustrates managing parallel statements using directive attributes.

- An Oracle-Supplied Mixed Workload Plan
  Oracle Database includes a predefined resource plan, `MIXED_WORKLOAD_PLAN`, that prioritizes interactive operations over batch operations, and includes the required subplans and consumer groups recommended by Oracle.

26.5.1 Multilevel Plan Example

An example illustrates a multilevel plan.

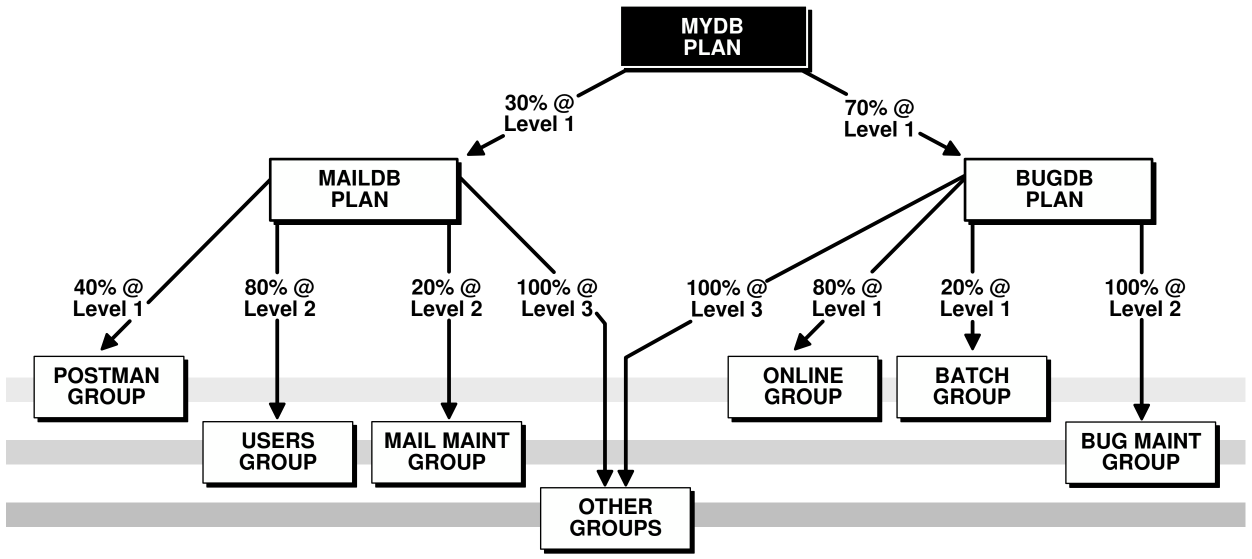
The following PL/SQL block creates a multilevel plan as illustrated in Figure 26-7. Default resource allocation method settings are used for all plans and resource consumer groups.

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
   COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
   COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
   COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Batch_group',
   COMMENT => 'Resource consumer group/method for batch job bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for bug db maint');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Postman_group',
   COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for mail db maint');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Online_group',
   COMMENT => 'online bug users sessions at level 1', MGMT_P1 => 80, MGMT_P2=> 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Batch_group',
   COMMENT => 'batch bug users sessions at level 1', MGMT_P1 => 20, MGMT_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Maint_group',
   COMMENT => 'bug maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Postman_group',
   COMMENT => 'mail postman at level 1', MGMT_P1 => 40, MGMT_P2 => 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Users_group',
   COMMENT => 'mail users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 80);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_Maint_group',
   COMMENT => 'mail maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'maildb_plan',
   COMMENT=> 'all mail users sessions at level 1', MGMT_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', MGMT_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

```
END;
/
```

The preceding call to `VALIDATE_PENDING_AREA` is optional because the validation is implicitly performed in `SUBMIT_PENDING_AREA`.

Figure 26-7 Multilevel Plan Schema



In this plan schema, CPU resources are allocated as follows:

- Under `mydb_plan`, 30% of CPU is allocated to the `maildb_plan` subplan, and 70% is allocated to the `bugdb_plan` subplan. Both subplans are at level 1. Because `mydb_plan` itself has no levels below level 1, any resource allocations that are unused by either subplan at level 1 can be used by its sibling subplan. Thus, if `maildb_plan` uses only 20% of CPU, then 80% of CPU is available to `bugdb_plan`.
- `maildb_plan` and `bugdb_plan` define allocations at levels 1, 2, and 3. The levels in these subplans are independent of levels in their parent plan, `mydb_plan`. That is, all plans and subplans in a plan schema have their own level 1, level 2, level 3, and so on.
- Of the 30% of CPU allocated to `maildb_plan`, 40% of that amount (effectively 12% of total CPU) is allocated to `Postman_group` at level 1. Because `Postman_group` has no siblings at level 1, there is an implied 60% remaining at level 1. This 60% is then shared by `Users_group` and `Mail_Maint_group` at level 2, at 80% and 20%, respectively. In addition to this 60%, `Users_group` and `Mail_Maint_group` can also use any of the 40% not used by `Postman_group` at level 1.
- CPU resources not used by either `Users_group` or `Mail_Maint_group` at level 2 are allocated to `OTHER_GROUPS`, because in multilevel plans, unused resources are reallocated to consumer groups or subplans at the next lower level, not to siblings at the same level. Thus, if `Users_group` uses only 70% instead of 80%, the remaining 10% cannot be used by `Mail_Maint_group`. That 10% is available only to `OTHER_GROUPS` at level 3.
- The 70% of CPU allocated to the `bugdb_plan` subplan is allocated to its consumer groups in a similar fashion. If either `Online_group` or `Batch_group` does not use its full allocation, the remainder may be used by `Bug_Maint_group`. If `Bug_Maint_group` does not use all of that allocation, the remainder goes to `OTHER_GROUPS`.

26.5.2 Examples of Using the Utilization Limit Attribute

You can use the `UTILIZATION_LIMIT` directive attribute to limit the CPU utilization for applications. One of the most common scenarios in which this attribute can be used is for database consolidation.

During database consolidation, you may need to be able to do the following:

- Manage the performance impact that one application can have on another.

  One method of managing this performance impact is to create a consumer group for each application and allocate resources to each consumer group.
- Limit the utilization of each application.

  Typically, in addition to allocating a specific percentage of the CPU resources to each consumer group, you may need to limit the maximum CPU utilization for each group. This limit prevents a consumer group from using all of the CPU resources when all the other consumer groups are idle.

  In some cases, you may want all application users to experience consistent performance regardless of the workload from other applications. This can be achieved by specifying a utilization limit for each consumer group in a resource plan.

The following examples demonstrate how to use the `UTILIZATION_LIMIT` resource plan directive attribute to:

- Restrict total database CPU utilization
- Quarantine runaway queries
- Limit CPU usage for applications
- Limit CPU utilization during maintenance windows

Example 1 - Restricting Overall Database CPU Utilization

In this example, regardless of database load, system workload from Oracle Database never exceeds 90% of CPU, leaving 10% of CPU for other applications sharing the server.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'MAXCAP_PLAN',
    COMMENT => 'Limit overall database CPU');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN              => 'MAXCAP_PLAN',
    GROUP_OR_SUBPLAN  => 'OTHER_GROUPS',
    COMMENT           => 'This group is mandatory',
    UTILIZATION_LIMIT => 90);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Because there is no plan directive other than the one for `OTHER_GROUPS`, all sessions are mapped to `OTHER_GROUPS`.

Example 2 - Quarantining Runaway Queries

In this example, runaway queries are switched to a consumer group with a utilization limit of 20%, limiting the amount of resources that they can consume until you can intervene. A runaway query is characterized here as one that takes more than 10 minutes of CPU time. Assume that session mapping rules start all sessions in `START_GROUP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'START_GROUP',
     COMMENT        => 'Sessions start here');

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'QUARANTINE_GROUP',
     COMMENT        => 'Sessions switched here to quarantine them');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'Quarantine_plan',
    COMMENT => 'Quarantine runaway queries');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                     => 'Quarantine_plan',
    GROUP_OR_SUBPLAN         => 'START_GROUP',
    COMMENT                  => 'Max CPU 10 minutes before switch',
    MGMT_P1                  => 75,
    switch_group             => 'QUARANTINE_GROUP',
    switch_time              => 600);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                     => 'Quarantine_plan',
    GROUP_OR_SUBPLAN         => 'OTHER_GROUPS',
    COMMENT                  => 'Mandatory',
    MGMT_P1                  => 25);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                     => 'Quarantine_plan',
    GROUP_OR_SUBPLAN         => 'QUARANTINE_GROUP',
    COMMENT                  => 'Limited CPU',
    MGMT_P2                  => 100,
    UTILIZATION_LIMIT        => 20);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Note:

Although you could set the utilization limit to zero for `QUARANTINE_GROUP`, thus completely quarantining runaway queries, it is recommended that you avoid doing this. If the runaway query is holding any resources—PGA memory, locks, and so on—required by any other session, then a zero allocation setting could lead to a deadlock.

Example 3 - Limiting CPU for Applications

In this example, assume that mapping rules map application sessions into one of four application groups. Each application group is allocated a utilization limit of 30%. This limits CPU utilization of any one application to 30%. The sum of the `UTILIZATION_LIMIT` values exceeds 100%, which is permissible and acceptable in a situation where all applications are not active simultaneously.

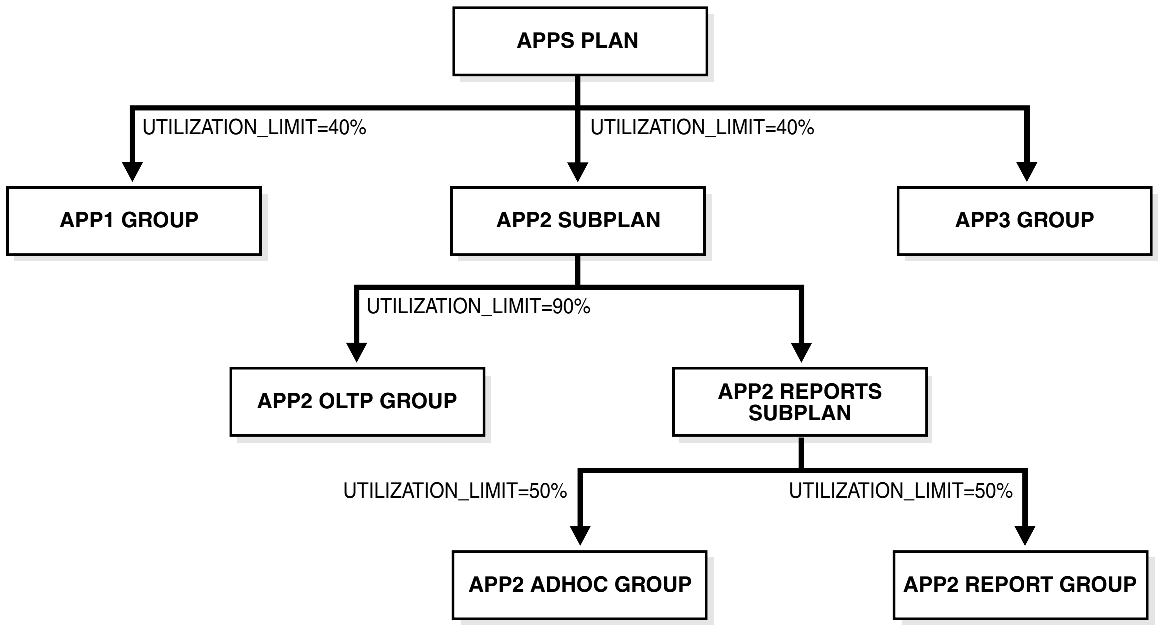
```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
     CONSUMER_GROUP => 'APP1_GROUP',
     COMMENT        => 'Apps group 1');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_GROUP',
     COMMENT        => 'Apps group 2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP3_GROUP',
     COMMENT        => 'Apps group 3');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP4_GROUP',
     COMMENT        => 'Apps group 4');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'apps_plan',
    COMMENT => 'Application consolidation');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                     => 'apps_plan',
    GROUP_OR_SUBPLAN         => 'APP1_GROUP',
    COMMENT                  => 'Apps group 1',
    UTILIZATION_LIMIT        => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                     => 'apps_plan',
    GROUP_OR_SUBPLAN         => 'APP2_GROUP',
    COMMENT                  => 'Apps group 2',
    UTILIZATION_LIMIT        => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                     => 'apps_plan',
    GROUP_OR_SUBPLAN         => 'APP3_GROUP',
    COMMENT                  => 'Apps group 3',
    UTILIZATION_LIMIT        => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                     => 'apps_plan',
    GROUP_OR_SUBPLAN         => 'APP4_GROUP',
    COMMENT                  => 'Apps group 4',
    UTILIZATION_LIMIT        => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                     => 'apps_plan',
    GROUP_OR_SUBPLAN         => 'OTHER_GROUPS',
    COMMENT                  => 'Mandatory',
    UTILIZATION_LIMIT        => 20);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

If all four application groups can fully use the CPU allocated to them (30% in this case), then the minimum CPU that is allocated to each application group is computed as a ratio of the application group's limit to the total of the limits of all application groups. In this example, all four application groups are allocated a utilization limit of 30%. Therefore, when all four groups fully use their limits, the CPU allocation to each group is 30/(30+30+30+30) = 25%.

Example 4 - Specifying a Utilization Limit for Consumer Groups and Subplans

The following example describes how the utilization limit is computed for scenarios, such as the one in Figure 26-8, where you set `UTILIZATION_LIMIT` for a subplan and for consumer groups within the subplan. For simplicity, the requirement to include the `OTHER_GROUPS` consumer group is ignored, and resource plan directives are not shown, even though they are part of the plan.

Figure 26-8   Resource Plan with Maximum Utilization for Subplan and Consumer Groups



The following PL/SQL block creates the plan described in Figure 26-8.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
    DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP1_GROUP',
     COMMENT        => 'Group for application #1');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_OLTP_GROUP',
     COMMENT        => 'Group for OLTP activity in application #2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_ADHOC_GROUP',
     COMMENT        => 'Group for ad-hoc queries in application #2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_REPORT_GROUP',
     COMMENT        => 'Group for reports in application #2');
   DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APPS_PLAN',
    COMMENT => 'Plan for managing 3 applications');
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APP2_SUBPLAN',
    COMMENT => 'Subplan for managing application #2',
    SUB_PLAN => TRUE);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN     => 'APP2_REPORTS_SUBPLAN',
    COMMENT => 'Subplan for managing reports in application #2',
    SUB_PLAN => TRUE);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APPS_PLAN',
    GROUP_OR_SUBPLAN          => 'APP1_GROUP',
    COMMENT                   => 'Limit CPU for application #1 to 40%',
    UTILIZATION_LIMIT         => 40);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APPS_PLAN',
    GROUP_OR_SUBPLAN          => 'APP2_SUBPLAN',
    COMMENT                   => 'Limit CPU for application #2 to 40%',
    UTILIZATION_LIMIT         => 40);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APP2_SUBPLAN',
    GROUP_OR_SUBPLAN          => 'APP2_OLTP_GROUP',
    COMMENT                   => 'Limit CPU for OLTP to 90% of application #2',
    UTILIZATION_LIMIT         => 90);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APP2_SUBPLAN',
    GROUP_OR_SUBPLAN          => 'APP2_REPORTS_SUBPLAN',
    COMMENT                   => 'Subplan for ad-hoc and normal reports for application #2');
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APP2_REPORTS_SUBPLAN',
    GROUP_OR_SUBPLAN          => 'APP2_ADHOC_GROUP',
    COMMENT                   => 'Limit CPU for ad-hoc queries to 50% of application #2
reports',
    UTILIZATION_LIMIT         => 50);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APP2_REPORTS_SUBPLAN',
    GROUP_OR_SUBPLAN          => 'APP2_REPORT_GROUP',
    COMMENT                   => 'Limit CPU for reports to 50% of application #2 reports',
    UTILIZATION_LIMIT         => 50);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                      => 'APPS_PLAN',
    GROUP_OR_SUBPLAN          => 'OTHER_GROUPS',
    COMMENT                   => 'No directives for default users');
   DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

In this example, the maximum CPU utilization for the consumer group `APP1_GROUP` and subplan `APP2_SUBPLAN` is set to 40%. The limit for the consumer groups `APP2_ADHOC_GROUP` and `APP2_REPORT_GROUP` is set to 50%.

Because there is no limit specified for the subplan `APP2_REPORTS_SUBPLAN`, it inherits the limit of its parent subplan `APP2_SUBPLAN`, which is 40%. The absolute limit for the consumer group `APP2_REPORT_GROUP` is computed as 50% of its parent subplan, which is 50% of 40%, or 20%.

Similarly, because the consumer group `APP2_ADHOC_GROUP` is contained in the subplan `APP2_REPORTS_SUBPLAN`, its limit is computed as a percentage of its parent subplan. The utilization limit for the consumer group `APP2_ADHOC_GROUP` is 50% of 40%, or 20%.

The maximum CPU utilization for the consumer group `APP2_OLTP_GROUP` is set to 90%. The parent subplan of `APP2_OLTP_GROUP`, `APP2_SUBPLAN`, has a limit of 40%. Therefore, the absolute limit for the group `APP2_OLTP_GROUP` is 90% of 40%, or 36%.

26.5.3 Example of Using Several Resource Allocation Methods

An example illustrates using several resource allocation methods.

The example presented here could represent a plan for a database supporting a packaged ERP (Enterprise Resource Planning) or CRM (Customer Relationship Management) application. The work in such an environment can be highly varied. There may be a mix of short transactions and quick queries, in combination with longer running batch jobs that include large parallel queries. The goal is to give good response time to OLTP (Online Transaction Processing), while allowing batch jobs to run in parallel.

The plan is summarized in the following table.

Group	CPU Resource Allocation %	Parallel Statement Queuing	Automatic Consumer Group Switching	Maximum Estimated Execution Time	Undo Pool	PGA Limit for Each Session
`oltp`	60%	--	Switch to group: `batch` Switch time: 3 secs	--	200K	20M
`batch`	30%	Parallel server limit: 8 Parallel queue timeout: 600 secs	--	3600 secs	--	--
`OTHER_GROUPS`	10%	--	--	--	--	--

The following statements create the preceding plan, which is named `erp_plan`:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
  COMMENT => 'Resource plan/method for ERP Database');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
  COMMENT => 'Resource consumer group/method for OLTP jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
  COMMENT => 'Resource consumer group/method for BATCH jobs');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', MGMT_P1 => 60,
  SWITCH_GROUP => 'batch', SWITCH_TIME => 3, UNDO_POOL => 200,
  SWITCH_FOR_CALL => TRUE, SESSION_PGA_LIMIT => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', MGMT_P1 => 30,
  PARALLEL_SERVER_LIMIT => 8, PARALLEL_QUEUE_TIMEOUT => 600,
  MAX_EST_EXEC_TIME => 3600);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', MGMT_P1 => 10);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

26.5.4 Example of Managing Parallel Statements Using Directive Attributes

An example illustrates managing parallel statements using directive attributes.

A typical data warehousing environment consists of different types of users with varying resource requirements. Users with common processing needs are grouped into a consumer group. The consumer group `URGENT_GROUP` consists of users who run reports that provide important information to top management. This group generates a large number of parallel queries. Users from the consumer group `ETL_GROUP` import data from source systems and perform extract, transform, and load (ETL) operations. The group `OTHER_GROUPS` contains users who execute ad-hoc queries. You must manage the requirements of these diverse groups of users while optimizing performance.

You can use the following directive attributes to manage and optimize the execution of parallel statements:

- `MGMT_P`*n*
- `PARALLEL_SERVER_LIMIT`
- `PARALLEL_STMT_CRITICAL`
- `PARALLEL_QUEUE_TIMEOUT`
- `PQ_TIMEOUT_ACTION`
- `PARALLEL_DEGREE_LIMIT_P1`
- `SHARES`

Note:

- The `MGMT_P`*n* management attributes and `SHARES` attribute control how a parallel statement is selected from the parallel statement queue for execution. You can prioritize the parallel statements of one consumer group over another by setting a higher value for the management attributes of that group.
- In a multitenant environment, if you want more per-workload management, then you can use the `SHARES` attribute to specify the share of resource allocation for pluggable databases (PDBs), which includes the parallel statement queuing resource. Alternatively, you can use the other directive attributes mentioned above.

Table 26-16 describes the resource allocations of the plan `DW_PLAN`, which can be used to manage the needs of the data warehouse users. This plan contains the consumer groups `URGENT_GROUP`, `ETL_GROUP`, and `OTHER_GROUPS`. This example demonstrates the use of directive attributes in ensuring that one application or consumer group does not use all the available parallel execution servers.

Table 26-16 Resource Plan with Parallel Statement Directives

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation	PARALLEL_DEGREE_LIMIT_P1	PARALLEL_SERVER_LIMIT	PARALLEL_QUEUE_TIMEOUT
`URGENT_GROUP`	100%			12		
`ETL_GROUP`		100%		8	50%	
`OTHER_GROUPS`			100%	2	50%	360

In this example, the parameter PARALLEL_SERVERS_TARGET initialization parameter is set to 64, which means that the number of parallel execution servers available is 64. The total number of parallel execution servers that can be used for parallel statement execution before URGENT_GROUP sessions with PARALLEL_DEGREE_POLICY set to AUTO are added to the parallel statement queue is equal to 64. Because the PARALLEL_SERVER_LIMIT attribute of ETL_GROUP and OTHER_GROUPS is 50%, the maximum number of parallel execution servers that can be used by these groups is 50% of 64, or 32 parallel execution servers each.

Note that parallel statements from a consumer group will only be queued if the PARALLEL_DEGREE_POLICY parameter is set to AUTO and the total number of active servers for the consumer group is higher than PARALLEL_SERVERS_TARGET. If PARALLEL_DEGREE_POLICY is set to MANUAL or LIMITED, then the statements are run provided there are enough parallel execution servers available. The parallel execution servers used by such a statement will count toward the total number of parallel execution servers used by the consumer group. However, the parallel statement will not be added to the parallel statement queue.

Tip:

For low-priority applications, it is a common practice to set low values for PARALLEL_DEGREE_LIMIT_P1 and PARALLEL_SERVER_LIMIT.

Because URGENT_GROUP has 100% of the allocation at level 1, its parallel statements will always be dequeued ahead of the other consumer groups from the parallel statement queue. Although URGENT_GROUP has no PARALLEL_SERVER_LIMIT directive attribute, a statement issued by a session in this group might still be queued if there are not enough available parallel execution servers to run it.

When you create the resource plan directive for the URGENT_GROUP, you can set the PARALLEL_STMT_CRITICAL parameter to BYPASS_QUEUE. With this setting, parallel statements from the consumer group bypass the parallel statements queue and are executed immediately. However, the number of parallel execution servers might exceed the setting of the PARALLEL_SERVERS_TARGET initialization parameter, and the degree of parallelism might be lower if the limit set by the PARALLEL_MAX_SERVERS initialization parameter is reached.

The degree of parallelism, represented by PARALLEL_DEGREE_LIMIT_P1, is set to 12 for URGENT_GROUP. Therefore, each parallel statement from URGENT_GROUP can use a maximum of 12 parallel execution servers. Similarly, each parallel statement from the ETL_GROUP can use a maximum of 8 parallel execution servers and each parallel statement from the OTHER_GROUPS can use 2 parallel execution servers.

Suppose, at a given time, the only parallel statements are from the ETL_GROUP, and they are using 26 out of the 32 parallel execution servers available to this group. Sessions from this consumer group have PARALLEL_DEGREE_POLICY set to AUTO. If another parallel statement with the PARALLEL_DEGREE_LIMIT_P1 attribute set to 8 is launched from ETL_GROUP, then this query cannot be run immediately because the available parallel execution servers in the ETL_GROUP is 32-26=6 parallel execution servers. The new parallel statement is queued until the number of parallel execution servers it requires is available in ETL_GROUP.

While the parallel statements in ETL_GROUP are being executed, suppose a parallel statement is launched from OTHER_GROUPS. This group still has 32 parallel execution servers available and so the parallel statement is executed.

The PARALLEL_QUEUE_TIMEOUT attribute for OTHER_GROUPS is set to 360. Therefore, any parallel statement from this group can remain in the parallel execution server queue for 360 seconds only. After this time, the parallel statement is removed from the queue and the error `ORA-07454` is returned.

See Also:

- "Parallel Execution Servers"
- "Creating Resource Plan Directives "

26.5.5 An Oracle-Supplied Mixed Workload Plan

Oracle Database includes a predefined resource plan, `MIXED_WORKLOAD_PLAN`, that prioritizes interactive operations over batch operations, and includes the required subplans and consumer groups recommended by Oracle.

`MIXED_WORKLOAD_PLAN` is defined as follows:

Group or Subplan	CPU Resource Allocation				
	Level 1	Level 2	Level 3	Automatic Consumer Group Switching	Max Degree of Parallelism
`BATCH_GROUP`			100%		
`INTERACTIVE_GROUP`		85%		Switch to group: `BATCH_GROUP` Switch time: 60 seconds Switch for call: `TRUE`	1
`ORA$AUTOTASK`		5%			
`OTHER_GROUPS`		5%			
`SYS_GROUP`	100%				

In this plan, because `INTERACTIVE_GROUP` is intended for short transactions, any call that consumes more than 60 seconds of CPU time is automatically switched to `BATCH_GROUP`, which is intended for longer batch operations.

You can use this predefined plan if it is appropriate for your environment. (You can modify the plan, or delete it if you do not intend to use it.) Note that there is nothing special about the names `BATCH_GROUP` and `INTERACTIVE_GROUP`. The names reflect only the intended purposes of the groups, and it is up to you to map application sessions to these groups and adjust CPU resource allocation percentages accordingly so that you achieve proper resource management for your interactive and batch applications. For example, to ensure that your interactive applications run under the `INTERACTIVE_GROUP` consumer group, you must map your interactive applications' user sessions to this consumer group based on user name, service name, program name, module name, or action, as described in "Specifying Session-to-Consumer Group Mapping Rules". You must map your batch applications to the `BATCH_GROUP` in the same way. Finally, you must enable this plan as described in "Enabling Oracle Database Resource Manager and Switching Plans".

See Table 26-17 and Table 26-18 for explanations of the other resource consumer groups and subplans in this plan.

26.6 Managing Multiple Database Instances on a Single Server

Oracle Database provides a method for managing CPU allocations on a multi-CPU server running multiple database instances. This method is called instance caging. Instance caging and Oracle Database Resource Manager (the Resource Manager) work together to support desired levels of service across multiple instances.

- About Instance Caging
  A simple way to limit CPU consumption for each database instance is to use instance caging. **Instance caging** is a method that uses an initialization parameter to limit the number of CPUs that an instance can use simultaneously.
- Enabling Instance Caging
  You can enable instance caging using by creating a resource plan with CPU directives and setting the `CPU_COUNT` initialization parameter.

26.6.1 About Instance Caging

A simple way to limit CPU consumption for each database instance is to use instance caging. **Instance caging** is a method that uses an initialization parameter to limit the number of CPUs that an instance can use simultaneously.

You might decide to run multiple Oracle database instances on a single multi-CPU server. A typical reason to do so would be server consolidation—using available hardware resources more efficiently. When running multiple instances on a single server, the instances compete for CPU. One resource-intensive database instance could significantly degrade the performance of the other instances. For example, on a 16-CPU system with four database instances, the operating system might be running one database instance on the majority of the CPUs during a period of heavy load for that instance. This could degrade performance in the other three instances. CPU allocation decisions such as this are made solely by the operating system; the user generally has no control over them.

In the previous example, if you use instance caging to limit the number of CPUs to four for each of the four instances, there is less likelihood that one instance can interfere with the others. When constrained to four CPUs, an instance might become CPU-bound. This is when the Resource Manager begins to do its work to allocate CPU among the various database sessions according to the resource plan that you set for the instance. Thus, instance caging and the Resource Manager together provide a simple, effective way to manage multiple instances on a single server.

There are two typical approaches to instance caging for a server:

- Over-subscribing—You would use this approach for non-critical databases such as development and test systems, or low-load non-critical production systems. In this approach, the sum of the CPU limits for each instance exceeds the actual number of CPUs on the system. For example, on a 4-CPU system with four database instances, you might limit each instance to three CPUs. When a server is over-subscribed in this way, the instances can impact each other's performance. However, instance caging limits the impact and helps provide somewhat predictable performance. However, if one of the instances has a period of high load, the CPUs are available to handle it. This is a reasonable approach for non-critical systems, because one or more of the instances may frequently be idle or at a very low load.

- Partitioning—This approach is for critical production systems, where you want to prevent instances from interfering with each other. You allocate CPUs such that the sum of all allocations is equal to the number of CPUs on the server. For example, on a 16-server system, you might allocate 8 CPUs to the first instance, 4 CPUs to the second, and 2 each to the remaining two instances. By dedicating CPU resources to each database instance, the load on one instance cannot affect another's, and each instance performs predictably.

Using Instance Caging with Utilization Limit

If you enable instance caging and set a utilization limit in your resource plan, then the absolute limit is computed as a percentage of the allocated CPU resources.

For example, if you enable instance caging and set the `CPU_COUNT` to 4, and a consumer group has a utilization limit of 50%, then the consumer group can use a maximum of 50% of 4 CPUs, which is 2 CPUs.

26.6.2 Enabling Instance Caging

You can enable instance caging using by creating a resource plan with CPU directives and setting the `CPU_COUNT` initialization parameter.

To enable instance caging, do the following for each instance on the server:

1. Enable the Resource Manager by assigning a resource plan, and ensure that the resource plan has CPU directives, using the `MGMT_P1` through `MGMT_P8` parameters.

   See "Enabling Oracle Database Resource Manager and Switching Plans" for instructions.

2. Set the `cpu_count` initialization parameter.

   This is a dynamic parameter, and can be set with the following statement:

   ```
   ALTER SYSTEM SET CPU_COUNT = 4;
   ```

26.7 Maintaining Consumer Groups, Plans, and Directives

You can maintain consumer groups, resource plans, and resource plan directives for Oracle Database Resource Manager (the Resource Manager). You perform maintenance tasks using the `DBMS_RESOURCE_MANAGER` PL/SQL package.

- Updating a Consumer Group
  You use the `UPDATE_CONSUMER_GROUP` procedure to update consumer group information.
- Deleting a Consumer Group
  The `DELETE_CONSUMER_GROUP` procedure deletes the specified consumer group.
- Updating a Plan
  You use the `UPDATE_PLAN` procedure to update plan information.
- Deleting a Plan
  The `DELETE_PLAN` procedure deletes the specified plan as well as all the plan directives associated with it.
- Updating a Resource Plan Directive
  Use the `UPDATE_PLAN_DIRECTIVE` procedure to update plan directives.
- Deleting a Resource Plan Directive
  To delete a resource plan directive, use the `DELETE_PLAN_DIRECTIVE` procedure.

See Also:

- Predefined Consumer Group Mapping Rules
- *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_RESOURCE_MANAGER` PL/SQL package.

26.7.1 Updating a Consumer Group

You use the `UPDATE_CONSUMER_GROUP` procedure to update consumer group information.

To update a consumer group:

1. Create a pending area.
2. Run the `UPDATE_CONSUMER_GROUP` procedure .

   If you do not specify the arguments for the `UPDATE_CONSUMER_GROUP` procedure, then they remain unchanged in the data dictionary.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.2 Deleting a Consumer Group

The `DELETE_CONSUMER_GROUP` procedure deletes the specified consumer group.

To delete a consumer group:

1. Create a pending area.
2. Run the `DELETE_CONSUMER_GROUP` procedure .
3. Submit the pending area.

Upon deletion of a consumer group, all users having the deleted group as their initial consumer group are assigned the `OTHER_GROUPS` as their initial consumer group. All currently running sessions belonging to a deleted consumer group are assigned to a new consumer group, based on the consumer group mapping rules. If no consumer group is found for a session through mapping, the session is switched to the `OTHER_GROUPS`.

You cannot delete a consumer group if it is referenced by a resource plan directive.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.3 Updating a Plan

You use the UPDATE_PLAN procedure to update plan information.

To update a plan:

1. Create a pending area.
2. Run the UPDATE_PLAN procedure. For example, the following PL/SQL block updates the COMMENT parameter:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_PLAN(
      PLAN => 'DAYTIME',
      NEW_COMMENT => '50% more resources for OLTP applications');
   END;
   /
   ```

   If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the CREATE_PENDING_AREA procedure.
- Submitting the Pending Area
  After you have validated your changes, call the SUBMIT_PENDING_AREA procedure to make your changes active.

26.7.4 Deleting a Plan

The DELETE_PLAN procedure deletes the specified plan as well as all the plan directives associated with it.

To delete a plan:

1. Create a pending area.
2. Run the DELETE_PLAN_CASCADE procedure. For example, the following PL/SQL block deletes the great_bread plan and its directives.

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
   END;
   /
   ```

   If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary.

   The resource consumer groups referenced by the deleted directives are not deleted, but they are no longer associated with the great_bread plan.

   The DELETE_PLAN_CASCADE procedure deletes the specified plan as well as all its descendants: plan directives and those subplans and resource consumer groups that are not marked by the database as mandatory. If DELETE_PLAN_CASCADE encounters an error, then it rolls back, leaving the plan unchanged.

   You cannot delete the currently active plan.

3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.5 Updating a Resource Plan Directive

Use the `UPDATE_PLAN_DIRECTIVE` procedure to update plan directives.

To update a resource plan directive:

1. Create a pending area.
2. Run the `UPDATE_PLAN_DIRECTIVE` procedure.

   The following example adds a comment to a directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
         PLAN             => 'SIMPLE_PLAN1',
         GROUP_OR_SUBPLAN => 'MYGROUP1',
         NEW_COMMENT      => 'Higher priority'
        );
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

   To clear (nullify) a comment, pass a null string (`''`). To clear (zero or nullify) any numeric directive parameter, set its new value to -1:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
         PLAN                  => 'SIMPLE_PLAN1',
         GROUP_OR_SUBPLAN      => 'MYGROUP1',
         NEW_MAX_EST_EXEC_TIME => -1
        );
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

   If you do not specify an argument for the `UPDATE_PLAN_DIRECTIVE` procedure, then its corresponding parameter in the directive remains unchanged.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.6 Deleting a Resource Plan Directive

To delete a resource plan directive, use the `DELETE_PLAN_DIRECTIVE` procedure.

To delete a resource plan directive:

1. Create a pending area.
2. Run the `DELETE_PLAN_DIRECTIVE` procedure.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.8 Viewing Database Resource Manager Configuration and Status

You can use several static data dictionary views and dynamic performance views to view the current configuration and status of Oracle Database Resource Manager (the Resource Manager).

- About Resource Manager Views
  A set of dynamic performance views enable you to monitor the results of your Oracle Database Resource Manager settings.
- Viewing Consumer Groups Granted to Users or Roles
  The `DBA_RSRC_CONSUMER_GROUP_PRIVS` view displays the consumer groups granted to users or roles.
- Viewing Plan Information
  An example illustrates using the `DBA_RSRC_PLANS` view to display all of the resource plans defined in the database.
- Viewing Current Consumer Groups for Sessions
  You can use the `V$SESSION` view to display the consumer groups that are currently assigned to sessions.
- Viewing the Currently Active Plans
  The `V$RSRC_PLAN` view displays currently active plans.
- Monitoring PDBs Managed by Oracle Database Resource Manager
  A set of dynamic performance views enables you to monitor the results of your Oracle Database Resource Manager settings for PDBs.

See Also:

Oracle Database Reference for details on all static data dictionary views and dynamic performance views

26.8.1 About Resource Manager Views

A set of dynamic performance views enable you to monitor the results of your Oracle Database Resource Manager settings.

Use the following dynamic performance views to help you monitor the results of your Oracle Database Resource Manager settings:

- V$RSRC_PLAN
- V$RSRC_CONSUMER_GROUP
- V$RSRC_SESSION_INFO
- V$RSRC_PLAN_HISTORY
- V$RSRC_CONS_GROUP_HISTORY
- V$RSRCMGRMETRIC
- V$RSRCMGRMETRIC_HISTORY

These views provide:

- Current status information
- History of resource plan activations
- Current and historical statistics on resource consumption and CPU waits by both resource consumer group and session

In addition, historical statistics are available through the `DBA_HIST_RSRC_PLAN` and `DBA_HIST_RSRC_CONSUMER_GROUP` views, which contain Automatic Workload Repository (AWR) snapshots of the `V$RSRC_PLAN_HISTORY` and `V$RSRC_CONS_GROUP_HISTORY`, respectively.

For assistance with tuning, the views `V$RSRCMGRMETRIC` and `V$RSRCMGRMETRIC_HISTORY` show how much time was spent waiting for CPU and how much CPU was consumed per minute for every consumer group for the past hour. These metrics can also be viewed graphically with Cloud Control, on the Resource Manager Statistics page.

When Resource Manager is enabled, Resource Manager automatically records statistics about resource usage, and you can examine these statistics using real-time SQL monitoring and Resource Manager dynamic performance views.

You can use real-time SQL monitoring by accessing the SQL Monitor page in Cloud Control or by querying the `V$SQL_MONITOR` view and other related views. The `V$SQL_MONITOR` view also includes information about the last action performed by Resource Manager for a consumer group in the following columns: `RM_CONSUMER_GROUP`, `RM_LAST_ACTION`, `RM_LAST_ACTION_REASON`, and `RM_LAST_ACTION_TIME`.

In addition, the following dynamic performance views contain statistics about resource usage:

- `V$RSRCMGRMETRIC`
- `V$RSRCMGRMETRIC_HISTORY`
- `V$RSRC_CONSUMER_GROUP`
- `V$RSRC_CONS_GROUP_HISTORY`

See Also:

Oracle Database SQL Tuning Guide for more information about real-time SQL monitoring

V$RSRC_PLAN

This view displays the currently active resource plan and its subplans.

```
SELECT name, is_top_plan FROM v$rsrc_plan;

NAME                             IS_TOP_PLAN
-------------------------------- -----------
DEFAULT_PLAN                     TRUE
ORA$AUTOTASK                     FALSE
ORA$AUTOTASK_HIGH_SUB_PLAN       FALSE
```

The plan for which `IS_TOP_PLAN` is `TRUE` is the currently active (top) plan, and the other plans are subplans of either the top plan or of other subplans in the list.

This view also contains other information, including the following:

- The `INSTANCE_CAGING` column shows whether instance caging is enabled.
- The `CPU_MANAGED` column shows whether CPU is being managed.
- The `PARALLEL_EXECUTION_MANAGED` column shows whether parallel statement queuing is enabled.

See Also:

Oracle Database Reference

V$RSRC_CONSUMER_GROUP

Use the `V$RSRC_CONSUMER_GROUP` view to monitor resources consumed, including CPU, I/O, and parallel execution servers. It can also be used to monitor statistics related to CPU resource management, runaway query management, parallel statement queuing, and so on. All of the statistics are cumulative from the time when the plan was activated.

```
SELECT name, active_sessions, queue_length,
  consumed_cpu_time, cpu_waits, cpu_wait_time
  FROM v$rsrc_consumer_group;

NAME                 ACTIVE_SESSIONS QUEUE_LENGTH CONSUMED_CPU_TIME  CPU_WAITS CPU_WAIT_TIME
------------------ --------------- ------------ ----------------- ---------- -------------
OLTP_ORDER_ENTRY                 1            0             29690        467          6709
OTHER_GROUPS                     0            0           5982366       4089         60425
SYS_GROUP                        1            0           2420704        914         19540
DSS_QUERIES                      4            2           4594660       3004         55700
```

In the preceding query results, the `DSS_QUERIES` consumer group has four sessions in its active session pool and two more sessions queued for activation.

A key measure in this view is CPU_WAIT_TIME. This indicates the total time that sessions in the consumer group waited for CPU because of resource management. Not included in this measure are waits due to latch or enqueue contention, I/O waits, and so on.

Note:

The V$RSRC_CONSUMER_GROUP view records statistics for resources that are not currently being managed by Resource Manager. when the STATISTICS_LEVEL initialization parameter is set to ALL or TYPICAL.

See Also:

Oracle Database Reference

V$RSRC_SESSION_INFO

Use this view to monitor the status of one or more sessions. The view shows how the session has been affected by the Resource Manager. It provides information such as:

- The consumer group that the session currently belongs to.
- The consumer group that the session originally belonged to.
- The session attribute that was used to map the session to the consumer group.
- Session state (RUNNING, WAIT_FOR_CPU, QUEUED, and so on).
- Current and cumulative statistics for metrics, such as CPU consumed, wait times, queued time, and number of active parallel servers used. Current statistics reflect statistics for the session since it joined its current consumer group. Cumulative statistics reflect statistics for the session in all consumer groups to which it has belonged since it was created.

```
SELECT se.sid sess_id, co.name consumer_group,
 se.state, se.consumed_cpu_time cpu_time, se.cpu_wait_time, se.queued_time
 FROM v$rsrc_session_info se, v$rsrc_consumer_group co
 WHERE se.current_consumer_group_id = co.id;

SESS_ID CONSUMER_GROUP     STATE     CPU_TIME CPU_WAIT_TIME QUEUED_TIME
------- ------------------ -------- --------- ------------- -----------
    113 OLTP_ORDER_ENTRY   WAITING     137947         28846           0
    135 OTHER_GROUPS       IDLE        785669         11126           0
    124 OTHER_GROUPS       WAITING      50401         14326           0
    114 SYS_GROUP          RUNNING        495             0           0
    102 SYS_GROUP          IDLE         88054            80           0
    147 DSS_QUERIES        WAITING     460910        512154           0
```

CPU_WAIT_TIME in this view has the same meaning as in the V$RSRC_CONSUMER_GROUP view, but applied to an individual session.

You can join this view with the V$SESSION view for more information about a session.

See Also:

- *Oracle Database Reference*
- *Oracle Database VLDB and Partitioning Guide*

V$RSRC_PLAN_HISTORY

This view shows when resource plans were enabled or disabled on the instance. Each resource plan activation or deactivation is assigned a sequence number. For each entry in the view, the `V$RSRC_CONS_GROUP_HISTORY` view has a corresponding entry for each consumer group in the plan that shows the cumulative statistics for the consumer group. The two views are joined by the `SEQUENCE#` column in each.

```
SELECT sequence# seq, name plan_name,
to_char(start_time, 'DD-MON-YY HH24:MM') start_time,
to_char(end_time, 'DD-MON-YY HH24:MM') end_time, window_name
FROM v$rsrc_plan_history;

 SEQ PLAN_NAME                   START_TIME      END_TIME        WINDOW_NAME
---- -------------------------- --------------- --------------- ----------------
   1                             29-MAY-07 23:05 29-MAY-07 23:05
   2 DEFAULT_MAINTENANCE_PLAN    29-MAY-07 23:05 30-MAY-07 02:05 TUESDAY_WINDOW
   3                             30-MAY-07 02:05 30-MAY-07 22:05
   4 DEFAULT_MAINTENANCE_PLAN    30-MAY-07 22:05 31-MAY-07 02:05 WEDNESDAY_WINDOW
   5                             31-MAY-07 02:05 31-MAY-07 22:05
   6 DEFAULT_MAINTENANCE_PLAN    31-MAY-07 22:05                 THURSDAY_WINDOW
```

A null value under `PLAN_NAME` indicates that no plan was active.

AWR snapshots of this view are stored in the `DBA_HIST_RSRC_PLAN` view.

See Also:

Oracle Database Reference

V$RSRC_CONS_GROUP_HISTORY

This view helps you understand how resources were shared among the consumer groups over time. The `sequence#` column corresponds to the column of the same name in the `V$RSRC_PLAN_HISTORY` view. Therefore, you can determine the plan that was active for each row of consumer group statistics.

```
SELECT sequence# seq, name, cpu_wait_time, cpu_waits,
consumed_cpu_time FROM v$rsrc_cons_group_history;

 SEQ NAME                      CPU_WAIT_TIME  CPU_WAITS CONSUMED_CPU_TIME
---- ------------------------- ------------- ---------- -----------------
   2 SYS_GROUP                         18133        691          33364431
   2 OTHER_GROUPS                      51252        825         181058333
   2 ORA$AUTOTASK_MEDIUM_GROUP            21          5           4019709
   2 ORA$AUTOTASK_URGENT_GROUP            35          1            198760
   2 ORA$AUTOTASK_STATS_GROUP              0          0                 0
   2 ORA$AUTOTASK_SPACE_GROUP              0          0                 0
   2 ORA$AUTOTASK_SQL_GROUP                0          0                 0
```

```
  2 ORA$AUTOTASK_HEALTH_GROUP              0          0                  0
  4 SYS_GROUP                          40344         85           42519265
  4 OTHER_GROUPS                      123295       1040          371481422
  4 ORA$AUTOTASK_MEDIUM_GROUP              1          4            7433002
  4 ORA$AUTOTASK_URGENT_GROUP          22959        158           19964703
  4 ORA$AUTOTASK_STATS_GROUP               0          0                  0
    .
    .
```

AWR snapshots of this view are stored in the `DBA_HIST_RSRC_CONSUMER_GROUP` view. Use `DBA_HIST_RSRC_CONSUMER_GROUP` with `DBA_HIST_RSRC_PLAN` to determine the plan that was active for each historical set of consumer group statistics.

Note:

The `V$RSRC_CONS_GROUP_HISTORY` view records statistics for resources that are not currently being managed by Resource Manager. when the `STATISTICS_LEVEL` initialization parameter is set to `ALL` or `TYPICAL`.

See Also:

- *Oracle Database Reference*
- *Oracle Database Performance Tuning Guide* for information about the AWR.

V$RSRCMGRMETRIC

This view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute. It provides real-time metrics for each consumer group and is very useful in scenarios where you are running workloads and want to continuously monitor CPU resource utilization.

Use this view to compare the maximum possible CPU utilization and average CPU utilization percentage for consumer groups with other consumer group settings such as CPU time used, time waiting for CPU, average number of sessions that are consuming CPU, and number of sessions that are waiting for CPU allocation. For example, you can view the amount of CPU resources a consumer group used and how long it waited for resource allocation. Or, you can view how many sessions from each consumer group are executed against the total number of active sessions.

To track CPU consumption in terms of CPU utilization, use the `CPU_UTILIZATION_LIMIT` and `AVG_CPU_UTILIZATION` columns. `AVG_CPU_UTILIZATION` lists the average percentage of the server's CPU that is consumed by a consumer group. `CPU_UTILIZATION_LIMIT` represents the maximum percentage of the server's CPU that a consumer group can use. This limit is set using the `UTILIZATION_LIMIT` directive attribute.

```
SELECT consumer_group_name, cpu_utilization_limit,
avg_cpu_utilization FROM v$rsrcmgrmetric;
```

Use the `CPU_CONSUMED_TIME` and `CPU_TIME_WAIT` columns to track CPU consumption and throttling in milliseconds. The column `NUM_CPUS` represents the number of CPUs that Resource Manager is managing.

```
SELECT consumer_group_name, cpu_consumed_time,
cpu_wait_time, num_cpus FROM v$rsrcmgrmetric;
```

To track the CPU consumption and throttling in terms of number of sessions, use the `RUNNING_SESSIONS_LIMIT`, `AVG_RUNNING_SESSIONS`, and `AVG_WAITING_SESSIONS` columns. `RUNNING_SESSIONS_LIMIT` lists the maximum number of sessions, from a particular consumer group, that can be running at any time. This limit is defined by the `UTILIZATION_LIMIT` directive attribute that you set either for the consumer group or for a subplan that contains the consumer group. For each consumer group, `AVG_RUNNING_SESSIONS` lists the average number of sessions that are consuming CPU and `AVG_WAITING_SESSIONS` lists the average number of sessions that are waiting for CPU.

```
SELECT sequence#, consumer_group_name, running_sessions_limit,
avg_running_sessions, avg_waiting_sessions FROM v$rsrcmgrmetric;
```

To track parallel statements and parallel server use for a consumer group, use the `AVG_ACTIVE_PARALLEL_STMTS`, `AVG_QUEUED_PARALLEL_STMTS`, `AVG_ACTIVE_PARALLEL_SERVERS`, `AVG_QUEUED_PARALLEL_SERVERS`, and `PARALLEL_SERVERS_LIMIT` columns. `AVG_ACTIVE_PARALLEL_STMTS` and `AVG_ACTIVE_PARALLEL_SERVERS` list the average number of parallel statements running and the average number of parallel servers used by the parallel statements. `AVG_QUEUED_PARALLEL_STMTS` and `AVG_QUEUED_PARALLEL_SERVERS` list the average number of parallel statements queued and average number of parallel servers that were requested by queued parallel statements. `PARALLEL_SERVERS_LIMIT` lists the number of parallel servers allowed to be used by the consumer group.

```
SELECT avg_active_parallel_stmts, avg_queued_parallel_stmts,
avg_active_parallel_servers, avg_queued_parallel_servers, parallel_servers_limit
FROM v$rsrcmgrmetric;
```

Note:

The `V$RSRCMGRMETRIC` view records statistics for resources that are not currently being managed by Resource Manager. when the `STATISTICS_LEVEL` initialization parameter is set to `ALL` or `TYPICAL`.

See Also:

Oracle Database Reference

V$RSRCMGRMETRIC_HISTORY

The columns in the `V$RSRCMGRMETRIC_HISTORY` are the same view as V$RSRCMGRMETRIC. The only difference between these views is that V$RSRCMGRMETRIC contains metrics for the past one minute only, whereas V$`RSRCMGRMETRIC_HISTORY` contains metrics for the last 60 minutes.

Note:

The V$RSRCMGRMETRIC_HISTORY view records statistics for resources that are not currently being managed by Resource Manager. when the STATISTICS_LEVEL initialization parameter is set to ALL or TYPICAL.

See Also:

Oracle Database Reference

26.8.2 Viewing Consumer Groups Granted to Users or Roles

The DBA_RSRC_CONSUMER_GROUP_PRIVS view displays the consumer groups granted to users or roles.

Specifically, it displays the groups to which a user or role is allowed to belong or be switched. For example, in the view shown below, user SCOTT always starts in the SALES consumer group, can switch to the MARKETING group through a specific grant, and can switch to the DEFAULT_CONSUMER_GROUP (OTHER_GROUPS) and LOW_GROUP groups because they are granted to PUBLIC. SCOTT also can grant the SALES group but not the MARKETING group to other users.

```
SELECT * FROM dba_rsrc_consumer_group_privs;

GRANTEE            GRANTED_GROUP                  GRANT_OPTION INITIAL_GROUP
------------------ ------------------------------ ------------ -------------
PUBLIC             DEFAULT_CONSUMER_GROUP         YES          YES
PUBLIC             LOW_GROUP                      NO           NO
SCOTT              MARKETING                      NO           NO
SCOTT              SALES                          YES          YES
SYSTEM             SYS_GROUP                      NO           YES
```

SCOTT was granted the ability to switch to these groups using the DBMS_RESOURCE_MANAGER_PRIVS package.

26.8.3 Viewing Plan Information

An example illustrates using the DBA_RSRC_PLANS view to display all of the resource plans defined in the database.

All plans have a NULL status, meaning that they are not in the pending area.

Note:

Plans in the pending area have a status of PENDING. Plans in the pending area are being edited.

```
SELECT plan,status,comments FROM dba_rsrc_plans;
```

```
PLAN                        STATUS   COMMENTS
--------------------------- -------- ----------------------------------------
DSS_PLAN                             Example plan for DSS workloads that prio...
ETL_CRITICAL_PLAN                    Example plan for DSS workloads that prio...
MIXED_WORKLOAD_PLAN                  Example plan for a mixed workload that p...
DEFAULT_MAINTENANCE_PLAN             Default plan for maintenance windows tha...
DEFAULT_PLAN                         Default, basic, pre-defined plan that pr...
INTERNAL_QUIESCE                     Plan for quiescing the database.  This p...
INTERNAL_PLAN                        Internally-used plan for disabling the r...
.
.
.
```

26.8.4 Viewing Current Consumer Groups for Sessions

You can use the `V$SESSION` view to display the consumer groups that are currently assigned to sessions.

The following example queries the `V$SESSION` view:

```
SELECT sid,serial#,username,resource_consumer_group FROM v$session;

SID    SERIAL#  USERNAME                 RESOURCE_CONSUMER_GROUP
-----  -------  ------------------------ --------------------------------
   11       136 SYS                      SYS_GROUP
   13     16570 SCOTT                    SALES
   ...
```

26.8.5 Viewing the Currently Active Plans

The `V$RSRC_PLAN` view displays currently active plans.

This example sets `mydb_plan`, as created by the example shown in "Multilevel Plan Example", as the top level plan. It then queries the `V$RSRC_PLAN` view to display the currently active plans. The view displays the current top level plan and all of its descendent subplans.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;

System altered.

SELECT name, is_top_plan FROM v$rsrc_plan;

NAME             IS_TOP_PLAN
----------------------------
MYDB_PLAN        TRUE
MAILDB_PLAN      FALSE
BUGDB_PLAN       FALSE
```

26.8.6 Monitoring PDBs Managed by Oracle Database Resource Manager

A set of dynamic performance views enables you to monitor the results of your Oracle Database Resource Manager settings for PDBs.

- About Resource Manager Views for PDBs
  You can monitor the results of your Oracle Database Resource Manager settings for PDBs using views.

- Monitoring CPU Usage for PDBs
  The `V$RSRCPDBMETRIC` view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute.
- Monitoring Parallel Execution for PDBs
  The `V$RSRCPDBMETRIC` view enables you to track parallel statements and parallel server use for PDBs.
- Monitoring the I/O Generated by PDBs
  The `V$RSRCPDBMETRIC` view enables you to track the amount of I/O generated by PDBs.
- Monitoring Memory Usage for PDBs
  The `V$RSRCPDBMETRIC` view enables you to track the amount memory used by PDBs.

26.8.6.1 About Resource Manager Views for PDBs

You can monitor the results of your Oracle Database Resource Manager settings for PDBs using views.

The following views are available:

- `V$RSRCPDBMETRIC`

  The `V$RSRCPDBMETRIC` view provides current statistics on resource consumption for PDBs, including CPU usage, parallel execution, I/O generated, and memory usage.
- `V$RSRCPDBMETRIC_HISTORY`

  The columns in the `V$RSRCPDBMETRIC_HISTORY` view are the same as the columns in the `V$RSRCPDBMETRIC` view. The only difference between these views is that the `V$RSRCPDBMETRIC` view contains metrics for the past one minute only, whereas the `V$RSRCPDBMETRIC_HISTORY` view contains metrics for the last 60 minutes.
- `V$RSRC_PDB`

  The `V$RSRC_PDB` view provides cumulative statistics. The statistics are accumulated since the time that the CDB resource plan was set.
- `DBA_HIST_RSRC_PDB_METRIC`

  This view contains the historical statistics of `V$RSRCPDBMETRIC_HISTORY`, taken using Automatic Workload Repository (AWR) snapshots.

Note:

The `V$RSRCPDBMETRIC` and `V$RSRCPDBMETRIC_HISTORY` views record statistics for resources that are not currently being managed by Resource Manager when the `STATISTICS_LEVEL` initialization parameter is set to `ALL` or `TYPICAL`.

See Also:

- *Oracle Database SQL Tuning Guide* for more information about real-time SQL monitoring
- *Oracle Database Reference* to learn about `V$RSRCPDBMETRIC`, `V$RSRCPDBMETRIC_HISTORY`, `V$RSRC_PDB`, and `DBA_HIST_RSRC_PDB_METRIC`

26.8.6.2 Monitoring CPU Usage for PDBs

The `V$RSRCPDBMETRIC` view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute.

The view provides real-time metrics for each PDB and is very useful in scenarios where you are running workloads and want to continuously monitor CPU resource utilization.

The active CDB resource plan manages CPU usage for a PDB. Use this view to compare the maximum and average CPU utilization for PDBs with other PDB settings such as the following:

- CPU time used
- Time waiting for CPU
- Average number of sessions that are consuming CPU
- Number of sessions that are waiting for CPU allocation

For example, you can view the amount of CPU resources a PDB used and how long it waited for resource allocation. Alternatively, you can view how many sessions from each PDB are executed against the total number of active sessions.

Tracking CPU Consumption in Terms of CPU Utilization for PDBs

To track CPU consumption in terms of CPU utilization, query the `CPU_UTILIZATION_LIMIT` and `AVG_CPU_UTILIZATION` columns. `AVG_CPU_UTILIZATION` lists the average percentage of the server's CPU that is consumed by a PDB. `CPU_UTILIZATION_LIMIT` represents the maximum percentage of the server's CPU that a PDB can use. This limit is set using the `UTILIZATION_LIMIT` directive attribute.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.CPU_UTILIZATION_LIMIT,
       r.AVG_CPU_UTILIZATION
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Tracking CPU Consumption and Throttling for PDBs

Use the `CPU_CONSUMED_TIME` and `CPU_TIME_WAIT` columns to track CPU consumption and throttling in milliseconds for each PDB. The column `NUM_CPUS` represents the number of CPUs that Resource Manager is managing.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.CPU_CONSUMED_TIME,
       r.CPU_WAIT_TIME,
       r.NUM_CPUS
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Tracking CPU Consumption and Throttling in Terms of Number of Sessions for PDBs

To track the CPU consumption and throttling in terms of number of sessions, use the `RUNNING_SESSIONS_LIMIT`, `AVG_RUNNING_SESSIONS`, and `AVG_WAITING_SESSIONS` columns. `RUNNING_SESSIONS_LIMIT` lists the maximum number of sessions from a particular PDB that can be running at any time. This limit is defined by the `UTILIZATION_LIMIT` directive attribute that you set for the PDB. `AVG_RUNNING_SESSIONS` lists the average number of sessions that are consuming CPU, and `AVG_WAITING_SESSIONS` lists the average number of sessions that are waiting for CPU.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.RUNNING_SESSIONS_LIMIT,
       r.AVG_RUNNING_SESSIONS,
       r.AVG_WAITING_SESSIONS
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

26.8.6.3 Monitoring Parallel Execution for PDBs

The `V$RSRCPDBMETRIC` view enables you to track parallel statements and parallel server use for PDBs.

Parallel execution servers for a PDB are managed with the active CDB resource plan of the PDB's CDB. To track parallel statements and parallel server use for PDBs, use the `AVG_ACTIVE_PARALLEL_STMTS`, `AVG_QUEUED_PARALLEL_STMTS`, `AVG_ACTIVE_PARALLEL_SERVERS`, `AVG_QUEUED_PARALLEL_SERVERS`, and `PARALLEL_SERVERS_LIMIT` columns.

`AVG_ACTIVE_PARALLEL_STMTS` and `AVG_ACTIVE_PARALLEL_SERVERS` list the average number of parallel statements running and the average number of parallel servers used by the parallel statements. `AVG_QUEUED_PARALLEL_STMTS` and `AVG_QUEUED_PARALLEL_SERVERS` list the average number of parallel statements queued and average number of parallel servers that were requested by queued parallel statements. `PARALLEL_SERVERS_LIMIT` lists the number of parallel servers allowed to be used by the PDB.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.AVG_ACTIVE_PARALLEL_STMTS,
r.AVG_QUEUED_PARALLEL_STMTS,
   r.AVG_ACTIVE_PARALLEL_SERVERS, r.AVG_QUEUED_PARALLEL_SERVERS,
r.PARALLEL_SERVERS_LIMIT
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

26.8.6.4 Monitoring the I/O Generated by PDBs

The `V$RSRCPDBMETRIC` view enables you to track the amount of I/O generated by PDBs.

I/O is limited for a PDB by setting the `MAX_IOPS` initialization parameter or the `MAX_MBPS` initialization parameter in the PDB. Use this view to compare the I/O generated by PDBs in terms of the number of operations each second and the number of megabytes each second.

Tracking the Number of I/O Operations Generated Each Second by PDBs

To track the I/O operations generated each second by PDBs during the previous minute, use the `IOPS` column.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.IOPS
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

Tracking the Number Megabytes Generated for I/O Operations Each Second by PDBs

To track number of megabytes generated for I/O operations each second by PDBs during the previous minute, use the `IOMBPS` column.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.IOMBPS
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

26.8.6.5 Monitoring Memory Usage for PDBs

The V$RSRCPDBMETRIC view enables you to track the amount memory used by PDBs.

Use this view to track the amount of SGA, PGA, buffer cache, and shared pool memory currently used by PDBs.

To track the current memory usage, in bytes, for specific PDBs, use the SGA_BYTES, PGA_BYTES, BUFFER_CACHE_BYTES, and SHARED_POOL_BYTES columns.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.SGA_BYTES, r.PGA_BYTES, r.BUFFER_CACHE_BYTES, 
r.SHARED_POOL_BYTES
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

26.9 Interacting with Operating-System Resource Control

Many operating systems provide tools for resource management. These tools often contain "workload manager" or "resource manager" in their names, and are intended to allow multiple applications to share the resources of a single server, using an administrator-defined policy. Examples are Hewlett Packard's Process Resource Manager or Solaris Containers, Zones, and Resource Pools.

- Guidelines for Using Operating-System Resource Control
  Follow guidelines if you use operating-system resource control.

26.9.1 Guidelines for Using Operating-System Resource Control

Follow guidelines if you use operating-system resource control.

If you choose to use operating-system resource control with Oracle Database, then you must use it judiciously, according to the following guidelines:

- If you have multiple instances on a node, and you want to distribute resources among them, then each instance should be assigned to a dedicated operating-system resource manager group or managed entity. To run multiple instances in the managed entity, use instance caging to manage how the CPU resources within the managed entity should be distributed among the instances. When Oracle Database Resource Manager is managing CPU resources, it expects a fixed amount of CPU resources for the instance. Without instance caging, it expects the available CPU resources to be equal to the number of CPUs in the managed entity. With instance caging, it expects the available CPU resources to be equal to the value of the CPU_COUNT initialization parameter. If there are less CPU resources than expected, then the Oracle Database Resource Manager is not as effective at enforcing the resource allocations in the resource plan. The PDB-level parameter CPU_MIN_COUNT is used to set the PDB share in the resource plan and the PDB-level CPU_COUNT to set the PDB utilization limit in the resource plan. See "Managing Multiple Database Instances on a Single Server" for information about instance caging.
- The dedicated entity running all the instance's processes must run at one priority (or resource consumption) level.

- The CPU resources assigned to the dedicated entity cannot be changed more frequently than once every few minutes.

  If the operating-system resource manager is rapidly changing the CPU resources allocated to an Oracle instance, then the Oracle Database Resource Manager might not manage CPU resources effectively. In particular, if the CPU resources allocated to the Oracle instance changes more frequently than every couple of minutes, then these changes might not be observed by Oracle because it only checks for such changes every couple of minutes. In these cases, Oracle Database Resource Manager can over-schedule processes if it concludes that more CPU resources are available than there actually are, and it can under-schedule processes if it concludes that less CPU resources are available than there actually are. If it over-schedules processes, then the UTILIZATION_LIMIT directives might be exceeded, and the CPU directives might not be accurately enforced. If it under-schedules processes, then the Oracle instance might not fully use the server's resources.
- Process priority management must not be enabled.
- Management of individual database processes at different priority levels (for example, using the nice command on UNIX platforms) is not supported. Severe consequences, including instance crashes, can result. Similar undesirable results are possible if operating-system resource control is permitted to manage the memory to which an Oracle Database instance is pinned.

Related Topics

- CPU-Related Initialization Parameters for PDBs

26.10 Oracle Database Resource Manager Reference

Resource Manager includes predefined resource plans, consumer groups, and consumer groups mapping rules. You can query data dictionary views for information about your Resource Manager configuration.

- Predefined Resource Plans and Consumer Groups
  Oracle Database includes predefined resource plans.
- Predefined Consumer Group Mapping Rules
  Oracle Database includes predefined consumer group mapping rules.
- Resource Manager Data Dictionary Views
  You can query a set of data dictionary views for information relating to database resource management.

26.10.1 Predefined Resource Plans and Consumer Groups

Oracle Database includes predefined resource plans.

Table 26-17 lists the resource plans and Table 26-18 lists the resource consumer groups that are predefined in each Oracle database. You can verify these by querying the views DBA_RSRC_PLANS and DBA_RSRC_CONSUMER_GROUPS.

The following query displays the CPU allocations in the example plan DSS_PLAN:

```
SELECT group_or_subplan, mgmt_p1, mgmt_p2, mgmt_p3, mgmt_p4
   FROM dba_rsrc_plan_directives WHERE plan = 'DSS_PLAN';

GROUP_OR_SUBPLAN                  MGMT_P1    MGMT_P2    MGMT_P3    MGMT_P4
------------------------------ ---------- ---------- ---------- ----------
SYS_GROUP                              75          0          0          0
```

```
DSS_CRITICAL_GROUP                    18          0          0          0
DSS_GROUP                              3          0          0          0
ETL_GROUP                              1          0          0          0
BATCH_GROUP                            1          0          0          0
ORA$AUTOTASK                           1          0          0          0
OTHER_GROUPS                           1          0          0          0
```

Table 26-17   Predefined Resource Plans

Resource Plan	Description
`DEFAULT_MAINTENANCE_PLAN`	Default plan for maintenance windows. See "About Resource Allocations for Automated Maintenance Tasks" for details of this plan. Because maintenance windows are regular Oracle Scheduler windows, you can change the resource plan associated with them, if desired. If you do change a maintenance window resource plan, ensure that you include the subplan `ORA$AUTOTASK` in the new plan.
`DEFAULT_PLAN`	Basic default plan that prioritizes `SYS_GROUP` operations and allocates minimal resources for automated maintenance and diagnostics operations.
`DSS_PLAN`	Example plan for a data warehouse that prioritizes critical DSS queries over non-critical DSS queries and ETL operations.
`ETL_CRITICAL_PLAN`	Example plan for a data warehouse that prioritizes ETL operations over DSS queries.
`INTERNAL_PLAN`	For disabling the resource manager. For internal use only.
`INTERNAL_QUIESCE`	For quiescing the database. This plan cannot be activated directly. To activate, use the `QUIESCE` command.
`MIXED_WORKLOAD_PLAN`	Example plan for a mixed workload that prioritizes interactive operations over batch operations. See "An Oracle-Supplied Mixed Workload Plan" for details.

Table 26-18   Predefined Resource Consumer Groups

Resource Consumer Group	Description
`BATCH_GROUP`	Consumer group for batch operations. Referenced by the example plan `MIXED_WORKLOAD_PLAN`.
`DSS_CRITICAL_GROUP`	Consumer group for critical DSS queries. Referenced by the example plans `DSS_PLAN` and `ETL_CRITICAL_PLAN`.
`DSS_GROUP`	Consumer group for non-critical DSS queries. Referenced by the example plans `DSS_PLAN` and `ETL_CRITICAL_PLAN`.
`ETL_GROUP`	Consumer group for ETL jobs. Referenced by the example plans `DSS_PLAN` and `ETL_CRITICAL_PLAN`.
`INTERACTIVE_GROUP`	Consumer group for interactive, OLTP operations. Referenced by the example plan `MIXED_WORKLOAD_PLAN`.
`LOW_GROUP`	Consumer group for low-priority sessions.
`ORA$AUTOTASK`	Consumer group for maintenance tasks.

Table 26-18 (Cont.) Predefined Resource Consumer Groups

Resource Consumer Group	Description
`OTHER_GROUPS`	Default consumer group for all sessions that do not have an explicit initial consumer group, are not mapped to a consumer group with session-to-consumer group mapping rules, or are mapped to a consumer group that is not in the currently active resource plan. `OTHER_GROUPS` must have a resource plan directive specified in every plan. It cannot be assigned explicitly to sessions through mapping rules.
`SYS_GROUP`	Consumer group for system administrators. It is the initial consumer group for all sessions created by user accounts `SYS` or `SYSTEM`. This initial consumer group can be overridden by session-to-consumer group mapping rules.

26.10.2 Predefined Consumer Group Mapping Rules

Oracle Database includes predefined consumer group mapping rules.

Table 26-19 summarizes the consumer group mapping rules that are predefined in Oracle Database. You can verify these rules by querying the view `DBA_RSRC_GROUP_MAPPINGS`. You can use the `DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING` procedure to modify or delete any of these mapping rules.

Table 26-19 Predefined Consumer Group Mapping Rules

Attribute	Value	Mapped Consumer Group	Notes
`ORACLE_USER`	`SYS`	`SYS_GROUP`	
`ORACLE_USER`	`SYSTEM`	`SYS_GROUP`	
`ORACLE_FUNCTION`	`BACKUP`	`BATCH_GROUP`	The session is running a backup operation with RMAN. The session is automatically switched to `BATCH_GROUP` when the operation begins.
`ORACLE_FUNCTION`	`COPY`	`BATCH_GROUP`	The session is running a copy operation with RMAN. The session is automatically switched to `BATCH_GROUP` when the operation begins.
`ORACLE_FUNCTION`	`DATALOAD`	`ETL_GROUP`	The session is performing a data load operation with Data Pump. The session is automatically switched to `ETL_GROUP` when the operation begins.

See Also:

"Specifying Session-to-Consumer Group Mapping Rules"

26.10.3 Resource Manager Data Dictionary Views

You can query a set of data dictionary views for information relating to database resource management.

Table 26-20 lists views that are associated with the Resource Manager.

Table 26-20 Resource Manager Data Dictionary Views

View	Description
`DBA_RSRC_CONSUMER_GROUP_PRIVS` `USER_RSRC_CONSUMER_GROUP_PRIVS`	`DBA` view lists all resource consumer groups and the users and roles to which they have been granted. `USER` view lists all resource consumer groups granted to the user.
`DBA_RSRC_CONSUMER_GROUPS`	Lists all resource consumer groups that exist in the database.
`DBA_RSRC_MANAGER_SYSTEM_PRIVS` `USER_RSRC_MANAGER_SYSTEM_PRIVS`	`DBA` view lists all users and roles that have been granted Resource Manager system privileges. `USER` view lists all the users that are granted system privileges for the `DBMS_RESOURCE_MANAGER` package.
`DBA_RSRC_PLAN_DIRECTIVES`	Lists all resource plan directives that exist in the database.
`DBA_RSRC_PLANS`	Lists all resource plans that exist in the database.
`DBA_RSRC_GROUP_MAPPINGS`	Lists all of the various mapping pairs for all of the session attributes.
`DBA_RSRC_MAPPING_PRIORITY`	Lists the current mapping priority of each attribute.
`DBA_HIST_RSRC_PLAN`	Displays historical information about resource plan activation. This view contains AWR snapshots of `V$RSRC_PLAN_HISTORY`.
`DBA_HIST_RSRC_CONSUMER_GROUP`	Displays historical statistical information about consumer groups. This view contains AWR snapshots of `V$RSRC_CONS_GROUP_HISTORY`.
`DBA_USERS` `USER_USERS`	`DBA` view contains information about all users of the database. It contains the initial resource consumer group for each user. `USER` view contains information about the current user. It contains the current user's initial resource consumer group.
`V$RSRC_CONS_GROUP_HISTORY`	For each entry in the view `V$RSRC_PLAN_HISTORY`, contains an entry for each consumer group in the plan showing the cumulative statistics for the consumer group.
`V$RSRC_CONSUMER_GROUP`	Displays information about active resource consumer groups. This view can be used for tuning.
`V$RSRCMGRMETRIC`	Displays a history of resources consumed and cumulative CPU wait time (due to resource management) per consumer group for the past minute.
`V$RSRCMGRMETRIC_HISTORY`	Displays a history of resources consumed and cumulative CPU wait time (due to resource management) per consumer group for the past hour on a minute-by-minute basis. If a new resource plan is enabled, the history is cleared.
`V$RSRC_PLAN`	Displays the names of all currently active resource plans.
`V$RSRC_PLAN_HISTORY`	Shows when Resource Manager plans were enabled or disabled on the instance. It helps you understand how resources were shared among the consumer groups over time.

Table 26-20 (Cont.) Resource Manager Data Dictionary Views

View	Description
`V$RSRC_SESSION_INFO`	Displays Resource Manager statistics for each session. Shows how the session has been affected by the Resource Manager. Can be used for tuning.
`V$SESSION`	Lists session information for each current session. Specifically, lists the name of the resource consumer group of each current session.

27 Oracle Scheduler Concepts

You can schedule tasks with Oracle Scheduler.

- Overview of Oracle Scheduler
  Oracle Database includes Oracle Scheduler, an enterprise job scheduler to help you simplify the scheduling of hundreds or even thousands of tasks. Oracle Scheduler (the Scheduler) is implemented by the procedures and functions in the `DBMS_SCHEDULER` PL/SQL package.
- Jobs and Supporting Scheduler Objects
  You use jobs and other scheduler objects for task scheduling.
- More About Jobs
  There are different types of jobs. A job instance represents a specific run of a job. You can supply job arguments to override the default program argument values.
- Scheduler Architecture
  Scheduler components handle jobs.
- Processes to Close a PDB
  If a PDB is closed with the immediate option, then the coordinator terminates jobs running in the PDB, and the jobs must be recovered before they can run again.
- Scheduler Support for Oracle Data Guard
  Beginning with Oracle Database 11*g* Release 1 (11.1), the Scheduler can run jobs based on whether a database is a primary database or a logical standby in an Oracle Data Guard environment.

27.1 Overview of Oracle Scheduler

Oracle Database includes Oracle Scheduler, an enterprise job scheduler to help you simplify the scheduling of hundreds or even thousands of tasks. Oracle Scheduler (the Scheduler) is implemented by the procedures and functions in the `DBMS_SCHEDULER` PL/SQL package.

The Scheduler enables you to control when and where various computing tasks take place in the enterprise environment. The Scheduler helps you effectively manage and plan these tasks. By ensuring that many routine computing tasks occur without manual intervention, you can lower operating costs, implement more reliable routines, minimize human error, and shorten the time windows needed.

The Scheduler provides sophisticated, flexible enterprise scheduling functionality, which you can use to:

- Run **database program units**

  You can run program units, that is, PL/SQL anonymous blocks, PL/SQL stored procedures, and Java stored procedures on the local database or on one or more remote Oracle databases.
- Run external executables, (executables that are external to the database)

  You can run **external executables**, such as applications, shell scripts, and batch files, on the local system or on one or more remote systems. Remote systems do not require an

Oracle Database installation; they require only a Scheduler agent. Scheduler agents are available for all platforms supported by Oracle Database and some additional platforms.

- Schedule job execution using the following methods:
  - Time-based scheduling

    You can schedule a job to run at a particular date and time, either once or on a repeating basis. You can define complex repeat intervals, such as "every Monday and Thursday at 3:00 a.m. except on public holidays" or "the last Wednesday of each business quarter." See "Creating, Running, and Managing Jobs" for more information.
  - Event-based scheduling

    You can start jobs in response to system or business events. Your applications can detect events and then signal the Scheduler. Depending on the type of signal sent, the Scheduler starts a specific job. Examples of event-based scheduling include starting jobs when a file arrives on a system, when inventory falls below predetermined levels, or when a transaction fails. Beginning with Oracle Database 11*g* Release 2 (11.2), a Scheduler object called a file watcher simplifies the task of configuring a job to start when a file arrives on a local or remote system. See "Using Events to Start Jobs " for more information.
  - Dependency scheduling

    You can set the Scheduler to run tasks based on the outcome of one or more previous tasks. You can define complex dependency chains that include branching and nested chains. See "Creating and Managing Job Chains" for more information.
- Prioritize jobs based on business requirements.

  The Scheduler provides control over resource allocation among competing jobs, thus aligning job processing with your business needs. This is accomplished in the following ways:
  - Controlling Resources by Job Class

    You can group jobs that share common characteristics and behavior into larger entities called job classes. You can prioritize among the classes by controlling the resources allocated to each class. Therefore, you can ensure that your critical jobs have priority and enough resources to complete. For example, for a critical project to load a data warehouse, you can combine all the data warehousing jobs into one class and give it priority over other jobs by allocating a high percentage of the available resources to it. You can also assign relative priorities to the jobs within a job class.
  - Controlling Job Prioritization based on Schedules

    You can change job priorities based on a schedule. Because your definition of a critical job can change over time, the Scheduler also enables you to change the priority among your jobs over that time frame. For example, extract, transfer, and load (ETL) jobs used to load a data warehouse may be critical during non-peak hours but not during peak hours. Additionally, jobs that must run during the close of a business quarter may need to take priority over the ETL jobs. In these cases, you can change the priority among the job classes by changing the resources allocated to each class. See "Creating Job Classes" and "Creating Windows" for more information.
- Manage and monitor jobs

  You can manage and monitor the multiple states that jobs go through, from creation to completion. The Scheduler logs activity so that you can easily track information such as the status of the job and the last run time of the job by querying views using Oracle Enterprise Manager Cloud Control or SQL. These views provide valuable information about jobs and their execution that can help you schedule and better manage your jobs.

For example, a DBA can easily track all jobs that failed for a particular user. See "Scheduler Data Dictionary Views".

When you create a multiple-destination job, a job that is defined at one database but that runs on multiple remote hosts, you can monitor the status of the job at each destination individually or the overall status of the parent job as a whole.

For advanced job monitoring, your applications can subscribe to job state change notifications that the Scheduler delivers in event queues. The Scheduler can also send e-mail notifications when a job changes state.

See "Monitoring and Managing the Scheduler".

- Execute and manage jobs in a clustered environment

  A cluster is a set of database instances that cooperates to perform the same task. Oracle Real Application Clusters (Oracle RAC) provides scalability and reliability without any change to your applications. The Scheduler fully supports execution of jobs in such a clustered environment. To balance the load on your system and for better performance, you can also specify the database service where you want a job to run. See "The Scheduler and Real Application Clusters" for more information.

27.2 Jobs and Supporting Scheduler Objects

You use jobs and other scheduler objects for task scheduling.

- About Jobs and Supporting Scheduler Objects
  To use the Scheduler, you create *Scheduler objects*. Schema objects define the what, when, and where for job scheduling. Scheduler objects enable a modular approach to managing tasks. One advantage of the modular approach is that objects can be reused when creating new tasks that are similar to existing tasks.
- Programs
  A program object (program) describes what is to be run by the Scheduler.
- Schedules
  A schedule object (schedule) specifies when and how many times a job is run.
- Jobs
  A job describes a user-defined task.
- Destinations
  You can specify external and database destinations for running a job.
- File Watchers
  A file watcher object (file watcher) defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job.
- Credentials
  Credentials are user name and password pairs stored in a dedicated database object.
- Chains
  Chains are the means by which you can implement dependency scheduling, in which job starts depend on the outcomes of one or more previous jobs.
- Job Classes
  Job classes enable you to assign the same attributes to member jobs, set resource allocation for member jobs, and group jobs for prioritization.
- Windows
  A window is an interval of time to run a job.

- Groups
  A group designates a list of Scheduler objects.
- Incompatibilities
  An incompatibility definition (or, incompatibility) specifies incompatible jobs or programs, where only one of the group can be running at a time.

27.2.1 About Jobs and Supporting Scheduler Objects

To use the Scheduler, you create *Scheduler objects*. Schema objects define the what, when, and where for job scheduling. Scheduler objects enable a modular approach to managing tasks. One advantage of the modular approach is that objects can be reused when creating new tasks that are similar to existing tasks.

The principal Scheduler object is the job. A **job** defines the action to perform, the schedule for the action, and the location or locations where the action takes place. Most other scheduler objects are created to support jobs.

Note:

The Oracle Scheduler job replaces the `DBMS_JOB` package, which is still supported for backward compatibility. This chapter assumes that you are only using Scheduler jobs. If you are using both at once, or migrating from `DBMS_JOB` to Scheduler jobs, see Support for DBMS_JOB.

Each of these objects is described in detail later in this section.

Because Scheduler objects belong to schemas, you can grant object privileges on them. Some Scheduler objects, including job classes, windows, and window groups, are always created in the `SYS` schema, even if the user is not `SYS`. All other objects are created in the user's own schema or in a designated schema.

See Also:

"Scheduler Privileges"

27.2.2 Programs

A program object (program) describes what is to be run by the Scheduler.

A program includes:

- An action: For example, the name of a stored procedure, the name of an executable found in the operating system file system (an "external executable"), or the text of a PL/SQL anonymous block.
- A type: `STORED_PROCEDURE`, `PLSQL_BLOCK`, `SQL_SCRIPT`, `EXTERNAL_SCRIPT`, `BACKUP_SCRIPT`, or `EXECUTABLE`, where `EXECUTABLE` indicates an external executable.
- Number of arguments: The number of arguments that the stored procedure or external executable accepts.

A program is a separate entity from a job. A job runs at a certain time or because a certain event occurred, and invokes a certain program. You can create jobs that point to existing program objects, which means that different jobs can use the same program and run the program at different times and with different settings. With the right privileges, different users can use the same program without having to redefine it. Therefore, you can create program libraries, where users can select from a list of existing programs.

If a stored procedure or external executable referenced by the program accepts arguments, you define these arguments in a separate step after creating the program. You can optionally define a default value for each argument.

See Also:

- "Creating Programs"
- "Jobs" for an overview of jobs

27.2.3 Schedules

A schedule object (schedule) specifies when and how many times a job is run.

Schedules can be shared by multiple jobs. For example, the end of a business quarter may be a common time frame for many jobs. Rather than defining an end-of-quarter schedule each time a new job is defined, job creators can point to a named schedule.

There are two types of schedules:

- time schedules

  With time schedules, you can schedule jobs to run immediately or at a later time. Time schedules include a start date and time, optional end date and time, and optional repeat interval.
- event schedules

  With event schedules, you can specify that a job executes when a certain event occurs, such as inventory falling below a threshold or a file arriving on a system. For more information on events, see "Using Events to Start Jobs ".

See Also:

"Creating Schedules"

27.2.4 Jobs

A job describes a user-defined task.

- About Jobs
  A job object (job) is a collection of metadata that describes a user-defined task. It defines what must be executed (the action), when (the one-time or recurring schedule or a triggering event), where (the destinations), and with what credentials. A job has an owner, which is the schema in which it is created.

- Specifying a Job Action
  You can specify a job action by specifying the database program unit or external executable to be run or the name of an existing program object (program).
- Specifying a Job Schedule
  You can specify a job schedule by setting attributes of the job object or the name of an existing schedule object (schedule).
- Specifying a Job Destination
  You can specify a job destination in several different ways.
- Specifying a Job Credential
  You can specify a job credential by specifying a named credential object or by allowing the credential attribute of the job to remain `NULL`.

27.2.4.1 About Jobs

A job object (job) is a collection of metadata that describes a user-defined task. It defines what must be executed (the action), when (the one-time or recurring schedule or a triggering event), where (the destinations), and with what credentials. A job has an owner, which is the schema in which it is created.

A job that runs a database program unit is known as a **database job**. A job that runs an external executable is known as an **external job**.

Jobs that run database program units at one or more remote locations are called **remote database jobs**. Jobs that run external executables at one or more remote locations are called **remote external jobs**.

You define where a job runs by specifying a one or more destinations. Destinations are also Scheduler objects and are described later in this section. If you do not specify a destination, it is assumed that the job runs on the local database.

27.2.4.2 Specifying a Job Action

You can specify a job action by specifying the database program unit or external executable to be run or the name of an existing program object (program).

You specify the job action in one of the following ways:

- By specifying as a job attribute the database program unit or external executable to be run. This is known as specifying the job action **inline**.
- By specifying as a job attribute the name of an existing program, that specifies the database program unit or external executable to be run. The job owner must have the `EXECUTE` privilege on the program or the `EXECUTE ANY PROGRAM` system privilege.

27.2.4.3 Specifying a Job Schedule

You can specify a job schedule by setting attributes of the job object or the name of an existing schedule object (schedule).

You specify the job schedule in one of the following ways:

- By setting attributes of the job object to define start and end dates and a repeat interval, or to define an event that starts the job. This is known as specifying the schedule **inline**.
- By specifying as a job attribute the name of an existing schedule, which defines start and end dates and a repeat interval, or defines an event.

27.2.4.4 Specifying a Job Destination

You can specify a job destination in several different ways.

You specify the job destinations in one of the following ways:

- By specifying as a job attribute a single named destination object. In this case, the job runs on one remote location.
- By specifying as a job attribute a named destination group, which is equivalent to a list of remote locations. In this case, the job runs on all remote locations.
- By not specifying a destination attribute, in which case the job runs locally. The job runs either of the following:
  - A database program unit on the local database (the database on which the job is created)
  - An external executable on the local host, depending on the job action type

27.2.4.5 Specifying a Job Credential

You can specify a job credential by specifying a named credential object or by allowing the credential attribute of the job to remain `NULL`.

You specify the job credentials in one of the following ways:

- By specifying as a job attribute a named credential object, which contains a database user name and password (for database jobs).

  The job runs as the user named in the credential.
- By allowing the credential attribute of the job to remain `NULL`, in which case a local database job runs as the job owner. (See Table 27-1.) The job owner is the schema in which the job was created.

Note:

A local database job always runs as the user is who is the job owner and will ignore any named credential.

After you create a job and enable it, the Scheduler automatically runs the job according to its schedule or when the specified event is detected. You can view the run status of job and its job log by querying data dictionary views. If a job runs on multiple destinations, you can query the status of the job at each destination.

See Also:

- "Destinations"
- "More About Jobs"
- "Creating Jobs"
- "Scheduler Data Dictionary Views"

27.2.5 Destinations

You can specify external and database destinations for running a job.

- About Destinations
  A destination object (destination) defines a location for running a job.
- About Destinations and Scheduler Agents
  The remote location specified in a destination object must have a Scheduler agent running, and the agent must be registered with the database creating the job.

27.2.5.1 About Destinations

A destination object (destination) defines a location for running a job.

There are two types of destinations:

- External destination: Specifies a remote host name and IP address for running a remote external job.
- Database destination: Specifies a remote database instance for running a remote database job.

Jobs that run external executables (external jobs) must specify external destinations, and jobs that run database program units (database jobs) must specify database destinations.

If you specify a destination when you create a job, the job runs on that destination. If you do not specify a destination, the job runs locally, on the system on which it is created.

You can also create a destination group, which consists of a list of destinations, and reference this destination group when creating a job. In this case, the job runs on all destinations in the group.

Note:

Destination groups can also include the keyword `LOCAL` as a group member, indicating that the job also runs on the local host or local database.

See Also:

"Groups"

No object privileges are required to use a destination created by another user.

27.2.5.2 About Destinations and Scheduler Agents

The remote location specified in a destination object must have a Scheduler agent running, and the agent must be registered with the database creating the job.

The Scheduler agent enables the local Scheduler to communicate with the remote host, start and stop jobs there, and return remote job status to the local database. For complete details, see "Specifying Destinations".

- External Destinations
  You cannot explicitly create external destinations. They are created in your local database when you register a Scheduler agent with that database.
- Database Destinations
  You create database destinations with the `DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION` procedure.

27.2.5.2.1 External Destinations

You cannot explicitly create external destinations. They are created in your local database when you register a Scheduler agent with that database.

The name assigned to the external destination is the name of the agent. You can configure an agent name after you install it, or you can accept the default agent name, which is the first part of the host name (before the first dot separator). For example, if you install an agent on the host `dbhost1.us.example.com`, the agent name defaults to `DBHOST1`.

27.2.5.2.2 Database Destinations

You create database destinations with the `DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION` procedure.

Note:

If you have multiple database instances running on the local host, you can run jobs on the other instances by creating database destinations for those instances. Thus, "remote" database instances do not necessarily have to reside on remote hosts. The local host must be running a Scheduler agent to support running remote database jobs on these additional instances.

See Also:

- "Specifying Destinations"
- "Installing and Configuring the Scheduler Agent on a Remote Host"

27.2.6 File Watchers

A file watcher object (file watcher) defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job.

You create a file watcher and then create any number of event-based jobs or event schedules that reference the file watcher. When the file watcher detects the arrival of the designated file, it raises a file arrival event. The job started by the file arrival event can retrieve the event message to learn about the newly arrived file.

A file watcher can watch for a file on the local system (the same host computer running Oracle Database) or a remote system, provided that the remote system is running the Scheduler agent.

To use file watchers, the database Java virtual machine (JVM) component must be installed.

See "About File Watchers" for more information.

See Also:

"Creating File Watchers and File Watcher Jobs"

27.2.7 Credentials

Credentials are user name and password pairs stored in a dedicated database object.

Scheduler jobs use credentials to authenticate themselves with a database instance or the operating system in order to run. You use credentials for:

- Remote database jobs: The credential contains a database user name and password. The stored procedure or PL/SQL block specified in the remote database job runs as this database user.
- External jobs (local or remote): The credential contains a host operating system user name and password. The external executable of the job then runs with this user name and password.
- File watchers: The credential contains a host operating system user name and password. The job that processes the file arrival event uses this user name and password to access the arrived file.

You can query the `*_CREDENTIALS` views to see a list of credentials in the database. Credential passwords are stored obfuscated, and are not displayed in these views.

See Also:

- "Specifying Scheduler Job Credentials"
- *Oracle Database Security Guide* for information about creating a credential using the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure

27.2.8 Chains

Chains are the means by which you can implement dependency scheduling, in which job starts depend on the outcomes of one or more previous jobs.

A chain consists of multiple steps that are combined using dependency rules. The dependency rules define the conditions that can be used to start or stop a step or the chain itself. Conditions can include the success, failure, or completion- or exit-codes of previous steps. Logical expressions, such as AND/OR, can be used in the conditions. In a sense, a chain resembles a decision tree, with many possible paths for selecting which tasks run and when.

In its simplest form, a chain consists of two or more Scheduler program objects (programs) that are linked together for a single, combined objective. An example of a chain might be "run program A followed by program B, and then run program C only if programs A and B complete successfully, otherwise wait an hour and then run program D."

As an example, you might want to create a chain to combine the different programs necessary for a successful financial transaction, such as validating and approving a loan application, and then funding the loan.

A Scheduler job can point to a chain instead of pointing to a single program object. The job then serves to start the chain. This job is referred to as the **chain job**. Multiple chain jobs can point to the same chain, and more than one of these jobs can run simultaneously, thereby creating multiple instances of the same chain, each at a different point of progress in the chain.

Each position within a chain is referred to as a **step**. Typically, after an initial set of chain steps has started, the execution of successive steps depends on the completion of one or more previous steps. Each step can point to one of the following:

- A program object (program)

  The program can run a database program unit (such as a stored procedure or PL/SQL anonymous block) or an external executable.
- Another chain (a nested chain)

  Chains can be nested to any level.
- An event schedule, inline event, or file watcher

  After starting a step that points to an event schedule or that has an inline event specification, the step waits until the specified event is raised. Likewise, a step that references a file watcher inline or that points to an event schedule that references a file watcher waits until the file arrival event is raised. For a file arrival event or any other type of event, when the event occurs, the step completes, and steps that are dependent on the event step can run. A common example of an event in a chain is a user intervention, such an approval or rejection.

Multiple steps in the chain can invoke the same program or nested chain.

For each step, you can specify either a database destination or an external destination on which the step should run. If a destination is not specified, the step runs on the originating (local) database or the local host. Each step in a chain can run on a different destination.

Figure 27-1 shows a chain with multiple branches. The figure makes use of icons to indicate `BEGIN`, `END`, and a nested chain, which is Step 7, in the lower subbranch.

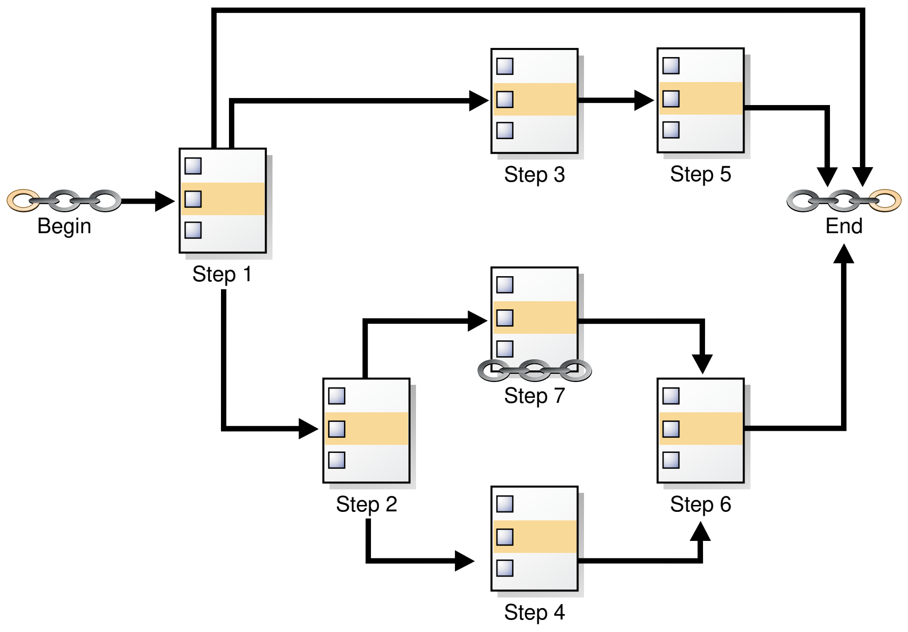
In this figure, rules could be defined as follows:

- If Step 1 completes successfully, start Step 2.

- If Step 1 fails with error code 20100, start Step 3.
- If Step 1 fails with any other error code, end the chain.

Additional rules govern the running of steps 4, 5, 6, and 7.

Figure 27-1 Chain with Multiple Branches



While a job pointing to a chain is running, the current state of all steps of the running chain can be monitored. For every step, the Scheduler creates a **step job** with the same job name and owner as the chain job. Each step job additionally has a step job subname to uniquely identify it. The step job subname is included as the `JOB_SUBNAME` column in the views `*_SCHEDULER_RUNNING_JOBS`, `*_SCHEDULER_JOB_LOG`, and `*_SCHEDULER_JOB_RUN_DETAILS`, and as the `STEP_JOB_SUBNAME` column in the `*_SCHEDULER_RUNNING_CHAINS` views.

See Also:

"Creating and Managing Job Chains"

27.2.9 Job Classes

Job classes enable you to assign the same attributes to member jobs, set resource allocation for member jobs, and group jobs for prioritization.

You typically create job classes only when you are in the role of Scheduler administrator.

Job classes provide a way to:

- Assign the same set of attribute values to member jobs

  Each job class specifies a set of attributes, such as logging level. When you assign a job to a job class, the job inherits those attributes. For example, you can specify the same policy for purging log entries for all payroll jobs.

- Set service affinity for member jobs

  You can set the `service` attribute of a job class to a desired database service name. This determines the instances in a Real Application Clusters environment that run the member jobs, and optionally, the system resources that are assigned to member jobs. See "Service Affinity when Using the Scheduler" for more information.
- Set resource allocation for member jobs

  Job classes provide the link between the Database Resource Manager and the Scheduler, because each job class can specify a resource consumer group as an attribute. Member jobs then belong to the specified consumer group and are assigned resources according to settings in the current resource plan.

  Alternatively, you can leave the `resource_consumer_group` attribute `NULL` and set the `service` attribute of a job class to a desired database service name. That service can in turn be mapped to a resource consumer group. If both the `resource_consumer_group` and `service` attributes are set, and the designated service maps to a resource consumer group, the resource consumer group named in the `resource_consumer_group` attribute takes precedence.

  See Managing Resources with Oracle Database Resource Manager for more information on mapping services to consumer groups.
- Group jobs for prioritization

  Within the same job class, you can assign priority values of 1-5 to individual jobs so that if two jobs in the class are scheduled to start at the same time, the one with the higher priority takes precedence. This ensures that you do not have a less important job preventing the timely completion of a more important one.

  If two jobs have the same assigned priority value, the job with the earlier start date takes precedence. If no priority is assigned to a job, its priority defaults to 3.

> **Note:**
>
> Job priorities are used only to prioritize among jobs in the same class.
>
> There is no guarantee that a high priority job in class A will be started before a low priority job in class B, even if they share the same schedule. Prioritizing among jobs of different classes depends on the current resource plan and on the designated resource consumer group or service name of each job class.

When defining job classes, try to classify jobs by functionality. Consider dividing jobs into groups that access similar data, such as marketing, production, sales, finance, and human resources.

The default public classes are as follows:

- `DEFAULT_JOB_CLASS`: This is the default job class for regular jobs.
- `DEFAULT_IN_MEMORY_JOB_CLASS`: This is the default job class for in-memory runtime and in-memory full jobs.
- `ORA$AUTOTASK_JOB_CLASS`: This is the job class used by autotask jobs. The resource consumer group of this job class is `ORA$AUTOTASK`.

Some of the restrictions to keep in mind are:

- A job must be part of exactly one class. When you create a job, you can specify which class the job is part of. If you do not specify a class, the job automatically becomes part of the class `DEFAULT_JOB_CLASS`.
- Dropping a class while there are still jobs in that class results in an error. You can force a class to be dropped even if there are still jobs that are members of that class, but all jobs referring to that class are then automatically disabled and assigned to the class `DEFAULT_JOB_CLASS`. Jobs belonging to the dropped class that are already running continue to run under class settings determined at the start of the job.

See Also:

- "Creating Job Classes"
- *Oracle Database Reference* to view job classes

27.2.10 Windows

A window is an interval of time to run a job.

- About Windows
  You create windows to automatically start jobs or to change resource allocation among jobs during various time periods of the day, week, and so on. A window is represented by an interval of time with a well-defined beginning and end, such as "from 12am-6am".
- Overlapping Windows
  Although Oracle does not recommend it, windows can overlap.

27.2.10.1 About Windows

You create windows to automatically start jobs or to change resource allocation among jobs during various time periods of the day, week, and so on. A window is represented by an interval of time with a well-defined beginning and end, such as "from 12am-6am".

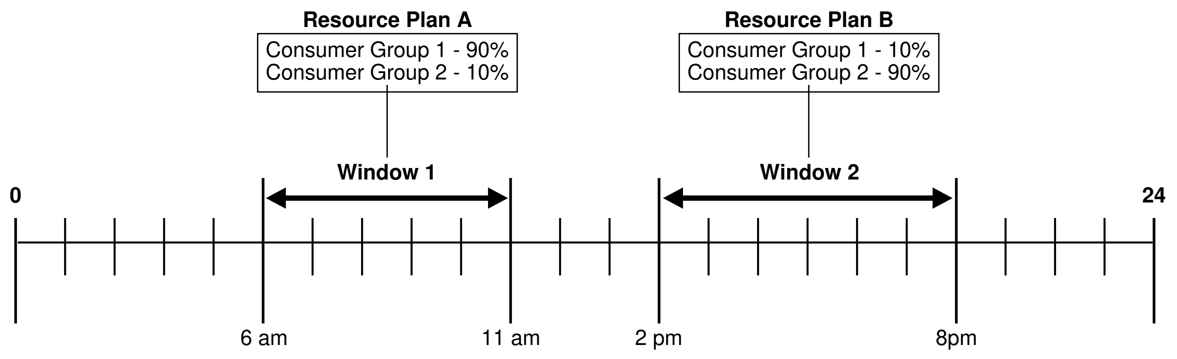
You typically create windows only when you are in the role of Scheduler administrator.

Windows work with job classes to control resource allocation. Each window specifies the resource plan to activate when the window **opens** (becomes active), and each job class specifies a resource consumer group or specifies a database service, which can map to a consumer group. A job that runs within a window, therefore, has resources allocated to it according to the consumer group of its job class and the resource plan of the window.

In a multitenant container database (CDB), there are two levels of windows. At the PDB level, windows can be used to set resource plans that allocate resources among consumer groups belonging to that PDB. At the root database level, windows can allocate resources to different PDBs. Therefore, at any time, there can be a window open in the root database and one in each PDB.

Figure 27-2 shows a workday that includes two windows. In this configuration, jobs belonging to the job class that links to `Consumer Group 1` get more resources in the morning than in the afternoon. The opposite is true for jobs in the job class that links to `Consumer Group 2`.

Figure 27-2 Windows help define the resources that are allocated to jobs



See Managing Resources with Oracle Database Resource Manager for more information on resource plans and consumer groups.

You can assign a priority to each window. If windows overlap, the window with the highest priority is chosen over other windows with lower priorities. The Scheduler automatically opens and closes windows as window start times and end times come and go.

A job can name a window in its `schedule_name` attribute. The Scheduler then starts the job when the window opens. If a window is already open, and a new job is created that points to that window, the new job does not start until the next time the window opens.

Note:

If necessary, you can temporarily block windows from switching the current resource plan. For more information, see "Enabling Oracle Database Resource Manager and Switching Plans", or the discussion of the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure in *Oracle Database PL/SQL Packages and Types Reference*.

See Also:

"Creating Windows"

27.2.10.2 Overlapping Windows

Although Oracle does not recommend it, windows can overlap.

Because only one window can be active at one time, the following rules are used to determine which window is active when windows overlap:

- If windows of the same priority overlap, the window that is active will stay open. However, if the overlap is with a window of higher priority, the lower priority window will close and the window with the higher priority will open. Jobs currently running that had a schedule naming the low priority window may be stopped depending on the behavior you assigned when you created the job.

- If, at the end of a window, there are multiple windows defined, the window with the highest priority opens. If all windows have the same priority, the window that has the highest percentage of time remaining opens.
- An open window that is dropped automatically closes. At that point, the previous rule applies.

Whenever two windows overlap, an entry is written in the Scheduler log.

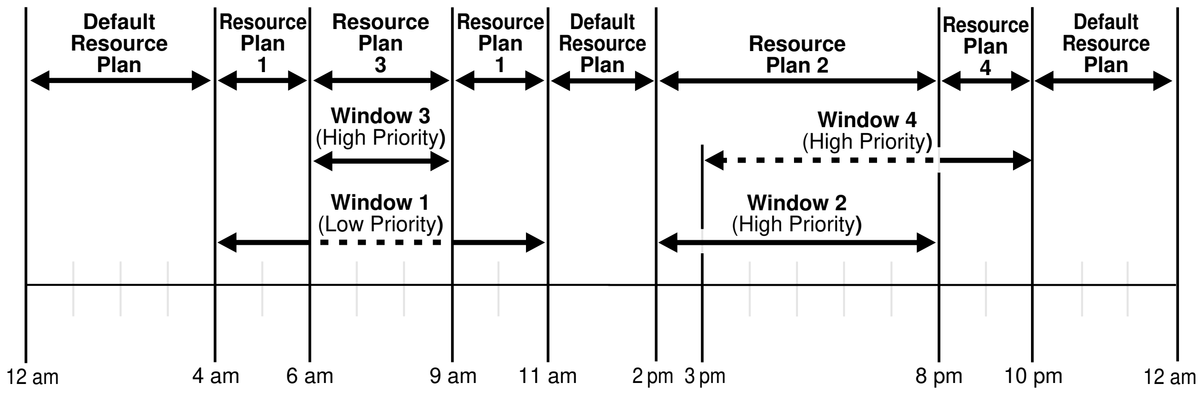
- Examples of Overlapping Windows
  Examples illustrate overlapping windows.

27.2.10.2.1 Examples of Overlapping Windows

Examples illustrate overlapping windows.

Figure 27-3 illustrates a typical example of how windows, resource plans, and priorities might be determined for a 24 hour schedule. In the following two examples, assume that Window1 has been associated with Resource Plan1, Window2 with Resource Plan2, and so on.

Figure 27-3 Windows and Resource Plans (Example 1)



In Figure 27-3, the following occurs:

- From 12AM to 4AM

  No windows are open, so a default resource plan is in effect.
- From 4AM to 6AM

  Window1 has been assigned a low priority, but it opens because there are no high priority windows. Therefore, Resource Plan 1 is in effect.
- From 6AM to 9AM

  Window3 will open because it has a higher priority than Window1, so Resource Plan 3 is in effect. The dotted line indicates Window1 is inactive.
- From 9AM to 11AM

  Even though Window1 was closed at 6AM because of a higher priority window opening, at 9AM, this higher priority window is closed and Window1 still has two hours remaining on its original schedule. It will be reopened for these remaining two hours and resource plan will be in effect.
- From 11AM to 2PM

A default resource plan is in effect because no windows are open.

- From 2PM to 3PM

  Window2 will open so Resource Plan 2 is in effect.
- From 3PM to 8PM

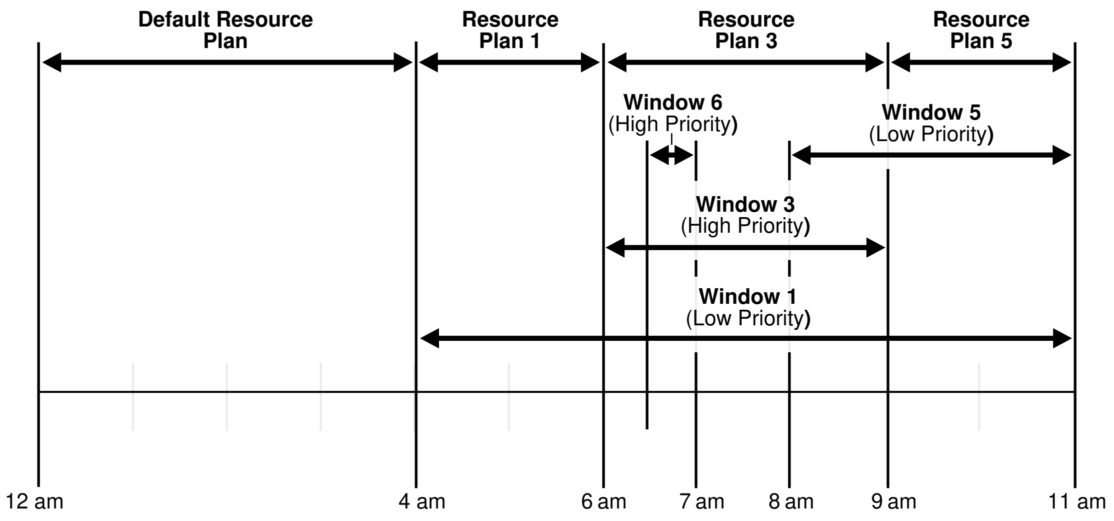
  Window4 is of the same priority as Window2, so it does not interrupt Window2 and Resource Plan 2 is in effect. The dotted line indicates Window4 is inactive.
- From 8PM to 10PM

  Window4 will open so Resource Plan 4 is in effect.
- From 10PM to 12AM

  A default resource plan is in effect because no windows are open.

Figure 27-4 illustrates another example of how windows, resource plans, and priorities might be determined for a 24 hour schedule.

Figure 27-4 Windows and Resource Plans (Example 2)



In Figure 27-4, the following occurs:

- From 12AM to 4AM

  A default resource plan is in effect.
- From 4AM to 6AM

  Window1 has been assigned a low priority, but it opens because there are no high priority windows, so Resource Plan 1 is in effect.
- From 6AM to 9AM

  Window3 will open because it has a higher priority than Window1. Note that Window6 does not open because another high priority window is already in effect.
- From 9AM to 11AM

  At 9AM, Window5 or Window1 are the two possibilities. They both have low priorities, so the choice is made based on which has a greater percentage of its duration remaining. Window5 has a larger percentage of time remaining compared to the total duration than Window1. Even if Window1 were to extend to, say, 11:30AM, Window5 would have 2/3 *

100% of its duration remaining, while Window1 would have only 2.5/7 * 100%, which is smaller. Thus, Resource Plan 5 will be in effect.

27.2.11 Groups

A group designates a list of Scheduler objects.

- About Groups
  Instead of passing a list of objects as an argument to a `DBMS_SCHEDULER` package procedure, you create a group that has those objects as its members, and then pass the group name to the procedure.
- Destination Groups
  When you want a job to run at multiple destinations, you create a database destination group or external destination group and assign it to the `destination_name` attribute of the job.
- Window Groups
  You can group windows for ease of use in scheduling jobs.

27.2.11.1 About Groups

Instead of passing a list of objects as an argument to a `DBMS_SCHEDULER` package procedure, you create a group that has those objects as its members, and then pass the group name to the procedure.

There are three types of groups:

- Database destination groups: Members are database destinations, for running remote database jobs.
- External destination groups: Members are external destinations, for running remote external jobs.
- Window groups: Members are Scheduler windows.

All members of a group must be of the same type and each member must be unique.

You create a group with the `DBMS_SCHEDULER.CREATE_GROUP` procedure.

27.2.11.2 Destination Groups

When you want a job to run at multiple destinations, you create a database destination group or external destination group and assign it to the `destination_name` attribute of the job.

Specifying a destination group as the `destination_name` attribute of a job is the only valid way to specify multiple destinations for the job.

27.2.11.3 Window Groups

You can group windows for ease of use in scheduling jobs.

You typically create window groups only when you are in the role of Scheduler administrator.

If a job must run during multiple time periods throughout the day, week, and so on, you can create a window for each time period, and then add the windows to a window group. You can then set the `schedule_name` attribute of the job to the name of this window group, and the job executes during all the time periods specified by the windows in the window group.

For example, if you had a window called "Weekends" and a window called "Weeknights," you could add these two windows to a window group called "Downtime." The data warehousing staff could then create a job to run queries according to this Downtime window group—on weeknights and weekends—when the queries could be assigned a high percentage of available resources.

If a window in a window group is already open, and a new job is created that points to that window group, the job is not started until the next window in the window group opens.

See Also:

- "Creating Destination Groups for Multiple-Destination Jobs"
- "Creating Window Groups"
- "Windows"

27.2.12 Incompatibilities

An incompatibility definition (or, incompatibility) specifies incompatible jobs or programs, where only one of the group can be running at a time.

For example, if jobs A and B are defined as incompatible, the Scheduler ensures that only one of them can be running at any given time, *even if* their respective job schedules would otherwise cause them to run at the same time.

An incompatibility can be defined at the job level (the default) or the program level. For example, assume the following:

- Jobs J1 and J2 are based on program P1.
- Jobs J3, J4, and J5 and based on program P2.
- Jobs J6 and J7 are based on program P3.

In this scenario:

- If a job-level incompatibility definition specifies J3, J4, and J5, and if job J3 is running, then J4 and J5 cannot be running until J3 finishes.
- If a program-level incompatibility definition specifies P1, P2, and P3, jobs J1 and J2 can run simultaneously (unless a job-level constraint prevents J1 and J2 from running simultaneously); however, no jobs based on programs P2 and P3 can be running until all jobs based on P1 finish.

See Also:

Using Incompatibility Definitions

27.3 More About Jobs

There are different types of jobs. A job instance represents a specific run of a job. You can supply job arguments to override the default program argument values.

- Job Categories
  Oracle Scheduler supports several types of jobs.
- Job Instances
  A job instance represents a specific run of a job. Jobs that are scheduled to run only once have only one instance. Jobs that have a repeating schedule or that run each time an event occurs have multiple instances, each run of the job representing an instance.
- Job Arguments
  When a job references a program object (program), you can supply job arguments to override the default program argument values, or provide values for program arguments that have no default value. You can also provide argument values to an inline action (for example, a stored procedure) that the job specifies.
- How Programs, Jobs, and Schedules are Related
  To define what is executed and when, you assign relationships among programs, jobs, and schedules.

See Also:

- "Creating Jobs"
- "Viewing the Job Log"

27.3.1 Job Categories

Oracle Scheduler supports several types of jobs.

- Database Jobs
  Database jobs run Oracle Database program units. You can run local and remote database jobs.
- External Jobs
  External jobs run executables outside of the database. You can run local and remote external jobs.
- Multiple-Destination Jobs
  A multiple-destination job is a job whose instances run on multiple target databases or hosts, but can be controlled and monitored from one central database.
- Chain Jobs
  The **chain** is the Scheduler mechanism that enables dependency-based scheduling.
- Detached Jobs
  You use a detached job to start a script or application that runs in a separate process, independently and asynchronously to the Scheduler.
- Lightweight Jobs
  Use lightweight jobs when you have many short-duration jobs that run frequently. Under certain circumstances, using lightweight jobs can deliver a small performance gain.
- In-Memory Jobs
  Use in-memory jobs when many jobs should be created and run during a short period of time. In-memory jobs have a slightly larger memory footprint, but use memory cache to reduce disk access and the time required for job creation and execution. Performance gains can be significant.

- Script Jobs
  Beginning with Oracle Database 12*c*, you can use several new script jobs to run custom user scripts with SQL*Plus, the RMAN interpreter, or a command shell such as `cmd.exe` for Windows and the `sh` shell or another interpreter for UNIX based systems.

27.3.1.1 Database Jobs

Database jobs run Oracle Database program units. You can run local and remote database jobs.

- About Database Jobs
  Database jobs run Oracle Database program units, including PL/SQL anonymous blocks, PL/SQL stored procedures, and Java stored procedures.
- Local Database Jobs
  A local database job runs on the originating database, as the database user who is the job owner. The job owner is the name of the schema in which the job was created.
- Remote Database Job
  The target database for a remote database job can be an Oracle database on a remote host or another database instance on the same host as the originating database.

27.3.1.1.1 About Database Jobs

Database jobs run Oracle Database program units, including PL/SQL anonymous blocks, PL/SQL stored procedures, and Java stored procedures.

For a database job where the action is specified inline, `job_type` is set to `'PLSQL_BLOCK'` or `'STORED_PROCEDURE'`, and `job_action` contains either the text of a PL/SQL anonymous block or the name of a stored procedure. (If a program is a named program object rather than program action specified inline, the corresponding `program_type` and `program_action` must be set accordingly.)

Database jobs that run on the originating database—the database on which they were created—are known as **local database jobs**, or just jobs. Database jobs that run on a target database other than the originating database are known as **remote database jobs**.

You can view run results for both local database and remote database jobs in the job log views on the originating database.

27.3.1.1.2 Local Database Jobs

A local database job runs on the originating database, as the database user who is the job owner. The job owner is the name of the schema in which the job was created.

27.3.1.1.3 Remote Database Job

The target database for a remote database job can be an Oracle database on a remote host or another database instance on the same host as the originating database.

You identify a remote database job by specifying the name of an existing database destination object in the `destination_name` attribute of the job.

Creating a remote database job requires Oracle Database 11*g* Release 2 (11.2) or later. However, the target database for the job can be any release of Oracle Database. No patch is required for the target database; you only need to install a Scheduler agent on the target database host (even if the target database host is the same as the originating database host) and register the agent with the originating database. The agent must be installed from Oracle Client 11*g* Release 2 (11.2) or later.

Remote database jobs must run as a user that is valid on the target database. You specify the required user name and password with a credential object that you assign to the remote database job.

See Also:

- "Credentials"
- "Creating Jobs"
- "Using the Oracle Scheduler Agent to Run Remote Jobs"
- "Viewing the Job Log"

27.3.1.2 External Jobs

External jobs run executables outside of the database. You can run local and remote external jobs.

- About External Jobs
  External jobs run external executables. An **external executable** is an operating system executable that runs outside the database, that is, external to the database.
- About Local External Jobs
  A local external job runs its external executable on the same computer as the Oracle database that schedules the job. For such a job, the `destination_name` job attribute is `NULL`.
- About Remote External Jobs
  A remote external job runs its external executable on a remote host. The remote host may or may not have Oracle Database installed.

27.3.1.2.1 About External Jobs

External jobs run external executables. An **external executable** is an operating system executable that runs outside the database, that is, external to the database.

For an external job, `job_type` is specified as `'EXECUTABLE'`. (If using named programs, the corresponding `program_type` would be `'EXECUTABLE'`.) The `job_action` (or corresponding `program_action`) is the full operating system–dependent path of the desired external executable, excluding any command line arguments. An example might be `/usr/local/bin/perl` or `C:\perl\bin\perl`.

Note that a Windows batch file is not directly executable and must be run a command prompt (`cmd.exe`).

Like a database job, you can assign a schema when you create the external job. That schema then becomes the job owner. Although it is possible to create an external job in the `SYS` schema, Oracle recommends against this practice.

Both the `CREATE JOB` and `CREATE EXTERNAL JOB` privileges are required to create local or remote external jobs.

External executables must run as some operating system user. Thus, the Scheduler enables you to assign operating system credentials to any external job that you create. Like remote database jobs, you specify these credentials with a credential object (a credential) and assign the credential to the external job.

There are two types of external jobs: local external jobs and remote external jobs. A **local external job** runs its external executable on the same computer as the database that schedules the job. A **remote external job** runs its executable on a remote host. The remote host does not need to have an Oracle database; you need only install and register a Scheduler agent.

Note:

On Windows, the host user that runs the external executable must be assigned the `Log on as a batch job` logon privilege.

See Also:

- "Credentials"
- "Using the Oracle Scheduler Agent to Run Remote Jobs"

27.3.1.2.2 About Local External Jobs

A local external job runs its external executable on the same computer as the Oracle database that schedules the job. For such a job, the `destination_name` job attribute is `NULL`.

Local external jobs write stdout and stderr output to log files in the directory *ORACLE_HOME*/scheduler/log. You can retrieve the contents of these files with `DBMS_SCHEDULER.GET_FILE`.

You do not have to assign a credential to a local external job, although Oracle strongly recommends that you do so for improved security. If you do not assign a credential, the job runs with default credentials. Table 27-1 shows the default credentials for different platforms and different job owners.

Table 27-1 Default Credentials for Local External Jobs

Job in SYS Schema?	Platform	Default Credentials
Yes	All	User who installed Oracle Database.
No	UNIX and Linux	Values of the `run-user` and `run-group` attributes specified in the file *ORACLE_HOME*/rdbms/admin/externaljob.ora
No	Windows	User that the `OracleJobScheduler`*`SID`* Windows service runs as (either the Local System account or a named local or domain user). Note: You must manually enable and start this service. For improved security, Oracle recommends using a named user instead of the Local System account.

Note:

Default credentials are included for compatibility with previous releases of Oracle Database, and may be deprecated in a future release. It is, therefore, best to assign a credential to every local external job.

To disable the running of local external jobs that were not assigned credentials, remove the `run_user` attribute from the *`ORACLE_HOME`*`/rdbms/admin/externaljob.ora` file (UNIX and Linux) or stop the `OracleJobScheduler` service (Windows). These steps do not disable the running of local external jobs in the `SYS` schema.

See Also:

- Your operating system–specific documentation for any post-installation configuration steps to support local external jobs
- Example 28-6

27.3.1.2.3 About Remote External Jobs

A remote external job runs its external executable on a remote host. The remote host may or may not have Oracle Database installed.

To enable remote external jobs to run on a specific remote host, you must install a Scheduler agent on the remote host and register it with the local database. The database communicates with the agent to start external executables and to retrieve execution results.

When creating a remote external job, you specify the name of an existing external destination object in the `destination_name` attribute of the job.

Remote external jobs write stdout and stderr output to log files in the directory *AGENT_HOME*/ data/log. You can retrieve the contents of these files with `DBMS_SCHEDULER.GET_FILE`. Example 28-6 illustrates how to retrieve stdout output. Although this example is for a local external job, the method is the same for remote external jobs.

See Also:

- "Credentials"
- "Using the Oracle Scheduler Agent to Run Remote Jobs"

27.3.1.3 Multiple-Destination Jobs

A multiple-destination job is a job whose instances run on multiple target databases or hosts, but can be controlled and monitored from one central database.

For DBAs or system administrators who must manage multiple databases or multiple hosts, a multiple-destination job can make administration considerably easier. With a multiple-destination job, you can:

- Specify several databases or hosts on which a job must run.
- Modify a job that is scheduled on multiple targets with a single operation.
- Stop jobs running on one or more remote targets.
- Determine the status (running, completed, failed, and so on) of the job instance at each of the remote targets.
- Determine the overall status of the collection of job instances.

A multiple-destination job can be viewed as a single entity for certain purposes and as a collection of independently running jobs for other purposes. When creating or altering the job metadata, the multiple-destination job looks like a single entity. However, when the job instances are running, they are better viewed as a collection of jobs that are nearly identical copies of each other. The job created at the source database is known as the **parent job**, and the job instances that run at the various destinations are known as **child jobs**.

You create a multiple-destination job by assigning a destination group to the `destination_name` attribute of the job. The job runs at all destinations in the group at its scheduled time, or upon the detection of a specified event. The local host can be included as one of the destinations on which the job runs.

For a job whose action is a database program unit, you must specify a database destination group in the `destination_name` attribute. The members of a database destination group include database destinations and the keyword `LOCAL`, which indicates the originating (local) database. For a job whose action is an external executable, you must specify an external destination group in the `destination_name` attribute. The members of an external destination group include external destinations and the keyword `LOCAL`, which indicates the local host.

Note:

Database destinations do not necessarily have to reference remote databases; they can reference additional database instances running on the same host as the database that creates the job.

Multiple-Destination Jobs and Time Zones

Some job destinations might be in time zones that are different from that of the database on which the parent job is created (the *originating database*). In this case, the start time of the job is always based on the time zone of the originating database. So, if you create the parent job in London, England, specify a start time of 8:00 p.m., and specify destinations at Tokyo, Los Angeles, and New York, then all child jobs start at 8:00 p.m. London time. Start times at all destinations may not be exact, due to varying system loads, issues that require retries, and so on.

Event-Based Multiple-Destination Jobs

In the case of a multiple-destination job that is event-based, when the parent job detects the event at its host, it starts all the child jobs at all destinations. The child jobs themselves do not detect events at their respective hosts.

See Also:

- "Creating Multiple-Destination Jobs"
- "Monitoring Multiple Destination Jobs"
- "Destination Groups"
- "Using Events to Start Jobs "

27.3.1.4 Chain Jobs

The **chain** is the Scheduler mechanism that enables dependency-based scheduling.

In its simplest form, it defines a group of program objects and the dependencies among them. A job can point to a chain instead of pointing to a single program object. The job then serves to start the chain. For a chain job, `job_type` is set to `'CHAIN'`.

See Also:

- "Chains"
- "Creating and Managing Job Chains"

27.3.1.5 Detached Jobs

You use a detached job to start a script or application that runs in a separate process, independently and asynchronously to the Scheduler.

A detached job typically starts another process and then exits. Upon exit (when the job action is completed) a detached job remains in the running state. The running state indicates that the asynchronous process that the job started is still active. When the asynchronous process finishes its work, it must connect to the database and call `DBMS_SCHEDULER.END_DETACHED_JOB_RUN`, which ends the job.

Detached jobs cannot be executed using `run_job` to manually trigger execution, when the `use_current_session` parameter set to `TRUE`.

A job is detached if it points to a program object (program) that has its `detached` attribute set to `TRUE` (a **detached program**).

You use a detached job under the following two circumstances:

- When it is impractical to wait for the launched asynchronous process to complete because would hold resources unnecessarily.

  An example is sending a request to an asynchronous Web service. It could take hours or days for the Web service to respond, and you do not want to hold a Scheduler job slave while waiting for the response. (See "Scheduler Architecture" for information about job slaves.)
- When it is impossible to wait for the launched asynchronous process to complete because the process shuts down the database.

An example would be using a Scheduler job to launch an RMAN script that shuts down the database, makes a cold backup, and then restarts the database. See "Creating Detached Jobs."

A detached job works as follows:

1. When it is time for the job to start, the job coordinator assigns a job slave to the job, and the job slave runs the program action defined in the detached program. The program action can be a PL/SQL block, a stored procedure, or an external executable.
2. The program action performs an immediate-return call of another script or executable, referred to here as Process A, and then exits. Because the work of the program action is complete, the job slave exits, but leaves the job in a running state.
3. Process A performs its processing. If it runs any DML against the database, it must commit its work. When processing is complete, Process A logs in to the database and calls `END_DETACHED_JOB_RUN`.
4. The detached job is logged as completed.

You can also call `STOP_JOB` to end a running detached job.

See Also:

"Creating Detached Jobs" for an example of performing a cold backup of the database with a detached job

27.3.1.6 Lightweight Jobs

Use lightweight jobs when you have many short-duration jobs that run frequently. Under certain circumstances, using lightweight jobs can deliver a small performance gain.

Lightweight jobs have the following characteristics:

- Unlike regular jobs, they are not schema objects.
- They have significantly better create and drop times over regular jobs because they do not have the overhead of creating a schema object.
- They have lower average session create time than regular jobs.
- They have a small footprint on disk for job metadata and run-time data.

You designate a lightweight job by setting the `job_style` job attribute to '`LIGHTWEIGHT`'. (The default job style is '`REGULAR`'.)

Like programs and schedules, regular jobs are schema objects. A regular job offers the maximum flexibility but does entail some overhead when it is created or dropped. The user has fine-grained control of the privileges on the job, and the job can have as its action a program or a stored procedure owned by another user.

If a relatively small number of jobs that run infrequently need to be created, then regular jobs are preferred over lightweight jobs.

A lightweight job must reference a program object (program) to specify a job action. The program must be already enabled when the lightweight job is created, and the program type must be either '`PLSQL_BLOCK`' or '`STORED_PROCEDURE`'. Because lightweight jobs are not schema objects, you cannot grant privileges on them. A lightweight job inherits privileges from its specified program. Thus, any user who has a certain set of privileges on the program has corresponding privileges on the lightweight job.

See Also:

"Creating Jobs Using a Named Program and Job Styles"

27.3.1.7 In-Memory Jobs

Use in-memory jobs when many jobs should be created and run during a short period of time. In-memory jobs have a slightly larger memory footprint, but use memory cache to reduce disk access and the time required for job creation and execution. Performance gains can be significant.

The following types of in-memory jobs are available: runtime (`IN_MEMORY_RUNTIME`) and full (`IN_MEMORY_FULL`).

- **In-memory runtime** jobs are based on Lightweight Jobs, so they are persistent. They can have a repeat interval and run multiple times.

  By default, in-memory jobs are associated to the job_class `DEFAULT_IN_MEMORY_JOB_CLASS`, which has a logging level of `NONE`. This means that by default, in-memory jobs produce no logging in the views related to Scheduler jobs, thus improving performance.
- **In-memory full** jobs exist only cached in memory, so they are not persistent. They must have a program associated, and they are meant to be run just once and discarded, so they cannot have a repeat interval. Because they do not have a backup on disk, they do not generate redo in creation or at run time, greatly speeding their operation.

  In-memory full jobs are only present in the instance where they were created or in one of the RAC instances. They are not propagated to a logical or physical standby instance, and therefore can no longer run (and will be discarded) if the primary instance is switched out for the standby.

See Also:

"Creating Jobs Using a Named Program and Job Styles"

27.3.1.8 Script Jobs

Beginning with Oracle Database 12*c*, you can use several new script jobs to run custom user scripts with SQL*Plus, the RMAN interpreter, or a command shell such as `cmd.exe` for Windows and the `sh` shell or another interpreter for UNIX based systems.

These executables all require OS credentials. These script jobs are:

- SQL Script Jobs: Requires a database destination.

  SQL script jobs use the SQL*Plus interpreter to run Scheduler jobs. Therefore, you can now use all SQL*Plus features, including query output formatting.

In order to connect to the database after spawning, SQL script jobs need an authentication step. Users can authenticate inline, in the job action, or using the `connect_credential` functionality provided by the Scheduler. To use the `connect_credential` functionality, the user sets the `connect_credential_name` attribute of a job. Then, the job attempts to connect to the database using the username, password, and role of that `connect_credential`.

- External Script Jobs: requires a normal destination

  External script jobs spawn a new shell interpreter, allowing a simple way to run command line scripts.
- Backup Script Jobs: Requires a database destination.

  Backup script jobs provide a more direct way to specify RMAN scripts that create and execute backup tasks.

  In order to connect to the database after spawning, backup script jobs need an authentication step. Users can authenticate inline, in the job action, or using the `connect_credential` functionality provided by the Scheduler. To use the `connect_credential` functionality, the user sets the `connect_credential_name` attribute of a job. Then, the job attempts to connect to the database using the username, password, and role of that `connect_credential`.

Note that job or program actions must point to an appropriate script for each interpreter or have an appropriate inline script. For further details, see the `job_action` parameters for the `CREATE_JOB` subprogram or the `program_action` parameters for the `CREATE_PROGRAM` subprogram.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for the CREATE_JOB parameters
- *Oracle Database PL/SQL Packages and Types Reference* for CREATE_PROGRAM parameters

27.3.2 Job Instances

A job instance represents a specific run of a job. Jobs that are scheduled to run only once have only one instance. Jobs that have a repeating schedule or that run each time an event occurs have multiple instances, each run of the job representing an instance.

For example, a job that is scheduled to run only on Tuesday, Oct. 8th 2009 has one instance, a job that runs daily at noon for a week has seven instances, and a job that runs when a file arrives on a remote system has one instance for each file arrival event.

Multiple-destination jobs have one instance for each destination. If a multiple-destination job has a repeating schedule, then there is one instance for each run of the job at each destination.

When a job is created, only one entry is added to the Scheduler's job table to represent the job. Depending on the logging level set, each time the job runs, an entry is added to the job log. Therefore, if you create a job that has a repeating schedule, there is one entry in the job views (`*_SCHEDULER_JOBS`) and multiple entries in the job log. Each job instance log entry provides information about a particular run, such as the job completion status and the start and end time. Each run of the job is assigned a unique log id that appears in both the job log and job run details views (`*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`).

See Also:

- "Monitoring Jobs"
- "Scheduler Data Dictionary Views"

27.3.3 Job Arguments

When a job references a program object (program), you can supply job arguments to override the default program argument values, or provide values for program arguments that have no default value. You can also provide argument values to an inline action (for example, a stored procedure) that the job specifies.

A job cannot be enabled until all required program argument values are defined, either as defaults in a referenced program object, or as job arguments.

A common example of a job is one that runs a set of nightly reports. If different departments require different reports, you can create a program for this task that can be shared among different users from different departments. The program action runs a reports script, and the program has one argument: the department number. Each user can then create a job that points to this program and can specify the department number as a job argument.

See Also:

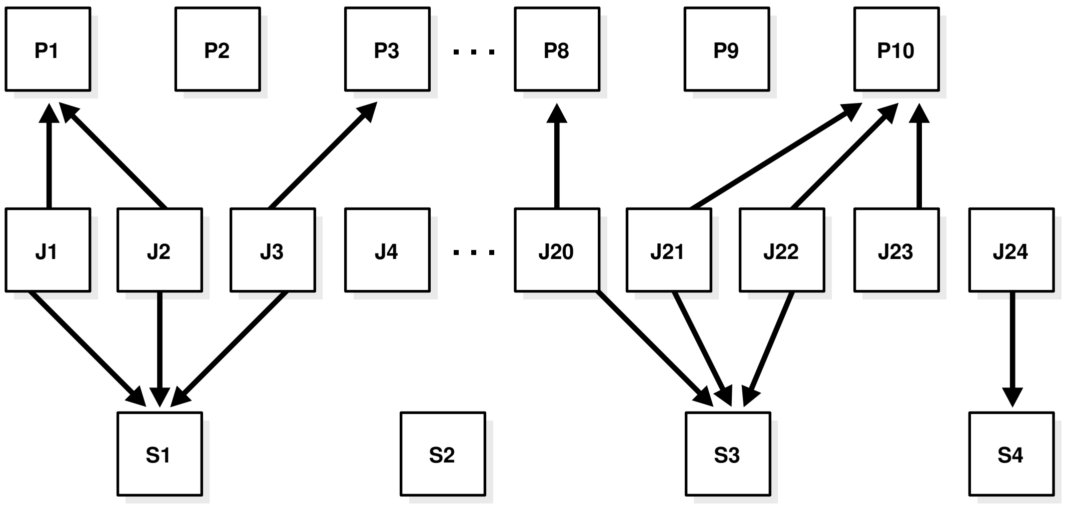
- "Setting Job Arguments"
- "Defining Program Arguments"
- "Creating Jobs"

27.3.4 How Programs, Jobs, and Schedules are Related

To define what is executed and when, you assign relationships among programs, jobs, and schedules.

Figure 27-5 illustrates examples of such relationships.

Figure 27-5 Relationships Among Programs, Jobs, and Schedules



To understand Figure 27-5, consider a situation where tables are being analyzed. In this example, program `P1` analyzes a table using the `DBMS_STATS` package. The program has an input parameter for the table name. Two jobs, `J1` and `J2`, both point to the same program, but each supplies a different table name. Additionally, schedule `S1` specifies a run time of 2:00 a.m. every day. The end result is that the two tables named in `J1` and `J2` are analyzed daily at 2:00 a.m.

Note that `J4` points to no other entity, so it is self-contained with all relevant information defined in the job itself. `P2`, `P9` and `S2` illustrate that you can leave a program or schedule unassigned if you want. You can, for example, create a program that calculates a year-end inventory and temporarily leave it unassigned to any job.

27.4 Scheduler Architecture

Scheduler components handle jobs.

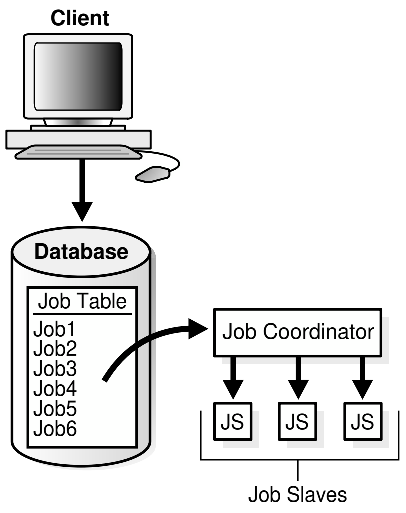
- Scheduler Components
  Scheduler components include the job table, the job coordinator, and job slaves.
- The Job Table
  The job table is a container for all the jobs, including those run from pluggable databases, with one table for each database. The job table stores information for all jobs such as the owner name or the level of logging. You can find this information in the `*_SCHEDULER_JOBS` views.
- The Job Coordinator
  The job coordinator starts job slaves.
- How Jobs Execute
  Job slaves execute the jobs you submit.
- After Jobs Complete
  The slaves perform several operations after a job completes.
- Using the Scheduler in Real Application Clusters Environments
  You can use the Scheduler in an Oracle Real Application Clusters environment.

27.4.1 Scheduler Components

Scheduler components include the job table, the job coordinator, and job slaves.

Figure 27-6 illustrates how jobs are handled by the database.

Figure 27-6 Scheduler Components



27.4.2 The Job Table

The job table is a container for all the jobs, including those run from pluggable databases, with one table for each database. The job table stores information for all jobs such as the owner name or the level of logging. You can find this information in the `*_SCHEDULER_JOBS` views.

Jobs are database objects, and therefore, can accumulate and take up too much space. To avoid this, job objects are automatically dropped by default after completion. This behavior is controlled by the `auto_drop` job attribute.

See "Scheduler Data Dictionary Views" for the available job views and administration.

27.4.3 The Job Coordinator

The job coordinator starts job slaves.

- About The Job Coordinator
  The job coordinator, under the control of the database, controls and starts job slaves, making use of the information in the job table.
- Job Coordinator Actions
  The job coordinator performs several actions.
- Maximum Number of Scheduler Job Processes
  The coordinator automatically determines how many job slaves to start based on CPU load and the number of outstanding jobs.

27.4.3.1 About The Job Coordinator

The job coordinator, under the control of the database, controls and starts job slaves, making use of the information in the job table.

The job coordinator background process (`cjqNNN`) starts automatically and stops on an as-needed basis. At database startup, the job coordinator is not started, but the database does monitor whether there are any jobs to be executed, or windows to be opened in the near future. If so, it starts the coordinator.

As long as there are jobs or windows running, the coordinator continues to run. After there has been a certain period of Scheduler inactivity and there are no jobs or windows scheduled in the near future, the coordinator is automatically stopped.

When the database determines whether to start the job coordinator, it takes the service affinity of jobs into account. For example, if there is only one job scheduled in the near future and this job belongs to a job class that has service affinity for only two out of the four Oracle RAC instances, only the job coordinators for those two instances are started. See "Service Affinity when Using the Scheduler" for more information.

27.4.3.2 Job Coordinator Actions

The job coordinator performs several actions.

The coordinator looks at the root database and all the PDBs and selects jobs based on the job priority, the job scheduled start time, and the availability of resources to run the job. The latter criterion depends on the consumer group of the job and the resource plan currently in effect. The coordinator makes no attempt to be fair to every PDB. The only way to ensure that jobs from a PDB are not starved is to allocate enough resources to it.

The job coordinator:

- Controls and spawns the job slaves
- Queries the job table
- Picks up jobs from the job table on a regular basis and places them in a memory cache. This improves performance by reducing trips to the disk
- Takes jobs from the memory cache and passes them to job slaves for execution
- Cleans up the job slave pool when slaves are no longer needed
- Goes to sleep when no jobs are scheduled
- Wakes up when a new job is about to be executed or a job was created using the `CREATE_JOB` procedure
- Upon database, startup after an abnormal database shutdown, recovers any jobs that were running.

You do not need to set the time that the job coordinator checks the job table; the system chooses the time frame automatically.

One job coordinator is used per instance. This is also the case in Oracle RAC environments.

See Also:

"Scheduler Data Dictionary Views" for job coordinator administration and "Using the Scheduler in Real Application Clusters Environments" for Oracle RAC information

27.4.3.3 Maximum Number of Scheduler Job Processes

The coordinator automatically determines how many job slaves to start based on CPU load and the number of outstanding jobs.

The `JOB_QUEUE_PROCESSES` initialization parameter can be used to limit the number of job slaves that the Scheduler can start. This parameter specifies the maximum number of job slaves per instance that can be created for the execution of `DBMS_JOB` jobs and Oracle Scheduler (`DBMS_SCHEDULER`) jobs. The range of values is 0 to 4000. The default value for `JOB_QUEUE_PROCESSES` across all containers is automatically derived from the number of sessions and CPUs configured in the system. The default is adequate for most use cases.

To limit job slaves in a CDB environment, you can set `JOB_QUEUE_PROCESSES` in the following locations:

- CDB root

  Set `JOB_QUEUE_PROCESSES` to the maximum number of slave processes that Scheduler can use simultaneously in the entire database instance.

  If `JOB_QUEUE_PROCESSES` is 0 in the CDB root, then `DBMS_JOB` and Oracle Scheduler jobs cannot run in the root or any PDB, regardless of the `JOB_QUEUE_PROCESSES` setting at the PDB level.

- PDB

  Set `JOB_QUEUE_PROCESSES` to the maximum number of simultaneous jobs for this PDB. The actual number depends on the resources assigned by Resource Manager and the demand in other containers. When multiple PDBs request jobs, Oracle Scheduler attempts to give all PDBs a fair share of the processes

  If `JOB_QUEUE_PROCESSES` is 0 in a PDB, then `DBMS_JOB` and Oracle Scheduler jobs cannot run in this PDB, regardless of the `JOB_QUEUE_PROCESSES` setting in the CDB root.

You must set all global Oracle Scheduler attributes at the PDB level. For example, if you set the `EMAIL_SENDER` attribute in the root database using `DBMS_SCHEDULER.SET_ATTRIBUTE`, then it applies to the jobs that run in the root, not the jobs running in a specific PDB. If you choose a new `EMAIL_SENDER` for a PDB, then you must set the global attribute in this PDB.

See Also:

Oracle Database Reference for more information about the `JOB_QUEUE_PROCESSES` initialization parameter

27.4.4 How Jobs Execute

Job slaves execute the jobs you submit.

They are awakened by the job coordinator when it is time for a job to be executed. They gather metadata to run the job from the job table.

When a job is picked for processing, the job slave does the following:

1. Gathers all the metadata needed to run the job, for example, program arguments and privilege information.
2. Starts a database session as the owner of the job, starts a transaction, and then starts executing the job.

   For jobs that are run from a pluggable database (PDB), the slave process switches to the PDB that the job belongs to and then executes it.
3. Once the job is complete, the slave commits and ends the transaction.
4. Closes the session.

27.4.5 After Jobs Complete

The slaves perform several operations after a job completes.

When a job is done, the slaves do the following:

- Reschedule the job if required.
- Update the state in the job table to reflect whether the job has completed or is scheduled to run again.
- Insert an entry into the job log table.
- Update the run count, and if necessary, failure and retry counts.
- Clean up.
- Look for new work (if none, they go to sleep).

The Scheduler dynamically sizes the slave pool as required.

27.4.6 Using the Scheduler in Real Application Clusters Environments

You can use the Scheduler in an Oracle Real Application Clusters environment.

- The Scheduler and Real Application Clusters
  In an Oracle Real Application Clusters (Oracle RAC) environment, the Scheduler uses one job table for each database and one job coordinator for each instance.
- Service Affinity when Using the Scheduler
  The Scheduler enables you to specify the database service under which a job should be run (service affinity).

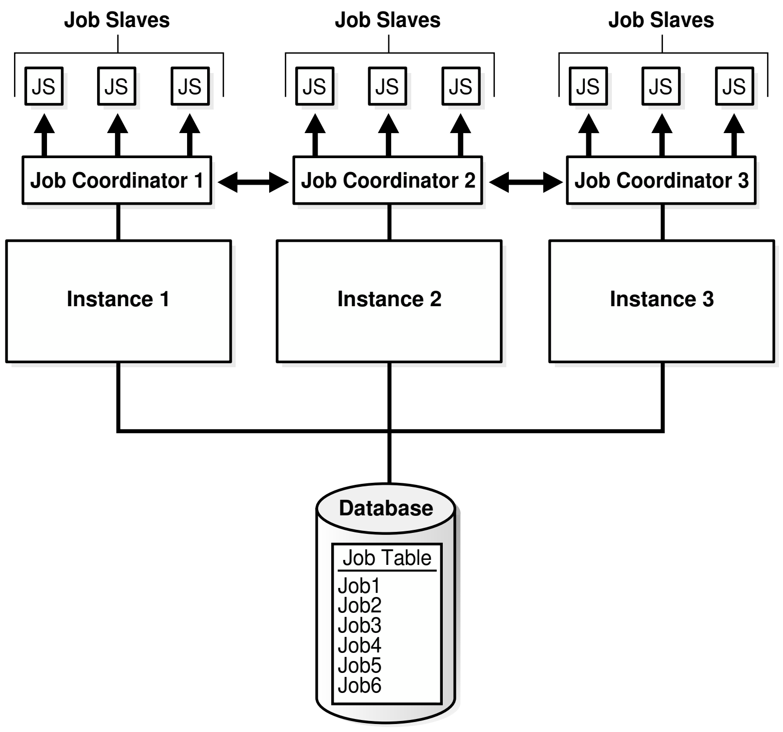
27.4.6.1 The Scheduler and Real Application Clusters

In an Oracle Real Application Clusters (Oracle RAC) environment, the Scheduler uses one job table for each database and one job coordinator for each instance.

The job coordinators communicate with each other to keep information current. The Scheduler attempts to balance the load of the jobs of a job class across all available instances when the job class has no service affinity, or across the instances assigned to a particular service when the job class does have service affinity.

Figure 27-7 illustrates a typical Oracle RAC architecture, with the job coordinator for each instance exchanging information with the others.

Figure 27-7 Oracle RAC Architecture and the Scheduler

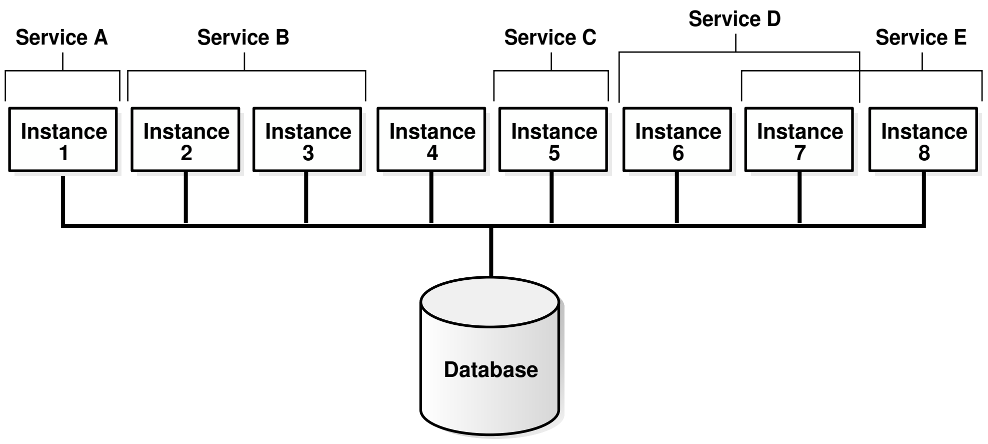


27.4.6.2 Service Affinity when Using the Scheduler

The Scheduler enables you to specify the database service under which a job should be run (service affinity).

This ensures better availability than instance affinity because it guarantees that other nodes can be dynamically assigned to the service if an instance goes down. Instance affinity does not have this capability, so, when an instance goes down, none of the jobs with an affinity to that instance can run until the instance comes back up. Figure 27-8 illustrates a typical example of how services and instances could be used.

Figure 27-8 Service Affinity and the Scheduler



In Figure 27-8, you could change the properties of the services and the Scheduler automatically recognizes the change.

Each job class can specify a database service. If a service is not specified, the job class belongs to an internal service that is guaranteed to be mapped to every running instance.

27.5 Processes to Close a PDB

If a PDB is closed with the immediate option, then the coordinator terminates jobs running in the PDB, and the jobs must be recovered before they can run again.

In an Oracle RAC database, the coordinator can, in most cases, recover the jobs on another instance where that PDB is open. So, if the coordinator on the first instance can find another instance where the PDB is still open, it moves the jobs there. In certain cases, moving the jobs to another instance may not be possible. For example, if the PDB in question is not open anywhere else, the jobs cannot be moved. Also, moving a job to another instance is not possible when the job has the `INSTANCE_ID` attribute set. In this case the job cannot run until the PDB on that instance is open again.

In a non-Oracle RAC case, the question of moving jobs does not arise. Terminated jobs can only be recovered after the PDB is opened again.

27.6 Scheduler Support for Oracle Data Guard

Beginning with Oracle Database 11*g* Release 1 (11.1), the Scheduler can run jobs based on whether a database is a primary database or a logical standby in an Oracle Data Guard environment.

For a physical standby database, any changes made to Scheduler objects or any database changes made by Scheduler jobs on the primary database are applied to the physical standby like any other database changes.

For the primary database and logical standby databases, there is additional functionality that enables you to specify that a job can run only when the database is in the role of the primary database or a logical standby. You do this using the `DBMS_SCHEDULER.SET_ATTRIBUTE` procedure to set the `database_role` job attribute to one of two values: `'PRIMARY'` or `'LOGICAL STANDBY'`. (To run a job in both roles, you can make a copy of the job and set `database_role` to `'PRIMARY'` for one job and to `'LOGICAL STANDBY'` for the other). On switchover or failover, the Scheduler automatically switches to running jobs specific to the new role. DML is replicated to the job event log so that on failover, there is an available record of what ran successfully on the primary database until it failed.

Replication of scheduler jobs from a primary to a logical standby is limited to the upgrade target in a rolling upgrade done using the `DBMS_ROLLING` package.

See Also:

- "Examples of Setting Attributes" for an example of setting the `database_role` attribute
- "Example of Creating a Job In an Oracle Data Guard Environment"
- *Oracle Data Guard Concepts and Administration*

28 Scheduling Jobs with Oracle Scheduler

You can create, run, and manage jobs with Oracle Scheduler.

Note:

This chapter describes how to use the `DBMS_SCHEDULER` package to work with Scheduler objects. You can accomplish the same tasks using Oracle Enterprise Manager Cloud Control and many of these tasks with Oracle SQL Developer.

See *Oracle Database PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` information and the Cloud Control online help for information on Oracle Scheduler pages.

- About Scheduler Objects and Their Naming
  You operate Oracle Scheduler by creating and managing a set of Scheduler objects. Each Scheduler object is a complete database schema object of the form `[schema.]name`. Scheduler objects follow the naming rules for database objects exactly and share the SQL namespace with other database objects.
- Creating, Running, and Managing Jobs
  A job is the combination of a schedule and a program, along with any additional arguments required by the program.
- Creating and Managing Programs to Define Jobs
  A program is a collection of metadata about a particular task. You optionally use a program to help define a job.
- Creating and Managing Schedules to Define Jobs
  You optionally use a schedule object (a schedule) to define when a job should be run. Schedules can be shared among users by creating and saving them as objects in the database.
- Using Events to Start Jobs
  Oracle Scheduler can start a job when an event is sent. An event is a message one application or system process sends to another.
- Creating and Managing Job Chains
  A job chain is a named series of tasks that are linked together for a combined objective.
- Using Incompatibility Definitions
  An incompatibility definition (or, incompatibility) specifies incompatible jobs or programs, where only one of the group can be running at a time.
- Managing Job Resources
  You can create and alter resources available for use by jobs, and control how many of a specified resource are available to a job.
- Prioritizing Jobs
  You prioritize Oracle Scheduler jobs using three Scheduler objects: job classes, windows, and window groups. These objects prioritize jobs by associating jobs with database resource manager consumer groups. This, in turn, controls the amount of resources

allocated to these jobs. In addition, job classes enable you to set relative priorities among a group of jobs if all jobs in the group are allocated identical resource levels.

- Monitoring Jobs
  You can monitor jobs in several different ways.

28.1 About Scheduler Objects and Their Naming

You operate Oracle Scheduler by creating and managing a set of Scheduler objects. Each Scheduler object is a complete database schema object of the form `[schema.]name`. Scheduler objects follow the naming rules for database objects exactly and share the SQL namespace with other database objects.

Follow SQL naming rules to name Scheduler objects in the `DBMS_SCHEDULER` package. By default, Scheduler object names are uppercase unless they are surrounded by double quotes. For example, when creating a job, `job_name => 'my_job'` is the same as `job_name => 'My_Job'` and `job_name => 'MY_JOB'`, but different from `job_name => '"my_job"'`. These naming rules are also followed in those cases where comma-delimited lists of Scheduler object names are used within the `DBMS_SCHEDULER` package.

See Also:

- *Oracle Database SQL Language Reference* for details regarding naming objects
- "About Jobs and Supporting Scheduler Objects"

28.2 Creating, Running, and Managing Jobs

A job is the combination of a schedule and a program, along with any additional arguments required by the program.

- Job Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common job tasks.
- Creating Jobs
  You create jobs using the `DBMS_SCHEDULER` package or Cloud Control.
- Altering Jobs
  You alter a job by modifying its attributes. You do so using the `SET_ATTRIBUTE`, `SET_ATTRIBUTE_NULL`, or `SET_JOB_ATTRIBUTES`procedures in the `DBMS_SCHEDULER` package or Cloud Control.
- Running Jobs
  A job can be run in several different ways.
- Stopping Jobs
  You stop one or more running jobs using the `STOP_JOB` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Stopping External Jobs
  The Scheduler offers implementors of external jobs a mechanism to gracefully clean up after their external jobs when `STOP_JOB` is called with `force` set to `FALSE`.

- Stopping a Chain Job
  If a job that points to a running chain is stopped, then all steps of the chain that are running are stopped.
- Dropping Jobs
  You drop one or more jobs using the `DROP_JOB` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Dropping Running Jobs
  If a job is running at the time of the `DROP_JOB` procedure call, then attempting to drop the job fails. You can modify this default behavior by setting either the `force` or `defer` option.
- Dropping Multiple Jobs
  When you specify multiple jobs to drop, the `commit_semantics` argument of the `DBMS_SCHEDULER.DROP_JOB` procedure determines the outcome if an error occurs on one of the jobs.
- Disabling Jobs
  You disable one or more jobs using the `DISABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Enabling Jobs
  You enable one or more jobs by using the `ENABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Copying Jobs
  You copy a job using the `COPY_JOB` procedure in the `DBMS_SCHEDULER` or Cloud Control.

See Also:

"Jobs" for an overview of jobs.

28.2.1 Job Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common job tasks.

Table 28-1 illustrates common job tasks and their appropriate procedures and privileges:

Table 28-1 Job Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a job	`CREATE_JOB` or `CREATE_JOBS`	`CREATE JOB` or `CREATE ANY JOB`
Alter a job	`SET_ATTRIBUTE` or `SET_JOB_ATTRIBUTES`	`ALTER` or `CREATE ANY JOB` or be the owner
Run a job	`RUN_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Copy a job	`COPY_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a job	`DROP_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Stop a job	`STOP_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Disable a job	`DISABLE`	`ALTER` or `CREATE ANY JOB` or be the owner
Enable a job	`ENABLE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "Scheduler Privileges" for further information regarding privileges.

28.2.2 Creating Jobs

You create jobs using the DBMS_SCHEDULER package or Cloud Control.

- Overview of Creating Jobs
  You create one or more jobs using the DBMS_SCHEDULER.CREATE_JOB or DBMS_SCHEDULER.CREATE_JOBS procedures or Cloud Control.
- Specifying Job Actions, Schedules, Programs, and Styles
  Because the CREATE_JOB procedure is overloaded, there are several different ways of using it.
- Specifying Scheduler Job Credentials
  Oracle Scheduler requires job credentials to authenticate with an Oracle database or the operating system before running.
- Specifying Destinations
  For remote external jobs and remote database jobs, you specify the job destination by creating a destination object and assigning it to the destination_name job attribute. A job with a NULL destination_name attribute runs on the host where the job is created.
- Creating Multiple-Destination Jobs
  You can create a job that runs on multiple destinations, but that is managed from a single location.
- Setting Job Arguments
  To set job arguments, use the SET_JOB_ARGUMENT_VALUE or SET_JOB_ANYDATA_VALUE procedures or Cloud Control. SET_JOB_ANYDATA_VALUE is used for complex data types that cannot be represented as a VARCHAR2 string.
- Setting Additional Job Attributes
  After creating a job, you can set additional job attributes or change attribute values by using the SET_ATTRIBUTE or SET_JOB_ATTRIBUTES procedures.
- Creating Detached Jobs
  A detached job must point to a program object (program) that has its detached attribute set to TRUE.
- Creating Multiple Jobs in a Single Transaction
  If you must create many jobs, then you may be able to reduce transaction overhead and experience a performance gain if you use the CREATE_JOBS procedure.
- Techniques for External Jobs
  This section contains the following examples, which demonstrate some practical techniques for external jobs.

28.2.2.1 Overview of Creating Jobs

You create one or more jobs using the DBMS_SCHEDULER.CREATE_JOB or DBMS_SCHEDULER.CREATE_JOBS procedures or Cloud Control.

You use the CREATE_JOB procedure to create a single job. This procedure is overloaded to enable you to create different types of jobs that are based on different objects. You can create multiple jobs in a single transaction using the CREATE_JOBS procedure.

You must have the CREATE JOB privilege to create a job in your own schema, and the CREATE ANY JOB privilege to create a job in any schema except SYS.

For each job being created, you specify a job type, an action, and a schedule. You can also optionally specify a credential name, a destination or destination group name, a job class, and other attributes. As soon as you enable a job, it is automatically run by the Scheduler at its next scheduled date and time. By default, jobs are disabled when created and must be enabled with `DBMS_SCHEDULER.ENABLE` to run. You can also set the `enabled` argument of the `CREATE_JOB` procedure to `TRUE`, in which case the job is ready to be automatically run, according to its schedule, as soon as you create it.

Some job attributes cannot be set with `CREATE_JOB`, and instead must be set with `DBMS_SCHEDULER.SET_ATTRIBUTE`. For example, to set the `logging_level` attribute for a job, you must call `SET_ATTRIBUTE` after calling `CREATE_JOB`.

You can create a job in another schema by specifying `schema.job_name`. The creator of a job is, therefore, not necessarily the job owner. The job owner is the user in whose schema the job is created. The NLS environment of the job, when it runs, is the existing environment at the time the job was created.

The following example demonstrates creating a database job called `update_sales`, which calls a package procedure in the `OPS` schema that updates a sales summary table:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'update_sales',
   job_type           =>  'STORED_PROCEDURE',
   job_action         =>  'OPS.SALES_PKG.UPDATE_SALES_SUMMARY',
   start_date         =>  '28-APR-08 07.00.00 PM Australia/Sydney',
   repeat_interval    =>  'FREQ=DAILY;INTERVAL=2', /* every other day */
   end_date           =>  '20-NOV-08 07.00.00 PM Australia/Sydney',
   auto_drop          =>   FALSE,
   job_class          =>  'batch_update_jobs',
   comments           =>  'My new job');
END;
/
```

Because no `destination_name` attribute is specified, the job runs on the originating (local) database. The job runs as the user who created the job.

The `repeat_interval` argument specifies that this job runs every other day until it reaches the end date and time. Another way to limit the number of times that a repeating job runs is to set its `max_runs` attribute to a positive number.

The job is disabled when it is created, by default. You must enable it with `DBMS_SCHEDULER.ENABLE` before the Scheduler will automatically run it.

Jobs are set to be automatically dropped by default after they complete. Setting the `auto_drop` attribute to `FALSE` causes the job to persist. Note that repeating jobs are not auto-dropped unless the job end date passes, the maximum number of runs (`max_runs`) is reached, or the maximum number of failures is reached (`max_failures`).

After a job is created, it can be queried using the `*_SCHEDULER_JOBS` views.

See Also:

"Specifying Scheduler Job Credentials"

28.2.2.2 Specifying Job Actions, Schedules, Programs, and Styles

Because the `CREATE_JOB` procedure is overloaded, there are several different ways of using it.

In addition to specifying the job action and job repeat interval as job attributes as shown in the example in "Overview of Creating Jobs", known as specifying the job action and job schedule *inline*, you can create a job that points to a program object (program) to specify the job action, a schedule object (schedule) to specify the repeat interval, or both a program and schedule. You can also create jobs by specifying job programs and job styles.

- Creating Jobs Using a Named Program
  You can create a job by pointing to a named program instead of inlining its action.
- Creating Jobs Using a Named Program and Job Styles
  You can create jobs using named programs and job styles. The following job styles are available: `'REGULAR'`, '`LIGHTWEIGHT`', `'IN_MEMORY_RUNTIME'`, '`IN_MEMORY_FULL`'.
- Creating Jobs Using a Named Schedule
  You can create a job by pointing to a named schedule instead of inlining its schedule.
- Creating Jobs Using Named Programs and Schedules
  A job can be created by pointing to both a named program and a named schedule.

See Also:

- "Programs"
- "Schedules"

28.2.2.2.1 Creating Jobs Using a Named Program

You can create a job by pointing to a named program instead of inlining its action.

To create a job using a named program, you specify the value for `program_name` in the `CREATE_JOB` procedure when creating the job and do not specify the values for `job_type`, `job_action`, and `number_of_arguments`.

To use an existing program when creating a job, the owner of the job must be the owner of the program or have `EXECUTE` privileges on it. The following PL/SQL block is an example of a `CREATE_JOB` procedure with a named program that creates a regular job called `my_new_job1`:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name          =>  'my_new_job1',
   program_name      =>  'my_saved_program',
   repeat_interval   =>  'FREQ=DAILY;BYHOUR=12',
   comments          =>  'Daily at noon');
END;
/
```

28.2.2.2.2 Creating Jobs Using a Named Program and Job Styles

You can create jobs using named programs and job styles. The following job styles are available: `'REGULAR'`, `'LIGHTWEIGHT'`, `'IN_MEMORY_RUNTIME'`, `'IN_MEMORY_FULL'`.

The default job style is `'REGULAR'` which is implied if no job style is provided. Examples of the other job types follow.

`LIGHTWEIGHT` Jobs

The following PL/SQL block creates a lightweight job. Lightweight jobs must reference a program, and the program type must be '`PLSQL_BLOCK`' or '`STORED_PROCEDURE`'. In addition, the program must be already enabled when you create the job.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name         =>  'my_lightweight_job1',
   program_name     =>  'polling_prog_n2',
   repeat_interval  =>  'FREQ=SECONDLY;INTERVAL=10',
   end_date         =>  '30-APR-09 04.00.00 AM Australia/Sydney',
   job_style        => 'LIGHTWEIGHT',
   comments         => 'Job that polls device n2 every 10 seconds');
END;
/
```

`IN_MEMORY_RUNTIME` Jobs

The following PL/SQL block creates an in-memory runtime job. In-memory runtime jobs have the same requirements and restrictions as lightweight jobs.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name => 'my_repeat_job',
    program_name => 'repeat_prog',
    start_date => systimestamp,
    repeat_interval => 'freq=secondly;interval=10',
    job_style => 'IN_MEMORY_RUNTIME',
    enabled => true);
END;
/
```

`IN_MEMORY_FULL` Jobs

The following PL/SQL creates an in-memory full job. In-memory full jobs require a program and cannot have a schedule or repeat interval. They run automatically when the job is enabled, and after running they are discarded.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name => 'my_immediate_job',
    program_name => 'fast_op',
    job_style => 'IN_MEMORY_FULL',
    enabled => true);
END;
/
```

See Also:

"In-Memory Jobs"

28.2.2.2.3 Creating Jobs Using a Named Schedule

You can create a job by pointing to a named schedule instead of inlining its schedule.

To create a job using a named schedule, you specify the value for `schedule_name` in the `CREATE_JOB` procedure when creating the job and do not specify the values for `start_date`, `repeat_interval`, and `end_date`.

You can use any named schedule to create a job because all schedules are created with access to `PUBLIC`. The following `CREATE_JOB` procedure has a named schedule and creates a regular job called `my_new_job2`:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name                 =>  'my_new_job2',
   job_type                 =>  'PLSQL_BLOCK',
   job_action               =>  'BEGIN SALES_PKG.UPDATE_SALES_SUMMARY; END;',
   schedule_name            =>  'my_saved_schedule');
END;
/
```

28.2.2.2.4 Creating Jobs Using Named Programs and Schedules

A job can be created by pointing to both a named program and a named schedule.

For example, the following `CREATE_JOB` procedure creates a regular job called `my_new_job3`, based on the existing program, `my_saved_program1`, and the existing schedule, `my_saved_schedule1`:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name             =>  'my_new_job3',
   program_name         =>  'my_saved_program1',
   schedule_name        =>  'my_saved_schedule1');
END;
/
```

See Also:

- "Creating and Managing Programs to Define Jobs"
- "Creating and Managing Schedules to Define Jobs"
- "Using Events to Start Jobs "

28.2.2.3 Specifying Scheduler Job Credentials

Oracle Scheduler requires job credentials to authenticate with an Oracle database or the operating system before running.

For local external jobs, remote external jobs, and remote database jobs, you must specify the credentials under which the job runs. You do so by creating a credential object and assigning it to the `credential_name` job attribute.

> **Note:**
>
> A local database job always runs as the user is who is the job owner and will ignore any named credential.

To create a credential, call the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure.

You must have the `CREATE CREDENTIAL` privilege to create a credential in your own schema, and the `CREATE ANY CREDENTIAL` privilege to create a credential in any schema except `SYS`. A credential can be used only by a job whose owner has `EXECUTE` privileges on the credential or whose owner also owns the credential. Because a credential belongs to a schema like any other schema object, you use the `GRANT` SQL statement to grant privileges on a credential.

Example 28-1 Creating a Credential

```
BEGIN
  DBMS_CREDENTIAL.CREATE_CREDENTIAL('DW_CREDENTIAL', 'dwuser', 'dW001515');
END;
/

GRANT EXECUTE ON DW_CREDENTIAL TO salesuser;
```

You can query the `*_CREDENTIALS` views to see a list of credentials in the database. Credential passwords are stored obfuscated and are not displayed in these views.

> **Note:**
>
> `*_SCHEDULER_CREDENTIALS` is deprecated in Oracle Database 12*c*, but remains available, for reasons of backward compatibility.

> **See Also:**
>
> *Oracle Database Security Guide* for information about creating a credential using the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure

28.2.2.4 Specifying Destinations

For remote external jobs and remote database jobs, you specify the job destination by creating a destination object and assigning it to the `destination_name` job attribute. A job with a `NULL` `destination_name` attribute runs on the host where the job is created.

- Destination Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer destination tasks.
- Creating Destinations
  A **destination** is a Scheduler object that defines a location for running a job.
- Creating Destination Groups for Multiple-Destination Jobs
  To create a job that runs on multiple destinations, you must create a destination group and assign that group to the `destination_name` attribute of the job.
- Example: Creating a Remote Database Job
  An example illustrates creating a remote database job.

28.2.2.4.1 Destination Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer destination tasks.

Table 28-2 illustrates destination tasks and their procedures and privileges:

Table 28-2 Destination Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create an external destination	(none)	See "Creating Destinations"
Drop an external destination	`DROP_AGENT_DESTINATION`	`MANAGE SCHEDULER`
Create a database destination	`CREATE_DATABASE_DESTINATION`	`CREATE JOB` or `CREATE ANY JOB`
Drop a database destination	`DROP_DATABASE_DESTINATION`	`CREATE ANY JOB` or be the owner
Create a destination group	`CREATE_GROUP`	`CREATE JOB` or `CREATE ANY JOB`
Drop a destination group	`DROP_GROUP`	`CREATE ANY JOB` or be the owner
Add members to a destination group	`ADD_GROUP_MEMBER`	`ALTER` or `CREATE ANY JOB` or be the owner
Remove members from a destination group	`REMOVE_GROUP_MEMBER`	`ALTER` or `CREATE ANY JOB` or be the owner

28.2.2.4.2 Creating Destinations

A **destination** is a Scheduler object that defines a location for running a job.

You designate the locations where a job runs by specifying either a single destination or a destination group in the `destination_name` attribute of the job. If you leave the `destination_name` attribute `NULL`, the job runs on the local host (the host where the job was created).

Use external destinations to specify locations where remote external jobs run. Use database destinations to specify locations where remote database jobs run.

You do not need object privileges to use a destination created by another user.

To create an external destination, register a remote Scheduler agent with the database.

See "Installing and Configuring the Scheduler Agent on a Remote Host" for instructions.

> **Note:**
>
> There is no `DBMS_SCHEDULER` package procedure to create an external destination. You create an external destination implicitly by registering a remote agent.
>
> You can also register a local Scheduler agent if you have other database instances on the same host that are targets for remote jobs. This creates an external destination that references the local host.

The external destination name is automatically set to the agent name. To verify that the external destination was created, query the views `DBA_SCHEDULER_EXTERNAL_DESTS` or `ALL_SCHEDULER_EXTERNAL_DESTS`.

To create a database destination, call the `DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION` procedure.

You must specify the name of an external destination as a procedure argument. This designates the remote host that the database destination points to. You also specify a net service name or complete connect descriptor that identifies the database instance being connected to. If you specify a net service name, it must be resolved by the local `tnsnames.ora` file. If you do not specify a database instance, the remote Scheduler agent connects to its default database, which is specified in the agent configuration file.

To create a database destination, you must have the `CREATE JOB` system privilege. To create a database destination in a schema other than your own, you must have the `CREATE ANY JOB` privilege.

Example 28-2   Creating a Database Destination

The following example creates a database destination named `DBHOST1_ORCLDW`. For this example, assume the following:

- You installed a Scheduler agent on the remote host `dbhost1.example.com`, and you registered the agent with the local database.
- You did not modify the agent configuration file to set the agent name. Therefore the agent name and the external destination name default to `DBHOST1`.
- You used Net Configuration Assistant on the local host to create a connect descriptor in tnsnames.ora for the Oracle Database instance named `orcldw`, which resides on the remote host `dbhost1.example.com`. You assigned a net service name (alias) of `ORCLDW` to this connect descriptor.

```
BEGIN
 DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION (
  destination_name     => 'DBHOST1_ORCLDW',
  agent                => 'DBHOST1',
  tns_name             => 'ORCLDW',
  comments             => 'Instance named orcldw on host dbhost1.example.com');
```

```
END;
/
```

To verify that the database destination was created, query the views `*_SCHEDULER_DB_DESTS`.

See Also:

- "Destinations" for more information about destinations
- "Jobs" to learn about remote external jobs and remote database jobs

28.2.2.4.3 Creating Destination Groups for Multiple-Destination Jobs

To create a job that runs on multiple destinations, you must create a destination group and assign that group to the `destination_name` attribute of the job.

You can specify group members (destinations) when you create the group, or you can add group members at a later time.

To create a destination group, call the `DBMS_SCHEDULER.CREATE_GROUP` procedure.

For remote external jobs you must specify a group of type '`EXTERNAL_DEST`', and all group members must be external destinations. For remote database jobs, you must specify a group of type '`DB_DEST`', and all members must be database destinations.

Members of destination groups have the following format:

```
[[schema.]credential@][schema.]destination
```

where:

- *`credential`* is the name of an existing credential.
- *`destination`* is the name of an existing database destination or external destination

The credential portion of a destination member is optional. If omitted, the job using this destination member uses its default credential.

You can include another group of the same type as a member of a destination group. Upon group creation, the Scheduler expands the included group into its members.

If you want the local host to be one of many destinations on which a job runs, you can include the keyword `LOCAL` as a group member for either type of destination group. `LOCAL` can be preceded by a credential only in an external destination group.

A group is owned by the user who creates it. You must have the `CREATE JOB` system privilege to create a group in your own schema, and the `CREATE ANY JOB` system privilege to create a group in another schema. You can grant object privileges on a group to other users by granting `SELECT` on the group.

See Also:

"Groups" for an overview of groups.

Example 28-3   Creating a Database Destination Group

This example creates a database destination group. Because some members do not include a credential, a job using this destination group must have default credentials.

```
BEGIN
  DBMS_SCHEDULER.CREATE_GROUP(
    GROUP_NAME     => 'all_dbs',
    GROUP_TYPE     => 'DB_DEST',
    MEMBER         => 'oltp_admin@orcl, orcldw1, LOCAL',
    COMMENTS       => 'All databases managed by me');
END;
/
```

The following code adds another member to the group.

```
BEGIN
  DBMS_SCHEDULER.ADD_GROUP_MEMBER(
    GROUP_NAME     => 'all_dbs',
    MEMBER         => 'dw_admin@orcldw2');
END;
/
```

28.2.2.4.4 Example: Creating a Remote Database Job

An example illustrates creating a remote database job.

The following example creates a remote database job by specifying a database destination object in the `destination_name` object of the job. A credential must also be specified so the job can authenticate with the remote database. The example uses the credential created in Example 28-1 and the database destination created in Example 28-2.

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'SALES_SUMMARY1',
   job_type            =>  'STORED_PROCEDURE',
   job_action          =>  'SALES.SALES_REPORT1',
   start_date          =>  '15-JUL-09 11.00.00 PM Europe/Warsaw',
   repeat_interval     =>  'FREQ=DAILY',
   credential_name     =>  'DW_CREDENTIAL',
   destination_name    =>  'DBHOST1_ORCLDW');
END;
/
```

28.2.2.5 Creating Multiple-Destination Jobs

You can create a job that runs on multiple destinations, but that is managed from a single location.

A typical reason to do this is to run a database maintenance job on all of the databases that you administer. Rather than create the job on each database, you create the job once and designate multiple destinations for the job. From the database where you created the job (the *local database*), you can monitor the state and results of all instances of the job at all locations.

To create a multiple-destination job:

- Call the `DBMS_SCHEDULER.CREATE_JOB` procedure and set the `destination_name` attribute of the job to the name of database destination group or external destination group.

  If not all destination group members include a credential prefix (the schema), assign a default credential to the job.

To include the local host or local database as one of the destinations on which the job runs, ensure that the keyword `LOCAL` is one of the members of the destination group.

To obtain a list of destination groups, submit this query:

```
SELECT owner, group_name, group_type, number_of_members FROM all_scheduler_groups
  WHERE group_type = 'DB_DEST' or group_type = 'EXTERNAL_DEST';

OWNER           GROUP_NAME      GROUP_TYPE    NUMBER_OF_MEMBERS
--------------- --------------- ------------- -----------------
DBA1            ALL_DBS         DB_DEST                       4
DBA1            ALL_HOSTS       EXTERNAL_DEST                 4
```

The following example creates a multiple-destination database job, using the database destination group created in Example 28-3. The user specified in the credential should have sufficient privileges to perform the job action.

```
BEGIN
 DBMS_CREDENTIAL.CREATE_CREDENTIAL('DBA_CREDENTIAL', 'dba1', 'sYs040533');
 DBMS_SCHEDULER.CREATE_JOB (
   job_name             =>  'MAINT_SET1',
   job_type             =>  'STORED_PROCEDURE',
   job_action           =>  'MAINT_PROC1',
   start_date           =>  '15-JUL-09 11.00.00 PM Europe/Warsaw',
   repeat_interval      =>  'FREQ=DAILY',
   credential_name      =>  'DBA_CREDENTIAL',
   destination_name     =>  'ALL_DBS');
END;
/
```

See Also:

- "Multiple-Destination Jobs"
- "Monitoring Multiple Destination Jobs"
- "Groups"

28.2.2.6 Setting Job Arguments

To set job arguments, use the `SET_JOB_ARGUMENT_VALUE` or `SET_JOB_ANYDATA_VALUE` procedures or Cloud Control. `SET_JOB_ANYDATA_VALUE` is used for complex data types that cannot be represented as a `VARCHAR2` string.

After creating a job, you may need to set job arguments if:

- The inline job action is a stored procedure or other executable that requires arguments
- The job references a named program object and you want to override one or more default program arguments
- The job references a named program object and one or more of the program arguments were not assigned a default value

An example of a job that might need arguments is one that starts a reporting program that requires a start date and end date. The following code example sets the end date job argument, which is the second argument expected by the reporting program:

```
BEGIN
  DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
   job_name                => 'ops_reports',
   argument_position       => 2,
   argument_value          => '12-DEC-03');
END;
/
```

If you use this procedure on an argument whose value has already been set, it will be overwritten. You can set argument values using either the argument name or the argument position. To use argument name, the job must reference a named program object, and the argument must have been assigned a name in the program object. If a program is inlined, only setting by position is supported. Arguments are not supported for jobs of type 'PLSQL_BLOCK'.

To remove a value that has been set, use the RESET_JOB_ARGUMENT procedure. This procedure can be used for both regular and ANYDATA arguments.

SET_JOB_ARGUMENT_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

See Also:

"Defining Program Arguments"

28.2.2.7 Setting Additional Job Attributes

After creating a job, you can set additional job attributes or change attribute values by using the SET_ATTRIBUTE or SET_JOB_ATTRIBUTES procedures.

You can also set job attributes with Cloud Control. Although many job attributes can be set with the call to CREATE_JOB, some attributes, such as destination and credential_name, can be set only with SET_ATTRIBUTE or SET_JOB_ATTRIBUTES after the job has been created.

28.2.2.8 Creating Detached Jobs

A detached job must point to a program object (program) that has its detached attribute set to TRUE.

The following example for Linux and UNIX creates a nightly job that performs a cold backup of the database. It contains three steps.

Step 1—Create the Script That Invokes RMAN

Create a shell script that calls an RMAN script to perform a cold backup. The shell script is in ORACLE_HOME/scripts/coldbackup.sh. It must be executable by the user who installed Oracle Database (typically the user oracle).

```
#!/bin/sh

export ORACLE_HOME=/u01/app/oracle/product/database_release_number/db_1
export ORACLE_SID=orcl
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

$ORACLE_HOME/bin/rman TARGET / @$ORACLE_HOME/scripts/coldbackup.rman
  trace /u01/app/oracle/backup/coldbackup.out &
exit 0
```

Step 2—Create the RMAN Script

Create an RMAN script that performs the cold backup and then ends the job. The script is in ORACLE_HOME/scripts/coldbackup.rman.

```
run {
# Shut down database for backups and put into MOUNT mode
shutdown immediate
startup mount

# Perform full database backup
backup full format "/u01/app/oracle/backup/%d_FULL_%U" (database) ;

# Open database after backup
alter database open;

# Call notification routine to indicate job completed successfully
sql " BEGIN  DBMS_SCHEDULER.END_DETACHED_JOB_RUN(''sys.backup_job'', 0,
  null); END; ";
}
```

Step 3—Create the Job and Use a Detached Program

Submit the following PL/SQL block:

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name   => 'sys.backup_program',
    program_type   => 'executable',
    program_action => '?/scripts/coldbackup.sh',
    enabled        =>  TRUE);

  DBMS_SCHEDULER.SET_ATTRIBUTE('sys.backup_program', 'detached', TRUE);

  DBMS_SCHEDULER.CREATE_JOB(
    job_name        => 'sys.backup_job',
    program_name    => 'sys.backup_program',
    repeat_interval => 'FREQ=DAILY;BYHOUR=1;BYMINUTE=0');

  DBMS_SCHEDULER.ENABLE('sys.backup_job');
END;
/
```

See Also:

"Detached Jobs"

28.2.2.9 Creating Multiple Jobs in a Single Transaction

If you must create many jobs, then you may be able to reduce transaction overhead and experience a performance gain if you use the CREATE_JOBS procedure.

Example 28-4 demonstrates how to use this procedure to create multiple jobs in a single transaction.

Example 28-4 Creating Multiple Jobs in a Single Transaction

```
DECLARE
 newjob sys.job_definition;
 newjobarr sys.job_definition_array;
BEGIN
 -- Create an array of JOB_DEFINITION object types
 newjobarr := sys.job_definition_array();

 -- Allocate sufficient space in the array
 newjobarr.extend(5);

 -- Add definitions for 5 jobs
 FOR i IN 1..5 LOOP
   -- Create a JOB_DEFINITION object type
   newjob := sys.job_definition(job_name => 'TESTJOB' || to_char(i),
                     job_style => 'REGULAR',
                     program_name => 'PROG1',
                     repeat_interval => 'FREQ=HOURLY',
                     start_date => systimestamp + interval '600' second,
                     max_runs => 2,
                     auto_drop => FALSE,
                     enabled => TRUE
                    );

   -- Add it to the array
   newjobarr(i) := newjob;
 END LOOP;

 -- Call CREATE_JOBS to create jobs in one transaction
 DBMS_SCHEDULER.CREATE_JOBS(newjobarr, 'TRANSACTIONAL');
END;
/

PL/SQL procedure successfully completed.

SELECT JOB_NAME FROM USER_SCHEDULER_JOBS;

JOB_NAME
------------------------------
TESTJOB1
TESTJOB2
TESTJOB3
TESTJOB4
TESTJOB5

5 rows selected.
```

See Also:

"Lightweight Jobs"

28.2.2.10 Techniques for External Jobs

This section contains the following examples, which demonstrate some practical techniques for external jobs.

Example 28-5   Creating a Local External Job That Runs a Command Interpreter

This example demonstrates how to create a local external job on Windows that runs an interpreter command (in this case, `mkdir`). The job runs `cmd.exe` with the `/c` option.

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
   job_name             => 'MKDIR_JOB',
   job_type             => 'EXECUTABLE',
   number_of_arguments  => 3,
   job_action           => '\windows\system32\cmd.exe',
   auto_drop            => FALSE,
   credential_name      => 'TESTCRED');

 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',1,'/c');
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',2,'mkdir');
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',3,'\temp\extjob_test_dir');
 DBMS_SCHEDULER.ENABLE('MKDIR_JOB');
END;
/
```

Example 28-6   Creating a Local External Job and Viewing the Job Output

This example for Linux and UNIX shows how to create and run a local external job and then view the job output. When an external job runs, the Scheduler automatically retrieves the output from the job and stores it inside the database.

To see the output, query `*_SCHEDULER_JOB_RUN_DETAILS` views.

```
-- User scott must have CREATE JOB, CREATE CREDENTIAL, and CREATE EXTERNAL JOB
-- privileges
GRANT CREATE JOB, CREATE EXTERNAL JOB TO scott ;

CONNECT scott/password
SET SERVEROUTPUT ON

-- Create a credential for the job to use
exec DBMS_CREDENTIAL.CREATE_CREDENTIAL('my_cred','host_username','host_passwd')

-- Create a job that lists a directory. After running, the job is dropped.
BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
  job_name            => 'lsdir',
  job_type            => 'EXECUTABLE',
  job_action          => '/bin/ls',
  number_of_arguments => 1,
  enabled             => false,
  auto_drop           => true,
  credential_name     => 'my_cred');
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('lsdir',1,'/tmp');
 DBMS_SCHEDULER.ENABLE('lsdir');
END;
/

-- Wait a bit for the job to run, and then check the job results.
SELECT job_name, status, error#, actual_start_date, additional_info
 FROM user_scheduler_job_run_details WHERE job_name='LSDIR';

-- Now use the external log id from the additional_info column to
-- formulate the log file name and retrieve the output
DECLARE
 my_clob clob;
 log_id varchar2(50);
```

```
BEGIN
 SELECT regexp_substr(additional_info,'job[_0-9]*') INTO log_id
   FROM user_scheduler_job_run_details WHERE job_name='LSDIR';
 DBMS_LOB.CREATETEMPORARY(my_clob, false);

SELECT job_name, status, error#, errors, output FROM user_scheduler_job_run_details
WHERE job_name = 'LSDIR';
END;
/
```

See Also:

- *Oracle Database Security Guide* for more information about external authentication
- "External Jobs"
- "Stopping External Jobs"
- "Troubleshooting Remote Jobs"

28.2.3 Altering Jobs

You alter a job by modifying its attributes. You do so using the `SET_ATTRIBUTE`, `SET_ATTRIBUTE_NULL`, or `SET_JOB_ATTRIBUTES`procedures in the `DBMS_SCHEDULER` package or Cloud Control.

See the `CREATE_JOB` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details on job attributes.

All jobs can be altered, and, except for the job name, all job attributes can be changed. If there is a running instance of the job when the change is made, it is not affected by the call. The change is only seen in future runs of the job.

In general, you should not alter a job that was automatically created for you by the database. Jobs that were created by the database have the column `SYSTEM` set to `TRUE` in job views. The attributes of a job are available in the `*_SCHEDULER_JOBS` views.

It is valid for running jobs to alter their own job attributes. However, these changes do not take effect until the next scheduled run of the job.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE`, `SET_ATTRIBUTE_NULL`, and `SET_JOB_ATTRIBUTES` procedures.

The following example changes the `repeat_interval` of the job `update_sales` to once per week on Wednesday.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name         =>  'update_sales',
   attribute    =>  'repeat_interval',
   value        =>  'freq=weekly; byday=wed');
END;
/
```

28.2.4 Running Jobs

A job can be run in several different ways.

There are three ways in which a job can be run:

- According to the job schedule—In this case, provided that the job is enabled, the job is automatically picked up by the Scheduler job coordinator and run under the control of a job slave. The job runs as the user who is the job owner, or in the case of a local external job with a credential, as the user named in the credential. To find out whether the job succeeded, you must query the job views (`*_SCHEDULER_JOBS`) or the job log (`*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`). See "How Jobs Execute" for more information job slaves and the Scheduler architecture.
- When an event occurs—Enabled event-based jobs start when a specified event is received on an event queue or when a file watcher raises a file arrival event. (See "Using Events to Start Jobs ".) Event-based jobs also run under the control of a job slave and run as the user who owns the job, or in the case of a local external job with a credential, as the user named in the credential. To find out whether the job succeeded, you must query the job views or the job log.
- By calling `DBMS_SCHEDULER.RUN_JOB`—You can use the `RUN_JOB` procedure to test a job or to run it outside of its specified schedule. You can run the job asynchronously, which is similar to the previous two methods of running a job, or synchronously, in which the job runs in the session that called `RUN_JOB`, and as the user logged in to that session. The `use_current_session` argument of `RUN_JOB` determines whether a job runs synchronously or asynchronously.

  `RUN_JOB` accepts a comma-delimited list of job names.

  The following example asynchronously runs two jobs:

  ```
  BEGIN
    DBMS_SCHEDULER.RUN_JOB(
      JOB_NAME            => 'DSS.ETLJOB1, DSS.ETLJOB2',
      USE_CURRENT_SESSION => FALSE);
  END;
  /
  ```

> **Note:**
>
> It is not necessary to call `RUN_JOB` to run a job according to its schedule. Provided that job is enabled, the Scheduler runs it automatically.

28.2.5 Stopping Jobs

You stop one or more running jobs using the `STOP_JOB` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

`STOP_JOB` accepts a comma-delimited list of jobs, job classes, and job destination IDs. A **job destination ID** is a number, assigned by the Scheduler, that represents a unique combination of a job, a credential, and a destination. It serves as a convenient method for identifying a particular child job of a multiple-destination job and for stopping just that child. You obtain the job destination ID for a child job from the `*_SCHEDULER_JOB_DESTS` views.

If a job class is supplied, all running jobs in the job class are stopped. For example, the following statement stops job `job1`, all jobs in the job class `dw_jobs`, and two child jobs of a multiple-destination job:

```
BEGIN
  DBMS_SCHEDULER.STOP_JOB('job1, sys.dw_jobs, 984, 1223');
END;
/
```

All instances of the designated jobs are stopped. After stopping a job, the state of a one-time job is set to `STOPPED`, and the state of a repeating job is set to `SCHEDULED` (because the next run of the job is scheduled). In addition, an entry is made in the job log with `OPERATION` set to '`STOPPED`', and `ADDITIONAL_INFO` set to '`REASON="Stop job called by user: `*`username`*`"`'.

By default, the Scheduler tries to gracefully stop a job using an interrupt mechanism. This method gives control back to the slave process, which can collect statistics of the job run. If the `force` option is set to `TRUE`, the job is abruptly terminated and certain run-time statistics might not be available for the job run.

Stopping a job that is running a chain automatically stops all running steps (by calling `STOP_JOB` with the `force` option set to `TRUE` on each step).

You can use the `commit_semantics` argument of `STOP_JOB` to control the outcome if multiple jobs are specified and errors occur when trying to stop one or more jobs. If you set this argument to `ABSORB_ERRORS`, the procedure may be able to continue after encountering an error and attempt to stop the remaining jobs. If the procedure indicates that errors occurred, you can query the view `SCHEDULER_BATCH_ERRORS` to determine the nature of the errors. See "Dropping Jobs" for a more detailed discussion of commit semantics.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `STOP_JOB` procedure.

Note:

When a job is stopped, only the current transaction is rolled back. This can cause data inconsistency.

28.2.6 Stopping External Jobs

The Scheduler offers implementors of external jobs a mechanism to gracefully clean up after their external jobs when `STOP_JOB` is called with `force` set to `FALSE`.

The mechanism described in this section applies only to remote external jobs on the UNIX and Linux platforms.

On UNIX and Linux, a `SIGTERM` signal is sent to the process launched by the Scheduler. The implementor of the external job is expected to trap the `SIGTERM` in an interrupt handler, clean up whatever work the job has done, and exit.

On Windows, `STOP_JOB` with `force` set to `FALSE` is supported. The process launched by the Scheduler is a console process. To stop it, the Scheduler sends a `CTRL+BREAK` to the process. The `CTRL+BREAK` can be handled by registering a handler with the `SetConsoleCtrlHandler()` routine.

28.2.7 Stopping a Chain Job

If a job that points to a running chain is stopped, then all steps of the chain that are running are stopped.

See "Stopping Individual Chain Steps" for information about stopping individual chain steps.

28.2.8 Dropping Jobs

You drop one or more jobs using the `DROP_JOB` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

`DROP_JOB` accepts a comma-delimited list of jobs and job classes. If a job class is supplied, all jobs in the job class are dropped, although the job class itself is not dropped. You cannot use job destination IDs with `DROP_JOB` to drop the child of a multiple-destination job.

Use the `DROP_JOB_CLASS` procedure to drop a job class, as described in "Dropping Job Classes".

The following statement drops jobs `job1` and `job3`, and all jobs in job classes `jobclass1` and `jobclass2`:

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB ('job1, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

28.2.9 Dropping Running Jobs

If a job is running at the time of the `DROP_JOB` procedure call, then attempting to drop the job fails. You can modify this default behavior by setting either the `force` or `defer` option.

When you set the `force` option to `TRUE`, the Scheduler first attempts to stop the running job by using an interrupt mechanism, calling `STOP_JOB` with the `force` option set to `FALSE`. If the job stops successfully, it is then dropped. Alternatively, you can first call `STOP_JOB` to stop the job and then call `DROP_JOB`. If `STOP_JOB` fails, you can call `STOP_JOB` with the `force` option, provided you have the `MANAGE SCHEDULER` privilege. You can then drop the job. By default, `force` is set to `FALSE` for both the `STOP_JOB` and `DROP_JOB` procedures.

When you set the `defer` option to `TRUE`, the running job is allowed to complete and then dropped. The `force` and `defer` options are mutually exclusive; setting both results in an error.

28.2.10 Dropping Multiple Jobs

When you specify multiple jobs to drop, the `commit_semantics` argument of the `DBMS_SCHEDULER.DROP_JOB` procedure determines the outcome if an error occurs on one of the jobs.

Possible values for this argument are:

- `STOP_ON_FIRST_ERROR`, the default—The call returns on the first error and commits previous successful drop operations to disk.
- `TRANSACTIONAL`—The call returns on the first error and rolls back previous drop operations before the error. `force` must be `FALSE`.

- ABSORB_ERRORS—The call tries to absorb any errors, attempts to drop the rest of the jobs, and commits all the drops that were successful.

Setting commit_semantics is valid only when no job classes are included in the job_name list. When you include job classes, default commit semantics (STOP_ON_FIRST_ERROR) are in effect.

The following example drops the jobs myjob1 and myjob2 with the defer option and uses transactional commit semantics:

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB(
     job_name         => 'myjob1, myjob2',
     defer            => TRUE,
     commit_semantics => 'TRANSACTIONAL');
END;
/
```

This next example illustrates the ABSORB_ERRORS commit semantics. Assume that myjob1 is running when the procedure is called and that myjob2 is not.

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB(
     job_name         => 'myjob1, myjob2',
     commit_semantics => 'ABSORB_ERRORS');
END;
/
Error report:
ORA-27362: batch API call completed with errors
```

You can query the view SCHEDULER_BATCH_ERRORS to determine the nature of the errors.

```
SELECT object_name, error_code, error_message FROM scheduler_batch_errors;

OBJECT_NAME    ERROR CODE ERROR_MESSAGE
-------------- ---------- ---------------------------------------------------
STEVE.MYJOB1        27478 "ORA-27478: job "STEVE.MYJOB1" is running
```

Checking USER_SCHEDULER_JOBS, you would find that myjob2 was successfully dropped and that myjob1 is still present.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DROP_JOB procedure.

28.2.11 Disabling Jobs

You disable one or more jobs using the DISABLE procedure in the DBMS_SCHEDULER package or Cloud Control.

Jobs can also become disabled by other means. For example, dropping a job class disables the class jobs. Dropping either the program or the schedule that jobs point to, disables the jobs. However, disabling either the program or the schedule that jobs point to does not disable the jobs, and therefore, results in errors when the Scheduler tries to run them.

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator does not pick up these jobs for processing. When a job is disabled, its state in the job table is changed to disabled.

When a currently running job is disabled with the force option set to FALSE, an error returns. When force is set to TRUE, the job is disabled, but the currently running instance is allowed to finish.

If `commit_semantics` is set to `STOP_ON_FIRST_ERROR`, then the call returns on the first error and the previous successful disable operations are committed to disk. If `commit_semantics` is set to `TRANSACTIONAL` and `force` is set to `FALSE`, then the call returns on the first error and rolls back the previous disable operations before the error. If `commit_semantics` is set to `ABSORB_ERRORS`, then the call tries to absorb any errors and attempts to disable the rest of the jobs and commits all the successful disable operations. If the procedure indicates that errors occurred, you can query the view `SCHEDULER_BATCH_ERRORS` to determine the nature of the errors.

By default, `commit_semantics` is set to `STOP_ON_FIRST_ERROR`.

You can also disable several jobs in one call by providing a comma-delimited list of job names or job class names to the `DISABLE` procedure call. For example, the following statement combines jobs with job classes:

```
BEGIN
  DBMS_SCHEDULER.DISABLE('job1, job2, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DISABLE` procedure.

28.2.12 Enabling Jobs

You enable one or more jobs by using the `ENABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

The effect of this procedure is that the job will be picked up by the job coordinator for processing. Jobs are created disabled by default, so you must enable them before they can run. When a job is enabled, a validity check is performed. If the check fails, the job is not enabled.

If you enable a disabled job, it begins to run immediately according to its schedule. Enabling a disabled job also resets the job `RUN_COUNT`, `FAILURE_COUNT`, and `RETRY_COUNT` attributes.

If `commit_semantics` is set to `STOP_ON_FIRST_ERROR`, then the call returns on the first error and the previous successful enable operations are committed to disk. If `commit_semantics` is set to `TRANSACTIONAL`, then the call returns on the first error and the previous enable operations before the error are rolled back. If `commit_semantics` is set to `ABSORB_ERRORS`, then the call tries to absorb any errors and attempts to enable the rest of the jobs and commits all the successful enable operations. If the procedure indicates that errors occurred, you can query the view `SCHEDULER_BATCH_ERRORS` to determine the nature of the errors.

By default, `commit_semantics` is set to `STOP_ON_FIRST_ERROR`.

You can enable several jobs in one call by providing a comma-delimited list of job names or job class names to the `ENABLE` procedure call. For example, the following statement combines jobs with job classes:

```
BEGIN
 DBMS_SCHEDULER.ENABLE ('job1, job2, job3,
   sys.jobclass1, sys.jobclass2, sys.jobclass3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `ENABLE` procedure.

28.2.13 Copying Jobs

You copy a job using the `COPY_JOB` procedure in the `DBMS_SCHEDULER` or Cloud Control.

This call copies all the attributes of the old job to the new job (except job name). The new job is created disabled.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `COPY_JOB` procedure.

28.3 Creating and Managing Programs to Define Jobs

A program is a collection of metadata about a particular task. You optionally use a program to help define a job.

- Program Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common program tasks.
- Creating Programs with Scheduler
  A program describes what is to be run by the Scheduler.
- Altering Programs
  You alter a program by modifying its attributes. You can use Cloud Control or the `DBMS_SCHEDULER.SET_ATTRIBUTE` and `DBMS_SCHEDULER.SET_ATTRIBUTE_NULL` package procedures to alter programs.
- Dropping Programs
  You drop one or more programs using the `DROP_PROGRAM` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Disabling Programs
  You disable one or more programs using the `DISABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Enabling Programs
  You enable one or more programs using the `ENABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

See Also:

"Programs" for an overview of programs.

28.3.1 Program Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common program tasks.

Table 28-3 illustrates common program tasks and their appropriate procedures and privileges:

Table 28-3 Program Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a program	`CREATE_PROGRAM`	`CREATE JOB` or `CREATE ANY JOB`

Table 28-3 (Cont.) Program Tasks and Their Procedures

Task	Procedure	Privilege Needed
Alter a program	`SET_ATTRIBUTE`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a program	`DROP_PROGRAM`	`ALTER` or `CREATE ANY JOB` or be the owner
Disable a program	`DISABLE`	`ALTER` or `CREATE ANY JOB` or be the owner
Enable a program	`ENABLE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "Scheduler Privileges" for further information regarding privileges.

28.3.2 Creating Programs with Scheduler

A program describes what is to be run by the Scheduler.

- Creating Programs
  You create programs by using the `CREATE_PROGRAM` procedure or Cloud Control.
- Defining Program Arguments
  After creating a program, you can define program arguments.

28.3.2.1 Creating Programs

You create programs by using the `CREATE_PROGRAM` procedure or Cloud Control.

By default, programs are created in the schema of the creator. To create a program in another user's schema, you must qualify the program name with the schema name. For other users to use your programs, they must have `EXECUTE` privileges on the program, therefore, once a program has been created, you must grant the `EXECUTE` privilege on it.

The following example creates a program called `my_program1`:

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name           => 'my_program1',
   program_action         => '/usr/local/bin/date',
   program_type           => 'EXECUTABLE',
   comments               => 'My comments here');
END;
/
```

Programs are created in the disabled state by default; you must enable them before you can enable jobs that point to them.

Do not attempt to enable a program that requires arguments before you define all program arguments, which you must do in a `DEFINE_`*`XXX`*`_ARGUMENT` procedure as described in "Defining Program Arguments".

28.3.2.2 Defining Program Arguments

After creating a program, you can define program arguments.

You can define arguments by position in the calling sequence, with an optional argument name and optional default value. If no default value is defined for a program argument, the job that references the program must supply an argument value. (The job can also override a default value.) All argument values must be defined before the job can be enabled.

To set program argument values, use the `DEFINE_PROGRAM_ARGUMENT` or `DEFINE_ANYDATA_ARGUMENT` procedures. Use `DEFINE_ANYDATA_ARGUMENT` for complex types that must be encapsulated in an `ANYDATA` object. An example of a program that might need arguments is one that starts a reporting program that requires a start date and end date. The following code example sets the end date argument, which is the second argument expected by the reporting program. The example also assigns a name to the argument so that you can refer to the argument by name (instead of position) from other package procedures, including `SET_JOB_ANYDATA_VALUE` and `SET_JOB_ARGUMENT_VALUE`.

```
BEGIN
 DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_position       => 2,
   argument_name           => 'end_date',
   argument_type           => 'VARCHAR2',
   default_value           => '12-DEC-03');
END;
/
```

Valid values for the `argument_type` argument must be SQL data types, therefore booleans are not supported. For external executables, only string types such as `CHAR` or `VARCHAR2` are permitted.

You can drop a program argument either by name or by position, as in the following:

```
BEGIN
 DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_position       => 2);

 DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_name           => 'end_date');
END;
/
```

In some special cases, program logic depends on the Scheduler environment. The Scheduler has some predefined metadata arguments that can be passed as an argument to the program for this purpose. For example, for some jobs whose schedule is a window name, it is useful to know how much longer the window will be open when the job is started. This is possible by defining the window end time as a metadata argument to the program.

If a program needs access to specific job metadata, you can define a special metadata argument using the `DEFINE_METADATA_ARGUMENT` procedure, so values will be filled in by the Scheduler when the program is executed.

See Also:

"Setting Job Arguments"

28.3.3 Altering Programs

You alter a program by modifying its attributes. You can use Cloud Control or the `DBMS_SCHEDULER.SET_ATTRIBUTE` and `DBMS_SCHEDULER.SET_ATTRIBUTE_NULL` package procedures to alter programs.

See the `DBMS_SCHEDULER.CREATE_PROGRAM` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details on program attributes.

If any currently running jobs use the program that you altered, they continue to run with the program as defined before the alter operation.

The following example changes the executable that program `my_program1` runs:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           => 'my_program1',
   attribute      => 'program_action',
   value          => '/usr/local/bin/salesreports1');
END;
/
```

28.3.4 Dropping Programs

You drop one or more programs using the `DROP_PROGRAM` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

When the program is dropped, any arguments that pertain it are also dropped. You can drop several programs in one call by providing a comma-delimited list of program names. For example, the following statement drops three programs:

```
BEGIN
  DBMS_SCHEDULER.DROP_PROGRAM('program1, program2, program3');
END;
/
```

Running jobs that point to the program are not affected by the `DROP_PROGRAM` call and are allowed to continue.

If you set the `force` argument to `TRUE`, jobs pointing to this program are disabled and the program is dropped. If you set the `force` argument to `FALSE`, the default, the call fails if there are any jobs pointing to the program.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DROP_PROGRAM` procedure.

28.3.5 Disabling Programs

You disable one or more programs using the `DISABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

When a program is disabled, the status is changed to `disabled`. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

The `DISABLE` call does not affect running jobs that point to the program and they are allowed to continue. Also, disabling the program does not affect any arguments that pertain to it.

A program can also be disabled by other means, for example, if a program argument is dropped or the `number_of_arguments` is changed so that no arguments are defined.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DISABLE` procedure.

28.3.6 Enabling Programs

You enable one or more programs using the `ENABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

When a program is enabled, the enabled flag is set to `TRUE`. Programs are created disabled by default, therefore, you have to enable them before you can enable jobs that point to them. Before programs are enabled, validity checks are performed to ensure that the action is valid and that all arguments are defined.

You can enable several programs in one call by providing a comma-delimited list of program names to the `ENABLE` procedure call. For example, the following statement enables three programs:

```
BEGIN
  DBMS_SCHEDULER.ENABLE('program1, program2, program3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `ENABLE` procedure.

28.4 Creating and Managing Schedules to Define Jobs

You optionally use a schedule object (a schedule) to define when a job should be run. Schedules can be shared among users by creating and saving them as objects in the database.

- Schedule Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common schedule tasks.
- Creating Schedules
  You create schedules by using the `CREATE_SCHEDULE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Altering Schedules
  You alter a schedule by using the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures in the `DBMS_SCHEDULER` package or Cloud Control.
- Dropping Schedules
  You drop a schedule using the `DROP_SCHEDULE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Setting the Repeat Interval
  You can control when and how often a job repeats.

See Also:

- "Schedules" for an overview of schedules.
- "Managing Job Scheduling and Job Priorities with Windows" and "Managing Job Scheduling and Job Priorities with Window Groups" to schedule jobs while managing job resource usage

28.4.1 Schedule Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common schedule tasks.

Table 28-4 illustrates common schedule tasks and the procedures you use to handle them.

Table 28-4 Schedule Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a schedule	`CREATE_SCHEDULE`	`CREATE JOB` or `CREATE ANY JOB`
Alter a schedule	`SET_ATTRIBUTE`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a schedule	`DROP_SCHEDULE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "Scheduler Privileges" for further information.

28.4.2 Creating Schedules

You create schedules by using the `CREATE_SCHEDULE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

Schedules are created in the schema of the user creating the schedule, and are enabled when first created. You can create a schedule in another user's schema. Once a schedule has been created, it can be used by other users. The schedule is created with access to `PUBLIC`. Therefore, there is no need to explicitly grant access to the schedule. The following example create a schedule:

```
BEGIN
 DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name     => 'my_stats_schedule',
  start_date        => SYSTIMESTAMP,
  end_date          => SYSTIMESTAMP + INTERVAL '30' day,
  repeat_interval   => 'FREQ=HOURLY; INTERVAL=4',
  comments          => 'Every 4 hours');
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_SCHEDULE` procedure.
- "Creating an Event Schedule"

28.4.3 Altering Schedules

You alter a schedule by using the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures in the `DBMS_SCHEDULER` package or Cloud Control.

Altering a schedule changes the definition of the schedule. With the exception of schedule name, all attributes can be changed. The attributes of a schedule are available in the `*_SCHEDULER_SCHEDULES` views.

If a schedule is altered, the change does not affect running jobs and open windows that use this schedule. The change goes into effect the next time the jobs runs or the window opens.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE` procedure.

28.4.4 Dropping Schedules

You drop a schedule using the `DROP_SCHEDULE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

This procedure call deletes the schedule object from the database.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DROP_SCHEDULE` procedure.

28.4.5 Setting the Repeat Interval

You can control when and how often a job repeats.

- About Setting the Repeat Interval
  You control when and how often a job repeats by setting the `repeat_interval` attribute of the job itself or the named schedule that the job references. You can set `repeat_interval` with `DBMS_SCHEDULER` package procedures or with Cloud Control.
- Using the Scheduler Calendaring Syntax
  The main way to set how often a job repeats is to set the `repeat_interval` attribute with a Scheduler calendaring expression.
- Using a PL/SQL Expression
  When you need more complicated capabilities than the calendaring syntax provides, you can use PL/SQL expressions. You cannot, however, use PL/SQL expressions for windows or in named schedules. The PL/SQL expression must evaluate to a date or a timestamp.
- Differences Between PL/SQL Expression and Calendaring Syntax Behavior
  There are important differences in behavior between a calendaring expression and PL/SQL repeat interval.

- Repeat Intervals and Daylight Savings
  For repeating jobs, the next time a job is scheduled to run is stored in a timestamp with time zone column.

28.4.5.1 About Setting the Repeat Interval

You control when and how often a job repeats by setting the `repeat_interval` attribute of the job itself or the named schedule that the job references. You can set `repeat_interval` with `DBMS_SCHEDULER` package procedures or with Cloud Control.

Evaluating the `repeat_interval` results in a set of timestamps. The Scheduler runs the job at each timestamp. Note that the start date from the job or schedule also helps determine the resulting set of timestamps. If no value for `repeat_interval` is specified, the job runs only once at the specified start date.

Immediately after a job starts, the `repeat_interval` is evaluated to determine the next scheduled execution time of the job. While this might arrive while the job is still running, a new instance of the job does not start until the current one completes.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about `repeat_interval` evaluation

28.4.5.2 Using the Scheduler Calendaring Syntax

The main way to set how often a job repeats is to set the `repeat_interval` attribute with a Scheduler calendaring expression.

See Also:

Oracle Database PL/SQL Packages and Types Reference for a detailed description of the calendaring syntax for `repeat_interval` as well as the `CREATE_SCHEDULE` procedure

Examples of Calendaring Expressions

The following examples illustrate simple repeat intervals. For simplicity, it is assumed that there is no contribution to the evaluation results by the start date.

Run every Friday. (All three examples are equivalent.)

```
FREQ=DAILY; BYDAY=FRI;
FREQ=WEEKLY; BYDAY=FRI;
FREQ=YEARLY; BYDAY=FRI;
```

Run every other Friday.

```
FREQ=WEEKLY; INTERVAL=2; BYDAY=FRI;
```

Run on the last day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=-1;
```

Run on the next to last day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=-2;
```

Run on March 10th. (Both examples are equivalent)

```
FREQ=YEARLY; BYMONTH=MAR; BYMONTHDAY=10;
FREQ=YEARLY; BYDATE=0310;
```

Run every 10 days.

```
FREQ=DAILY; INTERVAL=10;
```

Run daily at 4, 5, and 6PM.

```
FREQ=DAILY; BYHOUR=16,17,18;
```

Run on the 15th day of every other month.

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=15;
```

Run on the 29th day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=29;
```

Run on the second Wednesday of each month.

```
FREQ=MONTHLY; BYDAY=2WED;
```

Run on the last Friday of the year.

```
FREQ=YEARLY; BYDAY=-1FRI;
```

Run every 50 hours.

```
FREQ=HOURLY; INTERVAL=50;
```

Run on the last day of every other month.

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=-1;
```

Run hourly for the first three days of every month.

```
FREQ=HOURLY; BYMONTHDAY=1,2,3;
```

Here are some more complex repeat intervals:

Run on the last workday of every month (assuming that workdays are Monday through Friday).

```
FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1
```

Run on the last workday of every month, excluding company holidays. (This example references an existing named schedule called `Company_Holidays`.)

```
FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; EXCLUDE=Company_Holidays; BYSETPOS=-1
```

Run at noon every Friday and on company holidays.

```
FREQ=YEARLY;BYDAY=FRI;BYHOUR=12;INCLUDE=Company_Holidays
```

Run on these three holidays: July 4th, Memorial Day, and Labor Day. (This example references three existing named schedules, `JUL4`, `MEM`, and `LAB`, where each defines a single date corresponding to a holiday.)

```
JUL4,MEM,LAB
```

Examples of Calendaring Expression Evaluation

A repeat interval of "`FREQ=MINUTELY;INTERVAL=2;BYHOUR=17; BYMINUTE=2,4,5,50,51,7;`" with a start date of 28-FEB-2004 23:00:00 will generate the following schedule:

```
SUN 29-FEB-2004 17:02:00
SUN 29-FEB-2004 17:04:00
SUN 29-FEB-2004 17:50:00
MON 01-MAR-2004 17:02:00
MON 01-MAR-2004 17:04:00
MON 01-MAR-2004 17:50:00
...
```

A repeat interval of "`FREQ=MONTHLY;BYMONTHDAY=15,-1`" with a start date of 29-DEC-2003 9:00:00 will generate the following schedule:

```
WED 31-DEC-2003 09:00:00
THU 15-JAN-2004 09:00:00
SAT 31-JAN-2004 09:00:00
SUN 15-FEB-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 15-MAR-2004 09:00:00
WED 31-MAR-2004 09:00:00
...
```

A repeat interval of "`FREQ=MONTHLY;`" with a start date of 29-DEC-2003 9:00:00 will generate the following schedule. (Note that because there is no `BYMONTHDAY` clause, the day of month is retrieved from the start date.)

```
MON 29-DEC-2003 09:00:00
THU 29-JAN-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 29-MAR-2004 09:00:00
...
```

Example of Using a Calendaring Expression

As an example of using the calendaring syntax, consider the following statement:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job1',
   start_date           => '15-JUL-04 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'FREQ=MINUTELY; INTERVAL=30;',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

This creates `my_job1` in `scott`. It will run for the first time on July 15th and then run until September 15. The job is run every 30 minutes.

28.4.5.3 Using a PL/SQL Expression

When you need more complicated capabilities than the calendaring syntax provides, you can use PL/SQL expressions. You cannot, however, use PL/SQL expressions for windows or in named schedules. The PL/SQL expression must evaluate to a date or a timestamp.

Other than this restriction, there are no limitations, so with sufficient programming, you can create every possible repeat interval. As an example, consider the following statement:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job2',
   start_date           => '15-JUL-04 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'SYSTIMESTAMP + INTERVAL '30' MINUTE',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

This creates `my_job1` in `scott`. It will run for the first time on July 15th and then every 30 minutes until September 15. The job is run every 30 minutes because `repeat_interval` is set to `SYSTIMESTAMP + INTERVAL '30' MINUTE`, which returns a date 30 minutes into the future.

28.4.5.4 Differences Between PL/SQL Expression and Calendaring Syntax Behavior

There are important differences in behavior between a calendaring expression and PL/SQL repeat interval.

These differences include the following:

- Start date
  - Using the calendaring syntax, the start date is a reference date only. Therefore, the schedule is valid as of this date. It does not mean that the job will start on the start date.
  - Using a PL/SQL expression, the start date represents the actual time that the job will start executing for the first time.
- Next run time
  - Using the calendaring syntax, the next time the job runs is fixed.
  - Using the PL/SQL expression, the next time the job runs depends on the actual start time of the current job run.

  As an example of the difference, for a job that is scheduled to start at 2:00 PM and repeat every 2 hours, but actually starts at 2:10:

  - If calendaring syntax specified the repeat interval, then it would repeat at 4, 6 and so on.
  - If a PL/SQL expression is used, then the job would repeat at 4:10, and if the next job actually started at 4:11, then the subsequent run would be at 6:11.

To illustrate these two points, consider a situation where you have a start date of 15-July-2003 1:45:00 and you want it to repeat every two hours. A calendar expression of "`FREQ=HOURLY; INTERVAL=2; BYMINUTE=0;`" will generate the following schedule:

```
TUE 15-JUL-2003  03:00:00
TUE 15-JUL-2003  05:00:00
TUE 15-JUL-2003  07:00:00
TUE 15-JUL-2003  09:00:00
TUE 15-JUL-2003  11:00:00
...
```

Note that the calendar expression repeats every two hours on the hour.

A PL/SQL expression of "`SYSTIMESTAMP + interval '2' hour`", however, might have a run time of the following:

```
TUE 15-JUL-2003  01:45:00
TUE 15-JUL-2003  03:45:05
TUE 15-JUL-2003  05:45:09
TUE 15-JUL-2003  07:45:14
TUE 15-JUL-2003  09:45:20
...
```

28.4.5.5 Repeat Intervals and Daylight Savings

For repeating jobs, the next time a job is scheduled to run is stored in a timestamp with time zone column.

- Using the calendaring syntax, the time zone is retrieved from `start_date`. For more information on what happens when `start_date` is not specified, see *Oracle Database PL/SQL Packages and Types Reference*.
- Using PL/SQL repeat intervals, the time zone is part of the timestamp that the PL/SQL expression returns.

In both cases, it is important to use region names. For example, use "`Europe/Istanbul`", instead of absolute time zone offsets such as "`+2:00`". The Scheduler follows daylight savings adjustments that apply to that region only when a time zone is specified as a region name.

28.5 Using Events to Start Jobs

Oracle Scheduler can start a job when an event is sent. An event is a message one application or system process sends to another.

- About Events
  An **event** is a message one application or system process sends to another to indicate that some action or occurrence has been detected. An event is **raised** (sent) by one application or process, and **consumed** (received) by one or more applications or processes.
- Starting Jobs with Events Raised by Your Application
  Oracle Scheduler can start a job when an event is raised by your application.
- Starting a Job When a File Arrives on a System
  You can configure the Scheduler to start a job when a file arrives on the local system or a remote system. The job is an event-based job, and the file arrival event is raised by a file watcher, which is a Scheduler object introduced in Oracle Database 11*g* Release 2 (11.2).

See Also:

- "Examples of Creating Jobs and Schedules Based on Events"
- "Creating and Managing Job Chains" for information about using events with chains to achieve precise control over process flow

28.5.1 About Events

An **event** is a message one application or system process sends to another to indicate that some action or occurrence has been detected. An event is **raised** (sent) by one application or process, and **consumed** (received) by one or more applications or processes.

The Scheduler consumes two kinds of events:

- Events that your application raises

  An application can raise an event to be consumed by the Scheduler. The Scheduler reacts to the event by starting a job. For example, when an inventory tracking system notices that the inventory has gone below a certain threshold, it can raise an event that starts an inventory replenishment job.

  See "Starting Jobs with Events Raised by Your Application".
- File arrival events that a file watcher raises

  You can create a file watcher, a Scheduler object introduced in Oracle Database 11*g* Release 2 (11.2), to watch for the arrival of a file on a system. You can then configure a job to start when the file watcher detects the presence of the file. For example, a data warehouse for a chain of stores loads data from end-of-day revenue reports which are uploaded from the stores. The data warehouse load job starts each time a new end-of-day report arrives.

  See "Starting a Job When a File Arrives on a System"

See Also:

"Monitoring Job State with Events Raised by the Scheduler" for information about how your application can consume job state change events raised by the Scheduler

28.5.2 Starting Jobs with Events Raised by Your Application

Oracle Scheduler can start a job when an event is raised by your application.

- About Events Raised by Your Application
  Your application can raise an event to notify the Scheduler to start a job. A job started in this way is referred to as an event-based job.
- Creating an Event-Based Job
  You use the `CREATE_JOB` procedure or Cloud Control to create an event-based job. The job can include event information inline as job attributes or can specify event information by pointing to an event schedule. Like jobs based on time schedules, event-based jobs are not auto-dropped unless the job end date passes, `max_runs` is reached, or the maximum number of failures (`max_failures`) is reached.
- Altering an Event-Based Job
  You alter an event-based job by using the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package.
- Creating an Event Schedule
  You can create a schedule that is based on an event. You can then reuse the schedule for multiple jobs. To do so, use the `CREATE_EVENT_SCHEDULE` procedure, or use Cloud Control.

- Altering an Event Schedule
  You alter the event information in an event schedule in the same way that you alter event information in a job.
- Passing Event Messages into an Event-Based Job
  Through a metadata argument, the Scheduler can pass the message content of the event to the event-based job that started the job.

28.5.2.1 About Events Raised by Your Application

Your application can raise an event to notify the Scheduler to start a job. A job started in this way is referred to as an event-based job.

You can create a named schedule that references an event instead of containing date, time, and recurrence information. If a job is given such a schedule (an **event schedule**), the job runs when the event is raised.

To raise an event to notify the Scheduler to start a job, your application enqueues a message onto an Oracle Database Advanced Queuing queue that was specified when setting up the job. When the job starts, it can optionally retrieve the message content of the event.

To create an event-based job, you must set these two additional attributes:

- `queue_spec`

  A queue specification that includes the name of the queue where your application enqueues messages to raise job start events, or in the case of a secure queue, the queue name followed by a comma and the agent name.
- `event_condition`

  A conditional expression based on message properties that must evaluate to `TRUE` for the message to start the job. The expression must have the syntax of an Oracle Database Advanced Queuing rule. Accordingly, you can include user data properties in the expression, provided that the message payload is an object type, and that you prefix object attributes in the expression with `tab.user_data`.

See Also:

- `DBMS_AQADM.ADD_SUBSCRIBER` procedure in *Oracle Database PL/SQL Packages and Types Reference* for more information on queueing rules
- *Oracle Database Advanced Queuing User's Guide* for more information on how to create queues
- *Oracle Database Advanced Queuing User's Guide* for more information on how to enqueue messages

The following example sets `event_condition` to select only low-inventory events that occur after midnight and before 9:00 a.m. Assume that the message payload is an object with two attributes called `event_type` and `event_timestamp`.

```
event_condition = 'tab.user_data.event_type = ''LOW_INVENTORY'' and
extract hour from tab.user_data.event_timestamp < 9'
```

You can specify `queue_spec` and `event_condition` as inline job attributes, or you can create an **event schedule** with these attributes and point to this schedule from the job.

> **Note:**
>
> The Scheduler runs the event-based job for each occurrence of an event that matches `event_condition`. However, by default, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job. Beginning in Oracle Database 11*g* Release 1 (11.1), you can change this default behavior by setting the job attribute `PARALLEL_INSTANCES` to `TRUE`. In this case, an instance of the job is started for every instance of the event, and all job instances are lightweight jobs. See the `SET_ATTRIBUTE` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details.

Table 28-5 describes common administration tasks involving events raised by an application (and consumed by the Scheduler) and the procedures associated with them.

Table 28-5 Event Tasks and Their Procedures for Events Raised by an Application

Task	Procedure	Privilege Needed
Creating an Event-Based Job	`CREATE_JOB`	`CREATE JOB` or `CREATE ANY JOB`
Altering an Event-Based Job	`SET_ATTRIBUTE`	`CREATE ANY JOB` or ownership of the job being altered or `ALTER` privileges on the job
Creating an Event Schedule	`CREATE_EVENT_SCHEDULE`	`CREATE JOB` or `CREATE ANY JOB`
Altering an Event Schedule	`SET_ATTRIBUTE`	`CREATE ANY JOB` or ownership of the schedule being altered or `ALTER` privileges on the schedule

28.5.2.2 Creating an Event-Based Job

You use the `CREATE_JOB` procedure or Cloud Control to create an event-based job. The job can include event information inline as job attributes or can specify event information by pointing to an event schedule. Like jobs based on time schedules, event-based jobs are not auto-dropped unless the job end date passes, `max_runs` is reached, or the maximum number of failures (`max_failures`) is reached.

- Specifying Event Information as Job Attributes
  To specify event information as job attributes, you use an alternate syntax of `CREATE_JOB` that includes the `queue_spec` and `event_condition` attributes.
- Specifying Event Information in an Event Schedule
  To specify event information with an event schedule, you set the `schedule_name` attribute of the job to the name of an event schedule.

28.5.2.2.1 Specifying Event Information as Job Attributes

To specify event information as job attributes, you use an alternate syntax of `CREATE_JOB` that includes the `queue_spec` and `event_condition` attributes.

The following example creates a job that starts when an application signals to the Scheduler that inventory levels for an item have fallen to a low threshold level:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name              =>  'process_lowinv_j1',
   program_name          =>  'process_lowinv_p1',
   event_condition       =>  'tab.user_data.event_type = ''LOW_INVENTORY''',
   queue_spec            =>  'inv_events_q, inv_agent1',
   enabled               =>  TRUE,
   comments              =>  'Start an inventory replenishment job');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `CREATE_JOB` procedure.

28.5.2.2.2 Specifying Event Information in an Event Schedule

To specify event information with an event schedule, you set the `schedule_name` attribute of the job to the name of an event schedule.

The following example specifies event information in an event schedule:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name              =>  'process_lowinv_j1',
   program_name          =>  'process_lowinv_p1',
   schedule_name         =>  'inventory_events_schedule',
   enabled               =>  TRUE,
   comments              =>  'Start an inventory replenishment job');
END;
/
```

See "Creating an Event Schedule" for more information.

28.5.2.3 Altering an Event-Based Job

You alter an event-based job by using the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package.

For jobs that specify the event inline, you cannot set the `queue_spec` and `event_condition` attributes individually with `SET_ATTRIBUTE`. Instead, you must set an attribute called `event_spec`, and pass an event condition and queue specification as the third and fourth arguments, respectively, to `SET_ATTRIBUTE`.

The following example uses the `event_spec` attribute:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE ('my_job', 'event_spec',
   'tab.user_data.event_type = ''LOW_INVENTORY''', 'inv_events_q, inv_agent1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `SET_ATTRIBUTE` procedure.

28.5.2.4 Creating an Event Schedule

You can create a schedule that is based on an event. You can then reuse the schedule for multiple jobs. To do so, use the `CREATE_EVENT_SCHEDULE` procedure, or use Cloud Control.

The following example creates an event schedule:

```
BEGIN
  DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'inventory_events_schedule',
   start_date        =>  SYSTIMESTAMP,
   event_condition   =>  'tab.user_data.event_type = ''LOW_INVENTORY''',
   queue_spec        =>  'inv_events_q, inv_agent1');
END;
/
```

You can drop an event schedule using the `DROP_SCHEDULE` procedure. See *Oracle Database PL/SQL Packages and Types Reference* for more information on `CREATE_EVENT_SCHEDULE`.

28.5.2.5 Altering an Event Schedule

You alter the event information in an event schedule in the same way that you alter event information in a job.

For more information, see "Altering an Event-Based Job".

The following example demonstrates how to use the `SET_ATTRIBUTE` procedure and the `event_spec` attribute to alter event information in an event schedule.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE ('inventory_events_schedule', 'event_spec',
   'tab.user_data.event_type = ''LOW_INVENTORY''', 'inv_events_q, inv_agent1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `SET_ATTRIBUTE` procedure.

28.5.2.6 Passing Event Messages into an Event-Based Job

Through a metadata argument, the Scheduler can pass the message content of the event to the event-based job that started the job.

The following rules apply:

- The job must use a named program of type `STORED_PROCEDURE`.
- One of the named program arguments must be a metadata argument with `metadata_attribute` set to `EVENT_MESSAGE`.
- The stored procedure that implements the program must have an argument at the position corresponding to the metadata argument of the named program. The argument type must be the data type of the queue where your application queues the job-start event.

If you use the `RUN_JOB` procedure to manually run a job that has an `EVENT_MESSAGE` metadata argument, the value passed to that argument is `NULL`.

The following example shows how to construct an event-based job that can receive the event message content:

```
CREATE OR REPLACE PROCEDURE my_stored_proc (event_msg IN event_queue_type)
AS
BEGIN
  -- retrieve and process message body
END;
/

BEGIN
```

```
  DBMS_SCHEDULER.CREATE_PROGRAM (
      program_name => 'my_prog',
      program_action=> 'my_stored_proc',
      program_type => 'STORED_PROCEDURE',
      number_of_arguments => 1,
      enabled => FALSE) ;

  DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT (
      program_name => 'my_prog',
      argument_position => 1 ,
      metadata_attribute => 'EVENT_MESSAGE') ;

  DBMS_SCHEDULER.ENABLE ('my_prog');
EXCEPTION
  WHEN others THEN RAISE ;
END ;
/

BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
     job_name => 'my_evt_job' ,
     program_name => 'my_prog',
     schedule_name => 'my_evt_sch',
     enabled => true,
     auto_Drop => false) ;
EXCEPTION
  WHEN others THEN RAISE ;
END ;
/
```

28.5.3 Starting a Job When a File Arrives on a System

You can configure the Scheduler to start a job when a file arrives on the local system or a remote system. The job is an event-based job, and the file arrival event is raised by a file watcher, which is a Scheduler object introduced in Oracle Database 11*g* Release 2 (11.2).

- About File Watchers
  A **file watcher** is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job.
- Enabling File Arrival Events from Remote Systems
  To receive file arrival events from a remote system, you must install the Scheduler agent on that system, and you must register the agent with the database.
- Creating File Watchers and File Watcher Jobs
  You complete several steps to create a file watcher and a file watch job.
- File Arrival Example
  An example illustrates file arrival for a file watcher job.
- Managing File Watchers
  The `DBMS_SCHEDULER` PL/SQL package provides procedures for enabling, disabling, dropping, and setting attributes for file watchers.
- Viewing File Watcher Information
  You can view information about file watchers by querying the views `*_SCHEDULER_FILE_WATCHERS`.

28.5.3.1 About File Watchers

A **file watcher** is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job.

You create a file watcher and then create any number of event-based jobs or event schedules that reference the file watcher. When the file watcher detects the arrival of the designated file, a newly arrived file, it raises a file arrival event.

A newly arrived file is a file that has been changed and therefore has a timestamp that is later than either the latest execution or the time that the file watcher job began monitoring the target file directory.

The way the file watcher determines whether a file is a newly arrived one or not is equivalent to repeatedly executing the Unix command `ls -lrt` or the Windows DOS command `dir /od` to watch for new files in a directory. Both these commands ensure that the recently modified file is listed at the end, that is the oldest first and the newest last.

Note:

The following behaviors:

The UNIX `mv` command does not change the file modification time, while the `cp` command does.

The Windows `move`/`paste` and `copy`/`paste` commands do not change the file modification time. To do this, execute the following DOS command after the `move` or `copy` command: `copy /b file_name +,,`

The `steady_state_duration` parameter of the `CREATE_FILE_WATCHER` procedure, described in *Oracle Database PL/SQL Packages and Types Reference*, indicates the minimum time interval that the file must remain unchanged before the file watcher considers the file found. This cannot exceed one hour. If the parameter is `NULL`, an internal value is used.

The job started by the file arrival event can retrieve the event message to learn about the newly arrived file. The message contains the information required to find the file, open it, and process it.

A file watcher can watch for a file on the local system (the same host computer running Oracle Database) or a remote system. Remote systems must be running the Scheduler agent, and the agent must be registered with the database.

File watchers check for the arrival of files every 10 minutes. You can adjust this interval. See "Changing the File Arrival Detection Interval" for details.

To use file watchers, the database Java virtual machine (JVM) component must be installed.

You must have the `CREATE JOB` system privilege to create a file watcher in your own schema. You require the `CREATE ANY JOB` system privilege to create a file watcher in a schema different from your own (except the `SYS` schema, which is disallowed). You can grant the `EXECUTE` object privilege on a file watcher so that jobs in different schemas can reference it. You can also grant the `ALTER` object privilege on a file watcher so that another user can modify it.

28.5.3.2 Enabling File Arrival Events from Remote Systems

To receive file arrival events from a remote system, you must install the Scheduler agent on that system, and you must register the agent with the database.

The remote system does not require a running Oracle Database instance to generate file arrival events.

To enable the raising of file arrival events at remote systems:

1. Set up the local database to run remote external jobs.

   See "Enabling and Disabling Databases for Remote Jobs" for instructions.
2. Install, configure, register, and start the Scheduler agent on the first remote system.

   See "Installing and Configuring the Scheduler Agent on a Remote Host" for instructions.

   This adds the remote host to the list of external destinations maintained on the local database.
3. Repeat the previous step for each additional remote system.

28.5.3.3 Creating File Watchers and File Watcher Jobs

You complete several steps to create a file watcher and a file watch job.

You perform the following tasks to create a file watcher and create the event-based job that starts when the designated file arrives.

Task 1 - Create a Credential
The file watcher requires a credential object (a credential) with which to authenticate with the host operating system for access to the file. See "Credentials" for information on privileges required to create credentials.
Perform these steps:

1. Create a credential for the operating system user that must have access to the watched-for file.

   ```
   BEGIN
     DBMS_CREDENTIAL.CREATE_CREDENTIAL('WATCH_CREDENTIAL', 'salesapps',
         'sa324w1');
   END;
   /
   ```
2. Grant the `EXECUTE` object privilege on the credential to the schema that owns the event-based job that the file watcher will start.

   ```
   GRANT EXECUTE ON WATCH_CREDENTIAL to DSSUSER;
   ```

Task 2 - Create a File Watcher
Perform these steps:

1. Create the file watcher, assigning attributes as described in the `DBMS_SCHEDULER.CREATE_FILE_WATCHER` procedure documentation in *Oracle Database PL/SQL Packages and Types Reference*. You can specify wildcard parameters in the file name. A '?' prefix in the `DIRECTORY_PATH` attribute denotes the path to the Oracle home directory. A `NULL destination` indicates the local host. To watch for the file on a remote

host, provide a valid external destination name, which you can obtain from the view ALL_SCHEDULER_EXTERNAL_DESTS.

```
BEGIN
  DBMS_SCHEDULER.CREATE_FILE_WATCHER(
    file_watcher_name => 'EOD_FILE_WATCHER',
    directory_path    => '?/eod_reports',
    file_name         => 'eod*.txt',
    credential_name   => 'watch_credential',
    destination       => NULL,
    enabled           => FALSE);
END;
/
```

2. Grant EXECUTE on the file watcher to any schema that owns an event-based job that references the file watcher.

```
GRANT EXECUTE ON EOD_FILE_WATCHER to dssuser;
```

Task 3 - Create a Program Object with a Metadata Argument
So that your application can retrieve the file arrival event message content, which includes file name, file size, and so on, create a Scheduler program object with a metadata argument that references the event message.
Perform these steps:

1. Create the program.

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name        => 'dssuser.eod_program',
    program_type        => 'stored_procedure',
    program_action      => 'eod_processor',
    number_of_arguments => 1,
    enabled             => FALSE);
END;
/
```

2. Define the metadata argument using the event_message attribute.

```
BEGIN
  DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT(
    program_name       => 'DSSUSER.EOD_PROGRAM',
    metadata_attribute => 'event_message',
    argument_position  => 1);
END;
/
```

3. Create the stored procedure that the program invokes.

   The stored procedure that processes the file arrival event must have an argument of type SYS.SCHEDULER_FILEWATCHER_RESULT, which is the data type of the event message. The position of that argument must match the position of the defined metadata argument. The procedure can then access attributes of this abstract data type to learn about the arrived file.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DEFINE_METADATA_ARGUMENT` procedure
- *Oracle Database PL/SQL Packages and Types Reference* for a description of the `SYS.SCHEDULER_FILEWATCHER_RESULT` type

Task 4 - Create an Event-Based Job That References the File Watcher
Create the event-based job as described in "Creating an Event-Based Job", with the following exception: instead of providing a queue specification in the `queue_spec` attribute, provide the name of the file watcher. You would typically leave the `event_condition` job attribute null, but you can provide a condition if desired.
As an alternative to setting the `queue_spec` attribute for the job, you can create an event schedule, reference the file watcher in the `queue_spec` attribute of the event schedule, and reference the event schedule in the `schedule_name` attribute of the job.
Perform these steps to prepare the event-based job:

1. Create the job.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
    job_name        => 'dssuser.eod_job',
    program_name    => 'dssuser.eod_program',
    event_condition => NULL,
    queue_spec      => 'eod_file_watcher',
    auto_drop       => FALSE,
    enabled         => FALSE);
END;
/
```

2. If you want the job to run for each instance of the file arrival event, even if the job is already processing a previous event, set the `parallel_instances` attribute to `TRUE`. With this setting, the job runs as a lightweight job so that multiple instances of the job can be started quickly. To discard file watcher events that occur while the event-based job is already processing another, leave the `parallel_instances` attribute `FALSE` (the default).

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE('dssuser.eod_job','parallel_instances',TRUE);
END;
/
```

   For more information about this attribute, see the `SET_ATTRIBUTE` description in *Oracle Database PL/SQL Packages and Types Reference*.

See Also:

- "Creating an Event Schedule"
- "Creating Jobs Using Named Programs and Schedules"

Task 5 - Enable All Objects
Enable the file watcher, the program, and the job.

```
BEGIN
   DBMS_SCHEDULER.ENABLE('DSSUSER.EOD_PROGRAM,DSSUSER.EOD_JOB,EOD_FILE_WATCHER');
END;
/
```

28.5.3.4 File Arrival Example

An example illustrates file arrival for a file watcher job.

In this example, an event-based job watches for the arrival of end-of-day sales reports onto the local host from various locations. As each report file arrives, a stored procedure captures information about the file and stores the information in a table called `eod_reports`. A regularly scheduled report aggregation job can then query this table, process all unprocessed files, and mark any newly processed files as processed.

It is assumed that the database user running the following code has been granted `EXECUTE` on the `SYS.SCHEDULER_FILEWATCHER_RESULT` data type.

```
BEGIN
  DBMS_CREDENTIAL.CREATE_CREDENTIAL(
     credential_name => 'watch_credential',
     username        => 'pos1',
     password        => 'jk4545st');
END;
/

CREATE TABLE eod_reports (WHEN timestamp, file_name varchar2(100),
   file_size number, processed char(1));

CREATE OR REPLACE PROCEDURE q_eod_report
  (payload IN sys.scheduler_filewatcher_result) AS
BEGIN
  INSERT INTO eod_reports VALUES
     (payload.file_timestamp,
      payload.directory_path || '/' || payload.actual_file_name,
      payload.file_size,
      'N');
END;
/

BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name        => 'eod_prog',
    program_type        => 'stored_procedure',
    program_action      => 'q_eod_report',
    number_of_arguments => 1,
    enabled             => FALSE);
  DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT(
    program_name        => 'eod_prog',
    metadata_attribute  => 'event_message',
    argument_position   => 1);
  DBMS_SCHEDULER.ENABLE('eod_prog');
END;
/

BEGIN
  DBMS_SCHEDULER.CREATE_FILE_WATCHER(
    file_watcher_name => 'eod_reports_watcher',
    directory_path    => '?/eod_reports',
    file_name         => 'eod*.txt',
    credential_name   => 'watch_credential',
    destination        => NULL,
    enabled            => FALSE);
END;
/

BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
    job_name        => 'eod_job',
    program_name    => 'eod_prog',
    event_condition => 'tab.user_data.file_size > 10',
    queue_spec      => 'eod_reports_watcher',
    auto_drop       => FALSE,
    enabled         => FALSE);
  DBMS_SCHEDULER.SET_ATTRIBUTE('EOD_JOB','PARALLEL_INSTANCES',TRUE);
END;
/

EXEC DBMS_SCHEDULER.ENABLE('eod_reports_watcher,eod_job');
```

28.5.3.5 Managing File Watchers

The `DBMS_SCHEDULER` PL/SQL package provides procedures for enabling, disabling, dropping, and setting attributes for file watchers.

- Enabling File Watchers
  If a file watcher is disabled, then use `DBMS_SCHEDULER.ENABLE` to enable it.
- Altering File Watchers
  Use the `DBMS_SCHEDULER.SET_ATTRIBUTE` and `DBMS_SCHEDULER.SET_ATTRIBUTE_NULL` package procedures to modify the attributes of a file watcher.
- Disabling and Dropping File Watchers
  Use the `DBMS_SCHEDULER.DISABLE` procedure to disable a file watcher and the `DBMS_SCHEDULER.DROP_FILE_WATCHER` procedure to drop a file watcher.
- Changing the File Arrival Detection Interval
  File watchers check for the arrival of files every ten minutes by default. You can change this interval.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information about the `DBMS_SCHEDULER` PL/SQL package

28.5.3.5.1 Enabling File Watchers

If a file watcher is disabled, then use `DBMS_SCHEDULER.ENABLE` to enable it.

This is shown in Task 5, - Enable All Objects.

You can enable a file watcher only if all of its attributes are set to legal values and the file watcher owner has `EXECUTE` privileges on the specified credential.

28.5.3.5.2 Altering File Watchers

Use the `DBMS_SCHEDULER.SET_ATTRIBUTE` and `DBMS_SCHEDULER.SET_ATTRIBUTE_NULL` package procedures to modify the attributes of a file watcher.

See the `CREATE_FILE_WATCHER` procedure description for information about file watcher attributes.

28.5.3.5.3 Disabling and Dropping File Watchers

Use the `DBMS_SCHEDULER.DISABLE` procedure to disable a file watcher and the `DBMS_SCHEDULER.DROP_FILE_WATCHER` procedure to drop a file watcher.

You cannot disable or drop a file watcher if there are jobs that depend on it. To force a disable or drop operation in this case, set the `FORCE` attribute to `TRUE`. If you force disabling or dropping a file watcher, jobs that depend on it become disabled.

28.5.3.5.4 Changing the File Arrival Detection Interval

File watchers check for the arrival of files every ten minutes by default. You can change this interval.

To change the file arrival detection interval:

1. Connect to the database as the `SYS` user.
2. Change the `REPEAT_INTERVAL` attribute of the predefined schedule `SYS.FILE_WATCHER_SCHEDULE`. Use any valid calendaring syntax.

   Oracle does not recommend setting `REPEAT_INTERVAL` for file watchers to a value lower than any of the `STEADY_STATE_DURATION` attribute values.

   See Also:

   - *Oracle Database PL/SQL Packages and Types Reference* for File Watcher attribute values
   - *Oracle Database PL/SQL Packages and Types Reference* for `CREATE_FILE_WATCHER` parameters

   The following example changes the file arrival detection frequency to every two minutes.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE('FILE_WATCHER_SCHEDULE', 'REPEAT_INTERVAL',
    'FREQ=MINUTELY;INTERVAL=2');
END;
/
```

28.5.3.6 Viewing File Watcher Information

You can view information about file watchers by querying the views `*_SCHEDULER_FILE_WATCHERS`.

For example, run the following query:

```
SELECT file_watcher_name, destination, directory_path, file_name, credential_name
   FROM dba_scheduler_file_watchers;

FILE_WATCHER_NAME    DESTINATION          DIRECTORY_PATH       FILE_NAME  CREDENTIAL_NAME
-------------------- -------------------- -------------------- ---------- ----------------
MYFW                 dsshost.example.com  /tmp                 abc        MYFW_CRED
EOD_FILE_WATCHER                          ?/eod_reports        eod*.txt   WATCH_CREDENTIAL
```

See Also:

Oracle Database Reference for details on the `*_SCHEDULER_FILE_WATCHERS` views

28.6 Creating and Managing Job Chains

A job chain is a named series of tasks that are linked together for a combined objective.

- About Creating and Managing Job Chains
  Using job chains, you can implement dependency-based scheduling, in which jobs start depending on the outcomes of one or more previous jobs.
- Chain Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common chain tasks.
- Creating Chains
  You create a chain by using the `CREATE_CHAIN` procedure in the `DBMS_SCHEDULER` package.
- Defining Chain Steps
  After creating a chain object, you define one or more chain steps.
- Adding Rules to a Chain
  You add a rule to a chain with the `DEFINE_CHAIN_RULE` procedure in the `DBMS_SCHEDULER` package. You call this procedure once for each rule that you want to add to the chain.
- Setting an Evaluation Interval for Chain Rules
  The Scheduler evaluates all chain rules at the start of the chain job and at the end of each chain step.
- Enabling Chains
  You enable a chain with the `ENABLE` procedure in the `DBMS_SCHEDULER` package. A chain must be enabled before it can be run by a job. Enabling an already enabled chain does not return an error.
- Creating Jobs for Chains
  To run a chain, you must either use the `RUN_CHAIN` procedure in the `DBMS_SCHEDULER` package or create and schedule a job of type '`CHAIN`' (a **chain job**).
- Dropping Chains
  You drop a chain, including its steps and rules, using the `DROP_CHAIN` procedure in the `DBMS_SCHEDULER` package.
- Running Chains
  To run a chain immediately, use the `RUN_JOB` or `RUN_CHAIN` procedure in the `DBMS_SCHEDULER` package.

- Dropping Chain Rules
  You drop a rule from a chain by using the `DROP_CHAIN_RULE` procedure in the `DBMS_SCHEDULER` package.
- Disabling Chains
  You disable a chain using the `DISABLE` procedure in the `DBMS_SCHEDULER` package.
- Dropping Chain Steps
  You drop a step from a chain using the `DROP_CHAIN_STEP` procedure in the `DBMS_SCHEDULER` package.
- Stopping Chains
  To stop a running chain, you call the `DBMS_SCHEDULER.STOP_JOB` procedure, passing the name of the chain job (the job that started the chain).
- Stopping Individual Chain Steps
  You can stop individual chain steps by creating a chain rule that stops one or more steps when the rule condition is met or by calling the `STOP_JOB` procedure.
- Pausing Chains
  You can pause an entire chain or individual branches of a chain. You do so by setting the `PAUSE` attribute of one or more steps to `TRUE` with the `DBMS_SCHEDULER.ALTER_CHAIN` or `ALTER_RUNNING_CHAIN` procedure.
- Skipping Chain Steps
  You can skip one or more steps in a chain. You do so by setting the `SKIP` attribute of one or more steps to `TRUE` with the `DBMS_SCHEDULER.ALTER_CHAIN` or `ALTER_RUNNING_CHAIN` procedure.
- Running Part of a Chain
  You can run only part of a chain.
- Monitoring Running Chains
  You can view the status of running chains with the following two views: `*_SCHEDULER_RUNNING_JOBS` and `*_SCHEDULER_RUNNING_CHAINS`.
- Handling Stalled Chains
  At the completion of a step, the chain rules are always evaluated to determine the next steps to run. If none of the rules cause another step to start, none cause the chain to end, and the `evaluation_interval` for the chain is `NULL`, the chain enters the **stalled** state.

See Also:

- "Chains" for an overview of chains
- "Examples of Creating Chains"

28.6.1 About Creating and Managing Job Chains

Using job chains, you can implement dependency-based scheduling, in which jobs start depending on the outcomes of one or more previous jobs.

To create and use a chain, you complete these tasks in order:

Task	See...
1. Create a chain object	Creating Chains

Task	See...
2. Define the steps in the chain	Defining Chain Steps
3. Add rules	Adding Rules to a Chain
4. Enable the chain	Enabling Chains
5. Create a job (the "chain job") that points to the chain	Creating Jobs for Chains

28.6.2 Chain Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common chain tasks.

Table 28-6 illustrates common tasks involving chains and the procedures associated with them.

Table 28-6 Chain Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a chain	`CREATE_CHAIN`	`CREATE JOB`, `CREATE EVALUATION CONTEXT`, `CREATE RULE`, and `CREATE RULE SET` if the owner. `CREATE ANY JOB`, `CREATE ANY RULE`, `CREATE ANY RULE SET`, and `CREATE ANY EVALUATION CONTEXT` otherwise
Drop a chain	`DROP_CHAIN`	Ownership of the chain or `ALTER` privileges on the chain or `CREATE ANY JOB` privileges. If not owner, also requires `DROP ANY EVALUATION CONTEXT` and `DROP ANY RULE SET`
Alter a chain	`SET_ATTRIBUTE`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB`
Alter a running chain	`ALTER_RUNNING_CHAIN`	Ownership of the job, or `ALTER` privileges on the job or `CREATE ANY JOB`
Run a chain	`RUN_CHAIN`	`CREATE JOB` or `CREATE ANY JOB`. In addition, the owner of the new job must have `EXECUTE` privileges on the chain or `EXECUTE ANY PROGRAM`
Add rules to a chain	`DEFINE_CHAIN_RULE`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB` privileges. `CREATE RULE` if the owner of the chain, `CREATE ANY RULE` otherwise
Alter rules in a chain	`DEFINE_CHAIN_RULE`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB` privileges. If not owner of the chain, requires `ALTER` privileges on the rule or `ALTER ANY RULE`
Drop rules from a chain	`DROP_CHAIN_RULE`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB` privileges. `DROP ANY RULE` if not the owner of the chain
Enable a chain	`ENABLE`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB`
Disable a chain	`DISABLE`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB`
Create steps	`DEFINE_CHAIN_STEP`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB`
Drop steps	`DROP_CHAIN_STEP`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB`

Table 28-6 (Cont.) Chain Tasks and Their Procedures

Task	Procedure	Privilege Needed
Alter steps (including assigning additional attribute values to steps)	`ALTER_CHAIN`	Ownership of the chain, or `ALTER` privileges on the chain or `CREATE ANY JOB`

28.6.3 Creating Chains

You create a chain by using the `CREATE_CHAIN` procedure in the `DBMS_SCHEDULER` package.

You must ensure that you have the required privileges first. See "Setting Chain Privileges" for details.

After creating the chain object with `CREATE_CHAIN`, you define chain steps and chain rules separately.

The following example creates a chain:

```
BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name          => 'my_chain1',
   rule_set_name       => NULL,
   evaluation_interval => NULL,
   comments            => 'My first chain');
END;
/
```

The `rule_set_name` and `evaluation_interval` arguments are typically left `NULL`. `evaluation_interval` can define a repeating interval at which chain rules get evaluated. `rule_set_name` refers to a rule set as defined within Oracle Streams.

See Also:

- "Adding Rules to a Chain" for more information about the `evaluation_interval` attribute
- See *Oracle Database PL/SQL Packages and Types Reference* for more information on `CREATE_CHAIN`

28.6.4 Defining Chain Steps

After creating a chain object, you define one or more chain steps.

Each step can point to one of the following:

- A Scheduler program object (program)
- Another chain (a nested chain)
- An event schedule, inline event, or file watcher

You define a step that points to a program or nested chain by using the `DEFINE_CHAIN_STEP` procedure. The following example adds two steps to `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name      =>  'my_chain1',
   step_name       =>  'my_step1',
   program_name    =>  'my_program1');
  DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name      =>  'my_chain1',
   step_name       =>  'my_step2',
   program_name    =>  'my_chain2');
END;
/
```

The named program or chain does not have to exist when you define the step. However, it must exist and be enabled when the chain runs, otherwise an error is generated.

You define a step that waits for an event to occur by using the `DEFINE_CHAIN_EVENT_STEP` procedure. Procedure arguments can point to an event schedule, can include an inline queue specification and event condition, or can include a file watcher name. This example creates a third chain step that waits for the event specified in the named event schedule:

```
BEGIN
  DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
   chain_name           =>  'my_chain1',
   step_name            =>  'my_step3',
   event_schedule_name  =>  'my_event_schedule');
END;
/
```

An event step does not wait for its event until the step is started.

Steps That Run Local External Executables

After defining a step that runs a local external executable, you must use the `ALTER_CHAIN` procedure to assign a credential to the step, as shown in the following example:

```
BEGIN
  DBMS_SCHEDULER.ALTER_CHAIN('chain1','step1','credential_name','MY_CREDENTIAL');
END;
/
```

Steps That Run on Remote Destinations

After defining a step that is to run an external executable on a remote host or a database program unit on a remote database, you must use the `ALTER_CHAIN` procedure to assign both a credential and a destination to the step, as shown in the following example:

```
BEGIN
 DBMS_SCHEDULER.ALTER_CHAIN('chain1','step2','credential_name','DW_CREDENTIAL');
 DBMS_SCHEDULER.ALTER_CHAIN('chain1','step2','destination_name','DBHOST1_ORCLDW');
END;
/
```

Making Steps Restartable

After a database recovery, by default, steps that were running are marked as `STOPPED` and the chain continues. You can specify the chain steps to restart automatically after a database recovery by using `ALTER_CHAIN` to set the `restart_on_recovery` attribute to `TRUE` for those steps.

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `DEFINE_CHAIN_STEP`, `DEFINE_CHAIN_EVENT_STEP`, and `ALTER_CHAIN` procedures.

See Also:

- "About Events"
- "About File Watchers"
- "Credentials"
- "Destinations"

28.6.5 Adding Rules to a Chain

You add a rule to a chain with the `DEFINE_CHAIN_RULE` procedure in the `DBMS_SCHEDULER` package. You call this procedure once for each rule that you want to add to the chain.

Chain rules define when steps run and define dependencies between steps. Each rule has a condition and an action. Whenever rules are evaluated, if a condition of a rule evaluates to `TRUE`, its action is performed. The condition can contain Scheduler chain condition syntax or any syntax that is valid in a SQL `WHERE` clause. The syntax can include references to attributes of any chain step, including step completion status. A typical action is to run a specified step or to run a list of steps.

All chain rules work together to define the overall action of the chain. All rules are evaluated to see what action or actions occur next, when the chain job starts and at the end of each step. If more than one rule has a `TRUE` condition, multiple actions can occur. You can also cause rules to be evaluated at regular intervals by setting the `evaluation_interval` attribute of a chain.

Conditions are usually based on the outcome of one or more previous steps. For example, you might want one step to run if the two previous steps succeeded, and another to run if either of the two previous steps failed.

Scheduler chain condition syntax takes one of the following two forms:

```
stepname [NOT] {SUCCEEDED|FAILED|STOPPED|COMPLETED}
stepname ERROR_CODE {comparision_operator|[NOT] IN} {integer|list_of_integers}
```

You can combine conditions with boolean operators `AND`, `OR`, and `NOT()` to create conditional expressions. You can employ parentheses in your expressions to determine order of evaluation.

`ERROR_CODE` can be set with the `RAISE_APPLICATION_ERROR` PL/SQL statement within the program assigned to the step. Although the error codes that your program sets in this way are negative numbers, when testing `ERROR_CODE` in a chain rule, you test for positive numbers. For example, if your program contains the following statement:

```
RAISE_APPLICATION_ERROR(-20100, errmsg);
```

your chain rule condition must be the following:

```
stepname ERROR_CODE=20100
```

Step Attributes

The following is a list of step attributes that you can include in conditions when using SQL `WHERE` clause syntax:

```
completed
```

```
state
start_date
end_date
error_code
duration
```

The `completed` attribute is boolean and is `TRUE` when the `state` attribute is either `SUCCEEDED`, `FAILED`, or `STOPPED`.

Table 28-7 shows the possible values for the `state` attribute. These values are visible in the `STATE` column of the `*_SCHEDULER_RUNNING_CHAINS` views.

Table 28-7 Values for the State Attribute of a Chain Step

State Attribute Value	Meaning
`NOT_STARTED`	The chain of a step is running, but the step has not yet started.
`SCHEDULED`	A rule started the step with an `AFTER` clause and the designated wait time has not yet expired.
`RUNNING`	The step is running. For an event step, the step was started and is waiting for an event.
`PAUSED`	The `PAUSE` attribute of a step is set to `TRUE` and the step is paused. It must be unpaused before steps that depend on it can start.
`SUCCEEDED`	The step completed successfully. The `ERROR_CODE` of the step is 0.
`FAILED`	The step completed with a failure. `ERROR_CODE` is nonzero.
`STOPPED`	The step was stopped with the `STOP_JOB` procedure.
`STALLED`	The step is a nested chain that has stalled.

See the `DEFINE_CHAIN_RULE` procedure in *Oracle Database PL/SQL Packages and Types Reference* for rules and examples for SQL `WHERE` clause syntax.

Condition Examples Using Scheduler Chain Condition Syntax

These examples use Scheduler chain condition syntax.

Steps started by rules containing the following condition starts when the step named `form_validation_step` completes (`SUCCEEDED`, `FAILED`, or `STOPPED`).

```
form_validation_step COMPLETED
```

The following condition is similar, but indicates that the step must succeed for the condition to be met.

```
form_validation_step SUCCEEDED
```

The next condition tests for an error. It is `TRUE` if the step `form_validation_step` failed with any error code other than 20001.

```
form_validation_step FAILED AND form_validation_step ERROR_CODE != 20001
```

See the `DEFINE_CHAIN_RULE` procedure in *Oracle Database PL/SQL Packages and Types Reference* for more examples.

Condition Examples Using SQL WHERE Syntax

```
':step1.state=''SUCCEEDED'''
```

Starting the Chain

At least one rule must have a condition that always evaluates to `TRUE` so that the chain can start when the chain job starts. The easiest way to accomplish this is to set the condition to '`TRUE`' if you are using Schedule chain condition syntax, or '`1=1`' if you are using SQL syntax.

Ending the Chain

At least one chain rule must contain an `action` of '`END`'. A chain job does not complete until one of the rules containing the `END` action evaluates to `TRUE`. Several different rules with different `END` actions are common, some with error codes, and some without.

If a chain has no more running steps, and it is not waiting for an event to occur, and no rules containing the `END` action evaluate to `TRUE` (or there are no rules with the `END` action), the chain job enters the `CHAIN_STALLED` state. See "Handling Stalled Chains" for more information.

Example of Defining Rules

The following example defines a rule that starts the chain at `step1` and a rule that starts `step2` when `step1` completes. `rule_name` and `comments` are optional and default to `NULL`. If you do use `rule_name`, you can later redefine that rule with another call to `DEFINE_CHAIN_RULE`. The new definition overwrites the previous one.

```
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   condition    =>   'TRUE',
   action       =>   'START step1',
   rule_name    =>   'my_rule1',
   comments     =>   'start the chain');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   condition    =>   'step1 completed',
   action       =>   'START step2',
   rule_name    =>   'my_rule2');
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information on the `DEFINE_CHAIN_RULE` procedure and Scheduler chain condition syntax
- "Examples of Creating Chains"

28.6.6 Setting an Evaluation Interval for Chain Rules

The Scheduler evaluates all chain rules at the start of the chain job and at the end of each chain step.

You can also configure a chain to have Scheduler evaluate its rules at a repeating time interval, such as once per hour. This capability is useful to start chain steps based on time of day or based on occurrences external to the chain. Here are some examples:

- A chain step is resource-intensive and must therefore run at off-peak hours. You could condition the step on both the completion of another step and on the time of day being after 6:00 p.m and before midnight. The Scheduler would then have to evaluate rules every so often to determine when this condition becomes `TRUE`.
- A step must wait for data to arrive in a table from some other process that is external to the chain. You could condition this step on both the completion of another step and on a particular table containing rows. The Scheduler would then have to evaluate rules every so often to determine when this condition becomes `TRUE`. The condition would use SQL `WHERE` clause syntax, and would be similar to the following:

  ```
  ':step1.state=''SUCCEEDED'' AND select count(*) from oe.sync_table > 0'
  ```

To set an evaluation interval for a chain, you set the `evaluation_interval` attribute when you create the chain. The data type for this attribute is `INTERVAL DAY TO SECOND`.

```
BEGIN
  DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name          => 'my_chain1',
   rule_set_name       => NULL,
   evaluation_interval => INTERVAL '30' MINUTE,
   comments            => 'Chain with 30 minute evaluation interval');
END;
/
```

28.6.7 Enabling Chains

You enable a chain with the `ENABLE` procedure in the `DBMS_SCHEDULER` package. A chain must be enabled before it can be run by a job. Enabling an already enabled chain does not return an error.

This example enables chain `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.ENABLE ('my_chain1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `ENABLE` procedure.

Note:

Chains are automatically disabled by the Scheduler when one of the following is dropped:

- The program that one of the chain steps points to
- The nested chain that one of the chain steps points to
- The event schedule that one of the chain event steps points to

28.6.8 Creating Jobs for Chains

To run a chain, you must either use the `RUN_CHAIN` procedure in the `DBMS_SCHEDULER` package or create and schedule a job of type '`CHAIN`' (a **chain job**).

The job action must refer to a previously created chain name, as shown in the following example:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

For every step of a chain job that is running, the Scheduler creates a **step job** with the same job name and owner as the chain job. Each step job additionally has a job subname to uniquely identify it. You can view the job subname as a column in the views `*_SCHEDULER_RUNNING_JOBS`, `*_SCHEDULER_JOB_LOG`, and `*_SCHEDULER_JOB_RUN_DETAILS`. The job subname is normally the same as the step name except in the following cases:

- For nested chains, the current step name may have already been used as a job subname. In this case, the Scheduler appends '`_`*`N`*' to the step name, where *`N`* is an integer that results in a unique job subname.
- If there is a failure when creating a step job, the Scheduler logs a `FAILED` entry in the job log views (`*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`) with the job subname set to '*`step_name`*`_0`'.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `CREATE_JOB` procedure
- "Running Chains" for another way to run a chain without creating a chain job

28.6.9 Dropping Chains

You drop a chain, including its steps and rules, using the `DROP_CHAIN` procedure in the `DBMS_SCHEDULER` package.

The following example drops the chain named `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.DROP_CHAIN (
   chain_name   => 'my_chain1',
   force        => TRUE);
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `DROP_CHAIN` procedure.

28.6.10 Running Chains

To run a chain immediately, use the `RUN_JOB` or `RUN_CHAIN` procedure in the `DBMS_SCHEDULER` package.

If you already created a chain job for a chain, you can use the `RUN_JOB` procedure to run that job (and thus run the chain), but you must set the `use_current_session` argument of `RUN_JOB` to `FALSE`.

You can use the `RUN_CHAIN` procedure to run a chain without having to first create a chain job for the chain. You can also use `RUN_CHAIN` to run only part of a chain.

`RUN_CHAIN` creates a temporary job to run the specified chain. If you supply a job name, the job is created with that name, otherwise a default job name is assigned.

If you supply a list of *start steps*, only those steps are started when the chain begins running. (Steps that would normally have started do not run if they are not in the list.) If no list of start steps is given, the chain starts normally—that is, an initial evaluation is done to see which steps to start running. The following example immediately runs `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.RUN_CHAIN (
   chain_name    =>  'my_chain1',
   job_name      =>  'partial_chain_job',
   start_steps   =>  'my_step2, my_step4');
END;
/
```

See Also:

- "Running Part of a Chain"
- *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `RUN_CHAIN` procedure

28.6.11 Dropping Chain Rules

You drop a rule from a chain by using the `DROP_CHAIN_RULE` procedure in the `DBMS_SCHEDULER` package.

The following example drops `my_rule1`:

```
BEGIN
  DBMS_SCHEDULER.DROP_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   rule_name    =>   'my_rule1',
   force        =>   TRUE);
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `DROP_CHAIN_RULE` procedure.

28.6.12 Disabling Chains

You disable a chain using the `DISABLE` procedure in the `DBMS_SCHEDULER` package.

The following example disables `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.DISABLE ('my_chain1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `DISABLE` procedure.

Note:

Chains are automatically disabled by the Scheduler when one of the following is dropped:

- The program that one of the chain steps points to
- The nested chain that one of the chain steps points to
- The event schedule that one of the chain event steps points to

28.6.13 Dropping Chain Steps

You drop a step from a chain using the `DROP_CHAIN_STEP` procedure in the `DBMS_SCHEDULER` package.

The following example drops `my_step2` from `my_chain2`:

```
BEGIN
  DBMS_SCHEDULER.DROP_CHAIN_STEP (
   chain_name   =>   'my_chain2',
   step_name    =>   'my_step2',
   force        =>    TRUE);
```

```
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `DROP_CHAIN_STEP` procedure.

28.6.14 Stopping Chains

To stop a running chain, you call the `DBMS_SCHEDULER.STOP_JOB` procedure, passing the name of the chain job (the job that started the chain).

When you stop a chain job, all steps of the chain that are running are stopped and the chain ends.

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `STOP_JOB` procedure.

28.6.15 Stopping Individual Chain Steps

You can stop individual chain steps by creating a chain rule that stops one or more steps when the rule condition is met or by calling the `STOP_JOB` procedure.

For each step being stopped, you must specify the schema name, chain job name, and step job subname.

```
BEGIN
  DBMS_SCHEDULER.STOP_JOB('oe.chainrunjob.stepa');
END;
/
```

In this example, `chainrunjob` is the chain job name and `stepa` is the step job subname. The step job subname is typically the same as the step name, but not always. You can obtain the step job subname from the `STEP_JOB_SUBNAME` column of the `*_SCHEDULER_RUNNING_CHAINS` views.

When you stop a chain step, its `state` is set to `STOPPED`, and the chain rules are evaluated to determine the steps to run next.

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `STOP_JOB` procedure.

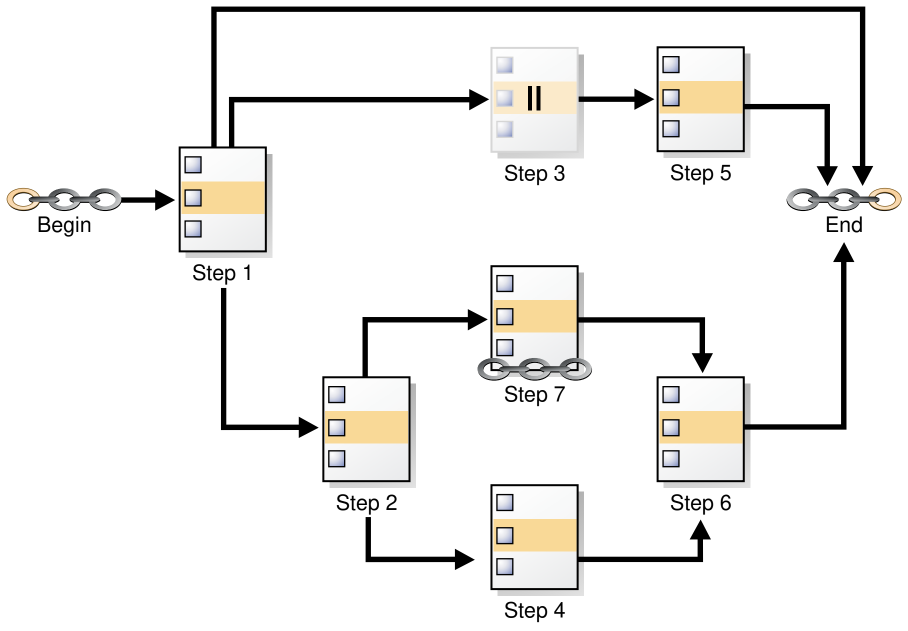
28.6.16 Pausing Chains

You can pause an entire chain or individual branches of a chain. You do so by setting the `PAUSE` attribute of one or more steps to `TRUE` with the `DBMS_SCHEDULER.ALTER_CHAIN` or `ALTER_RUNNING_CHAIN` procedure.

Pausing chain steps enables you to suspend the running of the chain after those steps run.

When you pause a step, after the step runs, its `state` attribute changes to `PAUSED`, and its `completed` attribute remains `FALSE`. Therefore, steps that depend on the completion of the paused step are not run. If you reset the `PAUSE` attribute to `FALSE` for a paused step, its `state` attribute is set to its completion state (`SUCCEEDED`, `FAILED`, or `STOPPED`), and steps that are awaiting the completion of the paused step can then run.

Figure 28-1 Chain with Step 3 Paused



In Figure 28-1, Step 3 is paused. Until Step 3 is unpaused, Step 5 will not run. If you were to pause only Step 2, then Steps 4, 6, and 7 would not run. However Steps 1, 3, and 5 could run. In either case, you are suspending only one branch of the chain.

To pause an entire chain, you pause all steps of the chain. To unpause a chain, you unpause one, many, or all of the chain steps. With the chain in Figure 28-1, pausing Step 1 pauses the entire chain after Step 1 runs.

See Also:

The `DBMS_SCHEDULER.ALTER_CHAIN` and `DBMS_SCHEDULER.ALTER_RUNNING_CHAIN` procedures in *Oracle Database PL/SQL Packages and Types Reference*

28.6.17 Skipping Chain Steps

You can skip one or more steps in a chain. You do so by setting the `SKIP` attribute of one or more steps to `TRUE` with the `DBMS_SCHEDULER.ALTER_CHAIN` or `ALTER_RUNNING_CHAIN` procedure.

If a `SKIP` attribute of a step is `TRUE`, then when a chain condition to run that step is met, instead of being run, the step is treated as immediately succeeded. Setting `SKIP` to `TRUE` has no effect on a step that is running, is scheduled to run after a delay, or has already run.

Skipping steps is especially useful when testing chains. For example, when testing the chain shown in Figure 28-1, skipping Step 7 could shorten testing time considerably, because this step is a nested chain.

See Also:

"Skipping Chain Steps"

28.6.18 Running Part of a Chain

You can run only part of a chain.

There are two ways to run only a part of a chain:

- Use the `ALTER_CHAIN` procedure to set the `PAUSE` attribute to `TRUE` for one or more steps, and then either start the chain job with `RUN_JOB` or start the chain with `RUN_CHAIN`. Any steps that depend on the paused steps do not run, but the paused steps do run.

  The disadvantage of this method is that you must set the `PAUSE` attribute back to `FALSE` for the affected steps for future runs of the chain.
- Use the `RUN_CHAIN` procedure to start only certain steps of the chain, skipping those steps that you do not want to run.

  This is a more straightforward approach, which also allows you to set the initial state of steps before starting them.

You may have to use both of these methods to skip steps both at the beginning and end of a chain.

See the discussion of the `RUN_CHAIN` procedure in *Oracle Database PL/SQL Packages and Types Reference* for more information.

28.6.19 Monitoring Running Chains

You can view the status of running chains with the following two views: `*_SCHEDULER_RUNNING_JOBS` and `*_SCHEDULER_RUNNING_CHAINS`.

The `*_SCHEDULER_RUNNING_JOBS` views contain one row for the chain job and one row for each running step. The `*_SCHEDULER_RUNNING_CHAINS` views contain one row for each chain step, including any nested chains, and include run status for each step such as `NOT_STARTED`, `RUNNING`, `STOPPED`, `SUCCEEDED`, and so on.

See Also:

- *Oracle Database Reference* for details about the `*_SCHEDULER_RUNNING_JOBS` views
- *Oracle Database Reference* for details about the `*_SCHEDULER_RUNNING_CHAINS` views

28.6.20 Handling Stalled Chains

At the completion of a step, the chain rules are always evaluated to determine the next steps to run. If none of the rules cause another step to start, none cause the chain to end, and the `evaluation_interval` for the chain is `NULL`, the chain enters the **stalled** state.

When a chain is stalled, no steps are running, no steps are scheduled to run (after waiting a designated time interval), and no event steps are waiting for an event. The chain can make no further progress unless you manually intervene. In this case, the state of the job that is running the chain is set to `CHAIN_STALLED`. However, the job is still listed in the `*_SCHEDULER_RUNNING_JOBS` views.

You can troubleshoot a stalled chain with the views `ALL_SCHEDULER_RUNNING_CHAINS`, which shows the state of all steps in the chain (including any nested chains), and `ALL_SCHEDULER_CHAIN_RULES`, which contains all the chain rules.

You can enable the chain to continue by altering the `state` of one of its steps with the `ALTER_RUNNING_CHAIN` procedure. For example, if step 11 is waiting for step 9 to succeed before it can start, and if it makes sense to do so, you can set the `state` of step 9 to '`SUCCEEDED`'.

Alternatively, if one or more rules are incorrect, you can use the `DEFINE_CHAIN_RULE` procedure to replace them (using the same rule names), or to create new rules. The new and updated rules apply to the running chain and all future chain runs. After adding or updating rules, you must run `EVALUATE_RUNNING_CHAIN` on the stalled chain job to trigger any required actions.

28.7 Using Incompatibility Definitions

An incompatibility definition (or, incompatibility) specifies incompatible jobs or programs, where only one of the group can be running at a time.

- Creating a Job or Program Incompatibility
  You can specify a job-level or program-level incompatibility by using the `CREATE_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.
- Adding a Job or Program to an Incompatibility
  You can add a job or program to an existing incompatibility definition by using the `ADD_TO_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.
- Removing a Job or Program from an Incompatibility
  You can remove a job or program from an existing incompatibility definition by using the `REMOVE_FROM_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.
- Dropping an Incompatibility
  You can drop an existing incompatibility definition by using the `DROP_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.

See Also:

- Incompatibilities
- DBMS_SCHEDULER in *Oracle Database PL/SQL Packages and Types Reference*

28.7.1 Creating a Job or Program Incompatibility

You can specify a job-level or program-level incompatibility by using the `CREATE_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.

For example, the following statement creates an incompatibility named `incompat1` specifying that only one of the jobs named `job1`, `job2`, or `job3` can be running at the same time:

```
BEGIN
dbms_scheduler.create_incompatibility(
  incompatibility_name => 'icompat1',
  object_name => 'job1,job2,job3',
  enabled => true );
END;
/
```

`object_name` contains a comma separated list of either all programs or all jobs that are incompatible with each other (that is, they cannot be run at the same time). In case of jobs the list, must consist of two or more jobs and `constraint_level` must be `'JOB_LEVEL'` (the default, and not included in the example). In case of programs, `constraint_level` can be either `'JOB_LEVEL'` or `'PROGRAM_LEVEL'`. When set to the default value `'JOB_LEVEL'`, only a single job that is based on the program (or programs) mentioned in `object_name` can run at the same time. When set to `'PROGRAM_LEVEL'`. the programs are incompatible, but the jobs based on the same program are not incompatible.

For example, if the value of `object_name` is `'P1,P2,P3'` and `constraint_level` is `'PROGRAM_LEVEL'`, manyjobs based on P1 can be running at the same time, but if any P1 based job is running, none based on P2 or P3 can be running. Or similarly, many jobs based on P3 can be running at the same time, but none based on P1 or P2. If `constraint_level` is set to `'JOB_LEVEL'`, then only a single job out of all the jobs based on programs P1, P2, and P3 can be running at any given time.

See Also:

CREATE INCOMPATIBILITY Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.7.2 Adding a Job or Program to an Incompatibility

You can add a job or program to an existing incompatibility definition by using the `ADD_TO_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.

For example, the following statement adds job `job1` to the incompatibility named `icomp1234`:

```
BEGIN
dbms_scheduler.add_to_incompatibility(
  incompatibility_name => 'icomp1234',
  object_name => 'job1');
END;
/
```

`incompatibility_name` is the name of an existing incompatibility definition.

`object_name` contains a comma separated list of jobs or programs.

This procedure does not raise an error if a specified job or program to be added is already included in the specified incompatibility definition.

See Also:

ADD_TO_INCOMPATIBILITY Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.7.3 Removing a Job or Program from an Incompatibility

You can remove a job or program from an existing incompatibility definition by using the `REMOVE_FROM_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.

For example, the following statement removes job `job1` from the incompatibility named `icomp1234`:

```
BEGIN
dbms_scheduler.remove_from_incompatibility(
  incompatibility_name => 'icomp1234',
  object_name => 'job1');
END;
/
```

`incompatibility_name` is the name of an existing incompatibility definition.

`object_name` contains a comma separated list of jobs or programs.

This procedure does not raise an error if a specified job or program to be removed does not exist in the specified incompatibility definition.

See Also:

REMOVE_FROM_INCOMPATIBILITY Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.7.4 Dropping an Incompatibility

You can drop an existing incompatibility definition by using the `DROP_INCOMPATIBILITY` procedure in the `DBMS_SCHEDULER` package.

For example, the following statement drops the incompatibility named `icomp1234`:

```
BEGIN
dbms_scheduler.drop_incompatibility(
  incompatibility_name => 'icomp1234';
END;
/
```

`incompatibility_name` is the name of an existing incompatibility definition.

See Also:

DROP_INCOMPATIBILITY Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.8 Managing Job Resources

You can create and alter resources available for use by jobs, and control how many of a specified resource are available to a job.

Customers have jobs that need access to resources. A limited number of such resources are available, so the scheduling system needs to keep track of which jobs use which resources and not schedule jobs until the resources that they need are available.

- Creating or Dropping a Resource
  You can create a resource by using the `CREATE_RESOURCE` procedure in the `DBMS_SCHEDULER` package.
- Altering a Resource
  You can alter a resource by using the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures in the `DBMS_SCHEDULER` package.
- Setting a Resource Constraint for a Job
  You can specify resources for use by jobs or programs by using the `SET_RESOURCE_CONSTRAINT` procedure in the `DBMS_SCHEDULER` package.

See Also:

- DBMS_SCHEDULER in *Oracle Database PL/SQL Packages and Types Reference*

28.8.1 Creating or Dropping a Resource

You can create a resource by using the `CREATE_RESOURCE` procedure in the `DBMS_SCHEDULER` package.

Resources are created in the schema of the user creating the resource.

For example, the following statement creates a resource named `my_resource` specifying that three units of the resource are to be made available initially, and that the Scheduler is manage the constraint so that no more than 3 units can be in use simultaneously by jobs.

```
BEGIN
   DBMS_SCHEDULER.CREATE_RESOURCE(
      resource_name => 'my_resource',
      units          => 3,
      state          => 'ENFORCE_CONSTRAINTS',
      comments       => 'Resource1'
   )
END;
/
```

If you no longer need a resource, you can drop it using the `DROP_RESOURCE` procedure in the `DBMS_SCHEDULER` package. For example:

```
BEGIN
   DBMS_SCHEDULER.DROP_RESOURCE(
      resource_name => 'my_resource',
      force         => true
   )
END;
/
```

See Also:

CREATE_RESOURCE Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.8.2 Altering a Resource

You can alter a resource by using the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures in the `DBMS_SCHEDULER` package.

If a resource is altered, the change does not affect currently running jobs that use this resource. The change goes into effect for subsequent jobs that use the resource.

See Also:

SET_ATTRIBUTE Procedure and SET_ATTRIBUTE_NULL Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.8.3 Setting a Resource Constraint for a Job

You can specify resources for use by jobs or programs by using the `SET_RESOURCE_CONSTRAINT` procedure in the `DBMS_SCHEDULER` package.

You can specify the number of units of the resource that a specified job or program can use.

For example, the following statement specifies that the object named `job1` can use one unit of the resource named `resource1`.

```
BEGIN
    DBME_SCHEDULER.SET_RESOURCE_CONSTRAINT(
         OBJECT_NAME   => 'job1',
         RESOURCE_NAME => 'resource1',
         UNITS         =>1);
END;
/
```

The `object_name` parameter can be the name of a program or a job, or a comma-separated list of names.

The `units` parameter specifies how many units of the resource this program or job can use. If `units` is set to 0, it means that the program or job does not use this resource anymore, and the constraint is deleted. If `units` is set to 0 on a resource for which there was no previous constraint, an error is generated.

If multiple constraints are defined on the same resource, the object types (job or program) must match. For example, if one or more constraints for a resource are based of jobs, and if a new constraint for a program is added for the same resource, an error is generated.

See Also:

SET_RESOURCE_CONSTRAINT Procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.9 Prioritizing Jobs

You prioritize Oracle Scheduler jobs using three Scheduler objects: job classes, windows, and window groups. These objects prioritize jobs by associating jobs with database resource manager consumer groups. This, in turn, controls the amount of resources allocated to these jobs. In addition, job classes enable you to set relative priorities among a group of jobs if all jobs in the group are allocated identical resource levels.

- Managing Job Priorities with Job Classes
  Job classes provide a way to group jobs for prioritization. They also provide a way to easily assign a set of attribute values to member jobs. Job classes influence the priorities of their member jobs through job class attributes that relate to the database resource manager.
- Setting Relative Job Priorities Within a Job Class
  You can change the relative priorities of jobs within the same job class by using the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package. Job priorities must be in the range of 1-5, where 1 is the highest priority.
- Managing Job Scheduling and Job Priorities with Windows
  You create windows to automatically start jobs or to change resource allocation among jobs during various time periods of the day, week, and so on. A window is represented by an interval of time.
- Managing Job Scheduling and Job Priorities with Window Groups
  Window groups provide an easy way to schedule jobs that must run during multiple time periods throughout the day, week, and so on. If you create a window group, add windows to it, and then name this window group in a job's `schedule_name` attribute, the job runs during all the windows in the window group. Window groups reside in the `SYS` schema.
- Allocating Resources Among Jobs Using Resource Manager
  The Database Resource Manager (Resource Manager) controls how resources are allocated among database sessions. It not only controls asynchronous sessions like Scheduler jobs, but also synchronous sessions like user sessions.
- Example of Resource Allocation for Jobs
  An example illustrates how resources are allocated for jobs.

See Also:

Managing Resources with Oracle Database Resource Manager

28.9.1 Managing Job Priorities with Job Classes

Job classes provide a way to group jobs for prioritization. They also provide a way to easily assign a set of attribute values to member jobs. Job classes influence the priorities of their member jobs through job class attributes that relate to the database resource manager.

A default job class is created with the database. If you create a job without specifying a job class, the job is assigned to this default job class (`DEFAULT_JOB_CLASS)`. The default job class has the `EXECUTE` privilege granted to `PUBLIC` so any database user who has the privilege to create a job can create a job in the default job class.

- Job Class Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common job class tasks.
- Creating Job Classes
  You create a job class using the `CREATE_JOB_CLASS` procedure in the `DBMS_SCHEDULER` package or Cloud Control. Job classes are always created in the `SYS` schema.
- Altering Job Classes
  You alter a job class by using the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Dropping Job Classes
  You drop one or more job classes using the `DROP_JOB_CLASS` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

See Also:

- *Oracle Database Reference* to view job classes
- "Allocating Resources Among Jobs Using Resource Manager"
- "Job Classes" for an overview of job classes

28.9.1.1 Job Class Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common job class tasks.

Table 28-8 illustrates common job class tasks and their appropriate procedures and privileges:

Table 28-8 Job Class Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a job class	`CREATE_JOB_CLASS`	`MANAGE SCHEDULER`
Alter a job class	`SET_ATTRIBUTE`	`MANAGE SCHEDULER`

Table 28-8 (Cont.) Job Class Tasks and Their Procedures

Task	Procedure	Privilege Needed
Drop a job class	`DROP_JOB_CLASS`	`MANAGE SCHEDULER`

See "Scheduler Privileges" for further information regarding privileges.

28.9.1.2 Creating Job Classes

You create a job class using the `CREATE_JOB_CLASS` procedure in the `DBMS_SCHEDULER` package or Cloud Control. Job classes are always created in the `SYS` schema.

The following statement creates a job class for all finance jobs:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group');
END;
/
```

All jobs in this job class are assigned to the `finance_group` resource consumer group.

To query job classes, use the `*_SCHEDULER_JOB_CLASSES` views.

See Also:

"About Resource Consumer Groups"

28.9.1.3 Altering Job Classes

You alter a job class by using the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

Other than the job class name, all the attributes of a job class can be altered. The attributes of a job class are available in the `*_SCHEDULER_JOB_CLASSES` views.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet.

28.9.1.4 Dropping Job Classes

You drop one or more job classes using the `DROP_JOB_CLASS` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

Dropping a job class means that all the metadata about the job class is removed from the database.

You can drop several job classes in one call by providing a comma-delimited list of job class names to the `DROP_JOB_CLASS` procedure call. For example, the following statement drops three job classes:

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB_CLASS('jobclass1, jobclass2, jobclass3');
END;
/
```

28.9.2 Setting Relative Job Priorities Within a Job Class

You can change the relative priorities of jobs within the same job class by using the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package. Job priorities must be in the range of 1-5, where 1 is the highest priority.

For example, the following statement changes the job priority for `my_job1` to a setting of `1`:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'job_priority',
   value          =>   1);
END;
/
```

You can verify that the attribute was changed by issuing the following statement:

```
SELECT JOB_NAME, JOB_PRIORITY FROM DBA_SCHEDULER_JOBS;

JOB_NAME                       JOB_PRIORITY
------------------------------ ------------
MY_EMP_JOB                                3
MY_EMP_JOB1                               1
MY_NEW_JOB1                               3
MY_NEW_JOB2                               3
MY_NEW_JOB3                               3
```

Overall priority of a job within the system is determined first by the combination of the resource consumer group that the job class of the job is assigned to and the current resource plan, and then by relative priority within the job class.

See Also:

- "Allocating Resources Among Jobs Using Resource Manager"
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE` procedure

28.9.3 Managing Job Scheduling and Job Priorities with Windows

You create windows to automatically start jobs or to change resource allocation among jobs during various time periods of the day, week, and so on. A window is represented by an interval of time.

- About Job Scheduling and Job Priorities with Windows
  Windows provide a way to automatically activate different resource plans at different times. Running jobs can then see a change in the resources that are allocated to them when there is a change in resource plan.

- Window Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common window tasks.
- Creating Windows
  You can use Cloud Control or the `DBMS_SCHEDULER.CREATE_WINDOW` procedure to create windows.
- Altering Windows
  You alter a window by modifying its attributes. You do so with the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures in the `DBMS_SCHEDULER` package or Cloud Control.
- Opening Windows
  When a window opens, the Scheduler switches to the resource plan that has been associated with it during its creation. If there are jobs running when the window opens, the resources allocated to them might change due to the switch in resource plan.
- Closing Windows
  A window can close based on a schedule, or it can be closed manually.
- Dropping Windows
  You drop one or more windows using the `DROP_WINDOW` procedure in the `DBMS_SCHEDULER` package or Cloud Control.
- Disabling Windows
  You disable one or more windows using the `DISABLE` procedure in the `DBMS_SCHEDULER` package or with Cloud Control.
- Enabling Windows
  You enable one or more windows using the `ENABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

28.9.3.1 About Job Scheduling and Job Priorities with Windows

Windows provide a way to automatically activate different resource plans at different times. Running jobs can then see a change in the resources that are allocated to them when there is a change in resource plan.

A job can name a window in its `schedule_name` attribute. The Scheduler then starts the job with the window *opens*. A window has a schedule associated with it, so it can open at various times during your workload cycle.

These are the key attributes of a window:

- Schedule

  This controls when the window is in effect.
- Duration

  This controls how long the window is open.
- Resource plan

  This names the resource plan that activates when the window opens.

Only one window can be in effect at any given time. Windows belong to the `SYS` schema.

All window activity is logged in the `*_SCHEDULER_WINDOW_LOG` views, otherwise known as the **window logs**. See "Window Log" for examples of window logging.

See Also:

"Windows" for an overview of windows.

28.9.3.2 Window Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common window tasks.

Table 28-9 illustrates common window tasks and the procedures you use to handle them.

Table 28-9 Window Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a window	`CREATE_WINDOW`	`MANAGE SCHEDULER`
Open a window	`OPEN_WINDOW`	`MANAGE SCHEDULER`
Close a window	`CLOSE_WINDOW`	`MANAGE SCHEDULER`
Alter a window	`SET_ATTRIBUTE`	`MANAGE SCHEDULER`
Drop a window	`DROP_WINDOW`	`MANAGE SCHEDULER`
Disable a window	`DISABLE`	`MANAGE SCHEDULER`
Enable a window	`ENABLE`	`MANAGE SCHEDULER`

See "Scheduler Privileges" for further information regarding privileges.

28.9.3.3 Creating Windows

You can use Cloud Control or the `DBMS_SCHEDULER.CREATE_WINDOW` procedure to create windows.

Using the procedure, you can leave the `resource_plan` parameter `NULL`. In this case, when the window opens, the current plan remains in effect.

You must have the `MANAGE SCHEDULER` privilege to create windows.

When you specify a schedule for a window, the Scheduler does not check if there is already a window defined for that schedule. Therefore, this may result in windows that overlap. Also, using a named schedule that has a PL/SQL expression as its repeat interval is not supported for windows

See the `CREATE_WINDOW` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details on window attributes.

The following example creates a window named `daytime` that enables the `mixed_workload_plan` resource plan during office hours:

```
BEGIN
   DBMS_SCHEDULER.CREATE_WINDOW (
     window_name      => 'daytime',
     resource_plan    => 'mixed_workload_plan',
     start_date       => '28-APR-09 08.00.00 AM',
     repeat_interval  => 'freq=daily; byday=mon,tue,wed,thu,fri',
     duration         => interval '9' hour,
     window_priority  => 'low',
    comments         => 'OLTP transactions have priority');
END;
/
```

To verify that the window was created properly, query the view `DBA_SCHEDULER_WINDOWS`. For example, issue the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN, DURATION, REPEAT_INTERVAL FROM DBA_SCHEDULER_WINDOWS;

WINDOW_NAME    RESOURCE_PLAN         DURATION       REPEAT_INTERVAL
-----------    -------------------   -------------  ---------------
DAYTIME        MIXED_WORKLOAD_PLAN   +000 09:00:00  freq=daily; byday=mon,tue,wed,thu,fri
```

28.9.3.4 Altering Windows

You alter a window by modifying its attributes. You do so with the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures in the `DBMS_SCHEDULER` package or Cloud Control.

With the exception of `WINDOW_NAME`, all the attributes of a window can be changed when it is altered. See the `CREATE_WINDOW` procedure in *Oracle Database PL/SQL Packages and Types Reference* for window attribute details.

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

All windows can be altered. If you alter a window that is disabled, it will remain disabled after it is altered. An enabled window will be automatically disabled, altered, and then reenabled, if the validity checks performed during the enable process are successful.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` procedures.

28.9.3.5 Opening Windows

When a window opens, the Scheduler switches to the resource plan that has been associated with it during its creation. If there are jobs running when the window opens, the resources allocated to them might change due to the switch in resource plan.

There are two ways a window can open:

- According to the window's schedule
- Manually, using the `OPEN_WINDOW` procedure

  This procedure opens the window independent of its schedule. This window will open and the resource plan associated with it will take effect immediately. Only an enabled window can be manually opened.

  In the `OPEN_WINDOW` procedure, you can specify the time interval that the window should be open for, using the `duration` attribute. The duration is of type interval day to second. If the duration is not specified, then the window will be opened for the regular duration as stored with the window.

  Opening a window manually has no impact on regular scheduled runs of the window.

  When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

  You can force a window to open even if there is one already open by setting the `force` option to `TRUE` in the `OPEN_WINDOW` call or Cloud Control.

When the `force` option is set to `TRUE`, the Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority. You can open a window that is already open. In this case, the window stays open for the duration specified in the call, from the time the `OPEN_WINDOW` command was issued.

Consider an example to illustrate this. `window1` was created with a duration of four hours. It has how been open for two hours. If at this point you reopen `window1` using the `OPEN_WINDOW` call and do not specify a duration, then `window1` will be open for another four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes.

When a window opens, an entry is made in the window log.

A window can fail to switch resource plans if the current resource plan has been manually switched using the `ALTER SYSTEM` statement with the `FORCE` option, or using the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure with the `allow_scheduler_plan_switches` argument set to `FALSE`. In this case, the failure to switch resource plans is written to the window log.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_SCHEDULER.OPEN_WINDOW` procedure
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` procedure

28.9.3.6 Closing Windows

A window can close based on a schedule, or it can be closed manually.

There are two ways a window can close:

- Based on a schedule

  A window will close based on the schedule defined at creation time.
- Manually, using the `CLOSE_WINDOW` procedure

  The `CLOSE_WINDOW` procedure will close an open window prematurely.

A closed window means that it is no longer in effect. When a window is closed, the Scheduler will switch the resource plan to the one that was in effect outside the window or in the case of overlapping windows to another window. If you try to close a window that does not exist or is not open, an error is generated.

A job that is running will not stop when the window it is running in closes unless the attribute `stop_on_window_close` was set to `TRUE` when the job was created. However, the resources allocated to the job may change because the resource plan may change.

When a running job has a window group as its schedule, the job will not be stopped when its window is closed if another window that is also a member of the same window group then becomes active. This is the case even if the job was created with the attribute `stop_on_window_close` set to `TRUE`.

When a window is closed, an entry will be added to the window log `DBA_SCHEDULER_WINDOW_LOG`.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CLOSE_WINDOW` procedure.

28.9.3.7 Dropping Windows

You drop one or more windows using the `DROP_WINDOW` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

When a window is dropped, all metadata about the window is removed from the `*_SCHEDULER_WINDOWS` views. All references to the window are removed from window groups.

You can drop several windows in one call by providing a comma-delimited list of window names or window group names to the `DROP_WINDOW` procedure. For example, the following statement drops both windows and window groups:

```
BEGIN
  DBMS_SCHEDULER.DROP_WINDOW ('window1, window2, window3,
   windowgroup1, windowgroup2');
END;
/
```

Note that if a window group name is provided, then the windows in the window group are dropped, but the window group is not dropped. To drop the window group, you must use the `DROP_GROUP` procedure.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DROP_GROUP` procedure.

28.9.3.8 Disabling Windows

You disable one or more windows using the `DISABLE` procedure in the `DBMS_SCHEDULER` package or with Cloud Control.

Therefore, the window will not open. However, the metadata of the window is still there, so it can be reenabled. Because the `DISABLE` procedure is used for several Scheduler objects, when disabling windows, they must be preceded by `SYS`.

A window can also become disabled for other reasons. For example, a window will become disabled when it is at the end of its schedule. Also, if a window points to a schedule that no longer exists, it becomes disabled.

If there are jobs that have the window as their schedule, you will not be able to disable the window unless you set `force` to `TRUE` in the procedure call. By default, `force` is set to `FALSE`. When the window is disabled, those jobs that have the window as their schedule will not be disabled.

You can disable several windows in one call by providing a comma-delimited list of window names or window group names to the `DISABLE` procedure call. For example, the following statement disables both windows and window groups:

```
BEGIN
  DBMS_SCHEDULER.DISABLE ('sys.window1, sys.window2,
   sys.window3, sys.windowgroup1, sys.windowgroup2');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DISABLE` procedure.

28.9.3.9 Enabling Windows

You enable one or more windows using the `ENABLE` procedure in the `DBMS_SCHEDULER` package or Cloud Control.

An enabled window is one that can be opened. Windows are, by default, created `enabled`. When a window is enabled using the `ENABLE` procedure, a validity check is performed and only if this is successful will the window be enabled. When a window is enabled, it is logged in the window log table. Because the `ENABLE` procedure is used for several Scheduler objects, when enabling windows, they must be preceded by `SYS`.

You can enable several windows in one call by providing a comma-delimited list of window names. For example, the following statement enables three windows:

```
BEGIN
  DBMS_SCHEDULER.ENABLE ('sys.window1, sys.window2, sys.window3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `ENABLE` procedure.

28.9.4 Managing Job Scheduling and Job Priorities with Window Groups

Window groups provide an easy way to schedule jobs that must run during multiple time periods throughout the day, week, and so on. If you create a window group, add windows to it, and then name this window group in a job's `schedule_name` attribute, the job runs during all the windows in the window group. Window groups reside in the `SYS` schema.

- Window Group Tasks and Their Procedures
  You use procedures in the `DBMS_SCHEDULER` package to administer common window group tasks.
- Creating Window Groups
  You create a window group by using the `DBMS_SCHEDULER.CREATE_GROUP` procedure, specifying a group type of '`WINDOW`'.
- Dropping Window Groups
  You drop one or more window groups by using the `DROP_GROUP` procedure in the `DBMS_SCHEDULER` package.
- Adding a Member to a Window Group
  You add windows to a window group by using the `ADD_GROUP_MEMBER` procedure in the `DBMS_SCHEDULER` package.
- Removing a Member from a Window Group
  You can remove one or more windows from a window group by using the `REMOVE_GROUP_MEMBER` procedure in the `DBMS_SCHEDULER` package.
- Enabling a Window Group
  You enable one or more window groups using the `ENABLE` procedure in the `DBMS_SCHEDULER` package.
- Disabling a Window Group
  You disable a window group using the `DISABLE` procedure in the `DBMS_SCHEDULER` package.

See Also:

"Window Groups" for an overview of window groups.

28.9.4.1 Window Group Tasks and Their Procedures

You use procedures in the `DBMS_SCHEDULER` package to administer common window group tasks.

Table 28-10 illustrates common window group tasks and the procedures you use to handle them.

Table 28-10 Window Group Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a window group	`CREATE_GROUP`	`MANAGE SCHEDULER`
Drop a window group	`DROP_GROUP`	`MANAGE SCHEDULER`
Add a member to a window group	`ADD_GROUP_MEMBER`	`MANAGE SCHEDULER`
Drop a member from a window group	`REMOVE_GROUP_MEMBER`	`MANAGE SCHEDULER`
Enable a window group	`ENABLE`	`MANAGE SCHEDULER`
Disable a window group	`DISABLE`	`MANAGE SCHEDULER`

See "Scheduler Privileges" for further information regarding privileges.

28.9.4.2 Creating Window Groups

You create a window group by using the `DBMS_SCHEDULER.CREATE_GROUP` procedure, specifying a group type of '`WINDOW`'.

You can specify the member windows of the group when you create the group, or you can add them later using the `ADD_GROUP_MEMBER` procedure. A window group cannot be a member of another window group. You can, however, create a window group that has no members.

If you create a window group and you specify a member window that does not exist, an error is generated and the window group is not created. If a window is already a member of a window group, it is not added again.

Window groups are created in the `SYS` schema. Window groups, like windows, are created with access to `PUBLIC`, therefore, no privileges are required to access window groups.

The following statement creates a window group called `downtime` and adds two windows (`weeknights` and `weekends`) to it:

```
BEGIN
  DBMS_SCHEDULER.CREATE_GROUP (
   group_name   =>  'downtime',
   group_type   =>  'WINDOW',
   member       =>  'weeknights, weekends');
END;
/
```

To verify the window group contents, issue the following queries as a user with the MANAGE SCHEDULER privilege:

```
SELECT group_name, enabled, number_of_members FROM dba_scheduler_groups
  WHERE group_type = 'WINDOW';

GROUP_NAME     ENABLED  NUMBER_OF_MEMBERS
-------------- -------- -----------------
DOWNTIME       TRUE                     2

SELECT group_name, member_name FROM dba_scheduler_group_members;

GROUP_NAME      MEMBER_NAME
--------------- --------------------
DOWNTIME        "SYS"."WEEKENDS"
DOWNTIME        "SYS"."WEEKNIGHTS"
```

28.9.4.3 Dropping Window Groups

You drop one or more window groups by using the DROP_GROUP procedure in the DBMS_SCHEDULER package.

This call will drop the window group but not the windows that are members of this window group. To drop all the windows that are members of this group but not the window group itself, you can use the DROP_WINDOW procedure and provide the name of the window group to the call.

You can drop several window groups in one call by providing a comma-delimited list of window group names to the DROP_GROUP procedure call. You must precede each window group name with the SYS schema. For example, the following statement drops three window groups:

```
BEGIN
DBMS_SCHEDULER.DROP_GROUP('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

28.9.4.4 Adding a Member to a Window Group

You add windows to a window group by using the ADD_GROUP_MEMBER procedure in the DBMS_SCHEDULER package.

You can add several members to a window group in one call, by specifying a comma-delimited list of windows. For example, the following statement adds two windows to the window group window_group1:

```
BEGIN
  DBMS_SCHEDULER.ADD_GROUP_MEMBER ('sys.windowgroup1','window2, window3');
END;
/
```

If an already open window is added to a window group, the Scheduler will not start jobs that point to this window group until the next window in the window group opens.

28.9.4.5 Removing a Member from a Window Group

You can remove one or more windows from a window group by using the REMOVE_GROUP_MEMBER procedure in the DBMS_SCHEDULER package.

Jobs with the stop_on_window_close flag set will only be stopped when a window closes. Dropping an open window from a window group has no impact on this.

You can remove several members from a window group in one call by specifying a comma-delimited list of windows. For example, the following statement drops two windows:

```
BEGIN
  DBMS_SCHEDULER.REMOVE_GROUP_MEMBER('sys.window_group1', 'window2, window3');
END;
/
```

28.9.4.6 Enabling a Window Group

You enable one or more window groups using the `ENABLE` procedure in the `DBMS_SCHEDULER` package.

By default, window groups are created `ENABLED`. For example:

```
BEGIN
  DBMS_SCHEDULER.ENABLE('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

28.9.4.7 Disabling a Window Group

You disable a window group using the `DISABLE` procedure in the `DBMS_SCHEDULER` package.

A job with a disabled window group as its schedule does not run when the member windows open. Disabling a window group does not disable its member windows.

You can also disable several window groups in one call by providing a comma-delimited list of window group names. For example, the following statement disables three window groups:

```
BEGIN
  DBMS_SCHEDULER.DISABLE('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

28.9.5 Allocating Resources Among Jobs Using Resource Manager

The Database Resource Manager (Resource Manager) controls how resources are allocated among database sessions. It not only controls asynchronous sessions like Scheduler jobs, but also synchronous sessions like user sessions.

It groups all "units of work" in the database into resource consumer groups and uses a resource plan to specify how the resources are allocated among the various consumer groups. The primary system resource that the Resource Manager allocates is CPU.

For Scheduler jobs, resources are allocated by first assigning each job to a job class, and then associating a job class with a consumer group. Resources are then distributed among the Scheduler jobs and other sessions within the consumer group. You can also assign relative priorities to the jobs in a job class, and resources are distributed to those jobs accordingly.

You can manually change the current resource plan at any time. Another way to change the current resource plan is by creating Scheduler windows. Windows have a resource plan attribute. When a window opens, the current plan is switched to the window's resource plan.

The Scheduler tries to limit the number of jobs that are running simultaneously so that at least some jobs can complete, rather than running a lot of jobs concurrently but without enough resources for any of them to complete.

The Scheduler and the Resource Manager are tightly integrated. The job coordinator obtains database resource availability from the Resource Manager. Based on that information, the coordinator determines how many jobs to start. It will only start jobs from those job classes that will have enough resources to run. The coordinator will keep starting jobs in a particular job class that maps to a consumer group until the Resource Manager determines that the maximum resource allocated for that consumer group has been reached. Therefore, there might be jobs in the job table that are ready to run but will not be picked up by the job coordinator because there are no resources to run them. Therefore, there is no guarantee that a job will run at the exact time that it was scheduled. The coordinator picks up jobs from the job table on the basis of which consumer groups still have resources available.

The Resource Manager continues to manage the resources that are assigned to each running job based on the specified resource plan. Keep in mind that the Resource Manager can only manage database processes. The active management of resources does not apply to external jobs.

See Also:

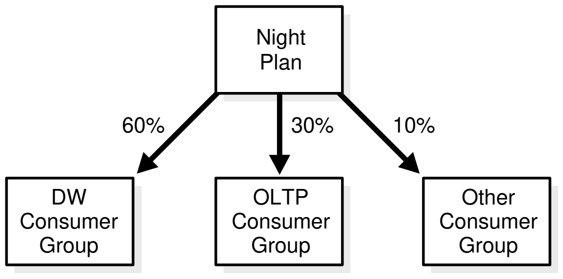
Managing Resources with Oracle Database Resource Manager

28.9.6 Example of Resource Allocation for Jobs

An example illustrates how resources are allocated for jobs.

Assume that the active resource plan is called "Night Plan" and that there are three job classes: `JC1`, which maps to consumer group `DW`; `JC2`, which maps to consumer group `OLTP`; and `JC3`, which maps to the default consumer group. Figure 28-2 offers a simple graphical illustration of this scenario.

Figure 28-2 Sample Resource Plan



This resource plan clearly gives priority to jobs that are part of job class `JC1`. Consumer group `DW` gets 60% of the resources, thus jobs that belong to job class `JC1` will get 60% of the resources. Consumer group `OLTP` has 30% of the resources, which implies that jobs in job class `JC2` will get 30% of the resources. The consumer group `Other` specifies that all other consumer groups will be getting 10% of the resources. Therefore, all jobs that belong in job class `JC3` will share 10% of the resources and can get a maximum of 10% of the resources.

Note that resources that remain unused by one consumer group are available from use by the other consumer groups. So if the jobs in job class JC1 do not fully use the allocated 60%, the unused portion is available for use by jobs in classes JC2 and JC3. Note also that the Resource Manager does not begin to restrict resource usage at all until CPU usage reaches 100%. See Managing Resources with Oracle Database Resource Manager for more information.

28.10 Monitoring Jobs

You can monitor jobs in several different ways.

- About Monitoring Jobs
  There are several ways to monitor Scheduler jobs.
- The Job Log
  You can view results for both local and remote jobs in the job log.
- Monitoring Multiple Destination Jobs
  For multiple-destination jobs, the overall parent job state depends on the outcome of the child jobs.
- Monitoring Job State with Events Raised by the Scheduler
  Scheduler can raise an event when a job changes state.
- Monitoring Job State with E-mail Notifications
  Scheduler an send an e-mail when a job changes state.

28.10.1 About Monitoring Jobs

There are several ways to monitor Scheduler jobs.

You can monitor Scheduler jobs in the following ways:

- Viewing the job log

  The job log includes the data dictionary views `*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`, where:

  * = {`DBA`|`ALL`|`USER`}

  See "Viewing the Job Log".
- Querying additional data dictionary views

  Query views such as `DBA_SCHEDULER_RUNNING_JOBS` and `DBA_SCHEDULER_RUNNING_CHAINS` to show the status and details of running jobs and chains.
- Writing applications that receive job state events from the Scheduler

  See "Monitoring Job State with Events Raised by the Scheduler"
- Configuring jobs to send e-mail notifications upon a state change

  See "Monitoring Job State with E-mail Notifications"

28.10.2 The Job Log

You can view results for both local and remote jobs in the job log.

- Viewing the Job Log
  You can view information about job runs, job state changes, and job failures in the job log. The job log shows results for both local and remote jobs.
- Run Details
  For every row in `*_SCHEDULER_JOB_LOG` for which the operation is `RUN`, `RETRY_RUN`, or `RECOVERY_RUN`, there is a corresponding row in the `*_SCHEDULER_JOB_RUN_DETAILS` view.
- Precedence of Logging Levels in Jobs and Job Classes
  Both jobs and job classes have a `logging_level` attribute.

28.10.2.1 Viewing the Job Log

You can view information about job runs, job state changes, and job failures in the job log. The job log shows results for both local and remote jobs.

The job log is implemented as the following two data dictionary views:

- `*_SCHEDULER_JOB_LOG`
- `*_SCHEDULER_JOB_RUN_DETAILS`

Depending on the logging level that is in effect, the Scheduler can make job log entries whenever a job is run and when a job is created, dropped, enabled, and so on. For a job that has a repeating schedule, the Scheduler makes multiple entries in the job log—one for each job instance. Each log entry provides information about a particular run, such as the job completion status.

The following example shows job log entries for a repeating job that has a value of 4 for the `max_runs` attribute:

```
SELECT job_name, job_class, operation, status FROM USER_SCHEDULER_JOB_LOG;

JOB_NAME         JOB_CLASS            OPERATION       STATUS
---------------- -------------------- --------------- ----------
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               COMPLETED
```

You can control how frequently information is written to the job log by setting the `logging_level` attribute of either a job or a job class. Table 28-11 shows the possible values for `logging_level`.

Table 28-11 Job Logging Levels

Logging Level	Description
`DBMS_SCHEDULER.LOGGING_OFF`	No logging is performed.
`DBMS_SCHEDULER.LOGGING_FAILED_RUNS`	A log entry is made only if the job fails.
`DBMS_SCHEDULER.LOGGING_RUNS`	A log entry is made each time the job is run.
`DBMS_SCHEDULER.LOGGING_FULL`	A log entry is made every time the job runs and for every operation performed on a job, including create, enable/disable, update (with `SET_ATTRIBUTE`), stop, and drop.

Log entries for job runs are not made until after the job run completes successfully, fails, or is stopped.

The following example shows job log entries for a complete job lifecycle. In this case, the logging level for the job class is `LOGGING_FULL`, and the job is a non-repeating job. After the first successful run, the job is enabled again, so it runs once more. It is then stopped and dropped.

```
SELECT to_char(log_date, 'DD-MON-YY HH24:MI:SS') TIMESTAMP, job_name,
  job_class, operation, status FROM USER_SCHEDULER_JOB_LOG
  WHERE job_name = 'JOB2' ORDER BY log_date;

TIMESTAMP            JOB_NAME  JOB_CLASS  OPERATION  STATUS
```

```
-------------------- --------- ---------- ---------- ---------
18-DEC-07 23:10:56   JOB2      CLASS1     CREATE
18-DEC-07 23:12:01   JOB2      CLASS1     UPDATE
18-DEC-07 23:12:31   JOB2      CLASS1     ENABLE
18-DEC-07 23:12:41   JOB2      CLASS1     RUN        SUCCEEDED
18-DEC-07 23:13:12   JOB2      CLASS1     ENABLE
18-DEC-07 23:13:18   JOB2                 RUN        STOPPED
18-DEC-07 23:19:36   JOB2      CLASS1     DROP
```

28.10.2.2 Run Details

For every row in `*_SCHEDULER_JOB_LOG` for which the operation is `RUN`, `RETRY_RUN`, or `RECOVERY_RUN`, there is a corresponding row in the `*_SCHEDULER_JOB_RUN_DETAILS` view.

Rows from the two different views are correlated with their `LOG_ID` columns. You can consult the run details views to determine why a job failed or was stopped.

```
SELECT to_char(log_date, 'DD-MON-YY HH24:MI:SS') TIMESTAMP, job_name, status,
   SUBSTR(additional_info, 1, 40) ADDITIONAL_INFO
   FROM user_scheduler_job_run_details ORDER BY log_date;

TIMESTAMP            JOB_NAME   STATUS    ADDITIONAL_INFO
-------------------- ---------- --------- ----------------------------------------
18-DEC-07 23:12:41   JOB2       SUCCEEDED
18-DEC-07 23:12:18   JOB2       STOPPED   REASON="Stop job called by user:'SYSTEM'
19-DEC-07 14:12:20   REMOTE_16  FAILED    ORA-29273: HTTP request failed ORA-06512
```

The run details views also contain actual job start times and durations.

You can also use the attribute `STORE_OUTPUT` to direct the `*_SCHEDULER_JOB_RUN_DETAILS` view to store the output sent to `stdout` for external jobs or `DBMS_OUTPUT` for database jobs. When `STORE_OUTPUT` is set to `TRUE` and the `LOGGING_LEVEL` indicates that the job run should be logged, then all the output is collected and put inside the `BINARY_OUTPUT` column of this view. A `char` representation can be queried from the `OUTPUT` column.

28.10.2.3 Precedence of Logging Levels in Jobs and Job Classes

Both jobs and job classes have a `logging_level` attribute.

The possible values for this attribute are listed in Table 28-11. The default logging level for job classes is `LOGGING_RUNS`, and the default level for individual jobs is `LOGGING_OFF`. If the logging level of the job class is higher than that of a job in the class, then the logging level of the job class takes precedence. Thus, by default, all job runs are recorded in the job log.

For job classes that have very short and highly frequent jobs, the overhead of recording every single run might be too much and you might choose to turn the logging off or set logging to occur only when jobs fail. However, you might prefer to have complete logging of everything that happens with jobs in a specific class, in which case you would enable full logging for that class.

To ensure that there is logging for all jobs, the individual job creator must not be able to turn logging off. The Scheduler supports this by making the class-specified level the minimum level at which job information is logged. A job creator can only enable more logging for an individual job, not less. Thus, leaving all individual job logging levels set to `LOGGING_OFF` ensures that all jobs in a class get logged as specified in the class.

This functionality is provided for debugging purposes. For example, if the class-specific level is set to record job runs and logging is turned off at the job level, the Scheduler still logs job runs. If, however, the job creator turns on full logging and the class-specific level is set to record runs only, the higher logging level of the job takes precedence and all operations on this individual job are logged. This way, an end user can test his job by turning on full logging.

To set the logging level of an individual job, you must use the `SET_ATTRIBUTE` procedure on that job. For example, to turn on full logging for a job called `mytestjob`, issue the following statement:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'mytestjob', 'logging_level', DBMS_SCHEDULER.LOGGING_FULL);
END;
/
```

Only a user with the `MANAGE SCHEDULER` privilege can set the logging level of a job class.

See Also:

"Monitoring and Managing Window and Job Logs" for more information about setting the job class logging level

28.10.3 Monitoring Multiple Destination Jobs

For multiple-destination jobs, the overall parent job state depends on the outcome of the child jobs.

For example, if all child jobs succeed, the parent job state is set to `SUCCEEDED`. If all fail, the parent job state is set to `FAILED`. If some fail and some succeed, the parent job state is set to `SOME FAILED`.

Due to situations that might arise on some destinations that delay the start of child jobs, there might be a significant delay before the parent job state is finalized. For repeating multiple-destination jobs, there might even be a situation in which some child jobs are on their next scheduled run while others are still working on the previous scheduled run. In this case, the parent job state is set to `INCOMPLETE`. Eventually, however, lagging child jobs may catch up to their siblings, in which case the final state of the parent job can be determined.

Table 28-12 lists the contents of the job monitoring views for multiple-destination jobs.

Table 28-12 Scheduler Data Dictionary View Contents for Multiple-Destination Jobs

View Name	Contents
`*_SCHEDULER_JOBS`	One entry for the parent job
`*_SCHEDULER_RUNNING_JOBS`	One entry for the parent job when it starts and an entry for each running child job
`*_SCHEDULER_JOB_LOG`	One entry for the parent job when it starts (operation = '`MULTIDEST_START`'), one entry for each child job when the child job completes, and one entry for the parent job when the last child job completes and thus the parent completes (operation = '`MULTIDEST_RUN`')

Table 28-12 (Cont.) Scheduler Data Dictionary View Contents for Multiple-Destination Jobs

View Name	Contents
`*_SCHEDULER_JOB_RUN_DETAILS`	One entry for each child job when the child job completes, and one entry for the parent job when the last child job completes and thus the parent completes
`*_SCHEDULER_JOB_DESTS`	One entry for each destination of the parent job

In the `*_SCHEDULER_JOB_DESTS` views, you can determine the unique job destination ID (`job_dest_id`) that is assigned to each child job. This ID represents the unique combination of a job, a credential, and a destination. You can use this ID with the `STOP_JOB` procedure. You can also monitor the job state of each child job with the `*_SCHEDULER_JOB_DESTS` views.

See Also:

- "Multiple-Destination Jobs"
- "Creating Multiple-Destination Jobs"
- "Scheduler Data Dictionary Views"

28.10.4 Monitoring Job State with Events Raised by the Scheduler

Scheduler can raise an event when a job changes state.

- About Job State Events
  You can configure a job so that the Scheduler raises an event when the job changes state.
- Altering a Job to Raise Job State Events
  To enable job state events to be raised for a job, you use the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package to turn on bit flags in the `raise_events` job attribute.
- Consuming Job State Events with your Application
  To consume job state events, your application must subscribe to the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`. This queue is a secure queue and is owned by `SYS`.

28.10.4.1 About Job State Events

You can configure a job so that the Scheduler raises an event when the job changes state.

The Scheduler can raise an event when a job starts, when a job completes, when a job exceeds its allotted run time, and so on. The consumer of the event is your application, which takes some action in response to the event. For example, if due to a high system load, a job is still not started 30 minutes after its scheduled start time, the Scheduler can raise an event that causes a handler application to stop lower priority jobs to free up system resources. The Scheduler can raise job state events for local (regular) jobs, remote database jobs, local external jobs, and remote external jobs.

Table 28-13 describes the job state event types raised by the Scheduler.

Table 28-13 Job State Event Types Raised by the Scheduler

Event Type	Description
`job_all_events`	Not an event, but a constant that provides an easy way for you to enable all events
`job_broken`	The job has been disabled and has changed to the `BROKEN` state because it exceeded the number of failures defined by the `max_failures` job attribute
`job_chain_stalled`	A job running a chain was put into the `CHAIN_STALLED` state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain `evaluation_interval` is set to `NULL`. No progress will be made in the chain unless there is manual intervention.
`job_completed`	The job completed because it reached its `max_runs` or `end_date`
`job_disabled`	The job was disabled by the Scheduler or by a call to `SET_ATTRIBUTE`
`job_failed`	The job failed, either by throwing an error or by abnormally terminating
`job_over_max_dur`	The job exceeded the maximum run duration specified by its `max_run_duration` attribute.
`job_run_completed`	A job run either failed, succeeded, or was stopped
`job_sch_lim_reached`	The job's schedule limit was reached. The job was not started because the delay in starting the job exceeded the value of the `schedule_limit` job attribute.
`job_started`	The job started
`job_stopped`	The job was stopped by a call to `STOP_JOB`
`job_succeeded`	The job completed successfully

You enable the raising of job state events by setting the `raise_events` job attribute. By default, a job does not raise any job state events.

The Scheduler uses Oracle Database Advanced Queuing to raise events. When raising a job state change event, the Scheduler enqueues a message onto a default event queue. Your applications subscribe to this queue, dequeue event messages, and take appropriate actions.

After you enable job state change events for a job, the Scheduler raises these events by enqueuing messages onto the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`. This queue is a secure queue, so depending on your application, you may have to configure the queue to enable certain users to perform operations on it.

To prevent unlimited growth of the Scheduler event queue, events raised by the Scheduler expire in 24 hours by default. You can change this expiry time by setting the `event_expiry_time` Scheduler attribute with the `DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE` procedure. Expired events are deleted from the event queue.

See Also:

- *Oracle Database Advanced Queuing User's Guide* for information about configuring secure queues using Oracle Database Advanced Queuing
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE` procedure

28.10.4.2 Altering a Job to Raise Job State Events

To enable job state events to be raised for a job, you use the `SET_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package to turn on bit flags in the `raise_events` job attribute.

Each bit flag represents a different job state to raise an event for. For example, turning on the least significant bit enables `job started` events to be raised. To enable multiple state change event types in one call, you add the desired bit flag values together and supply the result as an argument to `SET_ATTRIBUTE`.

The following example enables multiple state change events for job `dw_reports`. It enables the following event types, both of which indicate some kind of error.

- `JOB_FAILED`
- `JOB_SCH_LIM_REACHED`

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE('dw_reports', 'raise_events',
   DBMS_SCHEDULER.JOB_FAILED + DBMS_SCHEDULER.JOB_SCH_LIM_REACHED);
END;
/
```

Note:

You do not need to enable the `JOB_OVER_MAX_DUR` event with the `raise_events` job attribute; it is always enabled.

See Also:

The discussion of `DBMS_SCHEDULER.SET_ATTRIBUTE` in *Oracle Database PL/SQL Packages and Types Reference* for the names and values of job state bit flags

28.10.4.3 Consuming Job State Events with your Application

To consume job state events, your application must subscribe to the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`. This queue is a secure queue and is owned by `SYS`.

To create a subscription to this queue for a user, do the following:

1. Log in to the database as the `SYS` user or as a user with the `MANAGE ANY QUEUE` privilege.

2. Subscribe to the queue using a new or existing agent.
3. Run the package procedure `DBMS_AQADM.ENABLE_DB_ACCESS` as follows:

   ```
   DBMS_AQADM.ENABLE_DB_ACCESS(agent_name, db_username);
   ```

   where *`agent_name`* references the agent that you used to subscribe to the events queue, and *`db_username`* is the user for whom you want to create a subscription.

There is no need to grant dequeue privileges to the user. The dequeue privilege is granted on the Scheduler event queue to `PUBLIC`.

As an alternative, the user can subscribe to the Scheduler event queue using the `ADD_EVENT_QUEUE_SUBSCRIBER` procedure, as shown in the following example:

```
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER(subscriber_name);
```

where *`subscriber_name`* is the name of the Oracle Database Advanced Queuing (AQ) agent to be used to subscribe to the Scheduler event queue. (If it is `NULL`, an agent is created whose name is the user name of the calling user.) This call both creates a subscription to the Scheduler event queue and grants the user permission to dequeue using the designated agent. The subscription is rule-based. The rule permits the user to see only events raised by jobs that the user owns, and filters out all other messages. After the subscription is in place, the user can either poll for messages at regular intervals or register with AQ for notification.

See *Oracle Database Advanced Queuing User's Guide* for more information.

Scheduler Event Queue

The Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE` is of type `scheduler$_event_info`. See *Oracle Database PL/SQL Packages and Types Reference* for details on this type.

28.10.5 Monitoring Job State with E-mail Notifications

Scheduler an send an e-mail when a job changes state.

- About E-mail Notifications
  You can configure a job to send e-mail notifications when it changes state.
- Adding E-mail Notifications for a Job
  You use the `DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION` package procedure to add e-mail notifications for a job.
- Removing E-mail Notifications for a Job
  You use the `DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION` package procedure to remove e-mail notifications for a job.
- Viewing Information About E-mail Notifications
  You can view information about current e-mail notifications by querying the views `*_SCHEDULER_NOTIFICATIONS`.

28.10.5.1 About E-mail Notifications

You can configure a job to send e-mail notifications when it changes state.

The job state events for which e-mails can be sent are listed in Table 28-13. E-mail notifications can be sent to multiple recipients, and can be triggered by any event in a list of job state events that you specify. You can also provide a filter condition, and only generate notifications job state events that match the filter condition. You can include variables such as job owner, job name, event type, error code, and error message in both the subject and body of the message. The Scheduler automatically sets values for these variables before sending the e-mail notification.

You can configure many job state e-mail notifications for a single job. The notifications can differ by job state event list, recipients, and filter conditions.

For example, you can configure a job to send an e-mail to both the principle DBA and one of the senior DBAs whenever the job fails with error code 600 or 700. You can also configure the same job to send a notification to only the principle DBA if the job fails to start at its scheduled time.

Before you can configure jobs to send e-mail notifications, you must set the Scheduler attribute `email_server` to the address of the SMTP server to use to send the e-mail. You may also optionally set the Scheduler attribute `email_sender` to a default sender e-mail address for those jobs that do not specify a sender.

The Scheduler includes support for the SSL and TLS protocols when communicating with the SMTP server. The Scheduler also supports SMTP servers that require authentication.

See Also:

"Setting Scheduler Preferences" for details about setting e-mail notification–related attributes

28.10.5.2 Adding E-mail Notifications for a Job

You use the `DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION` package procedure to add e-mail notifications for a job.

For example, the following procedure adds an e-mail notification for the `OED_JOB` job:

```
BEGIN
 DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name    =>  'EOD_JOB',
  recipients =>  'jsmith@example.com, rjones@example.com',
  sender      =>  'do_not_reply@example.com',
  subject     =>  'Scheduler Job Notification-%job_owner%.%job_name%-%event_type%',
  body        =>   '%event_type% occurred at %event_timestamp%. %error_message%',
  events      =>  'JOB_FAILED, JOB_BROKEN, JOB_DISABLED, JOB_SCH_LIM_REACHED');
END;
/
```

Note the variables, enclosed in the '%' character, used in the `subject` and `body` arguments. When you specify multiple recipients and multiple events, each recipient is notified when any of the specified events is raised. You can verify this by querying the view `USER_SCHEDULER_NOTIFICATIONS`.

```
SELECT JOB_NAME, RECIPIENT, EVENT FROM USER_SCHEDULER_NOTIFICATIONS;

JOB_NAME    RECIPIENT            EVENT
----------- -------------------- -------------------
EOD_JOB     jsmith@example.com   JOB_FAILED
EOD_JOB     jsmith@example.com   JOB_BROKEN
EOD_JOB     jsmith@example.com   JOB_SCH_LIM_REACHED
EOD_JOB     jsmith@example.com   JOB_DISABLED
EOD_JOB     rjones@example.com   JOB_FAILED
EOD_JOB     rjones@example.com   JOB_BROKEN
EOD_JOB     rjones@example.com   JOB_SCH_LIM_REACHED
EOD_JOB     rjones@example.com   JOB_DISABLED
```

You call `ADD_JOB_EMAIL_NOTIFICATION` once for each different set of notifications that you want to configure for a job. You must specify `job_name` and `recipients`. All other arguments have defaults. The default `sender` is defined by a Scheduler attribute, as described in the previous section. See the `ADD_JOB_EMAIL_NOTIFICATION` procedure in *Oracle Database PL/SQL Packages and Types Reference* for defaults for the `subject`, `body`, and `events` arguments.

The following example configures an additional e-mail notification for the same job for a different event. This example accepts the defaults for the `sender`, `subject`, and `body` arguments.

```
BEGIN
 DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name         =>  'EOD_JOB',
  recipients       =>  'jsmith@example.com',
  events           =>  'JOB_OVER_MAX_DUR');
END;
/
```

This example could have also omitted the `events` argument to accept event defaults.

The next example is similar to the first, except that it uses a filter condition to specify that an e-mail notification is to be sent only when the error number that causes the job to fail is 600 or 700.

```
BEGIN
 DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name         => 'EOD_JOB',
  recipients       => 'jsmith@example.com, rjones@example.com',
  sender           => 'do_not_reply@example.com',
  subject          => 'Job Notification-%job_owner%.%job_name%-%event_type%',
  body             =>  '%event_type% at %event_timestamp%. %error_message%',
  events           => 'JOB_FAILED',
  filter_condition => ':event.error_code=600 or :event.error_code=700');
END;
/
```

See Also:

The `ADD_JOB_EMAIL_NOTIFICATION` procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.10.5.3 Removing E-mail Notifications for a Job

You use the `DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION` package procedure to remove e-mail notifications for a job.

For example, the following procedure removes an e-mail notification for the `OED_JOB` job:

```
BEGIN
 DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION (
  job_name   =>  'EOD_JOB',
  recipients =>  'jsmith@example.com, rjones@example.com',
  events     =>  'JOB_DISABLED, JOB_SCH_LIM_REACHED');
```

```
END;
/
```

When you specify multiple recipients and multiple events, the notification for each specified event is removed for each recipient. Running the same query as that of the previous section, the results are now the following:

```
SELECT JOB_NAME, RECIPIENT, EVENT FROM USER_SCHEDULER_NOTIFICATIONS;

JOB_NAME    RECIPIENT            EVENT
----------- -------------------- -------------------
EOD_JOB     jsmith@example.com   JOB_FAILED
EOD_JOB     jsmith@example.com   JOB_BROKEN
EOD_JOB     rjones@example.com   JOB_FAILED
EOD_JOB     rjones@example.com   JOB_BROKEN
```

Additional rules for specifying REMOVE_JOB_EMAIL_NOTIFICATION arguments are as follows:

- If you leave the events argument NULL, notifications for all events for the specified recipients are removed.
- If you leave recipients NULL, notifications for all recipients for the specified events are removed.
- If you leave both recipients and events NULL, then all notifications for the job are removed.
- If you include a recipient and event for which you did not previously create a notification, no error is generated.

See Also:

The REMOVE_JOB_EMAIL_NOTIFICATION procedure in *Oracle Database PL/SQL Packages and Types Reference*

28.10.5.4 Viewing Information About E-mail Notifications

You can view information about current e-mail notifications by querying the views *_SCHEDULER_NOTIFICATIONS.

See Also:

Oracle Database Reference for details on these views

29 Administering Oracle Scheduler

You can configure, manage, monitor, and troubleshoot Oracle Scheduler.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

Note:

This chapter describes how to use the `DBMS_SCHEDULER` package to administer Oracle Scheduler. You can accomplish many of the same tasks using Oracle Enterprise Manager Cloud Control.

See *Oracle Database PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` information and the Cloud Control online help for information on Oracle Scheduler pages.

See *Oracle Multitenant Administrator's Guide* for information on using Oracle Scheduler with CDB.

- Configuring Oracle Scheduler
  Configuring Oracle Scheduler includes tasks such as setting privileges and preferences, and using the Oracle Scheduler agent to run remote jobs.
- Monitoring and Managing the Scheduler
  You can view the currently active window and the resource plan associated with it, view information about currently running jobs, monitor and manage window and job logs, and manage Scheduler security.
- Import/Export and the Scheduler
  You must use the Data Pump utilities (`impdp` and `expdp`) to export Scheduler objects.
- Troubleshooting the Scheduler
  You can troubleshoot problems with Scheduler.
- Examples of Using the Scheduler
  Examples illustrate using Scheduler.
- Scheduler Reference
  There are several privileges and data dictionary views related to Scheduler.

29.1 Configuring Oracle Scheduler

Configuring Oracle Scheduler includes tasks such as setting privileges and preferences, and using the Oracle Scheduler agent to run remote jobs.

- Setting Oracle Scheduler Privileges
  You must have the SCHEDULER_ADMIN role to perform all Oracle Scheduler administration tasks. Typically, database administrators already have this role with the ADMIN option as part of the DBA role.
- Setting Scheduler Preferences
  There are several system-wide Scheduler preferences that you can set. You set these preferences by setting Scheduler attributes with the SET_SCHEDULER_ATTRIBUTE procedure in the DBMS_SCHEDULER package.
- Using the Oracle Scheduler Agent to Run Remote Jobs
  The Oracle Scheduler agent can schedule and run remote jobs.

29.1.1 Setting Oracle Scheduler Privileges

You must have the SCHEDULER_ADMIN role to perform all Oracle Scheduler administration tasks. Typically, database administrators already have this role with the ADMIN option as part of the DBA role.

For example, users SYS and SYSTEM are granted the DBA role. You can grant this role to another administrator by issuing the following statement:

```
GRANT SCHEDULER_ADMIN TO username;
```

Because the SCHEDULER_ADMIN role is a powerful role allowing a grantee to execute code as any user, you should consider granting individual Scheduler system privileges instead. Object and system privileges are granted using regular SQL grant syntax, for example, if the database administrator issues the following statement:

```
GRANT CREATE JOB TO scott;
```

After this statement is executed, scott can create jobs, schedules, programs, and file watchers in his schema. As another example, the database administrator can issue the following statement:

```
GRANT MANAGE SCHEDULER TO adam;
```

After this statement is executed, adam can create, alter, or drop windows, job classes, or window groups. adam will also be able to set and retrieve Scheduler attributes and purge Scheduler logs.

Setting Chain Privileges

Scheduler chains use underlying Oracle Rules Engine objects along with their associated privileges. To create a chain in their own schema, users must have the CREATE JOB privilege in addition to the Rules Engine privileges required to create rules, rule sets, and evaluation contexts in their own schema. These can be granted by issuing the following statement:

```
GRANT CREATE RULE, CREATE RULE SET, CREATE EVALUATION CONTEXT TO user;
```

To create a chain in a different schema, users must have the `CREATE ANY JOB` privilege in addition to the privileges required to create rules, rule sets, and evaluation contexts in schemas other than their own. These can be granted by issuing the following statement:

```
GRANT CREATE ANY RULE, CREATE ANY RULE SET,
   CREATE ANY EVALUATION CONTEXT TO user;
```

Altering or dropping chains in schemas other than the users's schema require corresponding system Rules Engine privileges for rules, rule sets, and evaluation contexts.

See Also:

"Chain Tasks and Their Procedures" for more information regarding chain privileges.

29.1.2 Setting Scheduler Preferences

There are several system-wide Scheduler preferences that you can set. You set these preferences by setting Scheduler attributes with the `SET_SCHEDULER_ATTRIBUTE` procedure in the `DBMS_SCHEDULER` package.

Setting these attributes requires the `MANAGE SCHEDULER` privilege. The attributes are:

- `default_timezone`

  It is very important that you set this attribute. Repeating jobs and windows that use the calendaring syntax need to know which time zone to use for their repeat intervals. See "Using the Scheduler Calendaring Syntax". They normally retrieve the time zone from `start_date`, but if no `start_date` is provided (which is not uncommon), they retrieve the time zone from the `default_timezone` Scheduler attribute.

  The Scheduler derives the value of `default_timezone` from the operating system environment. If the Scheduler can find no compatible value from the operating system, it sets `default_timezone` to `NULL`.

  It is crucial that you verify that `default_timezone` is set properly, and if not, that you set it. To verify it, run this query:

  ```
  SELECT DBMS_SCHEDULER.STIME FROM DUAL;

  STIME
  ---------------------------------------------------------------------------
  28-FEB-12 09.04.10.308959000 PM UTC
  ```

  To ensure that daylight savings adjustments are followed, it is recommended that you set `default_timezone` to a region name instead of an absolute time zone offset like '-8:00'. For example, if your database resides in Miami, Florida, USA, issue the following statement:

  ```
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('default_timezone','US/Eastern');
  ```

  Similarly, if your database resides in Paris, you would set this attribute to `'Europe/Warsaw'`. To see a list of valid region names, run this query:

  ```
  SELECT DISTINCT TZNAME FROM V$TIMEZONE_NAMES;
  ```

  If you do not properly set `default_timezone`, the default time zone for repeating jobs and windows will be the absolute offset retrieved from `SYSTIMESTAMP` (the time zone of the operating system environment of the database), which means that repeating jobs and windows that do not have their `start_date` set will not follow daylight savings adjustments.

- `email_server`

  This attribute specifies an SMTP server address that the Scheduler uses to send e-mail notifications for job state events. It takes the following format:

  ```
  host[:port]
  ```

  where:

  – *`host`* is the host name or IP address of the SMTP server.

  – *`port`* is the TCP port on which the SMTP server listens. If not specified, the default port of 25 is used.

  If this attribute is not specified, set to `NULL`, or set to an invalid SMTP server address, the Scheduler cannot send job state e-mail notifications.

- `email_sender`

  This attribute specifies the default e-mail address of the sender for job state e-mail notifications. It must be a valid e-mail address. If this attribute is not set or set to `NULL`, then job state e-mail notifications that do not specify a sender address do not have a FROM address in the e-mail header.

- `email_server_credential`

  This attribute specifies the schema and name of an existing credential object. The default is `NULL`.

  When an e-mail notification goes out, the Scheduler determines if the `email_server_credential` points to a valid credential object that `SYS` has execute object privileges on. If the SMTP server specified in the `email_server` attribute requires authentication, then the Scheduler uses the user name and password stored in the specified credential object to authenticate with the e-mail server.

  If the `email_server_credential` is specified, then the `email_server` attribute must specify an SMTP server that requires authentication.

  If the `email_server_credential` is not specified, then the Scheduler supports sending notification e-mails through an SMTP server for which authentication is not configured.

- `email_server_encryption`

  This attribute indicates whether encryption is enabled for this SMTP server connection, and if so, at what point encryption starts, and with which protocol.

  Values for `email_server_encryption` are:

  `NONE`: The default, indicates no encryption.

  `SSL_TLS`: Indicates that either `SSL` or `TLS` are used, from the beginning of the connection. The two sides determine which protocol is most secure. This is the most common setting for this parameter.

  `STARTTLS`: Indicates that the connection starts in an unencrypted state, but then the command `STARTTLS` directs the e-mail server to start encryption using `TLS`.

- `event_expiry_time`

  This attribute enables you to set the time in seconds before a job state event generated by the Scheduler expires (is automatically purged from the Scheduler event queue). If `NULL`, job state events expire after 24 hours.

- `log_history`

  This attribute controls the number of days that log entries for both the job log and the window log are retained. It helps prevent logs from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. Default value is 30. You can override this value at the job class level by setting a value for the `log_history` attribute of the job class.

See *Oracle Database PL/SQL Packages and Types Reference* for the syntax for the `SET_SCHEDULER_ATTRIBUTE` procedure.

29.1.3 Using the Oracle Scheduler Agent to Run Remote Jobs

The Oracle Scheduler agent can schedule and run remote jobs.

Using the Oracle Scheduler agent, the Scheduler can schedule and run two types of remote jobs:

- Remote database jobs: Remote database jobs must be run through an Oracle Scheduler agent. Oracle recommends that an agent be installed on the same host as the remote database.

  If you intend to run remote database jobs, the Scheduler agent must be Oracle Database 11*g* Release 2 (11.2) or later.
- Remote external jobs: Remote external jobs run on the same host that the Scheduler agent is installed on.

  If you intend to run only remote external jobs, Oracle Database 11*g* Release 1 (11.1) of the Scheduler agent is sufficient.

You must install Scheduler agents on all hosts that remote external jobs will run on. You should install Scheduler agents on all hosts running remote databases that remote database jobs will be run on.

Each database that runs remote jobs requires an initial setup to enable secure communications between databases and remote Scheduler agents, as described in "Setting up Databases for Remote Jobs".

Enabling remote jobs involves the following steps:

1. Enabling and Disabling Databases for Remote Jobs
2. Installing and Configuring the Scheduler Agent on a Remote Host
3. Performing Tasks with the Scheduler Agent

- Enabling and Disabling Databases for Remote Jobs
  You can set up databases for remote jobs and disable databases for remote jobs.
- Installing and Configuring the Scheduler Agent on a Remote Host
  Before you can run remote jobs on a particular host, you must install and configure the Scheduler agent.
- Performing Tasks with the Scheduler Agent
  The Scheduler agent is a standalone program that enables you to schedule and run external and database jobs on remote hosts. You start and stop the Scheduler agent using the `schagent` utility on UNIX and Linux, and the `OracleSchedulerExecutionAgent` service on Windows.

See Also:

- "About Remote External Jobs"
- "Database Jobs" for more information on remote database jobs

29.1.3.1 Enabling and Disabling Databases for Remote Jobs

You can set up databases for remote jobs and disable databases for remote jobs.

- Setting up Databases for Remote Jobs
  Before a database can run jobs using a remote Scheduler agent, the database must be properly configured, and the agent must be registered with the database.
- Disabling Remote Jobs
  You can disable remote jobs on a database by dropping the `REMOTE_SCHEDULER_AGENT` user.

29.1.3.1.1 Setting up Databases for Remote Jobs

Before a database can run jobs using a remote Scheduler agent, the database must be properly configured, and the agent must be registered with the database.

This section describes the configuration, including the required agent registration password in the database. You will later register the database, as shown in "Registering Scheduler Agents with Databases".

You can limit the number of Scheduler agents that can register, and you can set the password to expire after a specified duration.

Complete the following steps once for each database that creates and runs remote jobs.

To set up a database to create and run remote jobs:

1. Ensure that shared server is enabled.

   See "Enabling Shared Server".

   If several Scheduler agents are being used with the same database, set the value of the `SHARED_SERVERS` database initialization parameter high enough to avoid errors when all those agents try to work in parallel.

   Note:

   If you are running in multitenant mode, you must unlock the anonymous account in `CDB$ROOT`.

   Using SQL*Plus, connect to `CDB$ROOT` as `SYS` user, and enter the following command:

   ```
   SQL> alter session set container = CDB$ROOT;
   SQL> alter user anonymous account unlock container=current;
   ```

2. Using SQL*Plus, connect to the database (specify pluggable database under multitenant mode) as the `SYS` user.

3. Enter the following command to verify that the XML DB option is installed:

   ```
   SQL> DESC RESOURCE_VIEW
   ```

   If XML DB is not installed, this command returns an "object does not exist" error.

   > **Note:**
   >
   > If XML DB is not installed, you must install it before continuing.

4. If you are using HTTPS connections, then add a certificate to the database wallet as follows:

   > **Note:**
   >
   > Check that the database is not using the wallet while adding the certificate, or else shutdown the database while adding the certificate and then startup the database.

   a. If you do not have an existing database wallet in the `ORACLE_HOME/admin/$ORACLE_SID/xdb_wallet` directory, then create one using the `orapki` command line utility. For example:

   ```
   orapki wallet create -wallet $ORACLE_HOME/admin/$ORACLE_SID/xdb_wallet -
   pwd wallet_password -auto_login
   ```

   b. Add a certificate to the wallet using the `orapki` command line utility. For example:

   ```
   orapki wallet add -wallet $ORACLE_HOME/admin/$ORACLE_SID/xdb_wallet -dn
   CN=fully_qualified_domain_name -self_signed -pwd wallet_password -
   validity number_of_days -keysize key_size_for_the_certificate(512|1024|
   2048)
   ```

   > **Note:**
   >
   > The use of weaker encryption keys is deprecated in Oracle Database 21c.

   > **See Also:**
   >
   > *Oracle Database Security Guide* for more information about adding certificates to a database wallet using the `orapki` utility

5. Enable HTTP(S) connections to the database as follows:

   a. Determine whether or not the Oracle XML DBM HTTP(S) Server is enabled:

   Run the following command for HTTP connections:

   ```
   SQL> SELECT DBMS_XDB_CONFIG.GETHTTPPORT() FROM DUAL;
   ```

Run the following command for HTTPS connections:

```
SQL> SELECT DBMS_XDB_CONFIG.GETHTTPSPORT() FROM DUAL;
```

If the statement returns `0`, then Oracle XML DBM HTTP(S) Server is disabled.

b. Enable Oracle XML DB HTTP(S) Server on a nonzero port by logging in as `SYS` and run the following commands:

   If you are using HTTP connections:

   ```
   SQL> EXEC DBMS_XDB_CONFIG.SETHTTPPORT (port);
   SQL> COMMIT;
   ```

   If you are using HTTPS connections:

   ```
   SQL> EXEC DBMS_XDB_CONFIG.SETHTTPSPORT (port);
   SQL> COMMIT;
   ```

   where *`port`* is the TCP port number on which you want the database to listen for HTTP(S) connections.

   `port` must be an integer between 1 and 65536, and for UNIX and Linux must be greater than 1023. Choose a port number that is not already in use.

   Each pluggable database must use a unique port number so that the scheduler agent can determine the exact pluggable database later during the agent registration procedure.

   Note:

   - This enables HTTP(S) connections on all instances of an Oracle Real Application Clusters database.
   - Oracle Scheduler agent supports HTTPS connections starting with Oracle Database 18c.

6. Run the script `prvtrsch.plb` with following command:

   ```
   SQL> @?/rdbms/admin/prvtrsch.plb
   ```

7. Set a registration password for the Scheduler agents using the `SET_AGENT_REGISTRATION_PASS` procedure.

   The following example sets the agent registration password to `mypassword`.

   ```
   BEGIN
     DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS('mypassword');
   END;
   /
   ```

   Note:

   You must have the `MANAGE SCHEDULER` privilege to set an agent registration password. See *Oracle Database PL/SQL Packages and Types Reference* for more information on the `SET_AGENT_REGISTRATION_PASS` procedure.

You will do the actual registration further on, in "Registering Scheduler Agents with Databases".

29.1.3.1.2 Disabling Remote Jobs

You can disable remote jobs on a database by dropping the `REMOTE_SCHEDULER_AGENT` user.

To disable remote jobs:

- Submit the following SQL statement:

  ```
  DROP USER REMOTE_SCHEDULER_AGENT CASCADE;
  ```

Registration of new scheduler agents and execution of remote jobs is disabled until you run `prvtrsch.plb` again.

29.1.3.2 Installing and Configuring the Scheduler Agent on a Remote Host

Before you can run remote jobs on a particular host, you must install and configure the Scheduler agent.

After installing and configuring the Scheduler agent, you must register and start the Scheduler agent on the host, described in "Performing Tasks with the Scheduler Agent". The Scheduler agent must also be installed in its own Oracle home.

To install and configure the Scheduler agent on a remote host:

1. Download or retrieve the Scheduler agent software, which is available on the Oracle Database Client media included in the Database Media Pack, and online at:

   `http://www.oracle.com/technology/software/products/database`

2. Ensure that you have first properly set up any database on which you want to register the agent.

   See "Enabling and Disabling Databases for Remote Jobs" for instructions.

3. Log in to the host you want to install the Scheduler agent on. This host runs remote jobs.
   - For Windows, log in as an administrator.
   - For UNIX and Linux, log in as the user that you want the Scheduler agent to run as. This user requires no special privileges.

4. Run the Oracle Universal Installer (OUI) from the installation media for Oracle Database Client.
   - For Windows, run `setup.exe`.
   - For UNIX and Linux, use the following command:

     ```
     /directory_path/runInstaller
     ```

   where *`directory_path`* is the path to the Oracle Database Client installation media.

5. On the Select Installation Type page, select **Custom**, and then click **Next**.
6. On the Select Product Languages page, select the desired languages, and click **Next**.
7. On the Specify Install Location page, enter the path for a new Oracle home for the agent, and then click **Next**.
8. On the Available Product Components page, select **Oracle Scheduler Agent**, and click **Next**.
9. On the Oracle Database Scheduler Agent page:

   a. In the Scheduler Agent Hostname field, enter the host name of the computer that the Scheduler agent is installed on.
   b. In the Scheduler Agent Port Number field, enter the TCP port number that the Scheduler agent is to listen on for connections, or accept the default, and then click **Next**.

      Choose an integer between 1 and 65535. On UNIX and Linux, the number must be greater than 1023. Ensure that the port number is not already in use.

      OUI performs a series of prerequisite checks. If any of the prerequisite checks fail, resolve the problems, and then click **Next**.

10. On the Summary page, click **Finish**.
11. (UNIX and Linux only) When OUI prompts you to run the script `root.sh`, enter the following command as the `root` user:

    ```
    script_path/root.sh
    ```

    The script is located in the directory that you chose for agent installation.

    When the script completes, click **OK** in the Execute Configuration Scripts dialog box.
12. Click **Close** to exit OUI when installation is complete.
13. Use a text editor to review the agent configuration parameter file `schagent.conf`, which is located in the Scheduler agent home directory, and verify the port number in the `PORT=` directive.
14. Ensure that any firewall software on the remote host or any other firewall that protects that host has an exception to accommodate the Scheduler agent.

29.1.3.3 Performing Tasks with the Scheduler Agent

The Scheduler agent is a standalone program that enables you to schedule and run external and database jobs on remote hosts. You start and stop the Scheduler agent using the `schagent` utility on UNIX and Linux, and the `OracleSchedulerExecutionAgent` service on Windows.

- About the schagent Utility
  The executable utility `schagent` performs certain tasks for the agent on Windows, UNIX and Linux.
- Using the Scheduler Agent on Windows
  The Windows Scheduler agent service is automatically created and started during installation. The name of the service ends with `OracleSchedulerExecutionAgent`.
- Starting the Scheduler Agent
  Starting the Scheduler agent enables the host on which it resides to run remote jobs.
- Stopping the Scheduler Agent
  Stopping the Scheduler agent prevents the host on which it resides from running remote jobs.
- Registering Scheduler Agents with Databases
  As soon as you have finished configuring the Scheduler Agent, you can register the Agent on one or more databases that are to run remote jobs.

29.1.3.3.1 About the schagent Utility

The executable utility `schagent` performs certain tasks for the agent on Windows, UNIX and Linux.

The options for `schagent` are indicated in Table 29-1.

Use `schagent` with the appropriate syntax and options as follows:

For example:

UNIX and Linux: *AGENT_HOME*`/bin/schagent -status`

Windows: *AGENT_HOME*`/bin/schagent.exe -status`

Table 29-1 schagent options

Option	Description
`-start`	Starts the Scheduler Agent. *UNIX and Linux only*
`-stop`	Prompts the Scheduler agent to stop all the currently running jobs and then stop execution gracefully. *UNIX and Linux only*
`-abort`	Stops the Scheduler agent forcefully, that is, without stopping jobs first. From Oracle Database 11*g* Release 2 (11.2). *UNIX and Linux only*
`-status`	Returns this information about the Scheduler Agent running locally: version, uptime, total number of jobs run since the agent started, number of jobs currently running, and their descriptions.
`-registerdatabase`	Register the Scheduler agent with the base database or additional databases that are to run remote jobs on the agent's host computer.
`-unregisterdatabase`	Unregister an agent from a database.

29.1.3.3.2 Using the Scheduler Agent on Windows

The Windows Scheduler agent service is automatically created and started during installation. The name of the service ends with `OracleSchedulerExecutionAgent`.

Note:

Do not confuse this service with the `OracleJobScheduler` service, which runs on a Windows computer on which an Oracle database is installed, and manages the running of local external jobs without credentials.

29.1.3.3.3 Starting the Scheduler Agent

Starting the Scheduler agent enables the host on which it resides to run remote jobs.

To start the Scheduler agent:

- Do one of the following:

- On UNIX and Linux, run the following command:

  ```
  AGENT_HOME/bin/schagent -start
  ```

- On Windows, start the service whose name ends with `OracleSchedulerExecutionAgent`.

29.1.3.3.4 Stopping the Scheduler Agent

Stopping the Scheduler agent prevents the host on which it resides from running remote jobs.

To stop the Scheduler agent:

- Do one of the following:
  - On UNIX and Linux, run the `schagent` utility with either the `-stop` or `-abort` option as described in Table 29-1:

    ```
    AGENT_HOME/bin/schagent -stop
    ```

  - On Windows, stop the service whose name ends with `OracleSchedulerExecutionAgent`. This is equivalent to the `-abort` option.

29.1.3.3.5 Registering Scheduler Agents with Databases

As soon as you have finished configuring the Scheduler Agent, you can register the Agent on one or more databases that are to run remote jobs.

You can also log in later on and register the agent with additional databases.

1. If you have already logged out, then log in to the host that is running the Scheduler agent, as follows:
   - For Windows, log in as an administrator.
   - For UNIX and Linux, log in as the user with which you installed the Scheduler agent.
2. Use the following command for each database that you want to register the Scheduler agent on:
   - On UNIX and Linux, run this command:

     ```
     AGENT_HOME/bin/schagent -registerdatabase db_host db_http(s)_port
     ```

   - On Windows, run this command:

     ```
     AGENT_HOME/bin/schagent.exe -registerdatabase db_host db_http(s)_port
     ```

   where:

   - *`db_host`* is the host name or IP address of the host on which the database resides. In an Oracle Real Application Clusters environment, you can specify any node.
   - *`db_http(s)_port`* is the port number that the database listens on for HTTP(S) connections. You set this parameter previously in "Enabling and Disabling Databases for Remote Jobs". You can check the port number by submitting the following SQL statement to the database:

     For HTTP connections:

     ```
     SELECT DBMS_XDB_CONFIG.GETHTTPPORT() FROM DUAL;
     ```

     For HTTPS connections:

     ```
     SELECT DBMS_XDB_CONFIG.GETHTTPSPORT() FROM DUAL;
     ```

A port number of 0 means that HTTP(S) connections are disabled.

The agent prompts you to enter the agent registration password that you set in "Enabling and Disabling Databases for Remote Jobs".

Starting with Oracle Database 18c, the agent automatically determines if the port is configured to use HTTP or HTTPS connections. For HTTPS connections, the agent also prompts you to select any untrusted certificate that you want to add as a trusted one.

3. Repeat the previous steps for any additional databases to run remote jobs on the agent's host.

29.2 Monitoring and Managing the Scheduler

You can view the currently active window and the resource plan associated with it, view information about currently running jobs, monitor and manage window and job logs, and manage Scheduler security.

- Viewing the Currently Active Window and Resource Plan
  You can view the currently active window and the plan associated with it by querying the `DBA_SCHEDULER_WINDOWS` view.
- Finding Information About Currently Running Jobs
  You can check the state of a job by querying the `DBA_SCHEDULER_JOBS` view.
- Monitoring and Managing Window and Job Logs
  The Scheduler supports two kinds of logs: the job log and the window log.
- Managing Scheduler Security
  You should grant the appropriate privileges to users based on the Scheduler operations they will perform.

29.2.1 Viewing the Currently Active Window and Resource Plan

You can view the currently active window and the plan associated with it by querying the `DBA_SCHEDULER_WINDOWS` view.

For example, issue the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN FROM DBA_SCHEDULER_WINDOWS
WHERE ACTIVE='TRUE';

WINDOW_NAME                    RESOURCE_PLAN
------------------------------ --------------------------
MY_WINDOW10                    MY_RESOURCEPLAN1
```

If there is no window active, you can view the active resource plan by issuing the following statement:

```
SELECT * FROM V$RSRC_PLAN;
```

29.2.2 Finding Information About Currently Running Jobs

You can check the state of a job by querying the `DBA_SCHEDULER_JOBS` view.

For example, issue the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME = 'MY_EMP_JOB1';
```

```
JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB1                    DISABLED
```

In this case, you could enable the job using the `ENABLE` procedure. Table 29-2 shows the valid values for job state.

Table 29-2 Job States

Job State	Description
`disabled`	The job is disabled.
`scheduled`	The job is scheduled to be executed.
`running`	The job is currently running.
`completed`	The job has completed, and is not scheduled to run again.
`stopped`	The job was scheduled to run once and was stopped while it was running.
`broken`	The job is broken.
`failed`	The job was scheduled to run once and failed.
`retry scheduled`	The job has failed at least once and a retry has been scheduled to be executed.
`succeeded`	The job was scheduled to run once and completed successfully.
`chain_stalled`	The job is of type chain and has no steps running, no steps scheduled to run, and no event steps waiting on an event, and the chain `evaluation_interval` is set to `NULL`. No progress will be made in the chain unless there is manual intervention.

You can check the progress of currently running jobs by issuing the following statement:

```
SELECT * FROM ALL_SCHEDULER_RUNNING_JOBS;
```

Note that, for the column `CPU_USED` to show valid data, the initialization parameter `RESOURCE_LIMIT` must be set to `true`.

You can check the status of all jobs at all remote and local destinations by issuing the following statement:

```
SELECT * FROM DBA_SCHEDULER_JOB_DESTS;
```

You can find out information about a job that is part of a running chain by issuing the following statement:

```
SELECT * FROM ALL_SCHEDULER_RUNNING_CHAINS WHERE JOB_NAME='MY_JOB1';
```

You can check whether the job coordinator is running by searching for a process of the form `cjqNNN`.

See Also:

- "Multiple-Destination Jobs"
- *Oracle Database Reference* for details regarding the `*_SCHEDULER_RUNNING_JOBS` view
- *Oracle Database Reference* for details regarding the `*_SCHEDULER_JOBS` view

29.2.3 Monitoring and Managing Window and Job Logs

The Scheduler supports two kinds of logs: the job log and the window log.

- Job Log
  You can view information about job runs, job state changes, and job failures in the job log.
- Window Log
  The window log records operations on windows.
- Purging Logs
  To prevent job and window logs from growing indiscriminately, use the `SET_SCHEDULER_ATTRIBUTE` procedure to specify how much history (in days) to keep.

29.2.3.1 Job Log

You can view information about job runs, job state changes, and job failures in the job log.

The job log is implemented as the following two data dictionary views:

- `*_SCHEDULER_JOB_LOG`
- `*_SCHEDULER_JOB_RUN_DETAILS`

You can control the amount of logging that the Scheduler performs on jobs at both the job class and individual job level. Normally, you control logging at the class level, as this offers you more control over logging for the jobs in the class.

See "Viewing the Job Log" for definitions of the various logging levels and for information about logging level precedence between jobs and their job class. By default, the logging level of job classes is `LOGGING_RUNS`, which causes all job runs to be logged.

You can set the `logging_level` attribute when you create the job class, or you can use the `SET_ATTRIBUTE` procedure to change the logging level at a later time. The following example sets the logging level of jobs in the `myclass1` job class to `LOGGING_FAILED_RUNS`, which means that only failed runs are logged. Note that all job classes are in the `SYS` schema.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'sys.myclass1', 'logging_level', DBMS_SCHEDULER.LOGGING_FAILED_RUNS);
END;
/
```

You must be granted the `MANAGE SCHEDULER` privilege to set the logging level of a job class.

See Also:

- "Viewing the Job Log" for more detailed information about the job log and for examples of queries against the job log views
- *Oracle Database Reference* for details on the `*_SCHEDULER_JOB_LOG` view
- *Oracle Database Reference* for details on the `*_SCHEDULER_JOB_RUN_DETAILS` view
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_JOB_CLASS` and `SET_ATTRIBUTE` procedures
- "Setting Scheduler Preferences" for information about setting retention for log entries

29.2.3.2 Window Log

The window log records operations on windows.

The Scheduler makes an entry in the window log each time that:

- You create or drop a window
- A window opens
- A window closes
- Windows overlap
- You enable or disable a window

There are no logging levels for window activity logging.

To see the contents of the window log, query the `DBA_SCHEDULER_WINDOW_LOG` view. The following statement shows sample output from this view:

```
SELECT log_id, to_char(log_date, 'DD-MON-YY HH24:MI:SS') timestamp,
   window_name, operation FROM DBA_SCHEDULER_WINDOW_LOG;

    LOG_ID TIMESTAMP            WINDOW_NAME       OPERATION
---------- -------------------- ----------------- --------
         4 10/01/2004 15:29:23  WEEKEND_WINDOW    CREATE
         5 10/01/2004 15:33:01  WEEKEND_WINDOW    UPDATE
        22 10/06/2004 22:02:48  WEEKNIGHT_WINDOW  OPEN
        25 10/07/2004 06:59:37  WEEKNIGHT_WINDOW  CLOSE
        26 10/07/2004 22:01:37  WEEKNIGHT_WINDOW  OPEN
        29 10/08/2004 06:59:51  WEEKNIGHT_WINDOW  CLOSE
```

The `DBA_SCHEDULER_WINDOWS_DETAILS` view provides information about every window that was active and is now closed (completed). The following statement shows sample output from that view:

```
SELECT LOG_ID, WINDOW_NAME, ACTUAL_START_DATE, ACTUAL_DURATION
  FROM DBA_SCHEDULER_WINDOW_DETAILS;

    LOG_ID WINDOW_NAME      ACTUAL_START_DATE                    ACTUAL_DURATION
---------- ---------------- ------------------------------------ ---------------
        25 WEEKNIGHT_WINDOW 06-OCT-04 10:02.48.832438 PM PST8PDT +000 01:02:32
        29 WEEKNIGHT_WINDOW 07-OCT-04 10.01.37.025704 PM PST8PDT +000 03:02:00
```

Notice that log IDs correspond in both of these views, and that in this case the rows in the `DBA_SCHEDULER_WINDOWS_DETAILS` view correspond to the `CLOSE` operations in the `DBA_SCHEDULER_WINDOW_LOG` view.

> **See Also:**
>
> - *Oracle Database Reference* for details on the `*_SCHEDULER_WINDOW_LOG` view
> - *Oracle Database Reference* for details on the `DBA_SCHEDULER_WINDOWS_DETAILS` view

29.2.3.3 Purging Logs

To prevent job and window logs from growing indiscriminately, use the `SET_SCHEDULER_ATTRIBUTE` procedure to specify how much history (in days) to keep.

Once per day, the Scheduler automatically purges all log entries that are older than the specified history period from both the job log and the window log. The default history period is 30 days. For example, to change the history period to 90 days, issue the following statement:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
```

Some job classes are more important than others. Because of this, you can override this global history setting by using a class-specific setting. For example, suppose that there are three job classes (`class1`, `class2`, and `class3`), and that you want to keep 10 days of history for the window log, `class1`, and `class3`, but 30 days for `class2`. To achieve this, issue the following statements:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','10');
DBMS_SCHEDULER.SET_ATTRIBUTE('class2','log_history','30');
```

You can also set the class-specific history when creating the job class.

Note that log entries pertaining to steps of a chain run are not purged until the entries for the main chain job are purged.

Purging Logs Manually

The `PURGE_LOG` procedure enables you to manually purge logs. As an example, the following statement purges all entries from both the job and window logs:

```
DBMS_SCHEDULER.PURGE_LOG();
```

Another example is the following, which purges all entries from the jog log that are older than three days. The window log is not affected by this statement.

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 3, which_log => 'JOB_LOG');
```

The following statement purges all window log entries older than 10 days and all job log entries older than 10 days that relate to `job1` and to the jobs in `class2`:

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 10, job_name => 'job1, sys.class2');
```

29.2.4 Managing Scheduler Security

You should grant the appropriate privileges to users based on the Scheduler operations they will perform.

You should grant the `CREATE JOB` system privilege to regular users who need to be able to use the Scheduler to schedule and run jobs. You should grant `MANAGE SCHEDULER` to any database administrator who needs to manage system resources. Grant any other Scheduler system privilege or role with great caution. In particular, the `CREATE ANY JOB` system privilege and the `SCHEDULER_ADMIN` role, which includes it, are very powerful because they allow execution of code as any user. They should only be granted to very powerful roles or users.

Handling external job is a particularly important issue from a security point of view. Only users that need to run jobs outside of the database should be granted the `CREATE EXTERNAL JOB` system privilege that allows them to do so. Security for the Scheduler has no other special requirements. See *Oracle Database Security Guide* for details regarding security.

If users need to create credentials to authenticate their jobs to the operating system or a remote database, grant them `CREATE CREDENTIAL` system privilege.

> **Note:**
>
> When upgrading from Oracle Database 10*g* Release 1 (10.1) to Oracle Database 10*g* Release 2 (10.2) or later, `CREATE EXTERNAL JOB` is automatically granted to all users and roles that have the `CREATE JOB` privilege. Oracle recommends that you revoke this privilege from users that do not need it.

29.3 Import/Export and the Scheduler

You must use the Data Pump utilities (`impdp` and `expdp`) to export Scheduler objects.

You cannot use the earlier import/export utilities (`IMP` and `EXP`) with the Scheduler. Also, Scheduler objects cannot be exported while the database is in read-only mode.

An export generates the DDL that was used to create the Scheduler objects. All attributes are exported. When an import is done, all the database objects are re-created in the new database. All schedules are stored with their time zones, which are maintained in the new database. For example, schedule "Monday at 1 PM PST in a database in San Francisco" would be the same if it was exported and imported to a database in Germany.

Although Scheduler credentials are exported, for security reasons, the passwords in these credentials are not exported. After you import Scheduler credentials, you must reset the passwords using the `SET_ATTRIBUTE` procedure of the `DBMS_SCHEDULER` package.

> **See Also:**
>
> *Oracle Database Utilities* for details on Data Pump

29.4 Troubleshooting the Scheduler

You can troubleshoot problems with Scheduler.

- A Job Does Not Run
  A job may fail to run for several reasons.
- A Program Becomes Disabled
  A program can become disabled if a program argument is dropped or `number_of_arguments` is changed so that all arguments are no longer defined.
- A Window Fails to Take Effect
  A window can fail to take effect for various reasons.

29.4.1 A Job Does Not Run

A job may fail to run for several reasons.

To begin troubleshooting a job that you suspect did not run, check the job state by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS;
```

Typical output will resemble the following:

```
JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB                     DISABLED
MY_EMP_JOB1                    FAILED
MY_NEW_JOB1                    DISABLED
MY_NEW_JOB2                    BROKEN
MY_NEW_JOB3                    COMPLETED
```

- About Job States
  If a job does not run, then it can be in one of the following states: failed, broken, disabled, or completed.
- Viewing the Job Log
  The job log is an important troubleshooting tool.
- Troubleshooting Remote Jobs
  Remote jobs must successfully communicate with a Scheduler agent on the remote host. If a remote job does not run, then check the `DBA_SCHEDULER_JOBS` view and the job log first.
- About Job Recovery After a Failure
  The Scheduler can attempt to recover jobs that are interrupted.

29.4.1.1 About Job States

If a job does not run, then it can be in one of the following states: failed, broken, disabled, or completed.

- Failed Jobs
  If a job has the status of `FAILED` in the job table, then it was scheduled to run once but the execution has failed. If the job was specified as restartable, then all retries have failed.
- Broken Jobs
  A broken job is one that has exceeded a certain number of failures. This number is set in `max_failures`, and can be altered.

- Disabled Jobs
  A job can become disabled for several reasons.
- Completed Jobs
  A job will be completed if `end_date` or `max_runs` is reached.

29.4.1.1.1 Failed Jobs

If a job has the status of `FAILED` in the job table, then it was scheduled to run once but the execution has failed. If the job was specified as restartable, then all retries have failed.

If a job fails in the middle of execution, only the last transaction of that job is rolled back. If your job executes multiple transactions, then you must be careful about setting `restartable` to `TRUE`. You can query failed jobs by querying the `*_SCHEDULER_JOB_RUN_DETAILS` views.

29.4.1.1.2 Broken Jobs

A broken job is one that has exceeded a certain number of failures. This number is set in `max_failures`, and can be altered.

In the case of a broken job, the entire job is broken, and it will not be run until it has been fixed. For debugging and testing, you can use the `RUN_JOB` procedure.

You can query broken jobs by querying the `*_SCHEDULER_JOBS` and `*_SCHEDULER_JOB_LOG` views.

29.4.1.1.3 Disabled Jobs

A job can become disabled for several reasons.

The reasons include the following:

- The job was manually disabled
- The job class it belongs to was dropped
- The program, chain, or schedule that it points to was dropped
- A window or window group is its schedule and the window or window group is dropped

29.4.1.1.4 Completed Jobs

A job will be completed if `end_date` or `max_runs` is reached.

If a job recently completed successfully but is scheduled to run again, then the job state is `SCHEDULED`.

29.4.1.2 Viewing the Job Log

The job log is an important troubleshooting tool.

For details and instructions, see "Viewing the Job Log".

29.4.1.3 Troubleshooting Remote Jobs

Remote jobs must successfully communicate with a Scheduler agent on the remote host. If a remote job does not run, then check the `DBA_SCHEDULER_JOBS` view and the job log first.

Then perform the following tasks:

1. Check that the remote system is reachable over the network with tools such as `nslookup` and `ping`.
2. Check the status of the Scheduler agent on the remote host by calling the `GET_AGENT_VERSION` package procedure.

```
DECLARE
  versionnum VARCHAR2(30);
BEGIN
  versionnum := DBMS_SCHEDULER.GET_AGENT_VERSION('remote_host.example.com');
  DBMS_OUTPUT.PUT_LINE(versionnum);
END;
/
```

   If an error is generated, the agent may not be installed or may not be registered with your local database. See "Using the Oracle Scheduler Agent to Run Remote Jobs" for instructions for installing, registering, and starting the Scheduler agent.

29.4.1.4 About Job Recovery After a Failure

The Scheduler can attempt to recover jobs that are interrupted.

The Scheduler attempts to recover jobs that are interrupted when:

- The database abnormally shuts down
- A job slave process is terminated or otherwise fails
- For an external job, the external job process that starts the executable or script is terminated or otherwise fails. (The external job process is `extjob` on UNIX. On Windows, it is the external job service.)
- For an external job, the process that runs the end-user executable or script is terminated or otherwise fails.

Job recovery proceeds as follows:

- The Scheduler adds an entry to the job log for the instance of the job that was running when the failure occurred. In the log entry, the `OPERATION` is '`RUN`', the `STATUS` is '`STOPPED`', and `ADDITIONAL_INFO` contains one of the following:
  - `REASON="Job slave process was terminated"`
  - `REASON="ORA-01014: ORACLE shutdown in progress"`
- If `restartable` is set to `TRUE` for the job, the job is restarted.
- If `restartable` is set to `FALSE` for the job:
  - If the job is a run-once job and `auto_drop` is set to `TRUE`, the job run is done and the job is dropped.
  - If the job is a run-once job and `auto_drop` is set to `FALSE`, the job is disabled and the job `state` is set to '`STOPPED`'.
  - If the job is a repeating job, the Scheduler schedules the next job run and the job `state` is set to '`SCHEDULED`'.

When a job is restarted as a result of this recovery process, the new run is entered into the job log with the operation '`RECOVERY_RUN`'.

29.4.2 A Program Becomes Disabled

A program can become disabled if a program argument is dropped or `number_of_arguments` is changed so that all arguments are no longer defined.

See "Creating and Managing Programs to Define Jobs" for more information regarding programs.

29.4.3 A Window Fails to Take Effect

A window can fail to take effect for various reasons.

A window can fail to take effect for the following reasons:

- A window becomes disabled when it is at the end of its schedule
- A window that points to a schedule that no longer exists is disabled

See "Managing Job Scheduling and Job Priorities with Windows" for more information regarding windows.

29.5 Examples of Using the Scheduler

Examples illustrate using Scheduler.

- Examples of Creating Job Classes
  Examples illustrate creating job classes.
- Examples of Setting Attributes
  Examples illustrate setting attributes.
- Examples of Creating Chains
  Examples illustrate creating chains.
- Examples of Creating Jobs and Schedules Based on Events
  Examples illustrate creating event-based jobs and event schedules.
- Example of Creating a Job In an Oracle Data Guard Environment
  In an Oracle Data Guard environment, the Scheduler includes additional support for two database roles: primary and logical standby. You can configure a job to run only when the database is in the primary role or only when the database is in the logical standby role.

29.5.1 Examples of Creating Job Classes

Examples illustrate creating job classes.

To create a job class, you use the `CREATE_JOB_CLASS` procedure.

Example 29-1   Creating a Job Class

The following statement creates a job class:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name              =>  'my_class1',
   service                     =>  'my_service1',
   comments                    =>  'This is my first job class');
END;
/
```

This creates `my_class1` in `SYS`. It uses a service called `my_service1`. To verify that the job class was created, issue the following statement:

```
SELECT JOB_CLASS_NAME FROM DBA_SCHEDULER_JOB_CLASSES
WHERE JOB_CLASS_NAME = 'MY_CLASS1';

JOB_CLASS_NAME
------------------------------
MY_CLASS1
```

Example 29-2   Creating a Job Class

The following statement creates a job class:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group',
   service                    =>  'accounting',
   comments                   =>  'All finance jobs');
END;
/
```

This creates `finance_jobs` in `SYS`. It assigns a resource consumer group called `finance_group`, and designates service affinity for the `accounting` service. Note that if the `accounting` service is mapped to a resource consumer group other than `finance_group`, jobs in this class run under the `finance_group` consumer group, because the `resource_consumer_group` attribute takes precedence.

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `CREATE_JOB_CLASS` procedure and "Creating Job Classes" for further information

29.5.2 Examples of Setting Attributes

Examples illustrate setting attributes.

To set attributes, you use `SET_ATTRIBUTE` and `SET_SCHEDULER_ATTRIBUTE` procedures.

Example 29-3   Setting the Repeat Interval Attribute

The following example resets the frequency that `my_emp_job1` runs daily:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'repeat_interval',
   value          =>   'FREQ=DAILY');
END;
/
```

To verify the change, issue the following statement:

```
SELECT JOB_NAME, REPEAT_INTERVAL FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME =  'MY_EMP_JOB1';
```

```
JOB_NAME                REPEAT_INTERVAL
----------------        ---------------
MY_EMP_JOB1             FREQ=DAILY
```

Example 29-4   Setting Multiple Job Attributes for a Set of Jobs

The following example sets four different attributes for each of five jobs:

```
DECLARE
 newattr sys.jobattr;
 newattrarr sys.jobattr_array;
 j number;
BEGIN
 -- Create new JOBATTR array
 newattrarr := sys.jobattr_array();

 -- Allocate enough space in the array
 newattrarr.extend(20);
 j := 1;
 FOR i IN 1..5 LOOP
   -- Create and initialize a JOBATTR object type
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'MAX_FAILURES',
                          attr_value => 5);
   -- Add it to the array.
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'COMMENTS',
                          attr_value => 'Test job');
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'END_DATE',
                          attr_value => systimestamp + interval '24' hour);
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'SCHEDULE_LIMIT',
                          attr_value => interval '1' hour);
   newattrarr(j) := newattr;
   j := j + 1;
 END LOOP;

 -- Call SET_JOB_ATTRIBUTES to set all 20 set attributes in one transaction
 DBMS_SCHEDULER.SET_JOB_ATTRIBUTES(newattrarr, 'TRANSACTIONAL');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `SET_SCHEDULER_ATTRIBUTE` procedure and "Setting Scheduler Preferences"

29.5.3 Examples of Creating Chains

Examples illustrate creating chains.

To create chains, you use the `CREATE_CHAIN` procedure. After creating a chain, you add steps to the chain with the `DEFINE_CHAIN_STEP` or `DEFINE_CHAIN_EVENT_STEP` procedures and define the rules with the `DEFINE_CHAIN_RULE` procedure.

Example 29-5 Creating a Chain

The following example creates a chain where `my_program1` runs before `my_program2` and `my_program3`. `my_program2` and `my_program3` run in parallel after `my_program1` has completed.

The user for this example must have the `CREATE EVALUATION CONTEXT`, `CREATE RULE`, and `CREATE RULE SET` privileges. See "Setting Chain Privileges" for more information.

```
BEGIN
  DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name            =>  'my_chain1',
   rule_set_name         =>  NULL,
   evaluation_interval   =>  NULL,
   comments              =>  NULL);
END;
/

--- define three steps for this chain. Referenced programs must be enabled.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepA', 'my_program1');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepB', 'my_program2');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepC', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE('my_chain1', 'TRUE', 'START stepA');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepA COMPLETED', 'Start stepB, stepC');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepB COMPLETED AND stepC COMPLETED', 'END');
END;
/

--- enable the chain
BEGIN
 DBMS_SCHEDULER.ENABLE('my_chain1');
END;
/

--- create a chain job to start the chain daily at 1:00 p.m.
BEGIN
 DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

Example 29-6   Creating a Chain

The following example creates a chain where first `my_program1` runs. If it succeeds, `my_program2` runs; otherwise, `my_program3` runs.

```
BEGIN
 DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name             => 'my_chain2',
   rule_set_name          => NULL,
   evaluation_interval    => NULL,
   comments               => NULL);
END;
/

--- define three steps for this chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step1', 'my_program1');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step2', 'my_program2');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step3', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain2', 'TRUE', 'START step1');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 SUCCEEDED', 'Start step2');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 COMPLETED AND step1 NOT SUCCEEDED', 'Start step3');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step2 COMPLETED OR step3 COMPLETED', 'END');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `CREATE_CHAIN`, `DEFINE_CHAIN_STEP`, and `DEFINE_CHAIN_RULE` procedures and "Setting Scheduler Preferences"

29.5.4 Examples of Creating Jobs and Schedules Based on Events

Examples illustrate creating event-based jobs and event schedules.

To create event-based jobs, you use the `CREATE_JOB` procedure. To create event-based schedules, you use the `CREATE_EVENT_SCHEDULE` procedure.

These examples assume the existence of an application that, when it detects the arrival of a file on a system, enqueues an event onto the queue `my_events_q`.

Example 29-7   Creating an Event-Based Schedule

The following example illustrates creating a schedule that can be used to start a job whenever the Scheduler receives an event indicating that a file arrived on the system before 9AM:

```
BEGIN
  DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'scott.file_arrival',
   start_date        =>  systimestamp,
   event_condition   =>  'tab.user_data.object_owner = ''SCOTT''
      and tab.user_data.event_name = ''FILE_ARRIVAL''
      and extract hour from tab.user_data.event_timestamp < 9',
   queue_spec        =>  'my_events_q');
END;
/
```

Example 29-8 Creating an Event-Based Job

The following example creates a job that starts when the Scheduler receives an event indicating that a file arrived on the system:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  my_job,
   program_name        =>  my_program,
   start_date          =>  '15-JUL-04 1.00.00AM US/Pacific',
   event_condition     =>  'tab.user_data.event_name = ''LOW_INVENTORY''',
   queue_spec          =>  'my_events_q'
   enabled             =>  TRUE,
   comments            =>  'my event-based job');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `CREATE_JOB` and `CREATE_EVENT_SCHEDULE` procedures

29.5.5 Example of Creating a Job In an Oracle Data Guard Environment

In an Oracle Data Guard environment, the Scheduler includes additional support for two database roles: primary and logical standby. You can configure a job to run only when the database is in the primary role or only when the database is in the logical standby role.

To do so, you set the `database_role` attribute. This example explains how to enable a job to run in both database roles. The method used is to create two copies of the job and assign a different `database_role` attribute to each.

By default, a job runs when the database is in the role that it was in when the job was created. You can run the same job in both roles using the following steps:

1. Copy the job
2. Enable the new job
3. Change the `database_role` attribute of the new job to the required role

The example starts by creating a job called `primary_job` on the primary database. It then makes a copy of this job and sets its `database_role` attribute to '`LOGICAL STANDBY`'. If the primary database then becomes a logical standby, the job continues to run according to its schedule.

When you copy a job, the new job is disabled, so you must enable the new job.

```
BEGIN DBMS_SCHEDULER.CREATE_JOB (
     job_name       => 'primary_job',
     program_name   => 'my_prog',
     schedule_name  => 'my_sched');

 DBMS_SCHEDULER.COPY_JOB('primary_job','standby_job');
 DBMS_SCHEDULER.ENABLE(name=>'standby_job', commit_semantics=>'ABSORB_ERRORS');
 DBMS_SCHEDULER.SET_ATTRIBUTE('standby_job','database_role','LOGICAL STANDBY');
END;
/
```

After you execute this example, the data in the `DBA_SCHEDULER_JOB_ROLES` view is as follows:

```
SELECT JOB_NAME, DATABASE_ROLE FROM DBA_SCHEDULER_JOB_ROLES
   WHERE JOB_NAME IN ('PRIMARY_JOB','STANDBY_JOB');

JOB_NAME                DATABASE_ROLE
--------                ----------------
PRIMARY_JOB             PRIMARY
STABDBY_JOB             LOGICAL STANDBY
```

Note:

For a physical standby database, any changes made to Scheduler objects or any database changes made by Scheduler jobs on the primary database are applied to the physical standby like any other database changes.

29.6 Scheduler Reference

There are several privileges and data dictionary views related to Scheduler.

- Scheduler Privileges
  Users can be granted various Scheduler privileges.
- Scheduler Data Dictionary Views
  You can query a set of views for information about Scheduler.

29.6.1 Scheduler Privileges

Users can be granted various Scheduler privileges.

Table 29-3 and Table 29-4 describe the various Scheduler privileges.

Table 29-3 Scheduler System Privileges

Privilege Name	Operations Authorized
`CREATE JOB`	This privilege enables you to create jobs, chains, schedules, programs, file watchers, destinations, and groups in your own schema. You can always alter and drop these objects in your own schema, even if you do not have the `CREATE JOB` privilege. In this case, the object would have been created in your schema by another user with the `CREATE ANY JOB` privilege.

Table 29-3 (Cont.) Scheduler System Privileges

Privilege Name	Operations Authorized
`CREATE ANY JOB`	This privilege enables you to create, alter, and drop jobs, chains, schedules, programs, file watchers, destinations, and groups in any schema except `SYS`. This privilege is extremely powerful and should be used with care because it allows the grantee to execute any PL/SQL code as any other database user.
`CREATE EXTERNAL JOB`	This privilege is required to create jobs that run outside of the database. Owners of jobs of type '`EXECUTABLE`' or jobs that point to programs of type '`EXECUTABLE`' require this privilege. To run a job of type '`EXECUTABLE`', you must have this privilege and the `CREATE JOB` privilege. This privilege is also required to retrieve files from a remote host and to save files to one or more remote hosts.
`EXECUTE ANY PROGRAM`	This privilege enables your jobs to use programs or chains from any schema.
`EXECUTE ANY CLASS`	This privilege enables your jobs to run under any job class.
`MANAGE SCHEDULER`	This is the most important privilege for administering the Scheduler. It enables you to create, alter, and drop job classes, windows, and window groups, and to stop jobs with the `force` option. It also enables you to set and retrieve Scheduler attributes, purge Scheduler logs, and set the agent password for a database.

Table 29-4 Scheduler Object Privileges

Privilege Name	Operations Authorized
`SELECT`	You can grant object privileges on a group to other users by granting `SELECT` on the group.
`EXECUTE`	You can grant this privilege only on programs, chains, file watchers, credentials, and job classes. The `EXECUTE` privilege enables you to reference the object in a job. It also enables you to view the object if the object is was not created in your schema.
`ALTER`	This privilege enables you to alter or drop the object it is granted on. Altering includes such operations as enabling, disabling, defining or dropping program arguments, setting or resetting job argument values and running a job. Certain restricted attributes of jobs of job type `EXECUTABLE` cannot be altered using the `ALTER` object privilege. These include `job_type`, `job_action`, `number_of_arguments`, `event_spec`, and setting PL/SQL date functions as schedules. For programs, jobs, chains, file watchers, and credentials, this privilege also enables schemas that do not own these objects to view them. This privilege can be granted on jobs, chains, programs, schedules, file watchers, and credentials. For other types of Scheduler objects, you must grant the `MANAGE SCHEDULER` system privilege.
`ALL`	This privilege authorizes operations allowed by all other object privileges possible for a given object. It can be granted on jobs, programs, chains, schedules, file watchers, credentials, and job classes.

Note:

No object privileges are required to use a destination object created by another user.

The `SCHEDULER_ADMIN` role is created with all of the system privileges shown in Table 29-3 (with the `ADMIN` option). The `SCHEDULER_ADMIN` role is granted to `DBA` (with the `ADMIN` option).

When calling DBMS_SCHEDULER procedures and functions from a definer's rights PL/SQL block, object privileges must be granted directly to the calling user. As with all PL/SQL stored procedures, DBMS_SCHEDULER ignores privileges granted through roles on database objects when called from a definer's rights PL/SQL block.

The following object privileges are granted to PUBLIC: SELECT ALL_SCHEDULER_* views, SELECT USER_SCHEDULER_* views, SELECT SYS.SCHEDULER$_JOBSUFFIX_S (for generating a job name), and EXECUTE SYS.DEFAULT_JOB_CLASS.

29.6.2 Scheduler Data Dictionary Views

You can query a set of views for information about Scheduler.

Some views are specific to multitenant container databases (CDBs), whereas others have a CDB-specific column. The V$ and GV$ views have a CON_ID column that identifies a container whose data is represented by a CDB_* row. CDB_* views correspond to all Scheduler DBA_* views. In a PDB, these views only show objects visible through a corresponding DBA_* view, but all objects are visible in the root. The CDB_* view contains all columns found in a given DBA_* view and the column (CON_ID).

Table 29-5 contains views associated with the Scheduler. The *_SCHEDULER_JOBS, *_SCHEDULER_SCHEDULES, *_SCHEDULER_PROGRAMS, *_SCHEDULER_RUNNING_JOBS, *_SCHEDULER_JOB_LOG, *_SCHEDULER_JOB_RUN_DETAILS views are particularly useful for managing jobs. See *Oracle Database Reference* for details regarding Scheduler views.

Note:

In the following table, the asterisk at the beginning of a view name can be replaced with DBA, ALL, or USER.

Example 29-9 Displaying Details About a Scheduler Job

This example shows information for completed instances of my_job1:

```
SELECT JOB_NAME, STATUS, ERROR#
FROM DBA_SCHEDULER_JOB_RUN_DETAILS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME     STATUS          ERROR#
--------     --------------  ------
MY_JOB1      FAILURE          20000
```

Table 29-5 Scheduler Views

View	Description
*_SCHEDULER_CHAIN_RULES	These views show all rules for all chains.
*_SCHEDULER_CHAIN_STEPS	These views show all steps for all chains.
*_SCHEDULER_CHAINS	These views show all chains.
*_SCHEDULER_CREDENTIALS *_CREDENTIALS	These views show all credentials. ** *_SCHEDULER_CREDENTIALS is deprecated in Oracle Database 12*c*, but remains available, for reasons of backward compatibility. The recommended view is *_CREDENTIALS.
*_SCHEDULER_DB_DESTS	These views show all database destinations.

Table 29-5 (Cont.) Scheduler Views

View	Description
`*_SCHEDULER_DESTS`	These views show all destinations, both database and external.
`*_SCHEDULER_EXTERNAL_DESTS`	These views show all external destinations.
`*_SCHEDULER_FILE_WATCHERS`	These views show all file watchers.
`*_SCHEDULER_GLOBAL_ATTRIBUTE`	These views show the current values of Scheduler attributes.
`*_SCHEDULER_GROUP_MEMBERS`	These views show all group members in all groups.
`*_SCHEDULER_GROUPS`	These views show all groups.
`*_SCHEDULER_INCOMPATIBILITY`	These views show all programs or jobs that are members of incompatibility definitions.
`*_SCHEDULER_JOB_ARGS`	These views show all set argument values for all jobs.
`*_SCHEDULER_JOB_CLASSES`	These views show all job classes.
`*_SCHEDULER_JOB_DESTS`	These views show the state of both local jobs and jobs at remote destinations, including child jobs of multiple-destination jobs. You obtain job destination IDs (`job_dest_id`) from these views.
`*_SCHEDULER_JOB_LOG`	These views show job runs and state changes, depending on the logging level set.
`*_SCHEDULER_JOB_ROLES`	These views show all jobs by Oracle Data Guard database role.
`*_SCHEDULER_JOB_RUN_DETAILS`	These views show all completed (failed or successful) job runs.
`*_SCHEDULER_JOBS`	These views show all jobs, enabled as well as disabled.
`*_SCHEDULER_NOTIFICATIONS`	These views show all job state e-mail notifications.
`*_SCHEDULER_PROGRAM_ARGS`	These views show all arguments defined for all programs as well as the default values if they exist.
`*_SCHEDULER_PROGRAMS`	These views show all programs.
`*_SCHEDULER_REMOTE_DATABASES`	These views show information about the remote databases accessible to the current user that have been registered as sources and destinations for remote database jobs.
`*_SCHEDULER_REMOTE_JOBSTATE`	These views displays information about the state of the jobs accessible to the current user at remote databases.
`*_SCHEDULER_RESOURCES`	These views describe the resource metadata.
`*_SCHEDULER_RUNNING_CHAINS`	These views show all chains that are running.
`*_SCHEDULER_RUNNING_JOBS`	These views show state information on all jobs that are currently being run.
`*_SCHEDULER_RSRC_CONSTRAINTS`	These views show the types of resources used by a job or program and the number of units of each resource it needs.
`*_SCHEDULER_SCHEDULES`	These views show all schedules.
`*_SCHEDULER_WINDOW_DETAILS`	These views show all completed window runs.
`*_SCHEDULER_WINDOW_GROUPS`	These views show all window groups.
`*_SCHEDULER_WINDOW_LOG`	These views show all state changes made to windows.
`*_SCHEDULER_WINDOWS`	These views show all windows.
`*_SCHEDULER_WINGROUP_MEMBERS`	These views show the members of all window groups, one row for each group member.

Part V
Distributed Database Management

You can manage a distributed database environment.

- Distributed Database Concepts
  Concepts related to distributed databases include distributed database architecture, database links, transaction processing, application development, and character set support.
- Managing a Distributed Database
  Managing a distributed database includes tasks such as managing global names, managing database links, and creating location and statement transparency.
- Developing Applications for a Distributed Database System
  Developing applications for a distributed database system includes tasks such as managing the distribution of application data, controlling connections established by database links, maintaining referential integrity, tuning distributed queries, and handling errors in remote procedures.
- Distributed Transactions Concepts
  Distributed transactions update data on two or more distinct nodes of a distributed database.
- Managing Distributed Transactions
  Managing distributed transactions includes tasks such as specifying the comment point strength of a node, naming transactions, and managing in-doubt transactions.

30 Distributed Database Concepts

Concepts related to distributed databases include distributed database architecture, database links, transaction processing, application development, and character set support.

- Distributed Database Architecture
  A **distributed database system** allows applications to access data from local and remote databases. In a **homogenous distributed database system**, each database is an Oracle Database. In a **heterogeneous distributed database system**, at least one of the databases is not an Oracle Database. Distributed databases use a **client/server** architecture to process information requests.
- Database Links
  The central concept in distributed database systems is a **database link**. A database link is a connection between two physical database servers that allows a client to access them as one logical database.
- Distributed Database Administration
  Distributed database administration includes topics related to site autonomy, security, auditing database links, and administration tools.
- Transaction Processing in a Distributed System
  A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user. A **remote transaction** contains only statements that access a single remote node. A **distributed transaction** contains statements that access multiple nodes.
- Distributed Database Application Development
  Application development in a distributed system raises issues that are not applicable in a non-distributed system.
- Character Set Support for Distributed Environments
  Different databases and clients can use different character sets in a distributed environment.

30.1 Distributed Database Architecture

A **distributed database system** allows applications to access data from local and remote databases. In a **homogenous distributed database system**, each database is an Oracle Database. In a **heterogeneous distributed database system**, at least one of the databases is not an Oracle Database. Distributed databases use a **client/server** architecture to process information requests.

- Homogenous Distributed Database Systems
  A homogenous distributed database system includes only Oracle databases.
- Heterogeneous Distributed Database Systems
  A heterogeneous distributed database system includes both Oracle databases and non-Oracle databases.
- Client/Server Database Architecture
  A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node

that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

30.1.1 Homogenous Distributed Database Systems

A homogenous distributed database system includes only Oracle databases.

- About Homogenous Distributed Database Systems
  A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more systems.
- Distributed Databases Versus Distributed Processing
  The terms **distributed database** and **distributed processing** are closely related, yet have distinct meanings.
- Distributed Databases Versus Replicated Databases
  The terms distributed database system and **database replication** are related, yet distinct.

30.1.1.1 About Homogenous Distributed Database Systems

A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more systems.

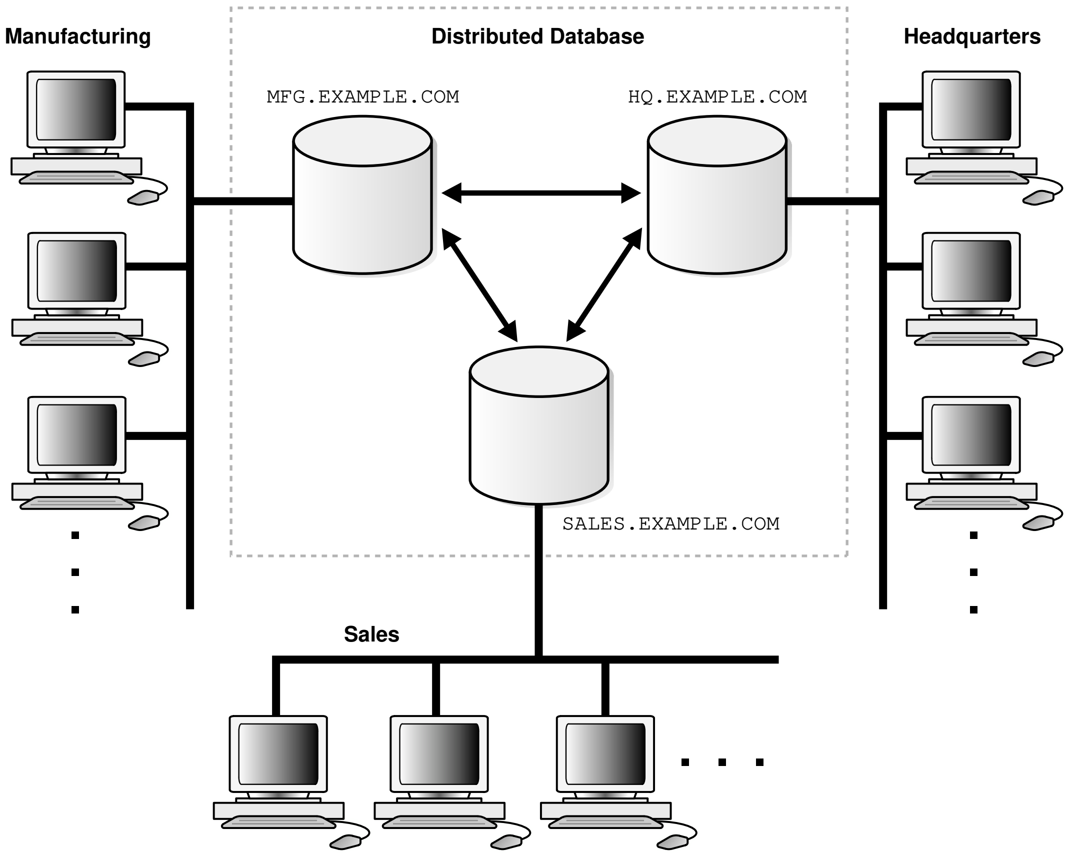
Figure 30-1 illustrates a distributed system that connects three databases: `hq`, `mfg`, and `sales`. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database `mfg` can retrieve joined data from the `products` table on the local database and the `dept` table on the remote `hq` database.

For a client application, the location and platform of the databases are transparent. You can also create **synonyms** for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database `mfg` but want to access data on database `hq`, creating a synonym on `mfg` for the remote `dept` table enables you to issue this query:

```
SELECT * FROM dept;
```

In this way, a distributed system gives the appearance of native data access. Users on `mfg` do not have to know that the data they access resides on remote databases.

Figure 30-1 Homogeneous Distributed Database



An Oracle Database distributed database system can incorporate Oracle Databases of different releases. All supported releases of Oracle Database can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system. A distributed database application cannot expect an Oracle7 database to understand the SQL extensions that are only available with Oracle Database.

30.1.1.2 Distributed Databases Versus Distributed Processing

The terms **distributed database** and **distributed processing** are closely related, yet have distinct meanings.

There definitions are as follows:

- Distributed database

  A set of databases in a distributed system that can appear to applications as a single data source.

- Distributed processing

  The operations that occurs when an application distributes its tasks among different computers in a network. For example, a database application typically distributes front-end presentation tasks to client computers and allows a back-end database server to manage shared access to a database. Consequently, a distributed database application processing system is more commonly referred to as a client/server database application system.

Distributed database systems employ a distributed processing architecture. For example, an Oracle Database server acts as a client when it requests data that another Oracle Database server manages.

30.1.1.3 Distributed Databases Versus Replicated Databases

The terms distributed database system and **database replication** are related, yet distinct.

In a **pure** (that is, not replicated) distributed database, the system manages a single copy of all data and supporting database objects. Typically, distributed database applications use distributed transactions to access both local and remote data and modify the global database in real-time.

> **Note:**
>
> This book discusses only pure distributed databases.

The term **replication** refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment.

Most commonly, replication is used to improve local database performance and protect the availability of applications because alternate data access options exist. For example, an application may normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible.

30.1.2 Heterogeneous Distributed Database Systems

A heterogeneous distributed database system includes both Oracle databases and non-Oracle databases.

- About Heterogeneous Distributed Database Systems
  In a heterogeneous distributed database system, at least one of the databases is a non-Oracle Database system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle Database. The local Oracle Database server hides the distribution and heterogeneity of the data.
- Heterogeneous Services
  Heterogeneous Services (HS) is an integrated component within the Oracle Database server and the enabling technology for the current suite of Oracle Transparent Gateway products.
- Transparent Gateway Agents
  For each non-Oracle Database system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle Database system. The agent is specific to the non-Oracle Database system, so each type of system requires a different agent.
- Generic Connectivity
  Generic connectivity enables you to connect to non-Oracle Database data stores by using either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent.

30.1.2.1 About Heterogeneous Distributed Database Systems

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle Database system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle Database. The local Oracle Database server hides the distribution and heterogeneity of the data.

The Oracle Database server accesses the non-Oracle Database system using Oracle Heterogeneous Services with an **agent**. If you access the non-Oracle Database data store using an Oracle Transparent Gateway, then the agent is a system-specific application. For example, if you include a Sybase database in an Oracle Database distributed system, then you must obtain a Sybase-specific transparent gateway so that the Oracle Database in the system can communicate with it.

Alternatively, you can use **generic connectivity** to access non-Oracle Database data stores so long as the non-Oracle Database system supports the ODBC or OLE DB protocols.

Note:

Other than the introductory material presented in this chapter, this book does not discuss Oracle Heterogeneous Services. See *Oracle Database Heterogeneous Connectivity User's Guide* for more detailed information about Heterogeneous Services.

30.1.2.2 Heterogeneous Services

Heterogeneous Services (HS) is an integrated component within the Oracle Database server and the enabling technology for the current suite of Oracle Transparent Gateway products.

HS provides the common architecture and administration mechanisms for Oracle Database gateway products and other heterogeneous access facilities. Also, it provides upwardly compatible functionality for users of most of the earlier Oracle Transparent Gateway releases.

30.1.2.3 Transparent Gateway Agents

For each non-Oracle Database system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle Database system. The agent is specific to the non-Oracle Database system, so each type of system requires a different agent.

The transparent gateway agent facilitates communication between Oracle Database and non-Oracle Database systems and uses the Heterogeneous Services component in the Oracle Database server. The agent executes SQL and transactional requests at the non-Oracle Database system on behalf of the Oracle Database server.

See Also:

Your Oracle-supplied gateway-specific documentation for information about transparent gateways

30.1.2.4 Generic Connectivity

Generic connectivity enables you to connect to non-Oracle Database data stores by using either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent.

Both are included with your Oracle product as a standard feature. Any data source compatible with the ODBC or OLE DB standards can be accessed using a generic connectivity agent.

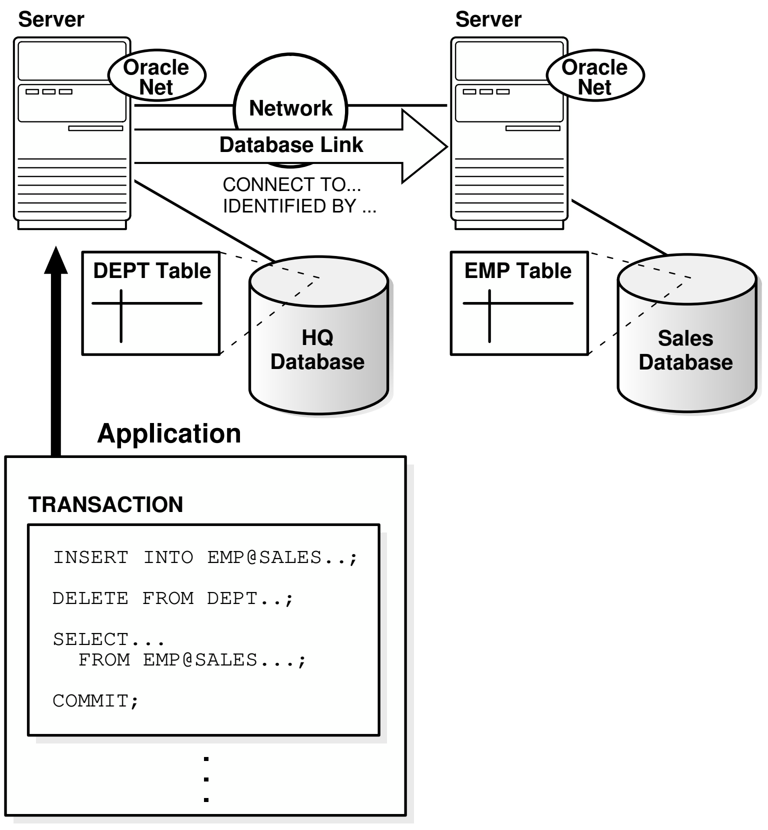
The advantage to generic connectivity is that it may not be required for you to purchase and configure a separate system-specific agent. You use an ODBC or OLE DB driver that can interface with the agent. However, some data access features are only available with transparent gateway agents.

30.1.3 Client/Server Database Architecture

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

In Figure 30-2, the host for the `hq` database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local `dept` table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table `emp` in the `sales` database).

Figure 30-2 An Oracle Database Distributed Database System



```
INSERT INTO EMP@SALES..;

DELETE FROM DEPT..;

SELECT...
  FROM EMP@SALES...;

COMMIT;
```

A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. For example, if you connect to the `hq` database and access the `dept` table on this database as in Figure 30-2, you can issue the following:

```
SELECT * FROM dept;
```

This query is direct because you are not accessing an object on a remote database.

In contrast, an indirect connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, if you connect to the `hq` database but access the `emp` table on the remote `sales` database as in Figure 30-2, you can issue the following:

```
SELECT * FROM emp@sales;
```

This query is indirect because the object you are accessing is not on the database to which you are directly connected.

30.2 Database Links

The central concept in distributed database systems is a **database link**. A database link is a connection between two physical database servers that allows a client to access them as one logical database.

- What Are Database Links?
  A database link is a pointer that defines a one-way communication path from an Oracle Database server to another database server.
- What Are Shared Database Links?
  A shared database link is a link between a local server process and the remote database. The link is shared because multiple client processes can use the same link simultaneously.
- Why Use Database Links?
  The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object owner. In other words, a local user can access a link to a remote database without having to be a user on the remote database.
- Global Database Names in Database Links
  To understand how a database link works, you must first understand what a global database name is. Each database in a distributed database is uniquely identified by its global database name.
- Global Name as a Loopback Database Link
  You can use the global name of a database as a loopback database link without explicitly creating a database link. When the database link in a SQL statement matches the global name of the current database, the database link is effectively ignored.
- Names for Database Links
  Typically, a database link has the same name as the global database name of the remote database that it references.
- Types of Database Links
  Oracle Database lets you create **private**, **public**, and **global** database links.
- Users of Database Links
  Users of database links include connect user, current user, and fixed user.
- Creation of Database Links: Examples
  Create database links using the `CREATE DATABASE LINK` statement.

- Schema Objects and Database Links
  After you have created a database link, you can execute SQL statements that access objects on the remote database. You must also be authorized in the remote database to access specific remote objects.
- Database Link Restrictions
  Several restrictions apply to database links.

30.2.1 What Are Database Links?

A database link is a pointer that defines a one-way communication path from an Oracle Database server to another database server.

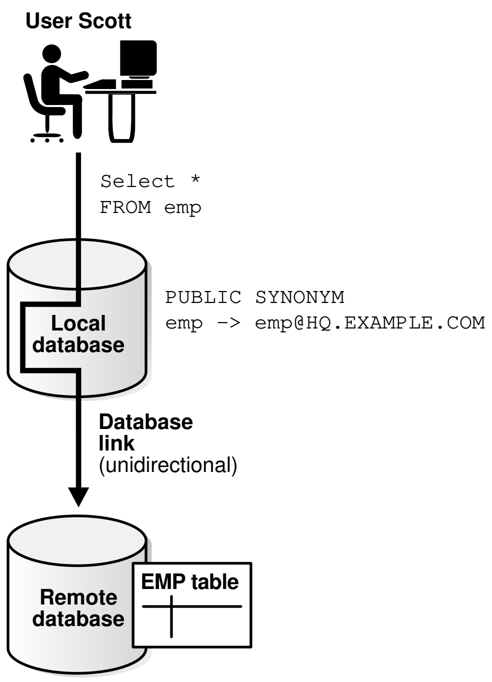
For public and private database links, the link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry. For global database links, the link pointer is defined in a directory service. The different types of database links are described in more detail in "Types of Database Links".

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique **global database name** in the network domain. The global database name uniquely identifies a database server in a distributed system.

Figure 30-3 shows an example of user `scott` accessing the `emp` table on the remote database with the global name `hq.example.com`:

Figure 30-3 Database Link



Database links are either private or public. If they are private, then only the user who created the link has access; if they are public, then all database users have access.

One principal difference among database links is the way that different link definitions determine how the link connection is authenticated. Users access a remote database through the following types of links:

Type of Link	Description
Connected user link	Users connect as themselves, which means that they must have an account on the remote database with the same user name and password as their account on the local database.
Fixed user link	Users connect using the user name and password referenced in the link. For example, if Jane uses a fixed user link that connects to the `hq` database with the user name and password `scott/`password, then she connects as `scott`, Jane has all the privileges in `hq` granted to `scott` directly, and all the default roles that `scott` has been granted in the `hq` database.
Current user link	A user connects as a global user. A local user can connect as a global user in the context of a stored procedure, without storing the global user's password in a link definition. For example, Jane can access a procedure that Scott wrote, accessing Scott's account and Scott's schema on the `hq` database.

Create database links using the `CREATE DATABASE LINK` statement. After a link is created, you can use it to specify schema objects in SQL statements.

See Also:

Oracle Database SQL Language Reference for syntax of the `CREATE DATABASE` statement

30.2.2 What Are Shared Database Links?

A shared database link is a link between a local server process and the remote database. The link is shared because multiple client processes can use the same link simultaneously.

When a local database is connected to a remote database through a database link, either database can run in dedicated or shared server mode. The following table illustrates the possibilities:

Local Database Mode	Remote Database Mode
Dedicated	Dedicated
Dedicated	Shared server
Shared server	Dedicated
Shared server	Shared server

A shared database link can exist in any of these four configurations. Shared links differ from standard database links in the following ways:

- Different users accessing the same schema object through a database link can share a network connection.

- When a user must establish a connection to a remote server from a particular server process, the process can reuse connections already established to the remote server. The reuse of the connection can occur if the connection was established on the same server process with the same database link, possibly in a different session. In a non-shared database link, a connection is not shared across multiple sessions.
- When you use a shared database link in a shared server configuration, a network connection is established directly out of the shared server process in the local server. For a non-shared database link on a local shared server, this connection would have been established through the local dispatcher, requiring context switches for the local dispatcher, and requiring data to go through the dispatcher.

> **See Also:**
>
> *Oracle Database Net Services Administrator's Guide* for information about shared server

30.2.3 Why Use Database Links?

The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object owner. In other words, a local user can access a link to a remote database without having to be a user on the remote database.

For example, assume that employees submit expense reports to Accounts Payable (A/P), and further suppose that a user using an A/P application must retrieve information about employees from the `hq` database. The A/P users should be able to connect to the `hq` database and execute a stored procedure in the remote `hq` database that retrieves the desired information. The A/P users should not need to be `hq` database users to do their jobs; they should only be able to access `hq` information in a controlled way as limited by the procedure.

> **See Also:**
>
> - "Users of Database Links" for an explanation of database link users
> - "Viewing Information About Database Links" for an explanation of how to hide passwords from non-administrative users

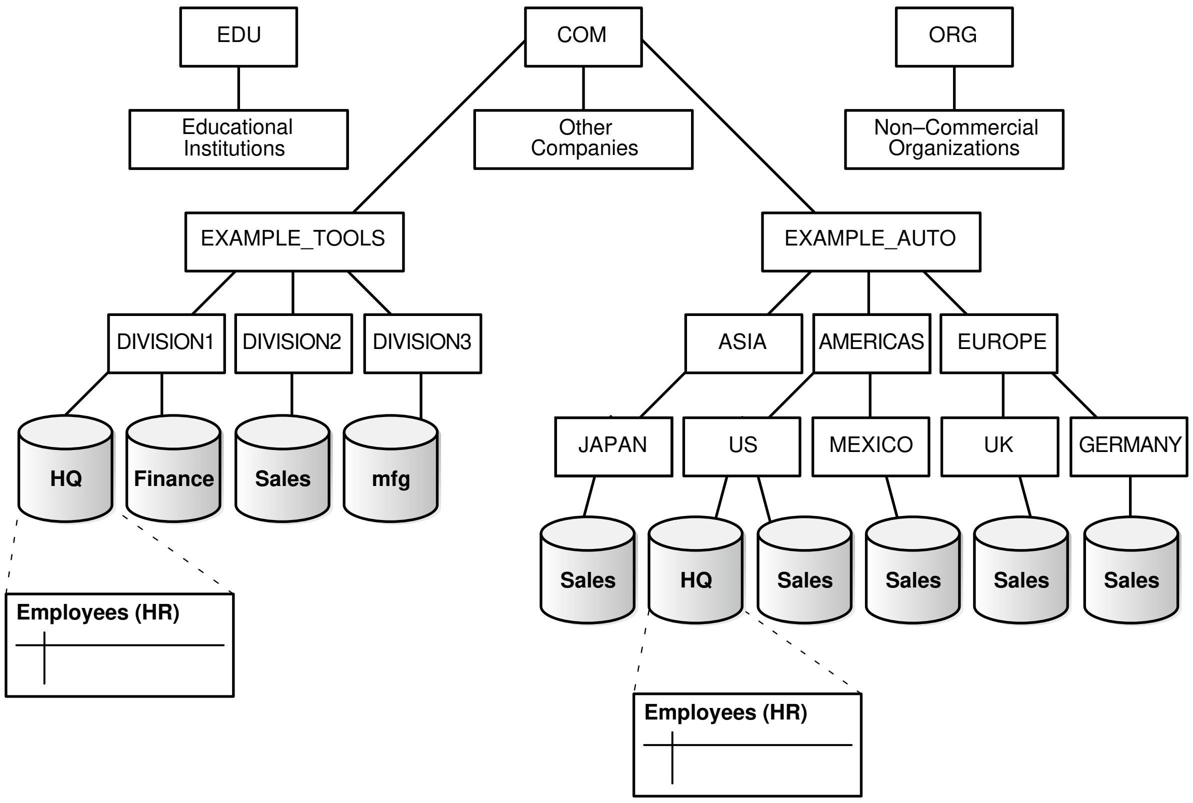
30.2.4 Global Database Names in Database Links

To understand how a database link works, you must first understand what a global database name is. Each database in a distributed database is uniquely identified by its global database name.

The database forms a global database name by prefixing the database network domain, specified by the `DB_DOMAIN` initialization parameter at database creation, with the individual database name, specified by the `DB_NAME` initialization parameter.

For example, Figure 30-4 illustrates a representative hierarchical arrangement of databases throughout a network.

Figure 30-4 Hierarchical Arrangement of Networked Databases



The name of a database is formed by starting at the leaf of the tree and following a path to the root. For example, the `mfg` database is in `division3` of the `example_tools` branch of the `com` domain. The global database name for `mfg` is created by concatenating the nodes in the tree as follows:

- `mfg.division3.example_tools.com`

While several databases can share an individual name, each database must have a unique global database name. For example, the network domains `us.americas.example_auto.com` and `uk.europe.example_auto.com` each contain a `sales` database. The global database naming system distinguishes the `sales` database in the `americas` division from the `sales` database in the `europe` division as follows:

- `sales.us.americas.example_auto.com`
- `sales.uk.europe.example_auto.com`

See Also:

"Managing Global Names in a Distributed System" to learn how to specify and change global database names

30.2.5 Global Name as a Loopback Database Link

You can use the global name of a database as a loopback database link without explicitly creating a database link. When the database link in a SQL statement matches the global name of the current database, the database link is effectively ignored.

For example, assume the global name of a database is `db1.example.com`. You can run the following SQL statement on this database:

```
SELECT * FROM hr.employees@db1.example.com;
```

In this case, the `@db1.example.com` portion of the SQL statement is effectively ignored.

30.2.6 Names for Database Links

Typically, a database link has the same name as the global database name of the remote database that it references.

For example, if the global database name of a database is `sales.us.example.com`, then the database link is also called `sales.us.example.com`.

When you set the initialization parameter `GLOBAL_NAMES` to `TRUE`, the database ensures that the name of the database link is the same as the global database name of the remote database. For example, if the global database name for `hq` is `hq.example.com`, and `GLOBAL_NAMES` is `TRUE`, then the link name must be called `hq.example.com`. Note that the database checks the domain part of the global database name as stored in the data dictionary, *not* the `DB_DOMAIN` setting in the initialization parameter file (see "Changing the Domain in a Global Database Name").

If you set the initialization parameter `GLOBAL_NAMES` to `FALSE`, then you are not required to use global naming. You can then name the database link whatever you want. For example, you can name a database link to `hq.example.com` as `foo`.

Note:

Oracle recommends that you use global naming because many useful features require global naming.

After you have enabled global naming, database links are essentially transparent to users of a distributed database because the name of a database link is the same as the global name of the database to which the link points. For example, the following statement creates a database link in the local database to remote database `sales`:

```
CREATE PUBLIC DATABASE LINK sales.division3.example.com USING 'sales1';
```

See Also:

Oracle Database Reference for more information about specifying the initialization parameter `GLOBAL_NAMES`

30.2.7 Types of Database Links

Oracle Database lets you create **private**, **public**, and **global** database links.

These basic link types differ according to which users are allowed access to the remote database:

Type	Owner	Description
Private	User who created the link. View ownership data through: • `DBA_DB_LINKS` • `ALL_DB_LINKS` • `USER_DB_LINKS`	Creates link in a specific schema of the local database. Only the owner of a private database link or PL/SQL subprograms in the schema can use this link to access database objects in the corresponding remote database.
Public	User called PUBLIC. View ownership data through views shown for private database links.	Creates a database-wide link. All users and PL/SQL subprograms in the database can use the link to access database objects in the corresponding remote database.
Global	No user owns the global database link. The global database link exists in a directory service.	Creates a network-wide link. When an Oracle network uses a directory server and the database is registered in the directory service, this information can be used as a database link. Users and PL/SQL subprograms in any database can use a global database link to access objects in the corresponding remote database. Global database links refer to the use of net service names from the directory server.

Determining the type of database links to employ in a distributed database depends on the specific requirements of the applications using the system. Consider these features when making your choice:

Type of Link	Features
Private database link	This link is more secure than a public or global link, because only the owner of the private link, or subprograms within the same schema, can use the link to access the remote database.
Public database link	When many users require an access path to a remote Oracle Database, you can create a single public database link for all users in a database.
Global database link	When an Oracle network uses a directory server, an administrator can conveniently manage global database links for all databases in the system. Database link management is centralized and simple. There is no user data associated with a global database link definition. A global database link must operate as a connected user database link.

See Also:

- "Specifying Link Types" to learn how to create different types of database links
- "Viewing Information About Database Links" to learn how to access information about links

30.2.8 Users of Database Links

Users of database links include connect user, current user, and fixed user.

- Overview of Database Link Users
  When creating the link, you determine which user should connect to the remote database to access the data.
- Connected User Database Links
  Connected user links have no connect string associated with them. The advantage of a connected user link is that a user referencing the link connects to the remote database as the same user, and credentials do not have to be stored in the link definition in the data dictionary.
- Fixed User Database Links
  A benefit of a fixed user link is that it connects a user in a primary database to a remote database with the security context of the user specified in the connect string.
- Current User Database Links
  Current user database links make use of a global user. A global user must be authenticated by an X.509 certificate or a password, and be a user on both databases involved in the link.

30.2.8.1 Overview of Database Link Users

When creating the link, you determine which user should connect to the remote database to access the data.

The following table explains the differences among the categories of users involved in database links:

User Type	Description	Sample Link Creation Syntax
Connected user	A local user accessing a database link in which no fixed username and password have been specified. If `SYSTEM` accesses a public link in a query, then the connected user is `SYSTEM`, and the database connects to the `SYSTEM` schema in the remote database. **Note:** A connected user does not have to be the user who created the link, but is any user who is accessing the link.	`CREATE PUBLIC DATABASE LINK hq USING 'hq';`
Current user	A global user in a `CURRENT_USER` database link. The global user must be authenticated by an X.509 certificate (an SSL-authenticated enterprise user) or a password (a password-authenticated enterprise user), and be a user on both databases involved in the link. See *Oracle Database Enterprise User Security Administrator's Guide* for information about global security	`CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER using 'hq';`
Fixed user	A user whose username/password is part of the link definition. If a link includes a fixed user, the fixed user's username and password are used to connect to the remote database.	`CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY` *`password`* `USING 'hq';`

Note:

The following users cannot be target users of database links: `SYS` and `PUBLIC`.

See Also:

"Specifying Link Users" to learn how to specify users when creating links

30.2.8.2 Connected User Database Links

Connected user links have no connect string associated with them. The advantage of a connected user link is that a user referencing the link connects to the remote database as the same user, and credentials do not have to be stored in the link definition in the data dictionary.

Connected user links have some disadvantages. Because these links require users to have accounts and privileges on the remote databases to which they are attempting to connect, they require more privilege administration for administrators. Also, giving users more privileges than they need violates the fundamental security concept of least privilege: users should only be given the privileges they need to perform their jobs.

The ability to use a connected user database link depends on several factors, chief among them whether the user is authenticated by the database using a password, or externally authenticated by the operating system or a network authentication service. If the user is externally authenticated, then the ability to use a connected user link also depends on whether the remote database accepts remote authentication of users, which is set by the `REMOTE_OS_AUTHENT` initialization parameter.

The `REMOTE_OS_AUTHENT` parameter operates as follows:

REMOTE_OS_AUTHENT Value	Consequences
`TRUE` for the remote database	An externally-authenticated user can connect to the remote database using a connected user database link.
`FALSE` for the remote database	An externally-authenticated user cannot connect to the remote database using a connected user database link unless a secure protocol or a network authentication service option is used.

If the connected user database link is accessed from within a definer's rights function, procedure, or package, then the definer's authorization ID is used to connect as a remote user. For example, if user `jane` calls procedure `scott.p` (a definer's rights procedure created by `scott`), and the link appears inside procedure `scott.p`, then `scott` is the connected user. To run a definer's rights function, procedure, or package that includes a connected user database link, the user who invokes the function, procedure, or package must be granted the `INHERIT REMOTE PRIVILEGES` privilege.

Note:

The `REMOTE_OS_AUTHENT` initialization parameter is deprecated. It is retained for backward compatibility only.

30.2.8.3 Fixed User Database Links

A benefit of a fixed user link is that it connects a user in a primary database to a remote database with the security context of the user specified in the connect string.

For example, local user `joe` can create a public database link in `joe`'s schema that specifies the fixed user `scott` with password *`password`*. If `jane` uses the fixed user link in a query, then `jane` is the user on the local database, but she connects to the remote database as `scott/`*`password`*.

Fixed user links have a user name and password associated with the connect string. The user name and password are stored with other link information in data dictionary tables.

30.2.8.4 Current User Database Links

Current user database links make use of a global user. A global user must be authenticated by an X.509 certificate or a password, and be a user on both databases involved in the link.

The user invoking the `CURRENT_USER` link does not have to be a global user. For example, if `jane` is authenticated (not as a global user) by password to the Accounts Payable database, she can access a stored procedure to retrieve data from the `hq` database. The procedure uses a current user database link, which connects her to `hq` as global user `scott`. User `scott` is a global user and authenticated through a certificate over SSL, but `jane` is not.

Note that current user database links have these consequences:

- If the current user database link is *not* accessed from within a stored object, then the current user is the same as the connected user accessing the link. For example, if `scott` issues a `SELECT` statement through a current user link, then the current user is `scott`.
- When executing a stored object such as a procedure, view, or trigger that accesses a database link, the current user is the user that *owns* the stored object, and not the user that *calls* the object. For example, if `jane` calls procedure `scott.p` (created by `scott`), and a current user link appears *within* the called procedure, then `scott` is the current user of the link.
- If the stored object is an invoker's rights function, procedure, or package, then the invoker's authorization ID is used to connect as a remote user. For example, if user `jane` calls procedure `scott.p` (an invoker's rights procedure created by `scott`), and the link appears inside procedure `scott.p`, then `jane` is the current user.
- You cannot connect to a database as an enterprise user and then use a current user link in a stored procedure that exists in a shared, global schema. For example, if user `jane` accesses a stored procedure in the shared schema `guest` on database `hq`, she cannot use a current user link in this schema to log on to a remote database.

See Also:

- "Distributed Database Security" for more information about security issues relating to database links
- *Oracle Database PL/SQL Language Reference* for more information about invoker's rights functions, procedures, or packages.
- *Oracle Database Security Guide* for information about running a definer's rights function, procedure, or package that includes a current user database link

30.2.9 Creation of Database Links: Examples

Create database links using the `CREATE DATABASE LINK` statement.

The table gives examples of SQL statements that create database links in a local database to the remote `sales.us.americas.example_auto.com` database:

SQL Statement	Connects To Database	Connects As	Link Type
`CREATE DATABASE LINK sales.us.americas.example_auto.com USING 'sales_us';`	`sales` using net service name `sales_us`	Connected user	Private connected user
`CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';`	`sales` using service name `am_sls`	Current global user	Private current user
`CREATE DATABASE LINK sales.us.americas.example_auto.com CONNECT TO scott IDENTIFIED BY` *`password`* `USING 'sales_us';`	`sales` using net service name `sales_us`	`scott` using password *`password`*	Private fixed user
`CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY` *`password`* `USING 'rev';`	`sales` using net service name `rev`	`scott` using password *`password`*	Public fixed user

SQL Statement	Connects To Database	Connects As	Link Type
`CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.example_auto.com CONNECT TO scott IDENTIFIED BY` *`password`* `AUTHENTICATED BY anupam IDENTIFIED BY` *`password1`* `USING 'sales';`	`sales` using net service name `sales`	`scott` using password *`password`*, authenticated as `anupam` using password *`password1`*	Shared public fixed user

See Also:

- "Creating Database Links" to learn how to create link
- *Oracle Database SQL Language Reference* for information about the `CREATE DATABASE LINK` statement syntax

30.2.10 Schema Objects and Database Links

After you have created a database link, you can execute SQL statements that access objects on the remote database. You must also be authorized in the remote database to access specific remote objects.

For example, to access remote object `emp` using database link `foo`, you can issue:

```
SELECT * FROM emp@foo;
```

Constructing properly formed object names using database links is an essential aspect of data manipulation in distributed systems.

If you call a procedure using a remote database link, and the procedure contains a `ROLLBACK` command, only DML operations performed at the remote site are rolled back. Any DML changes made at the originating site are not rolled back.

- Naming of Schema Objects Using Database Links
  Oracle Database uses the global database name to name the schema objects globally.
- Authorization for Accessing Remote Schema Objects
  To access a remote schema object, you must be granted access to the remote object in the remote database.
- Synonyms for Schema Objects
  Oracle Database lets you create synonyms so that you can hide the database link name from the user.
- Schema Object Name Resolution
  To resolve application references to schema objects (a process called **name resolution**), the database forms object names hierarchically.

30.2.10.1 Naming of Schema Objects Using Database Links

Oracle Database uses the global database name to name the schema objects globally.

Global database names are in the following form:

schema.schema_object@global_database_name

where:

- *schema* is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.
- *schema_object* is a logical data structure like a table, index, view, synonym, procedure, package, or a database link.
- *global_database_name* is the name that uniquely identifies a remote database. This name must be the same as the concatenation of the remote database initialization parameters `DB_NAME` and `DB_DOMAIN`, unless the parameter `GLOBAL_NAMES` is set to `FALSE`, in which case any name is acceptable.

For example, using a database link to database `sales.division3.example.com`, a user or application can reference remote data as follows:

```
SELECT * FROM scott.emp@sales.division3.example.com;  # emp table in scott's schema
SELECT loc FROM scott.dept@sales.division3.example.com;
```

If `GLOBAL_NAMES` is set to `FALSE`, then you can use any name for the link to `sales.division3.example.com`. For example, you can call the link `foo`. Then, you can access the remote database as follows:

```
SELECT name FROM scott.emp@foo;  # link name different from global name
```

30.2.10.2 Authorization for Accessing Remote Schema Objects

To access a remote schema object, you must be granted access to the remote object in the remote database.

Further, to perform any updates, inserts, or deletes on the remote object, you must be granted the `READ` or `SELECT` privilege on the object, along with the `UPDATE`, `INSERT`, or `DELETE` privilege. Unlike when accessing a local object, the `READ` or `SELECT` privilege is necessary for accessing a remote object because the database has no remote describe capability. The database must do a `SELECT *` on the remote object to determine its structure.

30.2.10.3 Synonyms for Schema Objects

Oracle Database lets you create synonyms so that you can hide the database link name from the user.

A synonym allows access to a table on a remote database using the same syntax that you would use to access a table on a local database. For example, assume you issue the following query against a table in a remote database:

```
SELECT * FROM emp@hq.example.com;
```

You can create the synonym `emp` for `emp@hq.example.com` so that you can issue the following query instead to access the same data:

```
SELECT * FROM emp;
```

See Also:

"Using Synonyms to Create Location Transparency" to learn how to create synonyms for objects specified using database links

30.2.10.4 Schema Object Name Resolution

To resolve application references to schema objects (a process called **name resolution**), the database forms object names hierarchically.

For example, the database guarantees that each schema within a database has a unique name, and that within a schema each object has a unique name. As a result, a schema object name is always unique within the database. Furthermore, the database resolves application references to the local name of the object.

In a distributed database, a schema object such as a table is accessible to all applications in the system. The database extends the hierarchical naming model with global database names to effectively create **global object names** and resolve references to the schema objects in a distributed database system. For example, a query can reference a remote table by specifying its fully qualified name, including the database in which it resides.

For example, assume that you connect to the local database as user `SYSTEM`:

```
CONNECT SYSTEM@sales1
```

You then issue the following statements using database link `hq.example.com` to access objects in the `scott` and `jane` schemas on remote database `hq`:

```
SELECT * FROM scott.emp@hq.example.com;
INSERT INTO jane.accounts@hq.example.com (acc_no, acc_name, balance)
  VALUES (5001, 'BOWER', 2000);
UPDATE jane.accounts@hq.example.com
  SET balance = balance + 500;
DELETE FROM jane.accounts@hq.example.com
  WHERE acc_name = 'BOWER';
```

30.2.11 Database Link Restrictions

Several restrictions apply to database links.

You *cannot* perform the following operations using database links:

- Grant privileges on remote objects
- Execute `DESCRIBE` operations on some remote objects. The following remote objects, however, do support `DESCRIBE` operations:
  - Tables
  - Views
  - Procedures
  - Functions
- Analyze remote objects

- Define or enforce referential integrity
- Grant roles to users in a remote database
- Obtain nondefault roles on a remote database. For example, if `jane` connects to the local database and executes a stored procedure that uses a fixed user link connecting as `scott`, `jane` receives `scott`'s default roles on the remote database. Jane cannot issue `SET ROLE` to obtain a nondefault role.
- Use a current user link without authentication through SSL, password, or Microsoft Windows native authentication

See Also:

- *Oracle Database Object-Relational Developer's Guide* for information about database link restrictions for user-defined types
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for information about database link restrictions for LOBs

30.3 Distributed Database Administration

Distributed database administration includes topics related to site autonomy, security, auditing database links, and administration tools.

- Site Autonomy
  Site autonomy means that each server participating in a distributed database is administered independently from all other databases.
- Distributed Database Security
  The database supports all of the security features that are available with a non-distributed database environment for distributed database systems, including password authentication for users and roles, some types of external authentication for users and roles including Kerberos version 5 for connected user links, and login packet encryption for client-to-server and server-to-server connections.
- Auditing Database Links
  You must always perform auditing operations locally. That is, if a user acts in a local database and accesses a remote database through a database link, the local actions are audited in the local database, and the remote actions are audited in the remote database, provided appropriate audit options are set in the respective databases.
- Administration Tools
  The database administrator has several choices for tools to use when managing an Oracle Database distributed database system.

See Also:

- Managing a Distributed Database to learn how to administer homogenous systems
- *Oracle Database Heterogeneous Connectivity User's Guide* to learn about heterogeneous services concepts

30.3.1 Site Autonomy

Site autonomy means that each server participating in a distributed database is administered independently from all other databases.

Although several databases can work together, each database is a separate repository of data that is managed individually. Some of the benefits of site autonomy in an Oracle Database distributed database include:

- Nodes of the system can mirror the logical organization of companies or groups that need to maintain independence.
- Local administrators control corresponding local data. Therefore, each database administrator's domain of responsibility is smaller and more manageable.
- Independent failures are less likely to disrupt other nodes of the distributed database. No single database failure need halt all distributed operations or be a performance bottleneck.
- Administrators can recover from isolated system failures independently from other nodes in the system.
- A data dictionary exists for each local database. A global catalog is not necessary to access local data.
- Nodes can upgrade software independently.

Although Oracle Database permits you to manage each database in a distributed database system independently, you should not ignore the global requirements of the system. For example, you may need to:

- Create additional user accounts in each database to support the links that you create to facilitate server-to-server connections.
- Set additional initialization parameters such as `COMMIT_POINT_STRENGTH`, and `OPEN_LINKS`.

30.3.2 Distributed Database Security

The database supports all of the security features that are available with a non-distributed database environment for distributed database systems, including password authentication for users and roles, some types of external authentication for users and roles including Kerberos version 5 for connected user links, and login packet encryption for client-to-server and server-to-server connections.

- Authentication Through Database Links
  Database links are either private or public, **authenticated** or **non-authenticated**.
- Authentication Without Passwords
  When using a connected user or current user database link, you can use an external authentication source such as Kerberos to obtain **end-to-end security**.

- Supporting User Accounts and Roles
  In a distributed database system, you must carefully plan the user accounts and roles that are necessary to support applications using the system.
- Centralized User and Privilege Management
  For centralized user and privilege management, you must consider the authentication method. You can also consider exclusively mapped global users or shared schema users.
- Data Encryption
  The Oracle Advanced Security option also enables Oracle Net and related products to use network data encryption and checksumming so that data cannot be read or altered. It protects data from unauthorized viewing by using the RSA Data Security RC4 or the Data Encryption Standard (DES) encryption algorithm.

See Also:

Oracle Database Enterprise User Security Administrator's Guide for more information about external authentication

30.3.2.1 Authentication Through Database Links

Database links are either private or public, **authenticated** or **non-authenticated**.

You create public links by specifying the `PUBLIC` keyword in the link creation statement. For example, you can issue:

```
CREATE PUBLIC DATABASE LINK foo USING 'sales';
```

You create authenticated links by specifying the `CONNECT TO` clause, `AUTHENTICATED BY` clause, or both clauses together in the database link creation statement. For example, you can issue:

```
CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY password USING 'sales';

CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO nick IDENTIFIED BY password1
     AUTHENTICATED BY david IDENTIFIED BY password2 USING 'sales';
```

This table describes how users access the remote database through the link:

Link Type	Authenticated	Security Access
Private	No	When connecting to the remote database, the database uses security information (userid/password) taken from the local session. Hence, the link is a connected user database link. Passwords must be synchronized between the two databases.
Private	Yes	The userid/password is taken from the link definition rather than from the local session context. Hence, the link is a fixed user database link. This configuration allows passwords to be different on the two databases, but the local database link password must match the remote database password.
Public	No	Works the same as a private nonauthenticated link, except that all users can reference this pointer to the remote database.
Public	Yes	All users on the local database can access the remote database, and all use the same userid/password to make the connection.

Note:

The following initialization parameters can provide enhanced security for database link connections:

- The `OUTBOUND_DBLINK_PROTOCOLS` initialization parameter can specify Oracle Net transport protocols restricting any outbound database link communication to use only the protocols from the specified list.
- The `ALLOW_GLOBAL_DBLINKS` initialization parameter can allow or disallow LDAP lookup for global database links information.

See Also:

- *Oracle Database Security Guide*
- *Oracle Database Reference* for more information about the `OUTBOUND_DBLINK_PROTOCOLS` initialization parameter
- *Oracle Database Reference* for more information about the `ALLOW_GLOBAL_DBLINKS` initialization parameter

30.3.2.2 Authentication Without Passwords

When using a connected user or current user database link, you can use an external authentication source such as Kerberos to obtain **end-to-end security**.

In end-to-end authentication, credentials are passed from server to server and can be authenticated by a database server belonging to the same domain. For example, if `jane` is authenticated externally on a local database, and wants to use a connected user link to connect as herself to a remote database, the local server passes the security ticket to the remote database.

30.3.2.3 Supporting User Accounts and Roles

In a distributed database system, you must carefully plan the user accounts and roles that are necessary to support applications using the system.

Note that:

- The user accounts necessary to establish server-to-server connections must be available in all databases of the distributed database system.
- The roles necessary to make available application privileges to distributed database application users must be present in all databases of the distributed database system.

As you create the database links for the nodes in a distributed database system, determine which user accounts and roles each site must support server-to-server connections that use the links.

In a distributed environment, users typically require access to many network services. When you must configure separate authentications for each user to access each network service, security administration can become unwieldy, especially for large systems.

See Also:

"Creating Database Links" for more information about the user accounts that must be available to support different types of database links in the system

30.3.2.4 Centralized User and Privilege Management

For centralized user and privilege management, you must consider the authentication method. You can also consider exclusively mapped global users or shared schema users.

- About Centralized User and Privilege Management
  The database provides different ways for you to manage the users and their privileges in a distributed system.
- Exclusively Mapped Global Users
  One option for centralizing user and privilege management is to create a global user in a centralized directory and a user in every database to which the global user must connect.
- Shared Schema Users
  The shared schema users functionality allows a global user to be centrally managed by an enterprise directory service. Users who are managed in the directory are called **enterprise users**.

30.3.2.4.1 About Centralized User and Privilege Management

The database provides different ways for you to manage the users and their privileges in a distributed system.

For example, you have these options:

- Enterprise user management

  You can create global users who are authenticated using passwords, Kerberos, or PKI certificates. You can then manage these users and their authorizations in a directory using an independent enterprise directory service.
- Network authentication service

  This common technique simplifies security management for distributed environments. You can use the Oracle Advanced Security option to enhance Oracle Net and the security of an Oracle Database distributed database system. Microsoft Windows native authentication is an example of a non-Oracle authentication solution.

See Also:

- *Oracle Database Security Guide* for more information about Oracle Database security
- *Oracle Database Enterprise User Security Administrator's Guide* for more information about enterprise user security in Oracle Database
- *Oracle Database Security Guide* for more information about configuring users with Microsoft Active Directory

30.3.2.4.2 Exclusively Mapped Global Users

One option for centralizing user and privilege management is to create a global user in a centralized directory and a user in every database to which the global user must connect.

For example, you can create a global user called `fred` with the following SQL statement:

```
CREATE USER fred IDENTIFIED GLOBALLY AS 'CN=fred adams,O=Oracle,C=England';
```

This solution allows a single global database user to be authenticated by a centralized directory and map the database user exclusively to a directory user.

The exclusively mapped global user solution has the consequence that you must create a user called `fred` on every database that this user must access. Because most users need permission to access an application schema but do not need their own schemas, the creation of a separate account in each database for every global user creates significant overhead. Because of this problem, the database also supports *shared schema users*, which are global users that can access a single, generic schema in every database.

30.3.2.4.3 Shared Schema Users

The shared schema users functionality allows a global user to be centrally managed by an enterprise directory service. Users who are managed in the directory are called **enterprise users**.

This directory may contain information about:

- Which databases in a distributed system an enterprise user can access
- Which role on each database an enterprise user can use
- Which schema on each database an enterprise user can connect to

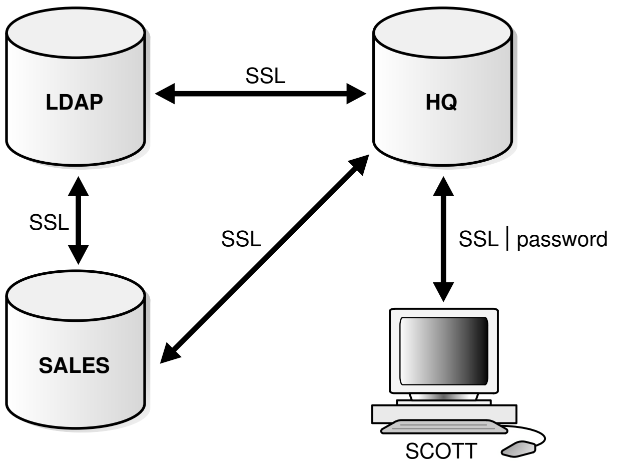
The administrator of each database is not required to create a global user account for each enterprise user on each database to which the enterprise user must connect. Instead, multiple enterprise users can connect to the same database schema, called a **shared schema**.

> **Note:**
>
> You cannot access a current user database link in a shared schema.

For example, suppose `jane`, `bill`, and `scott` all use a human resources application. The `hq` application objects are all contained in the `guest` schema on the `hq` database. In this case, you can create a local global user account to be used as a shared schema. This global user name, that is, shared schema name, is `guest`. `jane`, `bill`, and `scott` are all created as enterprise users in the directory service. They are also mapped to the `guest` schema in the directory, and can be assigned different authorizations in the `hq` application.

Figure 30-5 illustrates an example of global user security using the enterprise directory service:

Figure 30-5   Global User Security



Assume that the enterprise directory service contains the following information on enterprise users for `hq` and `sales`:

Database	Role	Schema	Enterprise Users
`hq`	`clerk1`	`guest`	`bill` `scott`
`sales`	`clerk2`	`guest`	`jane` `scott`

Also, assume that the local administrators for `hq` and `sales` have issued statements as follows:

Database	CREATE Statements

`hq`	`CREATE USER guest IDENTIFIED GLOBALLY AS '';`
	`CREATE ROLE clerk1 GRANT select ON emp;`
	`CREATE PUBLIC DATABASE LINK sales_link CONNECT AS CURRENT_USER USING 'sales';`

`sales`	`CREATE USER guest IDENTIFIED GLOBALLY AS '';`
	`CREATE ROLE clerk2 GRANT select ON dept;`

Assume that enterprise user `scott` requests a connection to local database `hq` in order to execute a distributed transaction involving `sales`. The following steps occur (not necessarily in this exact order):

1. Enterprise user `scott` is authenticated using SSL or a password.
2. User `scott` issues the following statement:

   ```
   SELECT e.ename, d.loc
   FROM emp e, dept@sales_link d
   WHERE e.deptno=d.deptno;
   ```

3. Databases `hq` and `sales` mutually authenticate one another using SSL.

4. `Database hq` queries the enterprise directory service to determine whether enterprise user `scott` has access to `hq`, and discovers `scott` can access local schema `guest` using role `clerk1`.
5. Database `sales` queries the enterprise directory service to determine whether enterprise user `scott` has access to `sales`, and discovers `scott` can access local schema `guest` using role `clerk2`.
6. Enterprise user `scott` logs into `sales` to schema `guest` with role `clerk2` and issues a `SELECT` to obtain the required information and transfer it to `hq`.
7. Database `hq` receives the requested data from `sales` and returns it to the client `scott`.

Note:

Starting with Oracle Database 18c:

- You can create a global role using the `GLOBALLY AS [`*`domain_name_of_directory_group`*`]` clause of the `CREATE ROLE` or `ALTER ROLE` statement to map a directory group to a global role. The global user must be authorized to use the global role by the enterprise directory service before the role is enabled.

  See *Oracle Database SQL Language Reference* for the syntax of `CREATE ROLE` and `ALTER ROLE` statements.
- You can authenticate and authorize users directly with Microsoft Active Directory. Thus, Oracle database users and roles can map directly to Active Directory users and groups without using Oracle Enterprise User Security (EUS) or any other intermediate directory service.

  See *Oracle Database Security Guide* for more information about configuring users with Microsoft Active Directory.

See Also:

- *Oracle Database Enterprise User Security Administrator's Guide* for more information about enterprise user security in Oracle Database
- *Oracle Database Security Guide* for more information about Oracle Database security

30.3.2.5 Data Encryption

The Oracle Advanced Security option also enables Oracle Net and related products to use network data encryption and checksumming so that data cannot be read or altered. It protects data from unauthorized viewing by using the RSA Data Security RC4 or the Data Encryption Standard (DES) encryption algorithm.

To ensure that data has not been modified, deleted, or replayed during transmission, the security services of the Oracle Advanced Security option can generate a cryptographically secure message digest and include it with each packet sent across the network.

Note:

Starting with Oracle Database Release 21c, the RC4 algorithm is deprecated and it may be desupported in a future release.

See Also:

Oracle Database Advanced Security Guide for more information about these and other features of the Oracle Advanced Security option

30.3.3 Auditing Database Links

You must always perform auditing operations locally. That is, if a user acts in a local database and accesses a remote database through a database link, the local actions are audited in the local database, and the remote actions are audited in the remote database, provided appropriate audit options are set in the respective databases.

The remote database cannot determine whether a successful connect request and subsequent SQL statements come from another server or from a locally connected client. For example, assume the following:

- Fixed user link `hq.example.com` connects local user `jane` to the remote `hq` database as remote user `scott`.
- User `scott` is audited on the remote database.

Actions performed during the remote database session are audited as if `scott` were connected locally to `hq` and performing the same actions there. You must set audit options in the remote database to capture the actions of the username--in this case, `scott` on the `hq` database--embedded in the link if the desired effect is to audit what `jane` is doing in the remote database.

Note:

You can audit the global username for global users.

You cannot set local auditing options on remote objects. Therefore, you cannot audit use of a database link, although access to remote objects can be audited on the remote database.

Note:

Starting with Oracle Database Release 21c, traditional auditing is deprecated. Oracle recommends that you use unified auditing, which enables selective and more effective auditing inside Oracle Database.

30.3.4 Administration Tools

The database administrator has several choices for tools to use when managing an Oracle Database distributed database system.

- Cloud Control and Distributed Databases
  Oracle Enterprise Manager Cloud Control is the Oracle Database administration tool that provides a graphical user interface (GUI). Cloud Control provides administrative functionality for distributed databases through an easy-to-use interface.
- Third-Party Administration Tools
  Currently more than 60 companies produce more than 150 products that help manage Oracle Databases and networks, providing a truly open environment.
- SNMP Support
  Besides its network administration capabilities, Oracle **Simple Network Management Protocol** (*SNMP*) support allows an Oracle Database server to be located and queried by any SNMP-based network management system.

30.3.4.1 Cloud Control and Distributed Databases

Oracle Enterprise Manager Cloud Control is the Oracle Database administration tool that provides a graphical user interface (GUI). Cloud Control provides administrative functionality for distributed databases through an easy-to-use interface.

You can use Cloud Control to:

- Administer multiple databases. You can use Cloud Control to administer a single database or to simultaneously administer multiple databases.
- Centralize database administration tasks. You can administer both local and remote databases running on any Oracle Database platform in any location worldwide. In addition, these Oracle Database platforms can be connected by any network protocols supported by Oracle Net.
- Dynamically execute SQL, PL/SQL, and Cloud Control commands. You can use Cloud Control to enter, edit, and execute statements. Cloud Control also maintains a history of statements executed.

  Thus, you can reexecute statements without retyping them, a particularly useful feature if you must execute lengthy statements repeatedly in a distributed database system.
- Manage security features such as global users, global roles, and the enterprise directory service.

30.3.4.2 Third-Party Administration Tools

Currently more than 60 companies produce more than 150 products that help manage Oracle Databases and networks, providing a truly open environment.

30.3.4.3 SNMP Support

Besides its network administration capabilities, Oracle **Simple Network Management Protocol** (*SNMP*) support allows an Oracle Database server to be located and queried by any SNMP-based network management system.

SNMP is the accepted standard underlying many popular network management systems such as:

- HP OpenView
- Digital POLYCENTER Manager on NetView
- IBM NetView/6000
- Novell NetWare Management System
- SunSoft SunNet Manager

Note:

Oracle has deprecated SNMP support in Oracle Net Listener. Oracle recommends not using SNMP in new implementations. See *Oracle Database Upgrade Guide* for more information.

30.4 Transaction Processing in a Distributed System

A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user. A **remote transaction** contains only statements that access a single remote node. A **distributed transaction** contains statements that access multiple nodes.

- Remote SQL Statements
  A remote SQL statement either queries or modifies one or more remote tables, all of which reside at the same remote node.
- Distributed SQL Statements
  A distributed SQL statement either queries or modifies data on two or more nodes.
- Shared SQL for Remote and Distributed Statements
  The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement.
- Remote Transactions
  A remote transaction contains one or more remote statements, all of which reference a single remote node.
- Distributed Transactions
  A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.
- Two-Phase Commit Mechanism
  The database **two-phase commit** mechanism guarantees that *all* database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction.
- Database Link Name Resolution
  Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name.
- Schema Object Name Resolution
  It is important to understand how the remote schema is determined when a local Oracle Database connects to a remote database.

- Global Name Resolution in Views, Synonyms, and Procedures
  A global object name can be complete or partial.

30.4.1 Remote SQL Statements

A remote SQL statement either queries or modifies one or more remote tables, all of which reside at the same remote node.

A **remote query** statement is a query that selects information from one or more remote tables, all of which reside at the same remote node. For example, the following query accesses data from the `dept` table in the `scott` schema of the remote `sales` database:

```
SELECT * FROM scott.dept@sales.us.americas.example_auto.com;
```

A **remote update** statement is an update that modifies data in one or more tables, all of which are located at the same remote node. For example, the following query updates the `dept` table in the `scott` schema of the remote `sales` database:

```
UPDATE scott.dept@mktng.us.americas.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
```

> **Note:**
>
> A remote update can include a subquery that retrieves data from one or more remote nodes, but because the update happens at only a single remote node, the statement is classified as a remote update.

30.4.2 Distributed SQL Statements

A distributed SQL statement either queries or modifies data on two or more nodes.

A **distributed query** statement retrieves information from two or more nodes. For example, the following query accesses data from the local database as well as the remote `sales` database:

```
SELECT ename, dname
  FROM scott.emp e, scott.dept@sales.us.americas.example_auto.com d
  WHERE e.deptno = d.deptno;
```

A **distributed update** statement modifies data on two or more nodes. A distributed update is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes. For example, the following PL/SQL program unit updates tables on the local database and the remote `sales` database:

```
BEGIN
  UPDATE scott.dept@sales.us.americas.example_auto.com
    SET loc = 'NEW YORK'
    WHERE deptno = 10;
  UPDATE scott.emp
    SET deptno = 11
    WHERE deptno = 10;
END;
COMMIT;
```

The database sends statements in the program to the remote nodes, and their execution succeeds or fails as a unit.

30.4.3 Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement.

The SQL text must match, and the referenced objects must match. If available, shared SQL areas can be used for the local and remote handling of any statement or decomposed query.

See Also:

Oracle Database Concepts for more information about shared SQL

30.4.4 Remote Transactions

A remote transaction contains one or more remote statements, all of which reference a single remote node.

For example, the following transaction contains two statements, each of which accesses the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.example_auto.com
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

30.4.5 Distributed Transactions

A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

For example, this transaction updates the local database and the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

Note:

If all statements of a transaction reference only a single remote node, the transaction is remote, not distributed.

30.4.6 Two-Phase Commit Mechanism

The database **two-phase commit** mechanism guarantees that *all* database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction.

A database must guarantee that all statements in a transaction, distributed or non-distributed, either commit or roll back as a unit. The effects of an ongoing transaction should be invisible to all other transactions at all nodes; this transparency should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a non-distributed database are discussed in the *Oracle Database Concepts*. In a distributed database, the database must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

See Also:

Distributed Transactions Concepts for more information about the Oracle Database two-phase commit mechanism

30.4.7 Database Link Name Resolution

Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name.

- About Database Link Name Resolution
  A **global object name** is an object specified using a database link.
- Name Resolution When the Global Database Name Is Complete
  For SQL statements with a complete global database name, the database searches only for links that match the specified global database name.
- Name Resolution When the Global Database Name Is Partial
  If any part of the domain is specified, then the database assumes that a complete global database name is specified.
- Name Resolution When No Global Database Name Is Specified
  If a global object name references an object in the local database and a database link name is *not* specified using the @ symbol, then the database automatically detects that the object is local and does not search for or use database links to resolve the object reference.
- Terminating the Search for Name Resolution
  The database does not necessarily stop searching for matching database links when it finds the first match. The database must search for matching private, public, and network database links until it determines a complete path to the remote database (both a remote account and service name).

30.4.7.1 About Database Link Name Resolution

A **global object name** is an object specified using a database link.

The essential components of a global object name are:

- Object name
- Database name
- Domain

The following table shows the components of an explicitly specified global database object name:

Statement	Object	Database	Domain
`SELECT * FROM joan.dept@sales.example.com`	`dept`	`sales`	`example.com`
`SELECT * FROM emp@mktg.us.example.com`	`emp`	`mktg`	`us.example.com`

Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name. For example, if you issue the following statement:

```
SELECT * FROM scott.emp@orders.us.example.com;
```

The database searches for a database link called `orders.us.example.com`. The database performs this operation to determine the path to the specified remote database.

The database always searches for matching database links in the following order:

1. Private database links in the schema of the user who issued the SQL statement.
2. Public database links in the local database.
3. Global database links (only if a directory server is available).

30.4.7.2 Name Resolution When the Global Database Name Is Complete

For SQL statements with a complete global database name, the database searches only for links that match the specified global database name.

Assume that you issue the following SQL statement, which specifies a complete global database name:

```
SELECT * FROM emp@prod1.us.example.com;
```

In this case, both the database name (`prod1`) and domain components (`us.example.com`) are specified, so the database searches for private, public, and global database links.

30.4.7.3 Name Resolution When the Global Database Name Is Partial

If any part of the domain is specified, then the database assumes that a complete global database name is specified.

If a SQL statement specifies a partial global database name (that is, only the database component is specified), the database appends the value in the `DB_DOMAIN` initialization parameter to the value in the `DB_NAME` initialization parameter to construct a complete name. For example, assume you issue the following statements:

```
CONNECT scott@locdb
SELECT * FROM scott.emp@orders;
```

If the network domain for `locdb` is `us.example.com`, then the database appends this domain to `orders` to construct the complete global database name of `orders.us.example.com`. The database searches for database links that match only the constructed global name. If a matching link is not found, the database returns an error and the SQL statement cannot execute.

30.4.7.4 Name Resolution When No Global Database Name Is Specified

If a global object name references an object in the local database and a database link name is *not* specified using the @ symbol, then the database automatically detects that the object is local and does not search for or use database links to resolve the object reference.

For example, assume that you issue the following statements:

```
CONNECT scott@locdb
SELECT * from scott.emp;
```

Because the second statement does not specify a global database name using a database link connect string, the database does not search for database links.

30.4.7.5 Terminating the Search for Name Resolution

The database does not necessarily stop searching for matching database links when it finds the first match. The database must search for matching private, public, and network database links until it determines a complete path to the remote database (both a remote account and service name).

The first match determines the remote schema as illustrated in the following table:

User Operation	Database Response	Example
Do *not* specify the `CONNECT` clause	Uses a connected user database link	`CREATE DATABASE LINK k1 USING 'prod'`
Do specify the `CONNECT TO ... IDENTIFIED BY` clause	Uses a fixed user database link	`CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY` *`password`* `USING 'prod'`
Specify the `CONNECT TO CURRENT_USER` clause	Uses a current user database link	`CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'`
Do *not* specify the `USING` clause	Searches until it finds a link specifying a database string. If matching database links are found and a string is never identified, the database returns an error.	`CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER`

After the database determines a complete path, it creates a remote session, assuming that an identical connection is not already open on behalf of the same local session. If a session already exists, the database reuses it.

30.4.8 Schema Object Name Resolution

It is important to understand how the remote schema is determined when a local Oracle Database connects to a remote database.

- About Schema Object Name Resolution
  After the local Oracle Database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement.
- Example of Global Object Name Resolution: Complete Object Name
  An example illustrates how the database resolves a complete global object name and determines the appropriate path to the remote database using both a private and public database link.
- Example of Global Object Name Resolution: Partial Object Name
  An example illustrates how the database resolves a partial global object name and determines the appropriate path to the remote database using both a private and public database link.

30.4.8.1 About Schema Object Name Resolution

After the local Oracle Database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement.

The first match determines the remote schema according to the following rules:

Type of Link Specified	Location of Object Resolution
A fixed user database link	Schema specified in the link creation statement
A connected user database link	Connected user's remote schema
A current user database link	Current user's schema

If the database cannot find the object, then it checks public objects of the remote database. If it cannot resolve the object, then the established remote session remains but the SQL statement cannot execute and returns an error.

The following are examples of global object name resolution in a distributed database system. For all the following examples, assume that:

30.4.8.2 Example of Global Object Name Resolution: Complete Object Name

An example illustrates how the database resolves a complete global object name and determines the appropriate path to the remote database using both a private and public database link.

For this example, assume the following:

- The remote database is named `sales.division3.example.com`.
- The local database is named `hq.division3.example.com`.
- A directory server (and therefore, global database links) is not available.
- A remote table `emp` is contained in the schema `tsmith`.

Consider the following statements issued by `scott` at the local database:

```
CONNECT scott@hq

CREATE PUBLIC DATABASE LINK sales.division3.example.com
CONNECT TO guest IDENTIFIED BY network
  USING 'dbstring';
```

Later, `JWARD` connects and issues the following statements:

```
CONNECT jward@hq

CREATE DATABASE LINK sales.division3.example.com
  CONNECT TO tsmith IDENTIFIED BY radio;

UPDATE tsmith.emp@sales.division3.example.com
  SET deptno = 40
  WHERE deptno = 10;
```

The database processes the final statement as follows:

1. The database determines that a complete global object name is referenced in `jward`'s `UPDATE` statement. Therefore, the system begins searching in the local database for a database link with a matching name.
2. The database finds a matching private database link in the schema `jward`. Nevertheless, the private database link `jward.sales.division3.example.com` does not indicate a complete path to the remote `sales` database, only a remote account. Therefore, the database now searches for a matching public database link.
3. The database finds the public database link in `scott`'s schema. From this public database link, the database takes the service name `dbstring`.
4. Combined with the remote account taken from the matching private fixed user database link, the database determines a complete path and proceeds to establish a connection to the remote `sales` database as user `tsmith/radio`.
5. The remote database can now resolve the object reference to the `emp` table. The database searches in the `tsmith` schema and finds the referenced `emp` table.
6. The remote database completes the execution of the statement and returns the results to the local database.

30.4.8.3 Example of Global Object Name Resolution: Partial Object Name

An example illustrates how the database resolves a partial global object name and determines the appropriate path to the remote database using both a private and public database link.

For this example, assume that:

- The remote database is named `sales.division3.example.com`.
- The local database is named `hq.division3.example.com`.
- A directory server (and therefore, global database links) is not available.
- A table `emp` on the remote database `sales` is contained in the schema `tsmith`, but not in schema `scott`.
- A public synonym named `emp` resides at remote database `sales` and points to `tsmith.emp` in the remote database `sales`.
- The public database link in "Example of Global Object Name Resolution: Complete Object Name " is already created on local database `hq`:

```
CREATE PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO guest IDENTIFIED BY network
  USING 'dbstring';
```

Consider the following statements issued at local database `hq`:

```
CONNECT scott@hq

CREATE DATABASE LINK sales.division3.example.com;

DELETE FROM emp@sales
  WHERE empno = 4299;
```

The database processes the final `DELETE` statement as follows:

1. The database notices that a partial global object name is referenced in `scott`'s `DELETE` statement. It expands it to a complete global object name using the domain of the local database as follows:

   ```
   DELETE FROM emp@sales.division3.example.com
     WHERE empno = 4299;
   ```

2. The database searches the local database for a database link with a matching name.
3. The database finds a matching *private* connected user link in the schema `scott`, but the private database link indicates no path at all. The database uses the connected username/ password as the remote account portion of the path and then searches for and finds a matching *public* database link:

   ```
   CREATE PUBLIC DATABASE LINK sales.division3.example.com
     CONNECT TO guest IDENTIFIED BY network
     USING 'dbstring';
   ```

4. The database takes the database net service name `dbstring` from the public database link. At this point, the database has determined a complete path.
5. The database connects to the remote database as `scott/`*`password`* and searches for and does not find an object named `emp` in the schema `scott`.
6. The remote database searches for a public synonym named `emp` and finds it.
7. The remote database executes the statement and returns the results to the local database.

30.4.9 Global Name Resolution in Views, Synonyms, and Procedures

A global object name can be complete or partial.

- About Global Name Resolution in Views, Synonyms, and Procedures
  A view, synonym, or PL/SQL program unit (for example, a procedure, function, or trigger) can reference a remote schema object by its global object name.
- What Happens When Global Names Change
  Global name changes can affect views, synonyms, and procedures that reference remote data using partial global object names.
- Scenarios for Global Name Changes
  Scenarios illustrate global name changes.

30.4.9.1 About Global Name Resolution in Views, Synonyms, and Procedures

A view, synonym, or PL/SQL program unit (for example, a procedure, function, or trigger) can reference a remote schema object by its global object name.

If the global object name is complete, then the database stores the definition of the object without expanding the global object name. If the name is partial, however, then the database expands the name using the domain of the local database name.

The following table explains when the database completes the expansion of a partial global object name for views, synonyms, and program units:

User Operation	Database Response
Create a view	Does *not* expand partial global names. The data dictionary stores the exact text of the defining query. Instead, the database expands a partial global object name each time a statement that uses the view is parsed.
Create a synonym	Expands partial global names. The definition of the synonym stored in the data dictionary includes the expanded global object name.
Compile a program unit	Expands partial global names.

30.4.9.2 What Happens When Global Names Change

Global name changes can affect views, synonyms, and procedures that reference remote data using partial global object names.

If the global name of the referenced database changes, views and procedures may try to reference a nonexistent or incorrect database. However, synonyms do not expand database link names at run time, so they do not change.

30.4.9.3 Scenarios for Global Name Changes

Scenarios illustrate global name changes.

For example, consider two databases named `sales.uk.example.com` and `hq.uk.example.com`. Also, assume that the `sales` database contains the following view and synonym:

```
CREATE VIEW employee_names AS
        SELECT ename FROM scott.emp@hr;

CREATE SYNONYM employee FOR scott.emp@hr;
```

The database expands the `employee` synonym definition and stores it as:

```
scott.emp@hr.uk.example.com
```

- Scenario 1: Both Databases Change Names
  A scenario illustrates a situation in which both global database names change.
- Scenario 2: One Database Changes Names
  A scenario illustrates a situation in which one global database name changes.

30.4.9.3.1 Scenario 1: Both Databases Change Names

A scenario illustrates a situation in which both global database names change.

First, consider the situation where both the Sales and Human Resources departments are relocated to the United States. Consequently, the corresponding global database names are both changed as follows:

- `sales.uk.example.com` becomes `sales.us.example.com`
- `hq.uk.example.com` becomes `hq.us.example.com`

The following table describes query expansion before and after the change in global names:

Query on `sales`	Expansion Before Change	Expansion After Change
`SELECT * FROM employee_names`	`SELECT * FROM scott.emp@hr.uk.example.com`	`SELECT * FROM scott.emp@hr.us.example.com`
`SELECT * FROM employee`	`SELECT * FROM scott.emp@hr.uk.example.com`	`SELECT * FROM scott.emp@hr.uk.example.com`

30.4.9.3.2 Scenario 2: One Database Changes Names

A scenario illustrates a situation in which one global database name changes.

Now consider that only the Sales department is moved to the United States; Human Resources remains in the UK. Consequently, the corresponding global database names are both changed as follows:

- `sales.uk.example.com` becomes `sales.us.example.com`
- `hq.uk.example.com` is not changed

The following table describes query expansion before and after the change in global names:

Query on `sales`	Expansion Before Change	Expansion After Change
`SELECT * FROM employee_names`	`SELECT * FROM scott.emp@hr.uk.example.com`	`SELECT * FROM scott.emp@hr.us.example.com`
`SELECT * FROM employee`	`SELECT * FROM scott.emp@hr.uk.example.com`	`SELECT * FROM scott.emp@hr.uk.example.com`

In this case, the defining query of the `employee_names` view expands to a nonexistent global database name. However, the `employee` synonym continues to reference the correct database, `hq.uk.example.com`.

30.5 Distributed Database Application Development

Application development in a distributed system raises issues that are not applicable in a non-distributed system.

- Transparency in a Distributed Database System
  With minimal effort, you can develop applications that make an Oracle Database distributed database system transparent to users that work with the system. The goal of transparency is to make a distributed database system appear as though it is a single Oracle Database. Consequently, the system does not burden developers and users of the system with complexities that would otherwise make distributed database application development challenging and detract from user productivity.
- PL/SQL and Remote Procedure Calls (RPCs)
  Developers can code PL/SQL packages and procedures to support applications that work with a distributed database. Applications can make local procedure calls to perform work at the local database and **remote procedure calls** (**RPCs**) to perform work at a remote database.

- Distributed Query Optimization
  Distributed query optimization is an Oracle Database feature that reduces the amount of data transfer required between sites when a transaction retrieves data from remote tables referenced in a distributed SQL statement.

> **See Also:**
>
> Developing Applications for a Distributed Database System to learn how to develop applications for distributed systems

30.5.1 Transparency in a Distributed Database System

With minimal effort, you can develop applications that make an Oracle Database distributed database system transparent to users that work with the system. The goal of transparency is to make a distributed database system appear as though it is a single Oracle Database. Consequently, the system does not burden developers and users of the system with complexities that would otherwise make distributed database application development challenging and detract from user productivity.

- Location Transparency
  An Oracle Database distributed database system has features that allow application developers and administrators to hide the physical location of database objects from applications and users.
- SQL and COMMIT Transparency
  The Oracle Database distributed database architecture provides query, update, and transaction transparency.

30.5.1.1 Location Transparency

An Oracle Database distributed database system has features that allow application developers and administrators to hide the physical location of database objects from applications and users.

Location transparency exists when a user can universally refer to a database object such as a table, regardless of the node to which an application connects. Location transparency has several benefits, including:

- Access to remote data is simple, because database users do not need to know the physical location of database objects.
- Administrators can move database objects with no impact on end-users or existing database applications.

Typically, administrators and developers use synonyms to establish location transparency for the tables and supporting objects in an application schema. For example, the following statements create synonyms in a database for tables in another, remote database.

```
CREATE PUBLIC SYNONYM emp
  FOR scott.emp@sales.us.americas.example_auto.com;
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.example_auto.com;
```

Now, rather than access the remote tables with a query such as:

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.example_auto.com e,
     scott.dept@sales.us.americas.example_auto.com d
  WHERE e.deptno = d.deptno;
```

An application can issue a much simpler query that does not have to account for the location of the remote tables.

```
SELECT ename, dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

In addition to synonyms, developers can also use views and stored procedures to establish location transparency for applications that work in a distributed database system.

30.5.1.2 SQL and COMMIT Transparency

The Oracle Database distributed database architecture provides query, update, and transaction transparency.

For example, standard SQL statements such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` work just as they do in a non-distributed database environment. Additionally, applications control transactions using the standard SQL statements `COMMIT`, `SAVEPOINT`, and `ROLLBACK`. There is no requirement for complex programming or other special operations to provide distributed transaction control.

- The statements in a single transaction can reference any number of local or remote tables.
- The database guarantees that all nodes involved in a distributed transaction take the same action: they either all commit or all roll back the transaction.
- If a network or system failure occurs during the commit of a distributed transaction, the transaction is automatically and transparently resolved globally. Specifically, when the network or system is restored, the nodes either all commit or all roll back the transaction.

Internal to the database, each committed transaction has an associated **system change number** (**SCN**) to uniquely identify the changes made by the statements within that transaction. In a distributed database, the SCNs of communicating nodes are coordinated when:

- A connection is established using the path described by one or more database links.
- A distributed SQL statement is executed.
- A distributed transaction is committed.

Among other benefits, the coordination of SCNs among the nodes of a distributed database system allows global distributed read-consistency at both the statement and transaction level. If necessary, global distributed time-based recovery can also be completed.

30.5.2 PL/SQL and Remote Procedure Calls (RPCs)

Developers can code PL/SQL packages and procedures to support applications that work with a distributed database. Applications can make local procedure calls to perform work at the local database and **remote procedure calls** (**RPCs**) to perform work at a remote database.

When a program calls a remote procedure, the local server passes all procedure parameters to the remote server in the call. For example, the following PL/SQL program unit calls the packaged procedure `del_emp` located at the remote `sales` database and passes it the parameter 1257:

```
BEGIN
 emp_mgmt.del_emp@sales.us.americas.example_auto.com(1257);
END;
```

In order for the RPC to succeed, the called procedure must exist at the remote site, and the user being connected to must have the proper privileges to execute the procedure.

When developing packages and procedures for distributed database systems, developers must code with an understanding of what program units should do at remote locations, and how to return the results to a calling application.

30.5.3 Distributed Query Optimization

Distributed query optimization is an Oracle Database feature that reduces the amount of data transfer required between sites when a transaction retrieves data from remote tables referenced in a distributed SQL statement.

Distributed query optimization uses cost-based optimization to find or generate SQL expressions that extract only the necessary data from remote tables, process that data at a remote site or sometimes at the local site, and send the results to the local site for final processing. This operation reduces the amount of required data transfer when compared to the time it takes to transfer all the table data to the local site for processing.

Using various cost-based optimizer hints such as `DRIVING_SITE`, `NO_MERGE`, and `INDEX`, you can control where Oracle Database processes the data and how it accesses the data.

See Also:

"Using Cost-Based Optimization" for more information about cost-based optimization

30.6 Character Set Support for Distributed Environments

Different databases and clients can use different character sets in a distributed environment.

- About Character Set Support for Distributed Environments
  Oracle Database supports environments in which clients, Oracle Database servers, and non-Oracle Database servers use different character sets. `NCHAR` support is provided for heterogeneous environments.
- Client/Server Environment
  In a client/server environment, set the client character set to be the same as or a subset of the Oracle Database server character set.
- Homogeneous Distributed Environment
  In a non-heterogeneous environment, the client and server character sets should be either the same as or subsets of the main server character set.
- Heterogeneous Distributed Environment
  In a heterogeneous environment, the globalization support parameter settings of the client, the transparent gateway, and the non-Oracle Database data source should be either the same or a subset of the database server character set.

30.6.1 About Character Set Support for Distributed Environments

Oracle Database supports environments in which clients, Oracle Database servers, and non-Oracle Database servers use different character sets. `NCHAR` support is provided for heterogeneous environments.

You can set a variety of National Language Support (NLS) and Heterogeneous Services (HS) environment variables and initialization parameters to control data conversion between different character sets.

Character settings are defined by the following NLS and HS parameters:

Parameters	Environment	Defined For
`NLS_LANG` (environment variable)	Client/Server	Client

`NLS_LANGUAGE`
`NLS_CHARACTERSET`
`NLS_TERRITORY`
	Client/Server
	Not Heterogeneous Distributed
	Heterogeneous Distributed
		Oracle Database server

`HS_LANGUAGE`	Heterogeneous Distributed	Non-Oracle Database server Transparent gateway
`NLS_NCHAR` (environment variable) `HS_NLS_NCHAR`	Heterogeneous Distributed	Oracle Database server Transparent gateway

See Also:

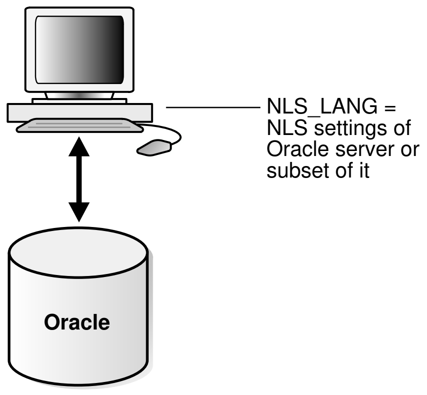
- *Oracle Database Globalization Support Guide* for information about NLS parameters
- *Oracle Database Heterogeneous Connectivity User's Guide* for information about HS parameters

30.6.2 Client/Server Environment

In a client/server environment, set the client character set to be the same as or a subset of the Oracle Database server character set.

Figure 30-6 illustrates a client/server environment.

Figure 30-6 NLS Parameter Settings in a Client/Server Environment

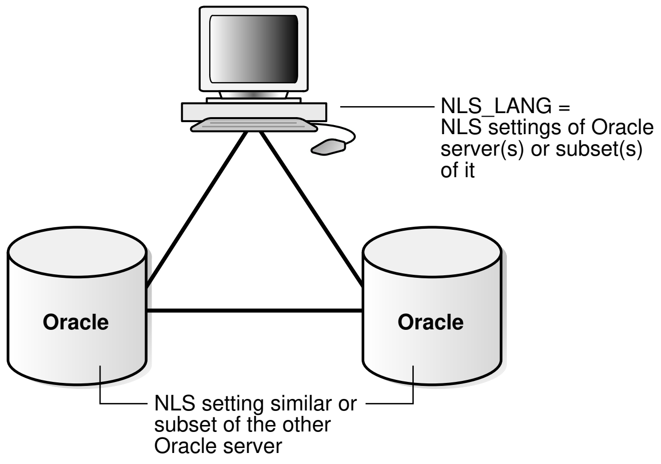


30.6.3 Homogeneous Distributed Environment

In a non-heterogeneous environment, the client and server character sets should be either the same as or subsets of the main server character set.

Figure 30-7 illustrates a homogeneous distributed environment:

Figure 30-7 NLS Parameter Settings in a Homogeneous Environment

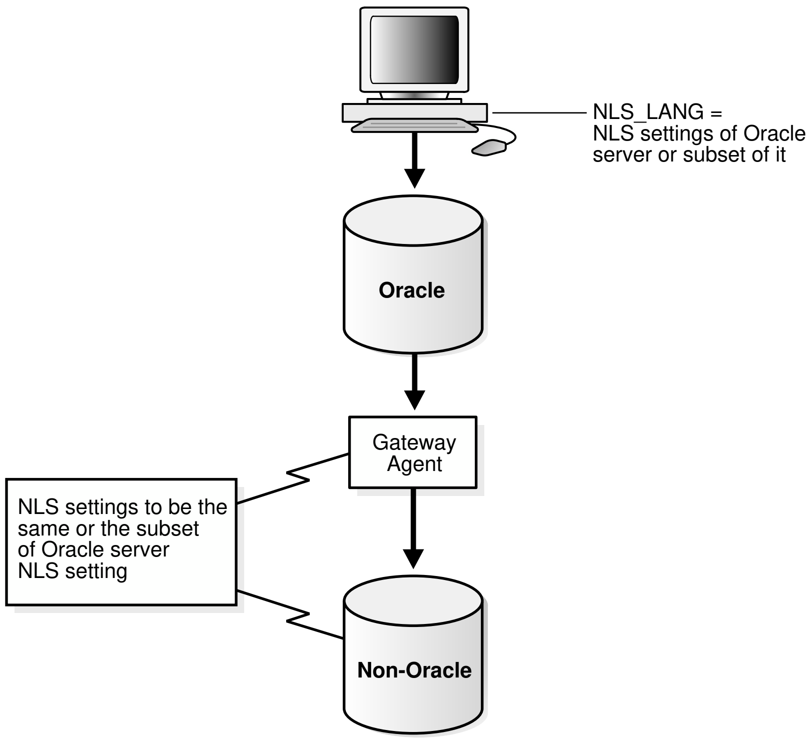


30.6.4 Heterogeneous Distributed Environment

In a heterogeneous environment, the globalization support parameter settings of the client, the transparent gateway, and the non-Oracle Database data source should be either the same or a subset of the database server character set.

Figure 30-8 illustrates a heterogeneous distributed environment. Transparent gateways have full globalization support.

Figure 30-8   NLS Parameter Settings in a Heterogeneous Environment



In a heterogeneous environment, only transparent gateways built with HS technology support complete `NCHAR` capabilities. Whether a specific transparent gateway supports `NCHAR` depends on the non-Oracle Database data source it is targeting. For information on how a particular transparent gateway handles `NCHAR` support, consult the system-specific transparent gateway documentation.

See Also:

Oracle Database Heterogeneous Connectivity User's Guide for more detailed information about Heterogeneous Services

31 Managing a Distributed Database

Managing a distributed database includes tasks such as managing global names, managing database links, and creating location and statement transparency.

- Managing Global Names in a Distributed System
  In a distributed database system, each database should have a unique **global database name**. Global database names uniquely identify a database in the system. A primary administration task in a distributed system is managing the creation and alteration of global database names.
- Creating Database Links
  To support application access to the data and schema objects throughout a distributed database system, you must create all necessary database links.
- Using Shared Database Links
  Every application that references a remote server using a standard database link establishes a connection between the local database and the remote database. Many users running applications simultaneously can cause a high number of connections between the local and remote databases. Shared database links enable you to limit the number of network connections required between the local server and the remote server.
- Managing Database Links
  Managing database links includes tasks such as closing them, dropping them, and limiting the number of active connections to them.
- Viewing Information About Database Links
  The data dictionary of each database stores the definitions of all the database links in the database. You can use data dictionary tables and views to gain information about the links.
- Creating Location Transparency
  After you have configured the necessary database links, you can use various tools to hide the distributed nature of the database system from users. In other words, users can access remote objects as if they were local objects.
- Managing Statement Transparency
  In a distributed database, some SQL statements can reference remote tables.
- Managing a Distributed Database: Examples
  Examples illustrate managing database links.

31.1 Managing Global Names in a Distributed System

In a distributed database system, each database should have a unique **global database name**. Global database names uniquely identify a database in the system. A primary administration task in a distributed system is managing the creation and alteration of global database names.

- Understanding How Global Database Names Are Formed
  A global database name is formed from two components: a database name and a domain.
- Determining Whether Global Naming Is Enforced
  The name that you give to a link on the local database depends on whether the local database enforces global naming.

- Viewing a Global Database Name
  Use the data dictionary view `GLOBAL_NAME` to view the database global name.
- Changing the Domain in a Global Database Name
  Use the `ALTER DATABASE` statement to change the domain in a database global name.
- Changing a Global Database Name: Scenario
  A scenario illustrates changing a global database name.

31.1.1 Understanding How Global Database Names Are Formed

A global database name is formed from two components: a database name and a domain.

The database name and the domain name are determined by the following initialization parameters at database creation:

Component	**Parameter**	**Requirements**	**Example**
Database name	`DB_NAME`	Must be 30 characters or less.	`sales`
Domain containing the database	`DB_DOMAIN`	Must follow standard Internet conventions. Levels in domain names must be separated by dots and the order of domain names is from leaf to root, left to right.	`us.example.com`

These are examples of valid global database names:

DB_NAME	**DB_DOMAIN**	**Global Database Name**
`sales`	`example.com`	`sales.example.com`
`sales`	`us.example.com`	`sales.us.example.com`
`mktg`	`us.example.com`	`mktg.us.example.com`
`payroll`	`example.org`	`payroll.example.org`

The `DB_DOMAIN` initialization parameter is only important at database creation time when it is used, together with the `DB_NAME` parameter, to form the database global name. At this point, the database global name is stored in the data dictionary. You must change the global name using an `ALTER DATABASE` statement, *not* by altering the `DB_DOMAIN` parameter in the initialization parameter file. It is good practice, however, to change the `DB_DOMAIN` parameter to reflect the change in the domain name before the next database startup.

31.1.2 Determining Whether Global Naming Is Enforced

The name that you give to a link on the local database depends on whether the local database enforces global naming.

If the local database enforces global naming, then you must use the remote database global database name as the name of the link. For example, if you are connected to the local `hq` server and want to create a link to the remote `mfg` database, and the local database enforces global naming, then you must use the `mfg` global database name as the link name.

You can also use service names as part of the database link name. For example, if you use the service names `sn1` and `sn2` to connect to database `hq.example.com`, and global naming is enforced, then you can create the following link names to `hq`:

- `HQ.EXAMPLE.COM@SN1`
- `HQ.EXAMPLE.COM@SN2`

> **See Also:**
>
> "Using Connection Qualifiers to Specify Service Names Within Link Names" for more information about using services names in link names

To determine whether global naming is enforced on a database, either examine the database initialization parameter file or query the `V$PARAMETER` view. For example, to see whether global naming is enforced on `mfg`, you could start a session on `mfg` and then create and execute the following `globalnames.sql` script (sample output included):

```
COL NAME FORMAT A12
COL VALUE FORMAT A6
SELECT NAME, VALUE FROM V$PARAMETER
   WHERE NAME = 'global_names'
/

SQL> @globalnames

NAME         VALUE
------------ ------
global_names FALSE
```

31.1.3 Viewing a Global Database Name

Use the data dictionary view `GLOBAL_NAME` to view the database global name.

For example, issue the following:

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
-------------------------------------------------------------------------------
SALES.EXAMPLE.COM
```

31.1.4 Changing the Domain in a Global Database Name

Use the `ALTER DATABASE` statement to change the domain in a database global name.

After the database is created, changing the initialization parameter `DB_DOMAIN` has no effect on the global database name or on the resolution of database link names.

The following example shows the syntax for the renaming statement, where *database* is a database name and *domain* is the network domain:

```
ALTER DATABASE RENAME GLOBAL_NAME TO database.domain;
```

Use the following procedure to change the domain in a global database name:

1. Determine the current global database name. For example, issue:

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
----------------------------------------------------------------------------
SALES.EXAMPLE.COM
```

2. Rename the global database name using an `ALTER DATABASE` statement. For example, enter:

   ```
   ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.example.com;
   ```

3. Query the `GLOBAL_NAME` table to check the new name. For example, enter:

   ```
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   ----------------------------------------------------------------------------
   SALES.US.EXAMPLE.COM
   ```

31.1.5 Changing a Global Database Name: Scenario

A scenario illustrates changing a global database name.

In this scenario, you change the domain part of the global database name of the local database. You also create database links using partially specified global names to test how Oracle Database resolves the names. You discover that the database resolves the partial names using the domain part of the current global database name of the local database, not the value for the initialization parameter `DB_DOMAIN`.

1. You connect to `SALES.US.EXAMPLE.COM` and query the `GLOBAL_NAME` data dictionary view to determine the current database global name:

   ```
   CONNECT SYSTEM@sales.us.example.com
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   ----------------------------------------------------------------------------
   SALES.US.EXAMPLE.COM
   ```

2. You query the `V$PARAMETER` view to determine the current setting for the `DB_DOMAIN` initialization parameter:

   ```
   SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'db_domain';

   NAME       VALUE
   ---------  -----------
   db_domain  US.EXAMPLE.COM
   ```

3. You then create a database link to a database called `hq`, using only a partially-specified global name:

   ```
   CREATE DATABASE LINK hq USING 'sales';
   ```

   The database expands the global database name for this link by appending the domain part of the global database name of the *local* database to the name of the database specified in the link.

4. You query `USER_DB_LINKS` to determine which domain name the database uses to resolve the partially specified global database name:

   ```
   SELECT DB_LINK FROM USER_DB_LINKS;

   DB_LINK
   ```

```
------------------
HQ.US.EXAMPLE.COM
```

This result indicates that the domain part of the global database name of the local database is `us.example.com`. The database uses this domain in resolving partial database link names when the database link is created.

5. Because you have received word that the `sales` database will move to Japan, you rename the `sales` database to `sales.jp.example.com`:

   ```
   ALTER DATABASE RENAME GLOBAL_NAME TO sales.jp.example.com;
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   ----------------------------------------------------------------------------
   SALES.JP.EXAMPLE.COM
   ```

6. You query `V$PARAMETER` again and discover that the value of `DB_DOMAIN` is *not* changed, although you renamed the domain part of the global database name:

   ```
   SELECT NAME, VALUE FROM V$PARAMETER
     WHERE NAME = 'db_domain';

   NAME       VALUE
   ---------  -----------
   db_domain  US.EXAMPLE.COM
   ```

   This result indicates that the value of the `DB_DOMAIN` initialization parameter is independent of the `ALTER DATABASE RENAME GLOBAL_NAME` statement. The `ALTER DATABASE` statement determines the domain of the global database name, not the `DB_DOMAIN` initialization parameter (although it is good practice to alter `DB_DOMAIN` to reflect the new domain name).

7. You create another database link to database `supply`, and then query `USER_DB_LINKS` to see how the database resolves the domain part of the global database name of `supply`:

   ```
   CREATE DATABASE LINK supply USING 'supply';
   SELECT DB_LINK FROM USER_DB_LINKS;

   DB_LINK
   ------------------
   HQ.US.EXAMPLE.COM
   SUPPLY.JP.EXAMPLE.COM
   ```

   This result indicates that the database resolves the partially specified link name by using the domain `jp.example.com`. This domain is used when the link is created because it is the domain part of the global database name of the local database. The database does *not* use the `DB_DOMAIN` initialization parameter setting when resolving the partial link name.

8. You then receive word that your previous information was faulty: `sales` will be in the `ASIA.JP.EXAMPLE.COM` domain, not the `JP.EXAMPLE.COM` domain. Consequently, you rename the global database name as follows:

   ```
   ALTER DATABASE RENAME GLOBAL_NAME TO sales.asia.jp.example.com;
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   ----------------------------------------------------------------------------
   SALES.ASIA.JP.EXAMPLE.COM
   ```

9. You query `V$PARAMETER` to again check the setting for the parameter `DB_DOMAIN`:

   ```
   SELECT NAME, VALUE FROM V$PARAMETER
     WHERE NAME = 'db_domain';
   ```

```
NAME        VALUE
----------  -----------
db_domain   US.EXAMPLE.COM
```

The result indicates that the domain setting in the parameter file is the same as it was before you issued *either* of the `ALTER DATABASE RENAME` statements.

10. Finally, you create a link to the `warehouse` database and again query `USER_DB_LINKS` to determine how the database resolves the partially-specified global name:

    ```
    CREATE DATABASE LINK warehouse USING 'warehouse';
    SELECT DB_LINK FROM USER_DB_LINKS;

    DB_LINK
    ------------------
    HQ.US.EXAMPLE.COM
    SUPPLY.JP.EXAMPLE.COM
    WAREHOUSE.ASIA.JP.EXAMPLE.COM
    ```

    Again, you see that the database uses the domain part of the global database name of the local database to expand the partial link name during link creation.

Note:

In order to correct the `supply` database link, it must be dropped and re-created.

See Also:

- *Oracle Database Reference* for more information about specifying the `DB_NAME` initialization parameter
- *Oracle Database Reference* for more information about specifying the `DB_DOMAIN` initialization parameter

31.2 Creating Database Links

To support application access to the data and schema objects throughout a distributed database system, you must create all necessary database links.

- Obtaining Privileges Necessary for Creating Database Links
  A database link is a pointer in the local database that lets you access objects on a remote database. To create a private database link, you must have been granted the proper privileges.
- Specifying Link Types
  When you create a database link, you must decide who will have access to it.
- Specifying Link Users
  A database link defines a communication path from one database to another. When an application uses a database link to access a remote database, Oracle Database establishes a database session in the remote database on behalf of the local application request. When you create a private or public database link, you can determine which

schema on the remote database the link will establish connections to by creating fixed user, current user, and connected user database links.

- Using Connection Qualifiers to Specify Service Names Within Link Names
  In some situations, you may want to have several database links of the same type (for example, public) that point to the same remote database, yet establish connections to the remote database using different communication pathways.

31.2.1 Obtaining Privileges Necessary for Creating Database Links

A database link is a pointer in the local database that lets you access objects on a remote database. To create a private database link, you must have been granted the proper privileges.

The following table illustrates which privileges are required on which database for which type of link:

Privilege	**Database**	**Required For**
`CREATE DATABASE LINK`	Local	Creation of a private database link.
`CREATE PUBLIC DATABASE LINK`	Local	Creation of a public database link.
`CREATE SESSION`	Remote	Creation of any type of database link.

To see which privileges you currently have available, query `ROLE_SYS_PRIVS`. For example, you could create and execute the following `privs.sql` script (sample output included):

```
SELECT DISTINCT PRIVILEGE AS "Database Link Privileges"
FROM ROLE_SYS_PRIVS
WHERE PRIVILEGE IN ( 'CREATE SESSION','CREATE DATABASE LINK',
                     'CREATE PUBLIC DATABASE LINK')
/

SQL> @privs

Database Link Privileges
----------------------------------------
CREATE DATABASE LINK
CREATE PUBLIC DATABASE LINK
CREATE SESSION
```

31.2.2 Specifying Link Types

When you create a database link, you must decide who will have access to it.

- Creating Private Database Links
  Use the `CREATE DATABASE LINK` statement to create private database links.
- Creating Public Database Links
  Use the `CREATE PUBLIC DATABASE LINK` statement to create public database links.
- Creating Global Database Links
  You can use a directory server in which databases are identified by net service names. In this document, these are what are referred to as global database links.

31.2.2.1 Creating Private Database Links

Use the `CREATE DATABASE LINK` statement to create private database links.

To create a private database link, specify the following (where *link_name* is the global database name or an arbitrary link name):

```
CREATE DATABASE LINK link_name ...;
```

Following are examples of private database links:

SQL Statement	Result
`CREATE DATABASE LINK supply.us.example.com;`	A private link using the global database name to the remote `supply` database. The link uses the userid/password of the connected user. So if `scott` (identified by *`password`*) uses the link in a query, the link establishes a connection to the remote database as `scott/`*`password`*.
`CREATE DATABASE LINK link_2 CONNECT TO jane IDENTIFIED BY` *`password`* `USING 'us_supply';`	A private fixed user link called `link_2` to the database with service name `us_supply`. The link connects to the remote database with the userid/password of `jane/`*`password`* regardless of the connected user.
`CREATE DATABASE LINK link_1 CONNECT TO CURRENT_USER USING 'us_supply';`	A private link called `link_1` to the database with service name `us_supply`. The link uses the userid/password of the current user to log onto the remote database. **Note:** The current user may not be the same as the connected user, and must be a global user on both databases involved in the link (see "Users of Database Links"). Current user links are part of the Oracle Advanced Security option.

See Also:

Oracle Database SQL Language Reference for `CREATE DATABASE LINK` syntax

31.2.2.2 Creating Public Database Links

Use the `CREATE PUBLIC DATABASE LINK` statement to create public database links.

To create a public database link, use the keyword `PUBLIC` (where *link_name* is the global database name or an arbitrary link name):

```
CREATE PUBLIC DATABASE LINK link_name ...;
```

Following are examples of public database links:

SQL Statement	Result
`CREATE PUBLIC DATABASE LINK supply.us.example.com;`	A public link to the remote `supply` database. The link uses the userid/password of the connected user. So if `scott` (identified by *`password`*) uses the link in a query, the link establishes a connection to the remote database as `scott/`*`password`*.
`CREATE PUBLIC DATABASE LINK pu_link CONNECT TO CURRENT_USER USING 'supply';`	A public link called `pu_link` to the database with service name `supply`. The link uses the userid/password of the current user to log onto the remote database. **Note:** The current user may not be the same as the connected user, and must be a global user on both databases involved in the link (see "Users of Database Links").
`CREATE PUBLIC DATABASE LINK sales.us.example.com CONNECT TO jane IDENTIFIED BY` *`password`*`;`	A public fixed user link to the remote `sales` database. The link connects to the remote database with the userid/password of `jane/`*`password`*.

See Also:

Oracle Database SQL Language Reference for `CREATE PUBLIC DATABASE LINK` syntax

31.2.2.3 Creating Global Database Links

You can use a directory server in which databases are identified by net service names. In this document, these are what are referred to as global database links.

See the *Oracle Database Net Services Administrator's Guide* to learn how to create directory entries that act as global database links.

31.2.3 Specifying Link Users

A database link defines a communication path from one database to another. When an application uses a database link to access a remote database, Oracle Database establishes a database session in the remote database on behalf of the local application request. When you create a private or public database link, you can determine which schema on the remote database the link will establish connections to by creating fixed user, current user, and connected user database links.

- Creating Fixed User Database Links
  When an application uses a fixed user database link, the local server always establishes a connection to a fixed remote schema in the remote database. The local server also sends the fixed user's credentials across the network when an application uses the link to access the remote database.
- Creating Connected User and Current User Database Links
  Connected user and current user database links do not include credentials in the definition of the link. The credentials used to connect to the remote database can change depending on the user that references the database link and the operation performed by the application.

31.2.3.1 Creating Fixed User Database Links

When an application uses a fixed user database link, the local server always establishes a connection to a fixed remote schema in the remote database. The local server also sends the fixed user's credentials across the network when an application uses the link to access the remote database.

To create a **fixed user database link**, you embed the credentials (in this case, a username and password) required to access the remote database in the definition of the link:

```
CREATE DATABASE LINK ... CONNECT TO username IDENTIFIED BY password ...;
```

Following are examples of fixed user database links:

SQL Statement	Result
`CREATE PUBLIC DATABASE LINK supply.us.example.com CONNECT TO scott IDENTIFIED BY` *`password`*`;`	A public link using the global database name to the remote `supply` database. The link connects to the remote database with the userid/password `scott`/*`password`*.
`CREATE DATABASE LINK foo CONNECT TO jane IDENTIFIED BY` *`password`* `USING 'finance';`	A private fixed user link called `foo` to the database with service name `finance`. The link connects to the remote database with the userid/password `jane`/*`password`*.

31.2.3.2 Creating Connected User and Current User Database Links

Connected user and current user database links do not include credentials in the definition of the link. The credentials used to connect to the remote database can change depending on the user that references the database link and the operation performed by the application.

> **Note:**
>
> For many distributed applications, you do not want a user to have privileges in a remote database. One simple way to achieve this result is to create a procedure that contains a fixed user or current user database link within it. In this way, the user accessing the procedure temporarily assumes someone else's privileges.

- Creating a Connected User Database Link
  To create a connected user database link, omit the `CONNECT TO` clause in the `CREATE DATABASE LINK` statement.
- Creating a Current User Database Link
  To create a current user database link, use the `CONNECT TO CURRENT_USER` clause in the `CREATE DATABASE LINK` statement.

Related Topics

- Users of Database Links
  Users of database links include connect user, current user, and fixed user.

31.2.3.2.1 Creating a Connected User Database Link

To create a connected user database link, omit the `CONNECT TO` clause in the `CREATE DATABASE LINK` statement.

The following syntax creates a connected user database link, where *dblink* is the name of the link and *net_service_name* is an optional connect string:

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink ... [USING 'net_service_name'];
```

For example, to create a connected user database link, use the following syntax:

```
CREATE DATABASE LINK sales.division3.example.com USING 'sales';
```

31.2.3.2.2 Creating a Current User Database Link

To create a current user database link, use the `CONNECT TO CURRENT_USER` clause in the `CREATE DATABASE LINK` statement.

Current user links are only available through the Oracle Advanced Security option.

The following syntax creates a current user database link, where *dblink* is the name of the link and *net_service_name* is an optional connect string:

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink CONNECT TO CURRENT_USER
[USING 'net_service_name'];
```

For example, to create a connected user database link to the `sales` database, you might use the following syntax:

```
CREATE DATABASE LINK sales CONNECT TO CURRENT_USER USING 'sales';
```

Note:

To use a current user database link, the current user must be a global user on both databases involved in the link.

See Also:

Oracle Database SQL Language Reference for more syntax information about creating database links

31.2.4 Using Connection Qualifiers to Specify Service Names Within Link Names

In some situations, you may want to have several database links of the same type (for example, public) that point to the same remote database, yet establish connections to the remote database using different communication pathways.

Some cases in which this strategy is useful are:

- A remote database is part of an Oracle Real Application Clusters configuration, so you define several public database links at your local node so that connections can be established to specific instances of the remote database.
- Some clients connect to the Oracle Database server using TCP/IP while others use DECNET.

To facilitate such functionality, the database lets you create a database link with an optional service name in the database link name. When creating a database link, a service name is specified as the trailing portion of the database link name, separated by an `@` sign, as in `@sales`. This string is called a **connection qualifier**.

For example, assume that remote database `hq.example.com` is managed in an Oracle Real Application Clusters environment. The `hq` database has two instances named `hq_1` and `hq_2`. The local database can contain the following public database links to define pathways to the remote instances of the `hq` database:

```
CREATE PUBLIC DATABASE LINK hq.example.com@hq_1
  USING 'string_to_hq_1';
CREATE PUBLIC DATABASE LINK hq.example.com@hq_2
  USING 'string_to_hq_2';
CREATE PUBLIC DATABASE LINK hq.example.com
  USING 'string_to_hq';
```

Notice in the first two examples that a service name is simply a part of the database link name. The text of the service name does not necessarily indicate how a connection is to be established; this information is specified in the service name of the `USING` clause. Also notice that in the third example, a service name is not specified as part of the link name. In this case, just as when a service name is specified as part of the link name, the instance is determined by the `USING` string.

To use a service name to specify a particular instance, include the service name at the end of the global object name:

```
SELECT * FROM scott.emp@hq.example.com@hq_1
```

Note that in this example, there are two @ symbols.

31.3 Using Shared Database Links

Every application that references a remote server using a standard database link establishes a connection between the local database and the remote database. Many users running applications simultaneously can cause a high number of connections between the local and remote databases. Shared database links enable you to limit the number of network connections required between the local server and the remote server.

- Determining Whether to Use Shared Database Links
  Look carefully at your application and shared server configuration to determine whether to use shared links. A simple guideline is to use shared database links when the number of users accessing a database link is expected to be much larger than the number of server processes in the local database.
- Creating Shared Database Links
  To create a shared database link, use the keyword `SHARED` in the `CREATE DATABASE LINK` statement.
- Configuring Shared Database Links
  You can configure shared database links in different ways.

See Also:

"What Are Shared Database Links?" for a conceptual overview of shared database links

31.3.1 Determining Whether to Use Shared Database Links

Look carefully at your application and shared server configuration to determine whether to use shared links. A simple guideline is to use shared database links when the number of users accessing a database link is expected to be much larger than the number of server processes in the local database.

The following table illustrates three possible configurations involving database links:

Link Type	Server Mode	Consequences
Nonshared	Dedicated/shared server	If your application uses a standard public database link, and 100 users simultaneously require a connection, then 100 direct network connections to the remote database are required.
Shared	Shared server	If 10 shared server processes exist in the local shared server mode database, then 100 users that use the same database link require 10 or fewer network connections to the remote server. Each local shared server process may only need one connection to the remote server.
Shared	Dedicated	If 10 clients connect to a local dedicated server, and each client has 10 sessions on the same connection (thus establishing 100 sessions overall), and each session references the same remote database, then only 10 connections are needed. With a nonshared database link, 100 connections are needed.

Shared database links are not useful in all situations. Assume that only one user accesses the remote server. If this user defines a shared database link and 10 shared server processes exist in the local database, then this user can require up to 10 network connections to the remote server. Because the user can use each shared server process, each process can establish a connection to the remote server.

Clearly, a nonshared database link is preferable in this situation because it requires only one network connection. Shared database links lead to more network connections in single-user scenarios, so use shared links only when many users need to use the same link. Typically, shared links are used for public database links, but can also be used for private database links when many clients access the same local schema (and therefore the same private database link).

Note:

In a multitiered environment, there is a restriction that if you use a shared database link to connect to a remote database, then that remote database cannot link to another database with a database link that cannot be migrated. That link must use a shared server, or it must be another shared database link.

31.3.2 Creating Shared Database Links

To create a shared database link, use the keyword SHARED in the CREATE DATABASE LINK statement.

Use the following syntax to create a shared database link:

```
CREATE SHARED DATABASE LINK dblink_name
[CONNECT TO username IDENTIFIED BY password]|[CONNECT TO CURRENT_USER]
AUTHENTICATED BY schema_name IDENTIFIED BY password
[USING 'service_name'];
```

Whenever you use the keyword SHARED, the clause AUTHENTICATED BY is required. The schema specified in the AUTHENTICATED BY clause must exist in the remote database and must be granted at least the CREATE SESSION privilege. The credentials of this schema can be considered the authentication method between the local database and the remote database. These credentials are required to protect the remote shared server processes from clients that masquerade as a database link user and attempt to gain unauthorized access to information.

After a connection is made with a shared database link, operations on the remote database proceed with the privileges of the CONNECT TO user or CURRENT_USER, not the AUTHENTICATED BY schema.

The following example creates a fixed user, shared link to database sales, connecting as scott and authenticated as linkuser:

```
CREATE SHARED DATABASE LINK link2sales
CONNECT TO scott IDENTIFIED BY password
AUTHENTICATED BY linkuser IDENTIFIED BY ostrich
USING 'sales';
```

See Also:

Oracle Database SQL Language Reference for information about the CREATE DATABASE LINK statement

31.3.3 Configuring Shared Database Links

You can configure shared database links in different ways.

- Creating Shared Links to Dedicated Servers
  A shared server process in the local server can own a dedicated remote server process.
- Creating Shared Links to Shared Servers
  You can create shared links using shared server processes on the remote server.

31.3.3.1 Creating Shared Links to Dedicated Servers

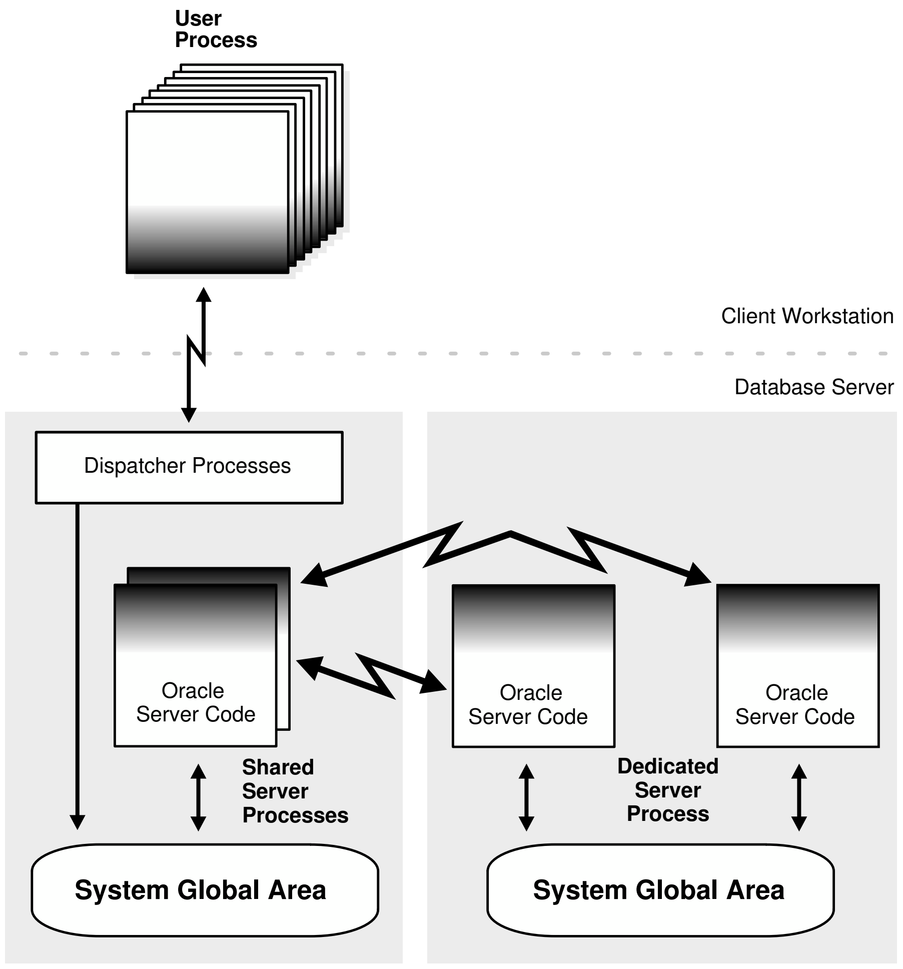
A shared server process in the local server can own a dedicated remote server process.

The advantage is that a direct network transport exists between the local shared server and the remote dedicated server. A disadvantage is that extra back-end server processes are needed.

Note:

The remote server can either be a shared server or dedicated server. There is a dedicated connection between the local and remote servers. When the remote server is a shared server, you can force a dedicated server connection by using the (`SERVER=DEDICATED`) clause in the definition of the service name.

Figure 31-1 A Shared Database Link to Dedicated Server Processes

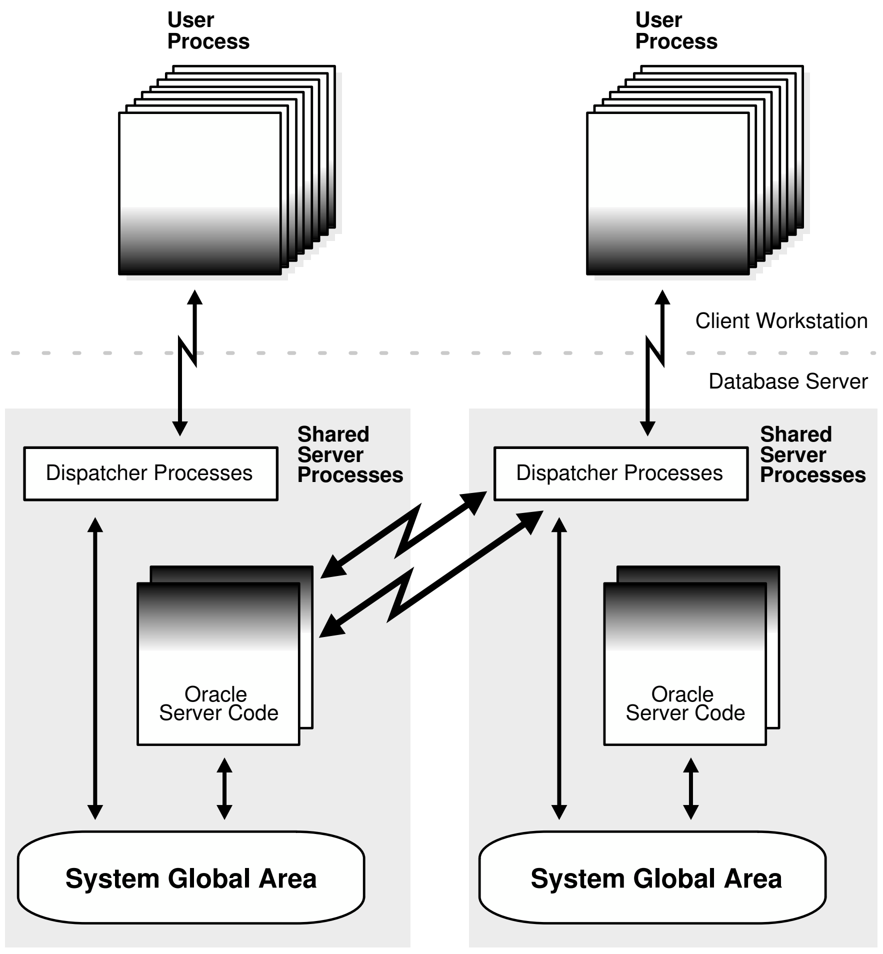


31.3.3.2 Creating Shared Links to Shared Servers

You can create shared links using shared server processes on the remote server.

This configuration eliminates the need for more dedicated servers, but requires the connection to go through the dispatcher on the remote server. Note that both the local and the remote server must be configured as shared servers.

Figure 31-2   Shared Database Link to Shared Server



See Also:

"Shared Server Processes" for information about the shared server option

31.4 Managing Database Links

Managing database links includes tasks such as closing them, dropping them, and limiting the number of active connections to them.

- Closing Database Links
  If you access a database link in a session, then the link remains open until you close the session. To close a database link manually, use the `ALTER SESSION CLOSE DATABASE LINK` statement.
- Dropping Database Links
  You can drop a database link just as you can drop a table or view. If the link is private, then it must be in your schema. If the link is public, then you must have the `DROP PUBLIC DATABASE LINK` system privilege.

- Limiting the Number of Active Database Link Connections
  You can limit the number of connections from a user process to remote databases using the static initialization parameter `OPEN_LINKS`.

31.4.1 Closing Database Links

If you access a database link in a session, then the link remains open until you close the session. To close a database link manually, use the `ALTER SESSION CLOSE DATABASE LINK` statement.

A link is open in the sense that a process is active on each of the remote databases accessed through the link. This situation has the following consequences:

- If 20 users open sessions and access the same public link in a local database, then 20 database link connections are open.
- If 20 users open sessions and each user accesses a private link, then 20 database link connections are open.
- If one user starts a session and accesses 20 different links, then 20 database link connections are open.

After you close a session, the links that were active in the session are automatically closed. You may have occasion to close the link manually. For example, close links when:

- The network connection established by a link is used infrequently in an application.
- The user session must be terminated.

To close a link, issue the following statement, where *linkname* refers to the name of the link:

```
ALTER SESSION CLOSE DATABASE LINK linkname;
```

Note that this statement only closes the links that are active in your current session.

31.4.2 Dropping Database Links

You can drop a database link just as you can drop a table or view. If the link is private, then it must be in your schema. If the link is public, then you must have the `DROP PUBLIC DATABASE LINK` system privilege.

The statement syntax is as follows, where *dblink* is the name of the link:

```
DROP [PUBLIC] DATABASE LINK dblink;
```

- Dropping a Private Database Link
  Use the `DROP DATABASE LINK` statement to drop a private database link.
- Dropping a Public Database Link
  Use the `DROP PUBLIC DATABASE LINK` statement to drop a public database link.

31.4.2.1 Dropping a Private Database Link

Use the `DROP DATABASE LINK` statement to drop a private database link.

1. Connect to the local database using SQL*Plus. For example, enter:

   ```
   CONNECT scott@local_db
   ```

2. Query `USER_DB_LINKS` to view the links that you own. For example, enter:

```
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
-----------------------------------
SALES.US.EXAMPLE.COM
MKTG.US.EXAMPLE.COM
2 rows selected.
```

3. Drop the desired link using the `DROP DATABASE LINK` statement. For example, enter:

   ```
   DROP DATABASE LINK sales.us.example.com;
   ```

31.4.2.2 Dropping a Public Database Link

Use the `DROP PUBLIC DATABASE LINK` statement to drop a public database link.

1. Connect to the local database as a user with the `DROP PUBLIC DATABASE LINK` privilege. For example, enter:

   ```
   CONNECT SYSTEM@local_db AS SYSDBA
   ```

2. Query `DBA_DB_LINKS` to view the public links. For example, enter:

   ```
   SELECT DB_LINK FROM DBA_DB_LINKS
     WHERE OWNER = 'PUBLIC';

   DB_LINK
   -----------------------------------
   DBL1.US.EXAMPLE.COM
   SALES.US.EXAMPLE.COM
   INST2.US.EXAMPLE.COM
   RMAN2.US.EXAMPLE.COM
   4 rows selected.
   ```

3. Drop the desired link using the `DROP PUBLIC DATABASE LINK` statement. For example, enter:

   ```
   DROP PUBLIC DATABASE LINK sales.us.example.com;
   ```

31.4.3 Limiting the Number of Active Database Link Connections

You can limit the number of connections from a user process to remote databases using the static initialization parameter `OPEN_LINKS`.

This parameter controls the number of remote connections that a single user session can use concurrently in distributed transactions.

Note the following considerations for setting this parameter:

- The value should be greater than or equal to the number of databases referred to in a single SQL statement that references multiple databases.
- Increase the value if several distributed databases are accessed over time. Thus, if you regularly access three databases, set `OPEN_LINKS` to 3 or greater.
- The default value for `OPEN_LINKS` is 4. If `OPEN_LINKS` is set to 0, then no distributed transactions are allowed.

See Also:

Oracle Database Reference for more information about the `OPEN_LINKS` initialization parameter

31.5 Viewing Information About Database Links

The data dictionary of each database stores the definitions of all the database links in the database. You can use data dictionary tables and views to gain information about the links.

- Determining Which Links Are in the Database
  A set of views shows the database links that have been defined at the local database and stored in the data dictionary.
- Determining Which Link Connections Are Open
  You may find it useful to determine which database link connections are currently open in your session.
- Determining the Host of Outgoing Database Links
  You can use the `RESOLVE_TNSNAME` function in the `DBMS_TNS` package to determine the host name of an outgoing database link.
- Determining Information About Incoming Database Links
  You can use the `DBA_DB_LINK_SOURCES` view to determine information about incoming database links.
- Determining the Source of High SCN Activity for Incoming Database Links
  You can use the following views to determine the source of high system change number (SCN) activity for incoming database links: `DBA_EXTERNAL_SCN_ACTIVITY`, `DBA_DB_LINK_SOURCES`, and `DBA_DB_LINK`.

31.5.1 Determining Which Links Are in the Database

A set of views shows the database links that have been defined at the local database and stored in the data dictionary.

:

View	Purpose
`DBA_DB_LINKS`	Lists all database links in the database.
`ALL_DB_LINKS`	Lists all database links accessible to the connected user.
`USER_DB_LINKS`	Lists all database links owned by the connected user.

These data dictionary views contain the same basic information about database links, with some exceptions:

Column	Which Views?	Description
`OWNER`	All except `USER_*`	The user who created the database link. If the link is public, then the user is listed as `PUBLIC`.
`DB_LINK`	All	The name of the database link.

Column	Which Views?	Description
USERNAME	All	If the link definition includes a fixed user, then this column displays the username of the fixed user. If there is no fixed user, the column is NULL.
PASSWORD	Only USER_*	Not used. Maintained for backward compatibility only.
HOST	All	The net service name used to connect to the remote database.
CREATED	All	Creation time of the database link.

Any user can query USER_DB_LINKS to determine which database links are available to that user. Only those with additional privileges can use the ALL_DB_LINKS or DBA_DB_LINKS view.

The following script queries the DBA_DB_LINKS view to access link information:

```
COL OWNER FORMAT a10
COL USERNAME FORMAT A8 HEADING "USER"
COL DB_LINK FORMAT A30
COL HOST FORMAT A7 HEADING "SERVICE"
SELECT * FROM DBA_DB_LINKS
/
```

Here, the script is invoked and the resulting output is shown:

```
SQL>@link_script

OWNER      DB_LINK                        USER     SERVICE CREATED
---------- ------------------------------ -------- ------- ----------
SYS        TARGET.US.EXAMPLE.COM             SYS      inst1   23-JUN-99
PUBLIC     DBL1.UK.EXAMPLE.COM               BLAKE    ora51   23-JUN-99
PUBLIC     RMAN2.US.EXAMPLE.COM                       inst2   23-JUN-99
PUBLIC     DEPT.US.EXAMPLE.COM                        inst2   23-JUN-99
JANE       DBL.UK.EXAMPLE.COM                BLAKE    ora51   23-JUN-99
SCOTT      EMP.US.EXAMPLE.COM                SCOTT    inst2   23-JUN-99
6 rows selected.
```

31.5.2 Determining Which Link Connections Are Open

You may find it useful to determine which database link connections are currently open in your session.

Note that if you connect as SYSDBA, you cannot query a view to determine all the links open for all sessions; you can only access the link information in the session within which you are working.

The following views show the database link connections that are currently open in your current session:

View	Purpose
V$DBLINK	Lists all open database links in your session, that is, all database links with the IN_TRANSACTION column set to YES.
GV$DBLINK	Lists all open database links in your session along with their corresponding instances. This view is useful in an Oracle Real Application Clusters configuration.

These data dictionary views contain the same basic information about database links, with one exception:

Column	Which Views?	Description
DB_LINK	All	The name of the database link.
OWNER_ID	All	The owner of the database link.
LOGGED_ON	All	Whether the database link is currently logged on.
HETEROGENEOUS	All	Whether the database link is homogeneous (NO) or heterogeneous (YES).
PROTOCOL	All	The communication protocol for the database link.
OPEN_CURSORS	All	Whether cursors are open for the database link.
IN_TRANSACTION	All	Whether the database link is accessed in a transaction that has not yet been committed or rolled back.
UPDATE_SENT	All	Whether there was an update on the database link.
COMMIT_POINT_STRENGTH	All	The commit point strength of the transactions using the database link.
INST_ID	GV$DBLINK only	The instance from which the view information was obtained.

For example, you can create and execute the script below to determine which links are open (sample output included):

```
COL DB_LINK FORMAT A25
COL OWNER_ID FORMAT 99999 HEADING "OWNID"
COL LOGGED_ON FORMAT A5 HEADING "LOGON"
COL HETEROGENEOUS FORMAT A5 HEADING "HETER"
COL PROTOCOL FORMAT A8
COL OPEN_CURSORS FORMAT 999 HEADING "OPN_CUR"
COL IN_TRANSACTION FORMAT A3 HEADING "TXN"
COL UPDATE_SENT FORMAT A6 HEADING "UPDATE"
COL COMMIT_POINT_STRENGTH FORMAT 99999 HEADING "C_P_S"

SELECT * FROM V$DBLINK
/

SQL> @dblink

DB_LINK                   OWNID LOGON HETER PROTOCOL OPN_CUR TXN UPDATE  C_P_S
------------------------- ------ ----- ----- -------- -------- --- ------ ------
INST2.EXAMPLE.COM                0 YES   YES    UNKN           0 YES YES       255
```

31.5.3 Determining the Host of Outgoing Database Links

You can use the RESOLVE_TNSNAME function in the DBMS_TNS package to determine the host name of an outgoing database link.

For outgoing database links, the HOST column value in the DBA_DB_LINKS view does not resolve the connect data when the database link is created with a connect identifier, such as the network service name. As a result, a tool that requires any part of the connect data, such as the host name, must check the tnsnames.ora file to find the information. However, the tnsnames.ora file might not be accessible to the tool.

1. Connect to the database that contains the outgoing database link as a user who can run subprograms in the `DBMS_TNS` package.
2. Query the `DBA_DB_LINKS` view.

Example 31-1 Determining the Host of Outgoing Database Links

This query results show the connect data, including the host name, for each outgoing database link.

```
SELECT DB_LINK, DBMS_TNS.RESOLVE_TNSNAME(HOST) FROM DBA_DB_LINKS;
```

See Also:

- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `RESOLVE_TNSNAME` function

31.5.4 Determining Information About Incoming Database Links

You can use the `DBA_DB_LINK_SOURCES` view to determine information about incoming database links.

The database records details about unique connections for incoming database links in a persistent table and the `DBA_DB_LINK_SOURCES` view. You can query this view for information about the incoming connections on database links.

1. Connect to the database as a user who can query the `DBA_DB_LINK_SOURCES` view.
2. Query the `DBA_DB_LINK_SOURCES` view.

Example 31-2 Querying the DBA_DB_LINK_SOURCES View

This example returns the database name and host name of each incoming database link. It also returns the time of the first login and last login to the current database with the database link.

```
SELECT DB_NAME, HOST_NAME, FIRST_LOGON_TIME, LAST_LOGON_TIME
   FROM DBA_DB_LINK_SOURCES;
```

See Also:

Oracle Database Reference

31.5.5 Determining the Source of High SCN Activity for Incoming Database Links

You can use the following views to determine the source of high system change number (SCN) activity for incoming database links: `DBA_EXTERNAL_SCN_ACTIVITY`, `DBA_DB_LINK_SOURCES`, and `DBA_DB_LINK`.

To implement distributed transactions and distributed read consistency in a distributed database environment, databases synchronize SCNs when calls are made over database links. A high SCN increase rate can cause a database to return errors.

It is often hard to determine the source of unusual increases of SCNs that result from distributed database operations. The `DBA_EXTERNAL_SCN_ACTIVITY` view enables you to determine which databases or clients are causing excessive SCN increases. You join this view in a query with the `DBA_DB_LINK_SOURCES` and `DBA_DB_LINK` views to return the information.

1. Connect to the database as a user who can query the `DBA_EXTERNAL_SCN_ACTIVITY`, `DBA_DB_LINK_SOURCES`, and `DBA_DB_LINK` views.
2. Run the following query to show the recent history of SCN increments and their sources:

```
(SELECT RESULT, OPERATION_TIMESTAMP, EXTERNAL_SCN, SCN_ADJUSTMENT,
HOST_NAME, DB_NAME, SESSION_ID, SESSION_SERIAL#
   FROM DBA_EXTERNAL_SCN_ACTIVITY a, DBA_DB_LINK_SOURCES s
   WHERE a.INBOUND_DB_LINK_SOURCE_ID = s.SOURCE_ID)
UNION
(SELECT RESULT, OPERATION_TIMESTAMP, EXTERNAL_SCN, SCN_ADJUSTMENT,
dbms_tns.resolve_tnsname(HOST) HOST_NAME, NULL DB_NAME, SESSION_ID,
SESSION_SERIAL#
   FROM DBA_EXTERNAL_SCN_ACTIVITY a, DBA_DB_LINKS o
   WHERE a.OUTBOUND_DB_LINK_NAME = o.DB_LINK AND
         a.OUTBOUND_DB_LINK_OWNER = o.OWNER)
UNION
(SELECT RESULT, OPERATION_TIMESTAMP, EXTERNAL_SCN,   SCN_ADJUSTMENT,
s.MACHINE HOST_NAME, NULL DB_NAME, SESSION_ID, SESSION_SERIAL#
   FROM DBA_EXTERNAL_SCN_ACTIVITY a, V$SESSION s
   WHERE a.SESSION_ID = s.SID AND
         a.SESSION_SERIAL#=s.SERIAL# AND
         INBOUND_DB_LINK_SOURCE_ID IS NULL AND
         OUTBOUND_DB_LINK_NAME IS NULL AND
         OUTBOUND_DB_LINK_OWNER IS NULL);
```

Note:

If no high SCN activity is recorded in the `DBA_EXTERNAL_SCN_ACTIVITY` view, then this query returns no results.

See Also:

Oracle Database Reference

31.6 Creating Location Transparency

After you have configured the necessary database links, you can use various tools to hide the distributed nature of the database system from users. In other words, users can access remote objects as if they were local objects.

- Using Views to Create Location Transparency
  Local views can provide location transparency for local and remote tables in a distributed database system.
- Using Synonyms to Create Location Transparency
  Synonyms are useful in both distributed and non-distributed environments because they hide the identity of the underlying object, including its location in a distributed database system. If you must rename or move the underlying object, you only need to redefine the synonym; applications based on the synonym continue to function normally. Synonyms also simplify SQL statements for users in a distributed database system.
- Using Procedures to Create Location Transparency
  PL/SQL program units called **procedures** can provide location transparency.

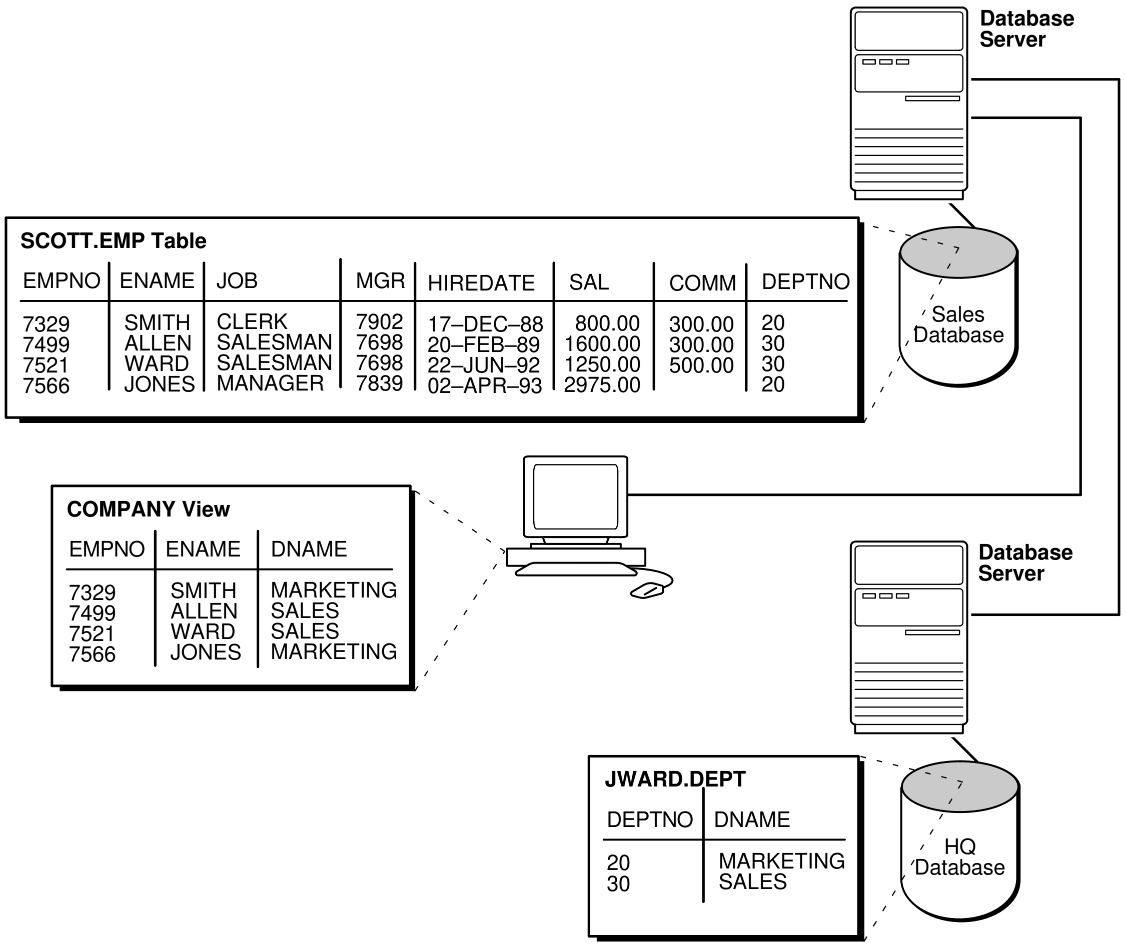
31.6.1 Using Views to Create Location Transparency

Local views can provide location transparency for local and remote tables in a distributed database system.

For example, assume that table `emp` is stored in a local database and table `dept` is stored in a remote database. To make these tables transparent to users of the system, you can create a view in the local database that joins local and remote data:

```
CREATE VIEW company AS
   SELECT a.empno, a.ename, b.dname
   FROM scott.emp a, jward.dept@hq.example.com b
   WHERE a.deptno = b.deptno;
```

Figure 31-3 Views and Location Transparency


Database Server
SCOTT.EMP Table
EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO
7329 SMITH CLERK 7902 17–DEC–88 800.00 300.00 20
7499 ALLEN SALESMAN 7698 20–FEB–89 1600.00 300.00 30
7521 WARD SALESMAN 7698 22–JUN–92 1250.00 500.00 30
7566 JONES MANAGER 7839 02–APR–93 2975.00 20
Sales Database
COMPANY View
EMPNO ENAME DNAME
7329 SMITH MARKETING
7499 ALLEN SALES
7521 WARD SALES
7566 JONES MARKETING
Database Server
JWARD.DEPT
DEPTNO DNAME
20 MARKETING
30 SALES
HQ Database

When users access this view, they do not need to know where the data is physically stored, or if data from more than one table is being accessed. Thus, it is easier for them to get required information. For example, the following query provides data from both the local and remote database table:

```
SELECT * FROM company;
```

The owner of the local view can grant only those object privileges on the local view that have been granted by the remote user. (The remote user is implied by the type of database link). This is similar to privilege management for views that reference local data.

31.6.2 Using Synonyms to Create Location Transparency

Synonyms are useful in both distributed and non-distributed environments because they hide the identity of the underlying object, including its location in a distributed database system. If you must rename or move the underlying object, you only need to redefine the synonym; applications based on the synonym continue to function normally. Synonyms also simplify SQL statements for users in a distributed database system.

- Creating Synonyms
  All synonyms are schema objects that are stored in the data dictionary of the database in which they are created. To simplify remote table access through database links, a synonym

can allow single-word access to remote data, hiding the specific object name and the location from users of the synonym.

- Managing Privileges and Synonyms
  A synonym is a reference to an actual object. A user who has access to a synonym for a particular schema object must also have privileges on the underlying schema object itself.

31.6.2.1 Creating Synonyms

All synonyms are schema objects that are stored in the data dictionary of the database in which they are created. To simplify remote table access through database links, a synonym can allow single-word access to remote data, hiding the specific object name and the location from users of the synonym.

You can create synonyms for the following:

- Tables
- Types
- Views
- Materialized views
- Sequences
- Procedures
- Functions
- Packages

The syntax to create a synonym is:

```
CREATE [PUBLIC] SYNONYM synonym_name
FOR [schema.]object_name[@database_link_name];
```

where:

- `PUBLIC` is a keyword specifying that this synonym is available to all users. Omitting this parameter makes a synonym private, and usable only by the creator. Public synonyms can be created only by a user with `CREATE PUBLIC SYNONYM` system privilege.
- *`synonym_name`* specifies the alternate object name to be referenced by users and applications.
- *`schema`* specifies the schema of the object specified in *`object_name`*. Omitting this parameter uses the schema of the creator as the schema of the object.
- *`object_name`* specifies either a table, view, sequence, materialized view, type, procedure, function or package as appropriate.
- *`database_link_name`* specifies the database link identifying the remote database and schema in which the object specified in *`object_name`* is located.

A synonym must be a uniquely named object for its schema. If a schema contains a schema object and a public synonym exists with the same name, then the database always finds the schema object when the user that owns the schema references that name.

Example: Creating a Public Synonym

Assume that in every database in a distributed database system, a public synonym is defined for the `scott.emp` table stored in the `hq` database:

```
CREATE PUBLIC SYNONYM emp FOR scott.emp@hq.example.com;
```

You can design an employee management application without regard to where the application is used because the location of the table `scott.emp@hq.example.com` is hidden by the public synonyms. SQL statements in the application access the table by referencing the public synonym `emp`.

Furthermore, if you move the `emp` table from the `hq` database to the `hr` database, then you only need to change the public synonyms on the nodes of the system. The employee management application continues to function properly on all nodes.

31.6.2.2 Managing Privileges and Synonyms

A synonym is a reference to an actual object. A user who has access to a synonym for a particular schema object must also have privileges on the underlying schema object itself.

For example, if the user attempts to access a synonym but does not have privileges on the table it identifies, an error occurs indicating that the table or view does not exist.

Assume `scott` creates local synonym `emp` as an alias for remote object `scott.emp@sales.example.com.` `scott` *cannot* grant object privileges on the synonym to another local user. `scott` cannot grant local privileges for the synonym because this operation amounts to granting privileges for the remote `emp` table on the `sales` database, which is not allowed. This behavior is different from privilege management for synonyms that are aliases for local tables or views.

Therefore, you cannot manage local privileges when synonyms are used for location transparency. Security for the base object is controlled entirely at the remote node. For example, user `admin` cannot grant object privileges for the `emp_syn` synonym.

Unlike a database link referenced in a view or procedure definition, a database link referenced in a synonym is resolved by first looking for a private link owned by the schema in effect at the time the reference to the synonym is parsed. Therefore, to ensure the desired object resolution, it is especially important to specify the schema of the underlying object in the definition of a synonym.

31.6.3 Using Procedures to Create Location Transparency

PL/SQL program units called **procedures** can provide location transparency.

- Using Local Procedures to Reference Remote Data
  Procedures or functions (either standalone or in packages) can contain SQL statements that reference remote data.
- Using Local Procedures to Call Remote Procedures
  You can use a local procedure to call a remote procedure. The remote procedure can then execute the required DML.
- Using Local Synonyms to Reference Remote Procedures
  You can use a local synonym to reference a remote procedure.
- Managing Procedures and Privileges
  Assume a local procedure includes a statement that references a remote table or view. The owner of the local procedure can grant the `execute` privilege to any user, thereby giving that user the ability to execute the procedure and, indirectly, access remote data.

31.6.3.1 Using Local Procedures to Reference Remote Data

Procedures or functions (either standalone or in packages) can contain SQL statements that reference remote data.

For example, consider the procedure created by the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
  DELETE FROM emp@hq.example.com
  WHERE empno = enum;
END;
```

When a user or application calls the `fire_emp` procedure, it is not apparent that a remote table is being modified.

A second layer of location transparency is possible when the statements in a procedure indirectly reference remote data using local procedures, views, or synonyms. For example, the following statement defines a local synonym:

```
CREATE SYNONYM emp FOR emp@hq.example.com;
```

Given this synonym, you can create the `fire_emp` procedure using the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
  DELETE FROM emp WHERE empno = enum;
END;
```

If you rename or move the table `emp@hq`, then you only need to modify the local synonym that references the table. None of the procedures and applications that call the procedure require modification.

31.6.3.2 Using Local Procedures to Call Remote Procedures

You can use a local procedure to call a remote procedure. The remote procedure can then execute the required DML.

For example, assume that `scott` connects to `local_db` and creates the following procedure:

```
CONNECT scott@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
  EXECUTE term_emp@hq.example.com;
END;
```

Now, assume that `scott` connects to the remote database and creates the remote procedure:

```
CONNECT scott@hq.example.com

CREATE PROCEDURE term_emp (enum NUMBER)
AS
BEGIN
  DELETE FROM emp WHERE empno = enum;
END;
```

When a user or application connected to `local_db` calls the `fire_emp` procedure, this procedure in turn calls the remote `term_emp` procedure on `hq.example.com`.

31.6.3.3 Using Local Synonyms to Reference Remote Procedures

You can use a local synonym to reference a remote procedure.

For example, `scott` connects to the local `sales.example.com` database and creates the following procedure:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp@hq.example.com
WHERE empno = enum;
END;
```

User `peggy` then connects to the `supply.example.com` database and creates the following synonym for the procedure that `scott` created on the remote `sales` database:

```
SQL> CONNECT peggy@supply
SQL> CREATE PUBLIC SYNONYM emp FOR scott.fire_emp@sales.example.com;
```

A local user on `supply` can use this synonym to execute the procedure on `sales`.

31.6.3.4 Managing Procedures and Privileges

Assume a local procedure includes a statement that references a remote table or view. The owner of the local procedure can grant the `execute` privilege to any user, thereby giving that user the ability to execute the procedure and, indirectly, access remote data.

In general, procedures aid in security. Privileges for objects referenced within a procedure do not need to be explicitly granted to the calling users.

31.7 Managing Statement Transparency

In a distributed database, some SQL statements can reference remote tables.

The database allows the following standard DML statements to reference remote tables:

- `SELECT` (queries)
- `INSERT`
- `UPDATE`
- `DELETE`
- `SELECT...FOR UPDATE` (not always supported in Heterogeneous Systems)
- `LOCK TABLE`

Queries including joins, aggregates, subqueries, and `SELECT...FOR UPDATE` can reference any number of local and remote tables and views. For example, the following query joins information from two remote tables:

```
SELECT e.empno, e.ename, d.dname
  FROM scott.emp@sales.division3.example.com e, jward.dept@hq.example.com d
  WHERE e.deptno = d.deptno;
```

In a homogeneous environment, `UPDATE`, `INSERT`, `DELETE`, and `LOCK TABLE` statements can reference both local and remote tables. No programming is necessary to update remote data. For example, the following statement inserts new rows into the remote table `emp` in the scott.sales schema by selecting rows from the emp table in the jward schema in the local database:

```
INSERT INTO scott.emp@sales.division3.example.com
  SELECT * FROM jward.emp;
```

Restrictions for Statement Transparency:

Several restrictions apply to statement transparency.

- Data manipulation language statements that update objects on a remote non-Oracle Database system cannot reference any objects on the local Oracle Database. For example, a statement such as the following will cause an error to be raised:

  ```
  INSERT INTO remote_table@link as SELECT * FROM local_table;
  ```

- Within a single SQL statement, all referenced LONG and LONG RAW columns, sequences, updated tables, and locked tables must be located at the same node.
- The database does not allow remote DDL statements (for example, CREATE, ALTER, and DROP) in homogeneous systems except through remote execution of procedures of the DBMS_SQL package, as in this example:

  ```
  DBMS_SQL.PARSE@link_name(crs, 'drop table emp', v7);
  ```

  Note that in Heterogeneous Systems, a pass-through facility lets you execute DDL.
- The LIST CHAINED ROWS clause of an ANALYZE statement cannot reference remote tables.
- In a distributed database system, the database always evaluates environmentally-dependent SQL functions such as SYSDATE, USER, UID, and USERENV with respect to the local server, no matter where the statement (or portion of a statement) executes.

  Note:

  Oracle Database supports the USERENV function for queries only.

- Several performance restrictions relate to access of remote objects:
  - Remote views do not have statistical data.
  - Queries on partitioned tables may not be optimized.
  - No more than 20 indexes are considered for a remote table.
  - No more than 20 columns are used for a composite index.
- There is a restriction in the Oracle Database implementation of distributed read consistency that can cause one node to be in the past with respect to another node. In accordance with read consistency, a query may end up retrieving consistent, but out-of-date data. See "Managing Read Consistency" to learn how to manage this problem.

  See Also:

  Oracle Database PL/SQL Packages and Types Reference for more information about the DBMS_SQL package

31.8 Managing a Distributed Database: Examples

Examples illustrate managing database links.

- Example 1: Creating a Public Fixed User Database Link
  An example illustrates creating a public fixed user database link.
- Example 2: Creating a Public Fixed User Shared Database Link
  An example illustrates creating a public fixed user shared database link.
- Example 3: Creating a Public Connected User Database Link
  An example illustrates creating a public connected user database link.
- Example 4: Creating a Public Connected User Shared Database Link
  An example illustrates creating a public connected user shared database link.
- Example 5: Creating a Public Current User Database Link
  An example illustrates creating a public current user database link.

31.8.1 Example 1: Creating a Public Fixed User Database Link

An example illustrates creating a public fixed user database link.

The following statements connect to the local database as `jane` and create a public fixed user database link to database `sales` for `scott`. The database is accessed through its net service name `sldb`:

```
CONNECT jane@local

CREATE PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO scott IDENTIFIED BY password
  USING 'sldb';
```

After executing these statements, any user connected to the local database can use the `sales.division3.example.com` database link to connect to the remote database. Each user connects to the schema `scott` in the remote database.

To access the table `emp` table in `scott`'s remote schema, a user can issue the following SQL query:

```
SELECT * FROM emp@sales.division3.example.com;
```

Note that each application or user session creates a separate connection to the common account on the server. The connection to the remote database remains open for the duration of the application or user session.

31.8.2 Example 2: Creating a Public Fixed User Shared Database Link

An example illustrates creating a public fixed user shared database link.

The following example connects to the local database as `dana` and creates a public link to the `sales` database (using its net service name `sldb`). The link allows a connection to the remote database as `scott` and authenticates this user as `scott`:

```
CONNECT dana@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO scott IDENTIFIED BY password
  AUTHENTICATED BY scott IDENTIFIED BY password
  USING 'sldb';
```

Now, any user connected to the local shared server can use this database link to connect to the remote `sales` database through a shared server process. The user can then query tables in the `scott` schema.

In the preceding example, each local shared server can establish one connection to the remote server. Whenever a local shared server process must access the remote server through the `sales.division3.example.com` database link, the local shared server process reuses established network connections.

31.8.3 Example 3: Creating a Public Connected User Database Link

An example illustrates creating a public connected user database link.

The following example connects to the local database as `larry` and creates a public link to the database with the net service name `sldb`:

```
CONNECT larry@local

CREATE PUBLIC DATABASE LINK redwood
  USING 'sldb';
```

Any user connected to the local database can use the `redwood` database link. The connected user in the local database who uses the database link determines the remote schema.

If `scott` is the connected user and uses the database link, then the database link connects to the remote schema `scott`. If `fox` is the connected user and uses the database link, then the database link connects to remote schema `fox`.

The following statement fails for local user `fox` in the local database when the remote schema `fox` cannot resolve the `emp` schema object. That is, if the `fox` schema in the `sales.division3.example.com` does not have `emp` as a table, view, or (public) synonym, an error will be returned.

```
CONNECT fox@local

SELECT * FROM emp@redwood;
```

31.8.4 Example 4: Creating a Public Connected User Shared Database Link

An example illustrates creating a public connected user shared database link.

The following example connects to the local database as `neil` and creates a shared, public link to the `sales` database (using its net service name `sldb`). The user is authenticated by the userid/password of `crazy/horse`. The following statement creates a public, connected user, shared database link:

```
CONNECT neil@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.example.com
  AUTHENTICATED BY crazy IDENTIFIED BY horse
  USING 'sldb';
```

Each user connected to the local server can use this shared database link to connect to the remote database and query the tables in the corresponding remote schema.

Each local, shared server process establishes one connection to the remote server. Whenever a local server process must access the remote server through the `sales.division3.example.com` database link, the local process reuses established network connections, even if the connected user is a different user.

If this database link is used frequently, eventually every shared server in the local database will have a remote connection. At this point, no more physical connections are needed to the remote server, even if new users use this shared database link.

31.8.5 Example 5: Creating a Public Current User Database Link

An example illustrates creating a public current user database link.

The following example connects to the local database as the connected user and creates a public link to the `sales` database (using its net service name `sldb`). The following statement creates a public current user database link:

```
CONNECT bart@local

CREATE PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO CURRENT_USER
  USING 'sldb';
```

Note:

To use this link, the current user must be a global user.

The consequences of this database link are as follows:

Assume `scott` creates local procedure `fire_emp` that deletes a row from the remote `emp` table, and grants execute privilege on `fire_emp` to `ford`.

```
CONNECT scott@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
   DELETE FROM emp@sales.division3.example.com
   WHERE empno=enum;
END;

GRANT EXECUTE ON fire_emp TO ford;
```

Now, assume that `ford` connects to the local database and runs `scott`'s procedure:

```
CONNECT ford@local_db

EXECUTE PROCEDURE scott.fire_emp (enum 10345);
```

When `ford` executes the procedure `scott.fire_emp`, the procedure runs under `scott`'s privileges. Because a current user database link is used, the connection is established to `scott`'s remote schema, not `ford`'s remote schema. Note that `scott` must be a global user while `ford` does not have to be a global user.

> **Note:**
>
> If a connected user database link were used instead, the connection would be to `ford`'s remote schema. For more information about invoker rights and privileges, see the *Oracle Database PL/SQL Language Reference*.

You can accomplish the same result by using a fixed user database link to `scott`'s remote schema.

32 Developing Applications for a Distributed Database System

Developing applications for a distributed database system includes tasks such as managing the distribution of application data, controlling connections established by database links, maintaining referential integrity, tuning distributed queries, and handling errors in remote procedures.

- Managing the Distribution of Application Data
  In a distributed database environment, coordinate with the database administrator to determine the best location for the data.
- Controlling Connections Established by Database Links
  When a global object name is referenced in a SQL statement or remote procedure call, database links establish a connection to a session in the remote database on behalf of the local user.
- Maintaining Referential Integrity in a Distributed System
  Design your application to check for any returned error messages that indicate that a portion of the distributed update has failed. If you detect a failure, then you should roll back the entire transaction before allowing the application to proceed.
- Tuning Distributed Queries
  The local Oracle Database server breaks the distributed query into a corresponding number of remote queries, which it then sends to the remote nodes for execution. The remote nodes execute the queries and send the results back to the local node. The local node then performs any necessary post-processing and returns the results to the user or application.
- Handling Errors in Remote Procedures
  Errors can occur when a database executes a procedure.

> **See Also:**
>
> *Oracle Database Development Guide* for more information about application development in an Oracle Database environment

32.1 Managing the Distribution of Application Data

In a distributed database environment, coordinate with the database administrator to determine the best location for the data.

Some issues to consider are:

- Number of transactions posted from each location
- Amount of data (portion of table) used by each node
- Performance characteristics and reliability of the network

- Speed of various nodes, capacities of disks
- Importance of a node or link when it is unavailable
- Need for referential integrity among tables

32.2 Controlling Connections Established by Database Links

When a global object name is referenced in a SQL statement or remote procedure call, database links establish a connection to a session in the remote database on behalf of the local user.

The remote connection and session are only created if the connection has not already been established previously for the local user session.

The connections and sessions established to remote databases persist for the duration of the local user's session, unless the application or user explicitly terminates them. Note that when you issue a `SELECT` statement across a database link, a transaction lock is placed on the undo segments. To rerelease the segment, you must issue a `COMMIT` or `ROLLBACK` statement.

Terminating remote connections established using database links is useful for disconnecting high cost connections that are no longer required by the application. You can terminate a remote connection and session using the `ALTER SESSION` statement with the `CLOSE DATABASE LINK` clause. For example, assume you issue the following transactions:

```
SELECT * FROM emp@sales;
COMMIT;
```

The following statement terminates the session in the remote database pointed to by the `sales` database link:

```
ALTER SESSION CLOSE DATABASE LINK sales;
```

To close a database link connection in your user session, you must have the `ALTER SESSION` system privilege.

Note:

Before closing a database link, first close all cursors that use the link and then end your current transaction if it uses the link.

See Also:

Oracle Database SQL Language Reference for more information about the `ALTER SESSION` statement

32.3 Maintaining Referential Integrity in a Distributed System

Design your application to check for any returned error messages that indicate that a portion of the distributed update has failed. If you detect a failure, then you should roll back the entire transaction before allowing the application to proceed.

If a part of a distributed statement fails, for example, due to an integrity constraint violation, the database returns error number `ORA-02055`. Subsequent statements or procedure calls return error number `ORA-02067` until a `ROLLBACK` or `ROLLBACK TO SAVEPOINT` is issued.

The database does not permit declarative referential integrity constraints to be defined across nodes of a distributed system. In other words, a declarative referential integrity constraint on one table cannot specify a foreign key that references a primary or unique key of a remote table. Nevertheless, you can maintain parent/child table relationships across nodes using triggers.

If you decide to define referential integrity across the nodes of a distributed database using triggers, be aware that network failures can limit the accessibility of not only the parent table, but also the child table. For example, assume that the child table is in the `sales` database and the parent table is in the `hq` database. If the network connection between the two databases fails, some DML statements against the child table (those that insert rows into the child table or update a foreign key value in the child table) cannot proceed because the referential integrity triggers must have access to the parent table in the `hq` database.

See Also:

Oracle Database PL/SQL Language Reference for more information about using triggers to enforce referential integrity

32.4 Tuning Distributed Queries

The local Oracle Database server breaks the distributed query into a corresponding number of remote queries, which it then sends to the remote nodes for execution. The remote nodes execute the queries and send the results back to the local node. The local node then performs any necessary post-processing and returns the results to the user or application.

Note:

SQL management objects, such as SQL profiles, SQL plan baselines, and SQL patches, and stored outlines might not always work as expected if your query references remote tables with database links. For example, for SQL plan management, when Oracle uses a SQL plan baseline for the query, the parts of the query that are remotely executed might use a different plan than when the SQL plan baseline was created.

- Using Collocated Inline Views
  The most effective way of optimizing distributed queries is to access the remote databases as little as possible and to retrieve only the required data.
- Using Cost-Based Optimization
  Using cost-based optimization includes completing tasks such as rewriting queries and setting up cost-based optimization.
- Using Hints
  Hints can extend the capability of cost-based optimization.
- Analyzing the Execution Plan
  An important aspect to tuning distributed queries is analyzing the execution plan.

32.4.1 Using Collocated Inline Views

The most effective way of optimizing distributed queries is to access the remote databases as little as possible and to retrieve only the required data.

For example, assume you reference five remote tables from two different remote databases in a distributed query and have a complex filter (for example, `WHERE r1.salary + r2.salary > 50000`). You can improve the performance of the query by rewriting the query to access the remote databases once and to apply the filter at the remote site. This rewrite causes less data to be transferred to the query execution site.

Rewriting your query to access the remote database once is achieved by using collocated inline views. The following terms need to be defined:

- **Collocated**

  Two or more tables located in the same database.

- **Inline view**

  A `SELECT` statement that is substituted for a table in a parent `SELECT` statement. The embedded `SELECT` statement, shown within the parentheses is an example of an inline view:

  ```
  SELECT e.empno,e.ename,d.deptno,d.dname
     FROM (SELECT empno, ename from
           emp@orc1.world) e, dept d;
  ```

- **Collocated inline view**

  An inline view that selects data from multiple tables from a single database only. It reduces the amount of times that the remote database is accessed, improving the performance of a distributed query.

Oracle recommends that you form your distributed query using collocated inline views to increase the performance of your distributed query. Oracle Database cost-based optimization can transparently rewrite many of your distributed queries to take advantage of the performance gains offered by collocated inline views.

32.4.2 Using Cost-Based Optimization

Using cost-based optimization includes completing tasks such as rewriting queries and setting up cost-based optimization.

- How Does Cost-Based Optimization Work?
  The main task of optimization is to rewrite a distributed query to use collocated inline views.
- Rewriting Queries for Cost-Based Optimization
  In addition to rewriting your queries with collocated inline views, the cost-based optimization method optimizes distributed queries according to the gathered statistics of the referenced tables and the computations performed by the optimizer.
- Setting Up Cost-Based Optimization
  After you have set up your system to use cost-based optimization to improve the performance of distributed queries, the operation is transparent to the user. In other words, the optimization occurs automatically when the query is issued.

32.4.2.1 How Does Cost-Based Optimization Work?

The main task of optimization is to rewrite a distributed query to use collocated inline views.

This optimization is performed in three steps:

1. All mergeable views are merged.
2. Optimizer performs collocated query block test.
3. Optimizer rewrites query using collocated inline views.

After the query is rewritten, it is executed and the data set is returned to the user.

While cost-based optimization is performed transparently to the user, it cannot improve the performance of several distributed query scenarios. Specifically, if your distributed query contains any of the following, cost-based optimization is not effective:

- Aggregates
- Subqueries
- Complex SQL

If your distributed query contains one of these elements, see "Using Hints" to learn how you can modify your query and use hints to improve the performance of your distributed query.

32.4.2.2 Rewriting Queries for Cost-Based Optimization

In addition to rewriting your queries with collocated inline views, the cost-based optimization method optimizes distributed queries according to the gathered statistics of the referenced tables and the computations performed by the optimizer.

For example, cost-based optimization analyzes the following query. The example assumes that table statistics are available. Note that it analyzes the query inside a `CREATE TABLE` statement:

```
CREATE TABLE AS (
                 SELECT l.a, l.b, r1.c, r1.d, r1.e, r2.b, r2.c
                 FROM local l, remote1 r1, remote2 r2
                    WHERE l.c = r.c
                    AND r1.c = r2.c
                    AND r.e > 300
                );
```

and rewrites it as:

```
CREATE TABLE AS (
                 SELECT l.a, l.b, v.c, v.d, v.e
                 FROM (
                        SELECT r1.c, r1.d, r1.e, r2.b, r2.c
                         FROM remote1 r1, remote2 r2
                         WHERE r1.c = r2.c
                         AND r1.e > 300
                      ) v, local l
                 WHERE l.c = r1.c
                );
```

The alias `v` is assigned to the inline view, which can then be referenced as a table in the preceding `SELECT` statement. Creating a collocated inline view reduces the amount of queries performed at a remote site, thereby reducing costly network traffic.

32.4.2.3 Setting Up Cost-Based Optimization

After you have set up your system to use cost-based optimization to improve the performance of distributed queries, the operation is transparent to the user. In other words, the optimization occurs automatically when the query is issued.

- Setting Up the Environment
  Set the `OPTIMIZER_MODE` initialization parameter to establish the default behavior for choosing an optimization approach for the instance.
- Analyzing Tables
  For cost-based optimization to select the most efficient path for a distributed query, you must provide accurate statistics for the tables involved. You do this using the `DBMS_STATS` package.

32.4.2.3.1 Setting Up the Environment

Set the `OPTIMIZER_MODE` initialization parameter to establish the default behavior for choosing an optimization approach for the instance.

You can set this parameter by:

- Modifying the `OPTIMIZER_MODE` parameter in the initialization parameter file
- Setting it at session level by issuing an `ALTER SESSION` statement

See Also:

Oracle Database SQL Tuning Guide for information on setting the `OPTIMIZER_MODE` initialization parameter in the parameter file and for configuring your system to use a cost-based optimization method

32.4.2.3.2 Analyzing Tables

For cost-based optimization to select the most efficient path for a distributed query, you must provide accurate statistics for the tables involved. You do this using the `DBMS_STATS` package.

Note:

You must connect locally with respect to the tables when executing the `DBMS_STATS` procedure.

You must first connect to the remote site and then execute a `DBMS_STATS` procedure.

The following `DBMS_STATS` procedures enable the gathering of certain classes of optimizer statistics:

- `GATHER_INDEX_STATS`
- `GATHER_TABLE_STATS`
- `GATHER_SCHEMA_STATS`

- `GATHER_DATABASE_STATS`

For example, assume that distributed transactions routinely access the `scott.dept` table. To ensure that the cost-based optimizer is still picking the best plan, execute the following:

```
BEGIN
   DBMS_STATS.GATHER_TABLE_STATS ('scott', 'dept');
END;
```

See Also:

- *Oracle Database SQL Tuning Guide* for information about generating statistics
- *Oracle Database PL/SQL Packages and Types Reference* for additional information on using the `DBMS_STATS` package

32.4.3 Using Hints

Hints can extend the capability of cost-based optimization.

- About Using Hints
  If a statement is not sufficiently optimized, then you can use hints to extend the capability of cost-based optimization. Specifically, if you write your own query to use collocated inline views, instruct the cost-based optimizer not to rewrite your distributed query.
- Using the NO_MERGE Hint
  The `NO_MERGE` hint prevents the database from merging an inline view into a potentially non-collocated SQL statement.
- Using the DRIVING_SITE Hint
  The `DRIVING_SITE` hint lets you specify the site where the query execution is performed.

32.4.3.1 About Using Hints

If a statement is not sufficiently optimized, then you can use hints to extend the capability of cost-based optimization. Specifically, if you write your own query to use collocated inline views, instruct the cost-based optimizer not to rewrite your distributed query.

Additionally, if you have special knowledge about the database environment (such as statistics, load, network and CPU limitations, distributed queries, and so forth), you can specify a hint to guide cost-based optimization. For example, if you have written your own optimized query using collocated inline views that are based on your knowledge of the database environment, specify the `NO_MERGE` hint to prevent the optimizer from rewriting your query.

This technique is especially helpful if your distributed query contains an aggregate, subquery, or complex SQL. Because this type of distributed query cannot be rewritten by the optimizer, specifying `NO_MERGE` causes the optimizer to skip the steps described in "How Does Cost-Based Optimization Work?".

The `DRIVING_SITE` hint lets you define a remote site to act as the query execution site. In this way, the query executes on the remote site, which then returns the data to the local site. This hint is especially helpful when the remote site contains the majority of the data.

See Also:

Oracle Database SQL Tuning Guide for more information about using hints

32.4.3.2 Using the NO_MERGE Hint

The `NO_MERGE` hint prevents the database from merging an inline view into a potentially non-collocated SQL statement.

This hint is embedded in the `SELECT` statement and can appear either at the beginning of the `SELECT` statement with the inline view as an argument or in the query block that defines the inline view.

```
/* with argument */

SELECT /*+NO_MERGE(v)*/ t1.x, v.avg_y
   FROM t1, (SELECT x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
   WHERE t1.x = v.x AND t1.y = 1;

/* in query block */

SELECT t1.x, v.avg_y
   FROM t1, (SELECT /*+NO_MERGE*/ x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
   WHERE t1.x = v.x AND t1.y = 1;
```

Typically, you use this hint when you have developed an optimized query based on your knowledge of your database environment.

Related Topics

- Using Hints
  Hints can extend the capability of cost-based optimization.

32.4.3.3 Using the DRIVING_SITE Hint

The `DRIVING_SITE` hint lets you specify the site where the query execution is performed.

It is best to let cost-based optimization determine where the execution should be performed, but if you prefer to override the optimizer, you can specify the execution site manually.

Following is an example of a `SELECT` statement with a `DRIVING_SITE` hint:

```
SELECT /*+DRIVING_SITE(dept)*/ * FROM emp, dept@remote.com
   WHERE emp.deptno = dept.deptno;
```

Related Topics

- Using Hints
  Hints can extend the capability of cost-based optimization.

32.4.4 Analyzing the Execution Plan

An important aspect to tuning distributed queries is analyzing the execution plan.

The feedback that you receive from your analysis is an important element to testing and verifying your database. Verification becomes especially important when you want to compare plans. For example, comparing the execution plan for a distributed query optimized by cost-based optimization to a plan for a query manually optimized using hints, collocated inline views, and other techniques.

- Generating the Execution Plan
  After you have prepared the database to store the execution plan, you are ready to view the plan for a specified query. Instead of directly executing a SQL statement, append the statement to the `EXPLAIN PLAN FOR` clause.
- Viewing the Execution Plan
  After you have executed the preceding SQL statement, the execution plan is stored temporarily in the `PLAN_TABLE`.

See Also:

Oracle Database SQL Tuning Guide for detailed information about execution plans, the `EXPLAIN PLAN` statement, and how to interpret the results

32.4.4.1 Generating the Execution Plan

After you have prepared the database to store the execution plan, you are ready to view the plan for a specified query. Instead of directly executing a SQL statement, append the statement to the `EXPLAIN PLAN FOR` clause.

For example, you can execute the following:

```
EXPLAIN PLAN FOR
   SELECT d.dname
   FROM dept d
      WHERE d.deptno
      IN (SELECT deptno
          FROM emp@orc2.world
          GROUP BY deptno
          HAVING COUNT (deptno) >3
          )
/
```

32.4.4.2 Viewing the Execution Plan

After you have executed the preceding SQL statement, the execution plan is stored temporarily in the `PLAN_TABLE`.

To view the results of the execution plan, execute the following script:

```
@utlxpls.sql
```

Note:

The `utlxpls.sql` can be found in the `$ORACLE_HOME/rdbms/admin` directory.

Executing the `utlxpls.sql` script displays the execution plan for the `SELECT` statement that you specified. The results are formatted as follows:

```
Plan Table
--------------------------------------------------------------------------------
| Operation                    |  Name     |  Rows | Bytes|  Cost   | Pstart| Pstop |
--------------------------------------------------------------------------------
| SELECT STATEMENT             |           |       |      |         |       |       |
|  NESTED LOOPS                |           |       |      |         |       |       |
|   VIEW                       |           |       |      |         |       |       |
|    REMOTE                    |           |       |      |         |       |       |
|   TABLE ACCESS BY INDEX RO|DEPT        |       |      |         |       |       |
|    INDEX UNIQUE SCAN         |PK_DEPT    |       |      |         |       |       |
--------------------------------------------------------------------------------
```

If you are manually optimizing distributed queries by writing your own collocated inline views or using hints, it is best to generate an execution plan before and after your manual optimization. With both execution plans, you can compare the effectiveness of your manual optimization and make changes as necessary to improve the performance of the distributed query.

To view the SQL statement that will be executed at the remote site, execute the following select statement:

```
SELECT OTHER
FROM PLAN_TABLE
   WHERE operation = 'REMOTE';
```

Following is sample output:

```
SELECT DISTINCT "A1"."DEPTNO" FROM "EMP" "A1"
   GROUP BY "A1"."DEPTNO" HAVING COUNT("A1"."DEPTNO")>3
```

Note:

If you are having difficulty viewing the entire contents of the OTHER column, execute the following SQL*Plus command:

```
SET LONG 9999999
```

32.5 Handling Errors in Remote Procedures

Errors can occur when a database executes a procedure.

When the database executes a procedure locally or at a remote location, four types of exceptions can occur:

- PL/SQL user-defined exceptions, which must be declared using the keyword EXCEPTION
- PL/SQL predefined exceptions such as the NO_DATA_FOUND keyword
- SQL errors such as ORA-00900 and ORA-02015
- Application exceptions generated using the RAISE_APPLICATION_ERROR() procedure

When using local procedures, you can trap these messages by writing an exception handler such as the following

```
BEGIN
  ...
EXCEPTION
   WHEN ZERO_DIVIDE THEN
   /* ... handle the exception */
END;
```

Notice that the `WHEN` clause requires an exception name. If the exception does not have a name, for example, exceptions generated with `RAISE_APPLICATION_ERROR`, you can assign one using `PRAGMA_EXCEPTION_INIT`. For example:

```
DECLARE
  null_salary EXCEPTION;
  PRAGMA EXCEPTION_INIT(null_salary, -20101);
BEGIN
  ...
  RAISE_APPLICATION_ERROR(-20101, 'salary is missing');
...
EXCEPTION
  WHEN null_salary THEN
  ...
END;
```

When calling a remote procedure, exceptions can be handled by an exception handler in the local procedure. The remote procedure must return an error number to the local, calling procedure, which then handles the exception as shown in the previous example. Note that PL/SQL user-defined exceptions always return `ORA-06510` to the local procedure.

Therefore, it is not possible to distinguish between two different user-defined exceptions based on the error number. All other remote exceptions can be handled in the same manner as local exceptions.

See Also:

Oracle Database PL/SQL Language Reference for more information about PL/SQL procedures

33 Distributed Transactions Concepts

Distributed transactions update data on two or more distinct nodes of a distributed database.

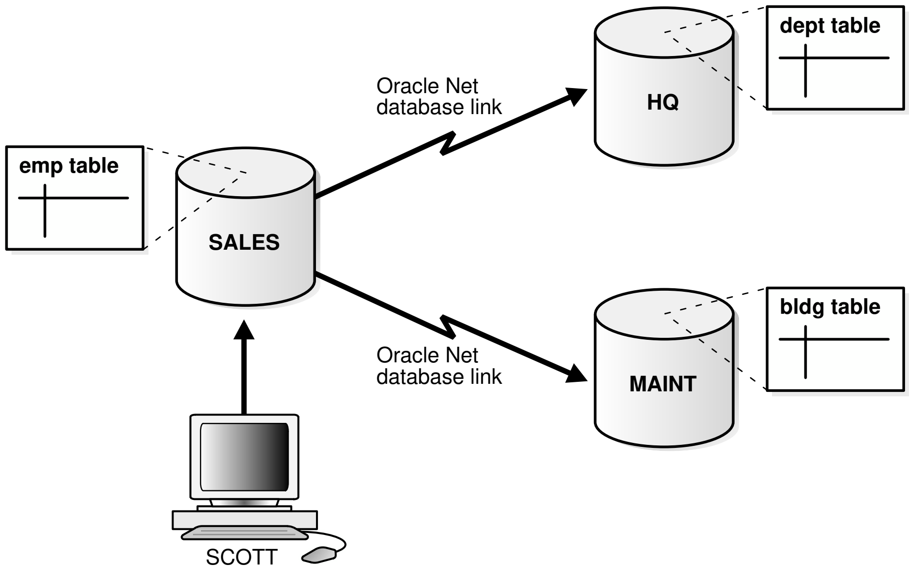
- What Are Distributed Transactions?
  A **distributed transaction** includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.
- Session Trees for Distributed Transactions
  A session tree is a hierarchical model that describes the relationships among sessions and their roles.
- Two-Phase Commit Mechanism
  In a distributed database environment, the database must coordinate the committing or rolling back of the changes in a distributed transaction as a self-contained unit.
- In-Doubt Transactions
  A transaction becomes in-doubt if the two-phase commit mechanism fails.
- Distributed Transaction Processing: Case Study
  A case study illustrates distributed transaction processing.

33.1 What Are Distributed Transactions?

A **distributed transaction** includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

For example, assume the database configuration depicted in Figure 33-1:

Figure 33-1 Distributed System



The following distributed transaction executed by `scott` updates the local `sales` database, the remote `hq` database, and the remote `maint` database:

```
UPDATE scott.dept@hq.us.example.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.example.com
  SET room = 1225
  WHERE room = 1163;
COMMIT;
```

Note:

If all statements of a transaction reference only a single remote node, then the transaction is remote, not distributed.

There are two types of permissible operations in distributed transactions: DML and DDL transactions, and transaction control statement.

- DML and DDL Transactions
  Some DML and DDL operations are supported in a distributed transaction.
- Transaction Control Statements
  Some transaction control statements are supported in distributed transactions.

33.1.1 DML and DDL Transactions

Some DML and DDL operations are supported in a distributed transaction.

The following are the DML and DDL operations supported in a distributed transaction:

- `CREATE TABLE AS SELECT`
- `DELETE`
- `INSERT` (default and direct load)
- `UPDATE`
- `LOCK TABLE`
- `SELECT`
- `SELECT FOR UPDATE`

You can execute DML and DDL statements in parallel, and `INSERT` direct load statements serially, but note the following restrictions:

- All remote operations must be `SELECT` statements.
- These statements must not be clauses in another distributed transaction.
- If the table referenced in the *table_expression_clause* of an `INSERT`, `UPDATE`, or `DELETE` statement is remote, then execution is serial rather than parallel.
- You cannot perform remote operations after issuing parallel DML/DDL or direct load `INSERT`.

- If the transaction begins using XA or OCI, it executes serially.
- No loopback operations can be performed on the transaction originating the parallel operation. For example, you cannot reference a remote object that is actually a synonym for a local object.
- If you perform a distributed operation other than a `SELECT` in the transaction, no DML is parallelized.

33.1.2 Transaction Control Statements

Some transaction control statements are supported in distributed transactions.

The following are the supported transaction control statements:

- `COMMIT`
- `ROLLBACK`
- `SAVEPOINT`

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about these SQL statements

33.2 Session Trees for Distributed Transactions

A session tree is a hierarchical model that describes the relationships among sessions and their roles.

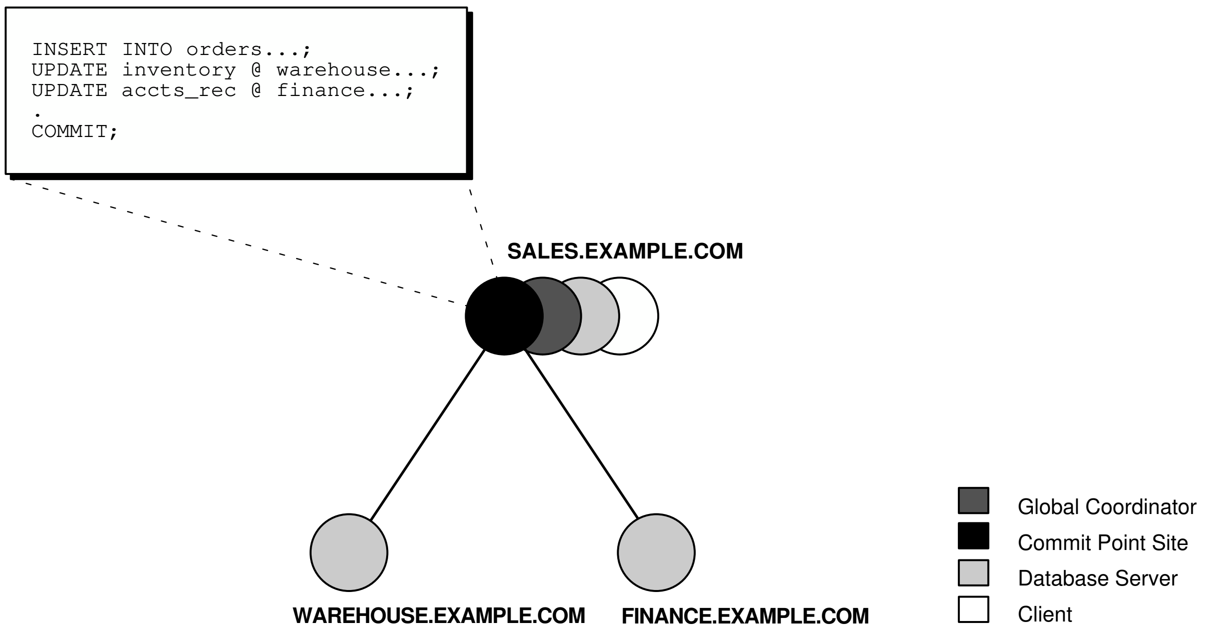
- About Session Trees for Distributed Transactions
  As the statements in a distributed transaction are issued, the database defines a **session tree** of all nodes participating in the transaction.
- Clients
  A node acts as a client when it references information from a database on another node.
- Database Servers
  A database server is a node that hosts a database from which a client requests data.
- Local Coordinators
  A node that must reference data on other nodes to complete its part in the distributed transaction is called a local coordinator.
- Global Coordinator
  The node where the distributed transaction originates is called the global coordinator.
- Commit Point Site
  The system administrator always designates one node to be the commit point site.

33.2.1 About Session Trees for Distributed Transactions

As the statements in a distributed transaction are issued, the database defines a **session tree** of all nodes participating in the transaction.

A session tree is a hierarchical model that describes the relationships among sessions and their roles. Figure 33-2 illustrates a session tree:

Figure 33-2 Example of a Session Tree



```
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
UPDATE accts_rec @ finance...;
.
COMMIT;
```

All nodes participating in the session tree of a distributed transaction assume one or more of the following roles:

Role	Description
Client	A node that references information in a database belonging to a different node.
Database server	A node that receives a request for information from another node.
Global coordinator	The node that originates the distributed transaction.
Local coordinator	A node that is forced to reference data on other nodes to complete its part of the transaction.
Commit point site	The node that commits or rolls back the transaction as instructed by the global coordinator.

The role a node plays in a distributed transaction is determined by:

- Whether the transaction is local or remote
- The **commit point strength** of the node ("Commit Point Site ")
- Whether all requested data is available at a node, or whether other nodes need to be referenced to complete the transaction
- Whether the node is read-only

33.2.2 Clients

A node acts as a client when it references information from a database on another node.

The referenced node is a database server. In Figure 33-2, the node `sales` is a client of the nodes that host the `warehouse` and `finance` databases.

33.2.3 Database Servers

A database server is a node that hosts a database from which a client requests data.

In Figure 33-2, an application at the `sales` node initiates a distributed transaction that accesses data from the `warehouse` and `finance` nodes. Therefore, `sales.example.com` has the role of client node, and `warehouse` and `finance` are both database servers. In this example, `sales` is a database server *and* a client because the application also modifies data in the `sales` database.

33.2.4 Local Coordinators

A node that must reference data on other nodes to complete its part in the distributed transaction is called a local coordinator.

In Figure 33-2, `sales` is a local coordinator because it coordinates the nodes it directly references: `warehouse` and `finance`. The node `sales` also happens to be the global coordinator because it coordinates all the nodes involved in the transaction.

A local coordinator is responsible for coordinating the transaction among the nodes it communicates directly with by:

- Receiving and relaying transaction status information to and from those nodes
- Passing queries to those nodes
- Receiving queries from those nodes and passing them on to other nodes
- Returning the results of queries to the nodes that initiated them

33.2.5 Global Coordinator

The node where the distributed transaction originates is called the global coordinator.

The database application issuing the distributed transaction is directly connected to the node acting as the global coordinator. For example, in Figure 33-2, the transaction issued at the node `sales` references information from the database servers `warehouse` and `finance`. Therefore, `sales.example.com` is the global coordinator of this distributed transaction.

The global coordinator becomes the parent or root of the session tree. The global coordinator performs the following operations during a distributed transaction:

- Sends all of the distributed transaction SQL statements, remote procedure calls, and so forth to the directly referenced nodes, thus forming the session tree
- Instructs all directly referenced nodes other than the commit point site to prepare the transaction
- Instructs the commit point site to initiate the global commit of the transaction if all nodes prepare successfully
- Instructs all nodes to initiate a global rollback of the transaction if there is a terminate response

33.2.6 Commit Point Site

The system administrator always designates one node to be the commit point site.

- About the Commit Point Site
  The job of the commit point site is to initiate a commit or roll back operation as instructed by the global coordinator.
- How a Distributed Transaction Commits
  A distributed transaction is considered committed after all non-commit-point sites are prepared, and the transaction has been actually committed at the commit point site.
- Commit Point Strength
  Every database server must be assigned a commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines which role it plays in the two-phase commit.

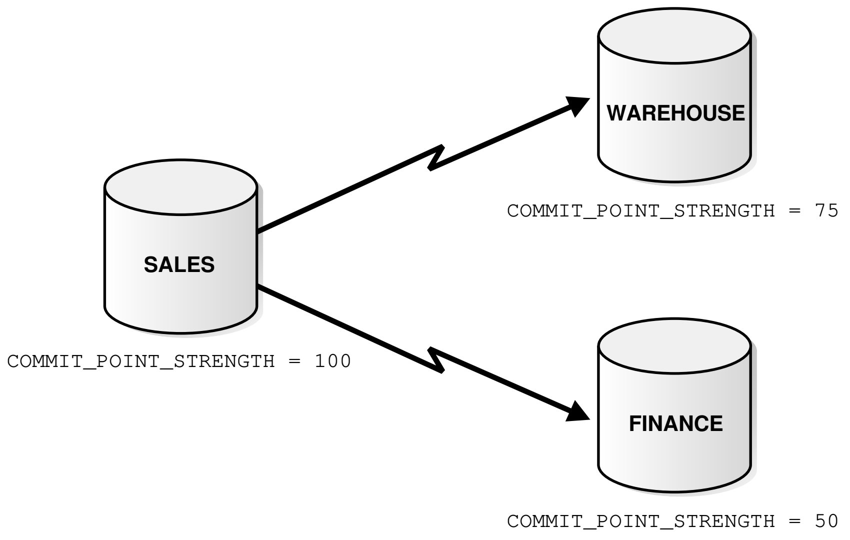
33.2.6.1 About the Commit Point Site

The job of the commit point site is to initiate a commit or roll back operation as instructed by the global coordinator.

The system administrator always designates one node to be the commit point site in the session tree by assigning all nodes a commit point strength. The node selected as commit point site should be the node that stores the most critical data.

Figure 33-3 illustrates an example of distributed system, with `sales` serving as the commit point site:

Figure 33-3 Commit Point Site


WAREHOUSE
COMMIT_POINT_STRENGTH = 75
SALES
COMMIT_POINT_STRENGTH = 100
FINANCE
COMMIT_POINT_STRENGTH = 50

The commit point site is distinct from all other nodes involved in a distributed transaction in these ways:

- The commit point site never enters the prepared state. Consequently, if the commit point site stores the most critical data, this data never remains in-doubt, even if a failure occurs. In failure situations, failed nodes remain in a prepared state, holding necessary locks on data until in-doubt transactions are resolved.
- The commit point site commits before the other nodes involved in the transaction. In effect, the outcome of a distributed transaction at the commit point site determines whether the transaction at all nodes is committed or rolled back: the other nodes follow the lead of the

commit point site. The global coordinator ensures that all nodes complete the transaction in the same manner as the commit point site.

33.2.6.2 How a Distributed Transaction Commits

A distributed transaction is considered committed after all non-commit-point sites are prepared, and the transaction has been actually committed at the commit point site.

The redo log at the commit point site is updated as soon as the distributed transaction is committed at this node.

Because the commit point log contains a record of the commit, the transaction is considered committed even though some participating nodes may still be only in the prepared state and the transaction not yet actually committed at these nodes. In the same way, a distributed transaction is considered *not* committed if the commit has not been logged at the commit point site.

33.2.6.3 Commit Point Strength

Every database server must be assigned a commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines which role it plays in the two-phase commit.

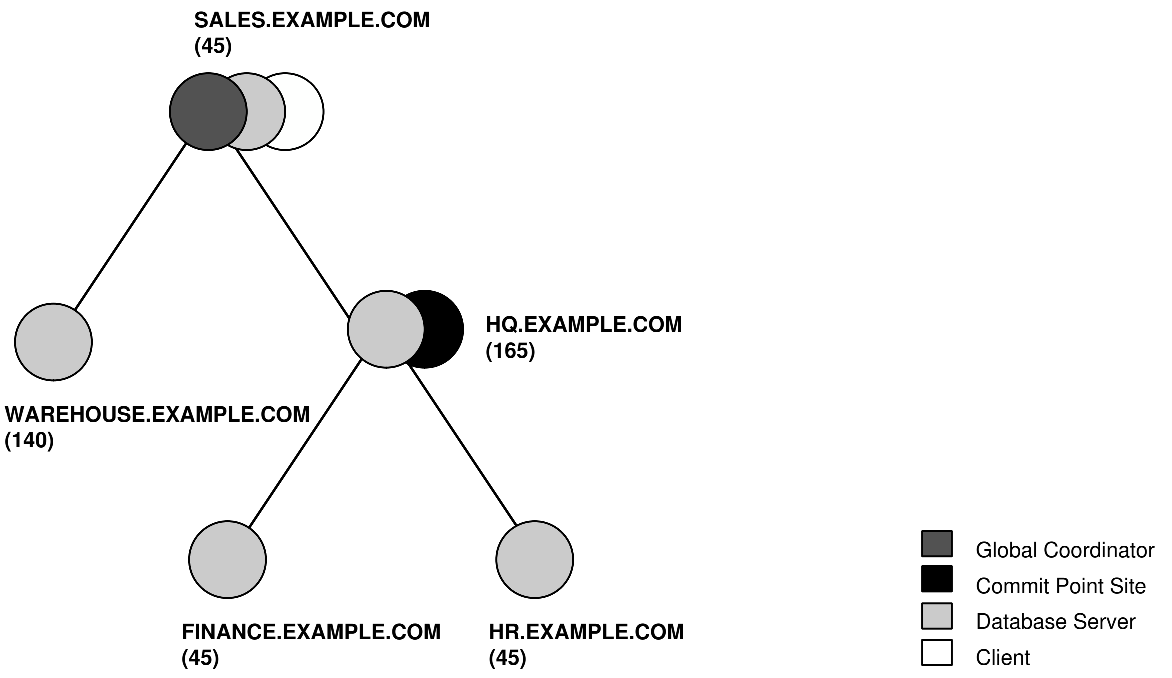
Specifically, the commit point strength determines whether a given node is the commit point site in the distributed transaction and thus commits before all of the other nodes. This value is specified using the initialization parameter `COMMIT_POINT_STRENGTH`. This section explains how the database determines the commit point site.

The commit point site, which is determined at the beginning of the prepare phase, is selected only from the nodes participating in the transaction. The following sequence of events occurs:

1. Of the nodes directly referenced by the global coordinator, the database selects the node with the highest commit point strength as the commit point site.
2. The initially-selected node determines if any of the nodes from which it has to obtain information for this transaction has a higher commit point strength.
3. Either the node with the highest commit point strength directly referenced in the transaction or one of its servers with a higher commit point strength becomes the commit point site.
4. After the final commit point site has been determined, the global coordinator sends prepare responses to all nodes participating in the transaction.

Figure 33-4 shows in a sample session tree the commit point strengths of each node (in parentheses) and shows the node chosen as the commit point site:

Figure 33-4 Commit Point Strengths and Determination of the Commit Point Site



The following conditions apply when determining the commit point site:

- A read-only node cannot be the commit point site.
- If multiple nodes directly referenced by the global coordinator have the same commit point strength, then the database designates one of these as the commit point site.
- If a distributed transaction ends with a rollback, then the prepare and commit phases are not needed. Consequently, the database never determines a commit point site. Instead, the global coordinator sends a `ROLLBACK` statement to all nodes and ends the processing of the distributed transaction.

As Figure 33-4 illustrates, the commit point site and the global coordinator can be different nodes of the session tree. The commit point strength of each node is communicated to the coordinators when the initial connections are made. The coordinators retain the commit point strengths of each node they are in direct communication with so that commit point sites can be efficiently selected during two-phase commits. Therefore, it is not necessary for the commit point strength to be exchanged between a coordinator and a node each time a commit occurs.

See Also:

- "Specifying the Commit Point Strength of a Node" to learn how to set the commit point strength of a node
- *Oracle Database Reference* for more information about the initialization parameter `COMMIT_POINT_STRENGTH`

33.3 Two-Phase Commit Mechanism

In a distributed database environment, the database must coordinate the committing or rolling back of the changes in a distributed transaction as a self-contained unit.

- About the Two-Phase Commit Mechanism
  Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because the database must coordinate the committing or rolling back of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.
- Prepare Phase
  Prepare phase is the first phase in committing a distributed transaction.
- Commit Phase
  The second phase in committing a distributed transaction is the commit phase. Before this phase occurs, *all* nodes other than the commit point site referenced in the distributed transaction have guaranteed that they are prepared, that is, they have the necessary resources to commit the transaction.
- Forget Phase
  After the participating nodes notify the commit point site that they have committed, the commit point site can forget about the transaction.

33.3.1 About the Two-Phase Commit Mechanism

Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because the database must coordinate the committing or rolling back of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.

The database ensures the integrity of data in a distributed transaction using the **two-phase commit mechanism**. In the **prepare phase**, the initiating node in the transaction asks the other participating nodes to promise to commit or roll back the transaction. During the **commit phase**, the initiating node asks all participating nodes to commit the transaction. If this outcome is not possible, then all nodes are asked to roll back.

All participating nodes in a distributed transaction should perform the same action: they should either all commit or all perform a rollback of the transaction. The database automatically controls and monitors the commit or rollback of a distributed transaction and maintains the integrity of the **global database** (the collection of databases participating in the transaction) using the two-phase commit mechanism. This mechanism is completely transparent, requiring no programming on the part of the user or application developer.

The commit mechanism has the following distinct phases, which the database performs automatically whenever a user commits a distributed transaction:

Phase	Description
Prepare phase	The initiating node, called the **global coordinator**, asks participating nodes other than the commit point site to promise to commit or roll back the transaction, even if there is a failure. If any node cannot prepare, the transaction is rolled back.

Phase	Description
Commit phase	If all participants respond to the coordinator that they are prepared, then the coordinator asks the commit point site to commit. After it commits, the coordinator asks all other nodes to commit the transaction.
Forget phase	The global coordinator forgets about the transaction.

33.3.2 Prepare Phase

Prepare phase is the first phase in committing a distributed transaction.

- About Prepare Phase
  The first phase in committing a distributed transaction is the prepare phase.
- Types of Responses in the Prepare Phase
  When a node is told to prepare, it can respond in the different ways.
- Steps in the Prepare Phase
  The prepare phase in the two-phase commit process includes specific steps.

33.3.2.1 About Prepare Phase

The first phase in committing a distributed transaction is the prepare phase.

In this phase, the database does not actually commit or roll back the transaction. Instead, all nodes referenced in a distributed transaction (except the commit point site, described in the "Commit Point Site ") are told to prepare to commit. By preparing, a node:

- Records information in the redo logs so that it can subsequently either commit or roll back the transaction, regardless of intervening failures
- Places a distributed lock on modified tables, which prevents reads

When a node responds to the global coordinator that it is prepared to commit, the prepared node *promises* to either commit or roll back the transaction later, but does not make a unilateral decision on whether to commit or roll back the transaction. The promise means that if an instance failure occurs at this point, the node can use the redo records in the online log to recover the database back to the prepare phase.

Note:

Queries that start after a node has prepared cannot access the associated locked data until all phases complete. The time is insignificant unless a failure occurs (see "Deciding How to Handle In-Doubt Transactions").

33.3.2.2 Types of Responses in the Prepare Phase

When a node is told to prepare, it can respond in the different ways.

Response	Meaning
Prepared	Data on the node has been modified by a statement in the distributed transaction, and the node has successfully prepared.

Response	Meaning
Read-only	No data on the node has been, or can be, modified (only queried), so no preparation is necessary.
Abort	The node cannot successfully prepare.

- Prepared Response
  When a node has successfully prepared, it issues a **prepared message**.
- Read-Only Response
  When a node is asked to prepare, and the SQL statements affecting the database do not change any data on the node, the node responds with a **read-only message**.
- Abort Response
  The **abort message** results in specific actions.

33.3.2.2.1 Prepared Response

When a node has successfully prepared, it issues a **prepared message**.

The message indicates that the node has records of the changes in the online log, so it is prepared either to commit or perform a rollback. The message also guarantees that locks held for the transaction can survive a failure.

33.3.2.2.2 Read-Only Response

When a node is asked to prepare, and the SQL statements affecting the database do not change any data on the node, the node responds with a **read-only message**.

The message indicates that the node will not participate in the commit phase.

There are three cases in which all or part of a distributed transaction is read-only:

Case	Conditions	Consequence
Partially read-only	Any of the following occurs: • Only queries are issued at one or more nodes. • No data is changed. • Changes rolled back due to triggers firing or constraint violations.	The read-only nodes recognize their status when asked to prepare. They give their local coordinators a read-only response. Thus, the commit phase completes faster because the database eliminates read-only nodes from subsequent processing.
Completely read-only with prepare phase	All of following occur: • No data changes. • Transaction is *not* started with `SET TRANSACTION READ ONLY` statement.	All nodes recognize that they are read-only during prepare phase, so no commit phase is required. The global coordinator, not knowing whether all nodes are read-only, must still perform the prepare phase.
Completely read-only without two-phase commit	All of following occur: • No data changes. • Transaction *is* started with `SET TRANSACTION READ ONLY` statement.	Only queries are allowed in the transaction, so global coordinator does not have to perform two-phase commit. Changes by other transactions do not degrade global transaction-level read consistency because of global SCN coordination among nodes. The transaction does not use undo segments.

Note that if a distributed transaction is set to read-only, then it does not use undo segments. If many users connect to the database and their transactions are *not* set to `READ ONLY`, then they allocate undo space even if they are only performing queries.

33.3.2.2.3 Abort Response

The **abort message** results in specific actions.

When a node cannot successfully prepare, it performs the following actions:

1. Releases resources currently held by the transaction and rolls back the local portion of the transaction.
2. Responds to the node that referenced it in the distributed transaction with an abort message.

These actions then propagate to the other nodes involved in the distributed transaction so that they can roll back the transaction and guarantee the integrity of the data in the global database. This response enforces the primary rule of a distributed transaction: *all nodes involved in the transaction either all commit or all roll back the transaction at the same logical time*.

33.3.2.3 Steps in the Prepare Phase

The prepare phase in the two-phase commit process includes specific steps.

To complete the prepare phase, each node excluding the commit point site performs the following steps:

1. The node requests that its **descendants**, that is, the nodes subsequently referenced, prepare to commit.
2. The node checks to see whether the transaction changes data on itself or its descendants. If there is no change to the data, then the node skips the remaining steps and returns a read-only response (see "Read-Only Response").
3. The node allocates the resources it must commit the transaction if data is changed.
4. The node saves redo records corresponding to changes made by the transaction to its redo log.
5. The node guarantees that locks held for the transaction are able to survive a failure.
6. The node responds to the initiating node with a prepared response (see "Prepared Response") or, if its attempt or the attempt of one of its descendents to prepare was unsuccessful, with an abort response (see "Abort Response").

These actions guarantee that the node can subsequently commit or roll back the transaction on the node. The prepared nodes then wait until a `COMMIT` or `ROLLBACK` request is received from the global coordinator.

After the nodes are prepared, the distributed transaction is said to be **in-doubt** (see "In-Doubt Transactions"). It retains in-doubt status until all changes are either committed or rolled back.

33.3.3 Commit Phase

The second phase in committing a distributed transaction is the commit phase. Before this phase occurs, *all* nodes other than the commit point site referenced in the distributed transaction have guaranteed that they are prepared, that is, they have the necessary resources to commit the transaction.

- Steps in the Commit Phase
  The commit phase in the two-phase commit process includes specific steps.
- Guaranteeing Global Database Consistency
  Each committed transaction has an associated system change number (SCN) to uniquely identify the changes made by the SQL statements within that transaction.

33.3.3.1 Steps in the Commit Phase

The commit phase in the two-phase commit process includes specific steps.

The commit phase consists of the following steps:

1. The global coordinator instructs the commit point site to commit.
2. The commit point site commits.
3. The commit point site informs the global coordinator that it has committed.
4. The global and local coordinators send a message to all nodes instructing them to commit the transaction.
5. At each node, the database commits the local portion of the distributed transaction and releases locks.
6. At each node, the database records an additional redo entry in the local redo log, indicating that the transaction has committed.
7. The participating nodes notify the global coordinator that they have committed.

When the commit phase is complete, the data on all nodes of the distributed system is consistent.

33.3.3.2 Guaranteeing Global Database Consistency

Each committed transaction has an associated system change number (SCN) to uniquely identify the changes made by the SQL statements within that transaction.

The SCN functions as an internal timestamp that uniquely identifies a committed version of the database.

In a distributed system, the SCNs of communicating nodes are coordinated when all of the following actions occur:

- A connection occurs using the path described by one or more database links
- A distributed SQL statement executes
- A distributed transaction commits

Among other benefits, the coordination of SCNs among the nodes of a distributed system ensures global read-consistency at both the statement and transaction level. If necessary, global time-based recovery can also be completed.

During the prepare phase, the database determines the highest SCN at all nodes involved in the transaction. The transaction then commits with the high SCN at the commit point site. The commit SCN is then sent to all prepared nodes with the commit decision.

See Also:

"Managing Read Consistency" for information about managing time lag issues in read consistency

33.3.4 Forget Phase

After the participating nodes notify the commit point site that they have committed, the commit point site can forget about the transaction.

The following steps occur:

1. After receiving notice from the global coordinator that all nodes have committed, the commit point site erases status information about this transaction.
2. The commit point site informs the global coordinator that it has erased the status information.
3. The global coordinator erases its own information about the transaction.

33.4 In-Doubt Transactions

A transaction becomes in-doubt if the two-phase commit mechanism fails.

- About In-Doubt Transactions
  The two-phase commit mechanism ensures that all nodes either commit or perform a rollback together. What happens if any of the three phases fails because of a system or network error? The transaction becomes in-doubt.
- Automatic Resolution of In-Doubt Transactions
  In the majority of cases, the database resolves the in-doubt transaction automatically. Assume that there are two nodes, `local` and `remote`, in the following scenarios. The local node is the commit point site. User `scott` connects to `local` and executes and commits a distributed transaction that updates `local` and `remote`.
- Manual Resolution of In-Doubt Transactions
  In some cases, you must resolve an in-doubt transaction manually.
- Relevance of System Change Numbers for In-Doubt Transactions
  A **system change number** (SCN) is an internal timestamp for a committed version of the database. The Oracle Database server uses the SCN clock value to guarantee transaction consistency.

33.4.1 About In-Doubt Transactions

The two-phase commit mechanism ensures that all nodes either commit or perform a rollback together. What happens if any of the three phases fails because of a system or network error? The transaction becomes in-doubt.

Distributed transactions can become in-doubt in the following ways:

- A server system running Oracle Database software crashes
- A network connection between two or more Oracle Databases involved in distributed processing is disconnected

- An unhandled software error occurs

The RECO process automatically resolves in-doubt transactions when the system, network, or software problem is resolved. Until RECO can resolve the transaction, the data is locked for both reads and writes. The database blocks reads because it cannot determine which version of the data to display for a query.

33.4.2 Automatic Resolution of In-Doubt Transactions

In the majority of cases, the database resolves the in-doubt transaction automatically. Assume that there are two nodes, `local` and `remote`, in the following scenarios. The local node is the commit point site. User `scott` connects to `local` and executes and commits a distributed transaction that updates `local` and `remote`.

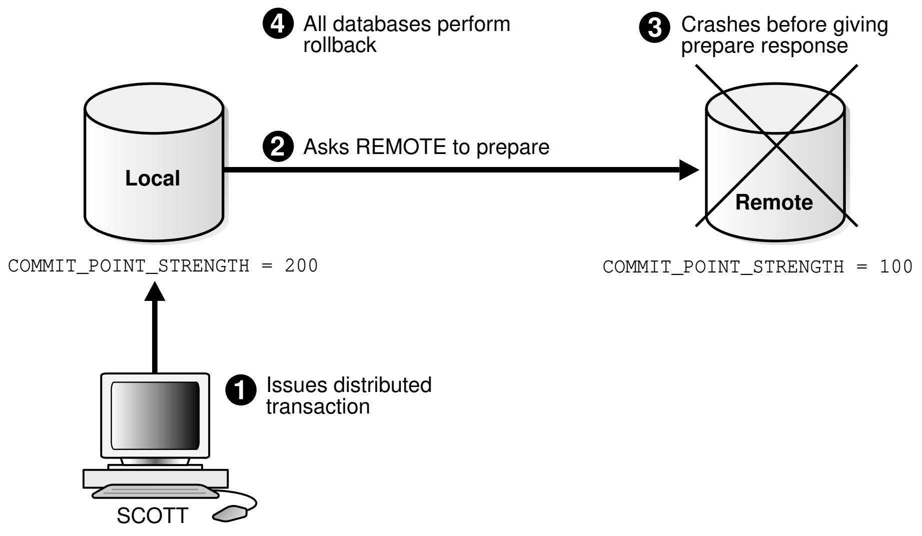
- Failure During the Prepare Phase
  An example illustrates the steps that are followed when there is a failure during the prepare phase of a two-phase transaction.
- Failure During the Commit Phase
  An example illustrates the steps that are followed when there is a failure during the commit phase of a two-phase transaction.

33.4.2.1 Failure During the Prepare Phase

An example illustrates the steps that are followed when there is a failure during the prepare phase of a two-phase transaction.

Figure 33-5 illustrates the sequence of events when there is a failure during the prepare phase of a distributed transaction:

Figure 33-5 Failure During Prepare Phase


4 All databases perform rollback
3 Crashes before giving prepare response
Local
2 Asks REMOTE to prepare
Remote
COMMIT_POINT_STRENGTH = 200
COMMIT_POINT_STRENGTH = 100
1 Issues distributed transaction
SCOTT

The following steps occur:

1. User `SCOTT` connects to `Local` and executes a distributed transaction.

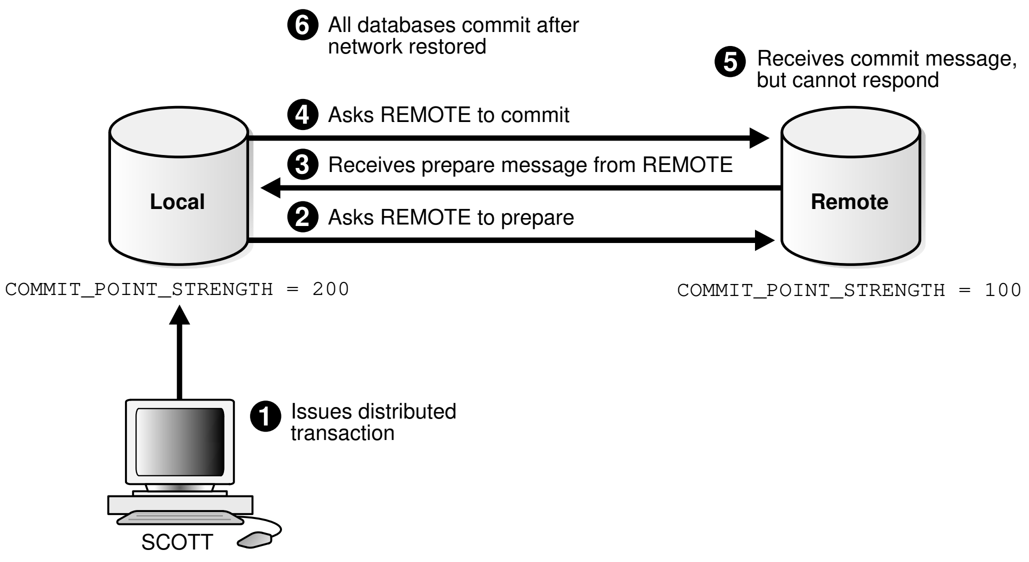
2. The global coordinator, which in this example is also the commit point site, requests all databases other than the commit point site to promise to commit or roll back when told to do so.
3. The `remote` database crashes before issuing the prepare response back to `local`.
4. The transaction is ultimately rolled back on each database by the RECO process when the remote site is restored.

33.4.2.2 Failure During the Commit Phase

An example illustrates the steps that are followed when there is a failure during the commit phase of a two-phase transaction.

Figure 33-6 illustrates the sequence of events when there is a failure during the commit phase of a distributed transaction:

Figure 33-6 Failure During Commit Phase


6 All databases commit after network restored
5 Receives commit message, but cannot respond
4 Asks REMOTE to commit
3 Receives prepare message from REMOTE
2 Asks REMOTE to prepare
Local
Remote
COMMIT_POINT_STRENGTH = 200
COMMIT_POINT_STRENGTH = 100
1 Issues distributed transaction
SCOTT

The following steps occur:

1. User `Scott` connects to `local` and executes a distributed transaction.
2. The global coordinator, which in this case is also the commit point site, requests all databases other than the commit point site to promise to commit or roll back when told to do so.
3. The commit point site receives a prepared message from `remote` saying that it will commit.
4. The commit point site commits the transaction locally, then sends a commit message to `remote` asking it to commit.
5. The `remote` database receives the commit message, but cannot respond because of a network failure.
6. The transaction is ultimately committed on the remote database by the RECO process after the network is restored.

See Also:

"Deciding How to Handle In-Doubt Transactions" for a description of failure situations and how the database resolves intervening failures during two-phase commit

33.4.3 Manual Resolution of In-Doubt Transactions

In some cases, you must resolve an in-doubt transaction manually.

You should only need to resolve an in-doubt transaction manually in the following cases:

- The in-doubt transaction has locks on critical data or undo segments.
- The cause of the system, network, or software failure cannot be repaired quickly.

Resolution of in-doubt transactions can be complicated. The procedure requires that you do the following:

- Identify the transaction identification number for the in-doubt transaction.
- Query the `DBA_2PC_PENDING` and `DBA_2PC_NEIGHBORS` views to determine whether the databases involved in the transaction have committed.
- If necessary, force a commit using the `COMMIT FORCE` statement or a rollback using the `ROLLBACK FORCE` statement.

See Also:

The following sections explain how to resolve in-doubt transactions:

– "Deciding How to Handle In-Doubt Transactions"

– "Manually Overriding In-Doubt Transactions"

33.4.4 Relevance of System Change Numbers for In-Doubt Transactions

A **system change number** (SCN) is an internal timestamp for a committed version of the database. The Oracle Database server uses the SCN clock value to guarantee transaction consistency.

For example, when a user commits a transaction, the database records an SCN for this commit in the redo log.

The database uses SCNs to coordinate distributed transactions among different databases. For example, the database uses SCNs in the following way:

1. An application establishes a connection using a database link.
2. The distributed transaction commits with the highest global SCN among all the databases involved.
3. The commit global SCN is sent to all databases involved in the transaction.

SCNs are important for distributed transactions because they function as a synchronized commit timestamp of a transaction, even if the transaction fails. If a transaction becomes in-doubt, an administrator can use this SCN to coordinate changes made to the global database.

The global SCN for the transaction commit can also be used to identify the transaction later, for example, in distributed recovery.

33.5 Distributed Transaction Processing: Case Study

A case study illustrates distributed transaction processing.

- About the Distributed Transaction Processing Case Study
  In this scenario, a company has separate Oracle Database servers, `sales.example.com` and `warehouse.example.com`. As users insert sales records into the `sales` database, associated records are being updated at the `warehouse` database.
- Stage 1: Client Application Issues DML Statements
  An example illustrates the first stage in distributed transaction processing.
- Stage 2: Oracle Database Determines Commit Point Site
  An example illustrates the second stage in distributed transaction processing.
- Stage 3: Global Coordinator Sends Prepare Response
  An example illustrates the third stage in distributed transaction processing.
- Stage 4: Commit Point Site Commits
  An example illustrates the fourth stage in distributed transaction processing.
- Stage 5: Commit Point Site Informs Global Coordinator of Commit
  An example illustrates the fifth stage in distributed transaction processing.
- Stage 6: Global and Local Coordinators Tell All Nodes to Commit
  An example illustrates the sixth stage in distributed transaction processing.
- Stage 7: Global Coordinator and Commit Point Site Complete the Commit
  An example illustrates the seventh stage in distributed transaction processing.

33.5.1 About the Distributed Transaction Processing Case Study

In this scenario, a company has separate Oracle Database servers, `sales.example.com` and `warehouse.example.com`. As users insert sales records into the `sales` database, associated records are being updated at the `warehouse` database.

This case study of distributed processing illustrates:

- The definition of a session tree
- How a commit point site is determined
- When prepare messages are sent
- When a transaction actually commits
- What information is stored locally about the transaction

33.5.2 Stage 1: Client Application Issues DML Statements

An example illustrates the first stage in distributed transaction processing.

At the Sales department, a salesperson uses SQL*Plus to enter a sales order and then commit it. The application issues several SQL statements to enter the order into the `sales` database and update the inventory in the `warehouse` database:

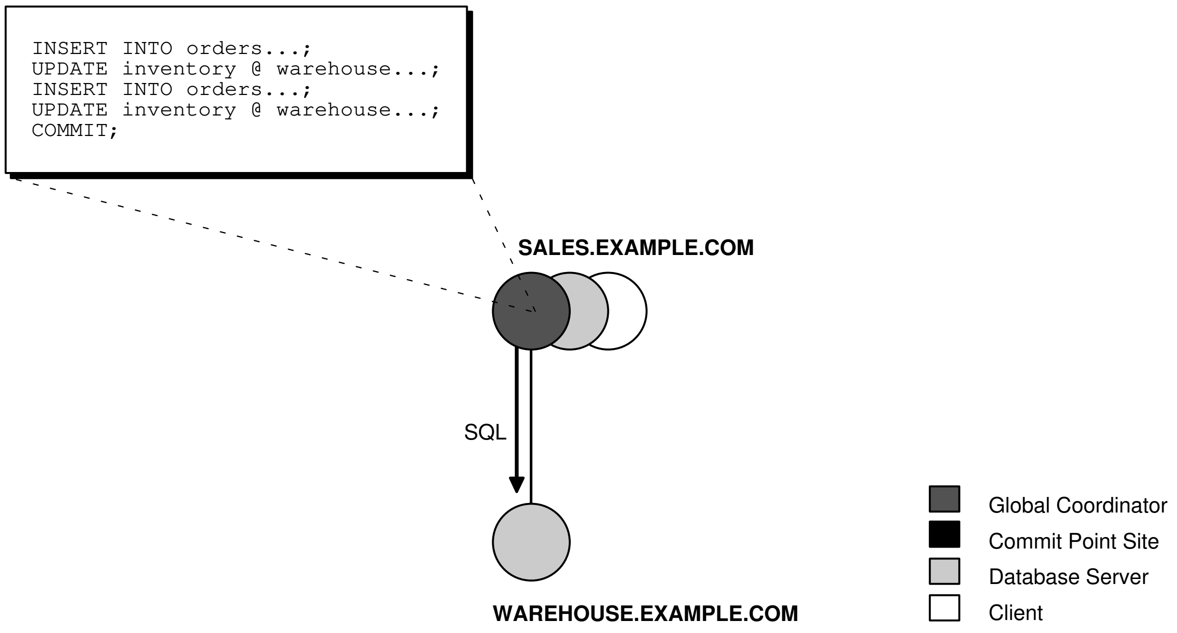
```
CONNECT scott@sales.example.com ...;
INSERT INTO orders ...;
```

```
UPDATE inventory@warehouse.example.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.example.com ...;
COMMIT;
```

These SQL statements are part of a single distributed transaction, guaranteeing that all issued SQL statements succeed or fail as a unit. Treating the statements as a unit prevents the possibility of an order being placed and then inventory not being updated to reflect the order. In effect, the transaction guarantees the consistency of data in the global database.

As each of the SQL statements in the transaction executes, the session tree is defined, as shown in Figure 33-7.

Figure 33-7 Defining the Session Tree



```
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
INSERT INTO orders...;
UPDATE inventory @ warehouse...;
COMMIT;
```

Note the following aspects of the transaction:

- An order entry application running on the `sales` database initiates the transaction. Therefore, `sales.example.com` is the global coordinator for the distributed transaction.
- The order entry application inserts a new sales record into the `sales` database and updates the inventory at the warehouse. Therefore, the nodes `sales.example.com` and `warehouse.example.com` are both database servers.
- Because `sales.example.com` updates the inventory, it is a client of `warehouse.example.com`.

This stage completes the definition of the session tree for this distributed transaction. Each node in the tree has acquired the necessary data locks to execute the SQL statements that reference local data. These locks remain even after the SQL statements have been executed until the two-phase commit is completed.

33.5.3 Stage 2: Oracle Database Determines Commit Point Site

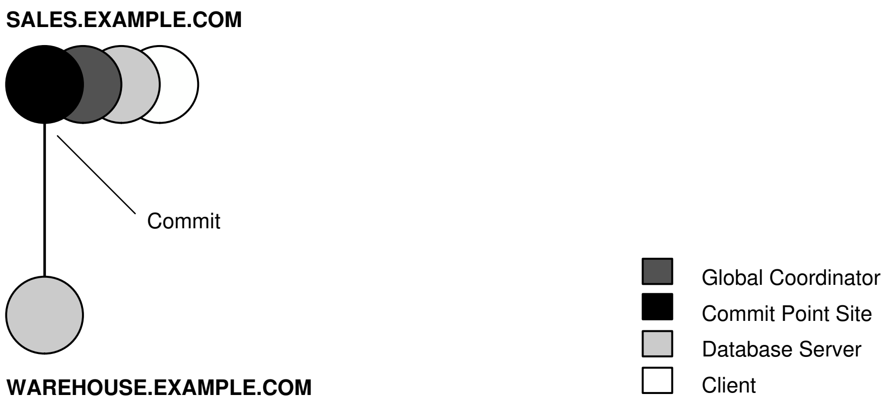
An example illustrates the second stage in distributed transaction processing.

The database determines the commit point site immediately following the `COMMIT` statement. `sales.example.com`, the global coordinator, is determined to be the commit point site, as shown in Figure 33-8.

See Also:

"Commit Point Strength" for more information about how the commit point site is determined

Figure 33-8 Determining the Commit Point Site



33.5.4 Stage 3: Global Coordinator Sends Prepare Response

An example illustrates the third stage in distributed transaction processing.

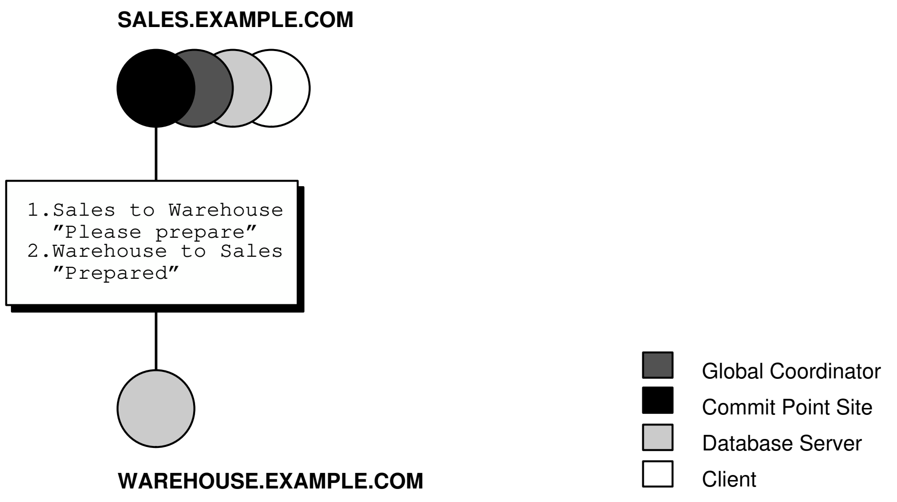
The prepare stage involves the following steps:

1. After the database determines the commit point site, the global coordinator sends the prepare message to all directly referenced nodes of the session tree, *excluding* the commit point site. In this example, `warehouse.example.com` is the only node asked to prepare.
2. Node `warehouse.example.com` tries to prepare. If a node can guarantee that it can commit the locally dependent part of the transaction and can record the commit information in its local redo log, then the node can successfully prepare. In this example, only `warehouse.example.com` receives a prepare message because `sales.example.com` is the commit point site.
3. Node `warehouse.example.com` responds to `sales.example.com` with a prepared message.

As each node prepares, it sends a message back to the node that asked it to prepare. Depending on the responses, one of the following can happen:

- If *any* of the nodes asked to prepare responds with an abort message to the global coordinator, then the global coordinator tells all nodes to roll back the transaction, and the operation is completed.
- If *all* nodes asked to prepare respond with a prepared or a read-only message to the global coordinator, that is, they have successfully prepared, then the global coordinator asks the commit point site to commit the transaction.

Figure 33-9 Sending and Acknowledging the Prepare Message



33.5.5 Stage 4: Commit Point Site Commits

An example illustrates the fourth stage in distributed transaction processing.

The committing of the transaction by the commit point site involves the following steps:

1. Node `sales.example.com`, receiving acknowledgment that `warehouse.example.com` is prepared, instructs the commit point site to commit the transaction.
2. The commit point site now commits the transaction locally and records this fact in its local redo log.

Even if `warehouse.example.com` has not yet committed, the outcome of this transaction is predetermined. In other words, the transaction *will* be committed at all nodes even if the ability of a given node to commit is delayed.

33.5.6 Stage 5: Commit Point Site Informs Global Coordinator of Commit

An example illustrates the fifth stage in distributed transaction processing.

This stage involves the following steps:

1. The commit point site tells the global coordinator that the transaction has committed. Because the commit point site and global coordinator are the same node in this example, no operation is required. The commit point site knows that the transaction is committed because it recorded this fact in its online log.
2. The global coordinator confirms that the transaction has been committed on all other nodes involved in the distributed transaction.

33.5.7 Stage 6: Global and Local Coordinators Tell All Nodes to Commit

An example illustrates the sixth stage in distributed transaction processing.

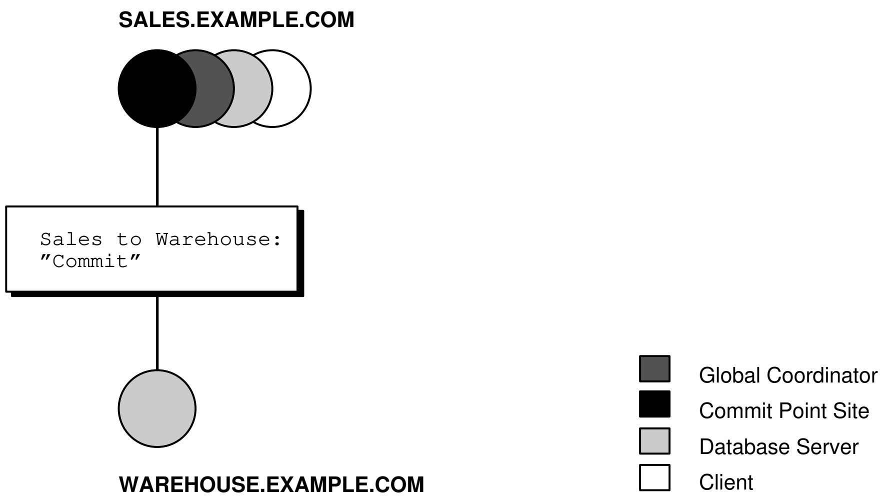
The committing of the transaction by all the nodes in the transaction involves the following steps:

1. After the global coordinator has been informed of the commit at the commit point site, it tells all other directly referenced nodes to commit.

2. In turn, any local coordinators instruct their servers to commit, and so on.
3. Each node, including the global coordinator, commits the transaction and records appropriate redo log entries locally. As each node commits, the resource locks that were being held locally for that transaction are released.

In Figure 33-10, `sales.example.com`, which is both the commit point site and the global coordinator, has already committed the transaction locally. `sales` now instructs `warehouse.example.com` to commit the transaction.

Figure 33-10 Instructing Nodes to Commit



33.5.8 Stage 7: Global Coordinator and Commit Point Site Complete the Commit

An example illustrates the seventh stage in distributed transaction processing.

The completion of the commit of the transaction occurs in the following steps:

1. After all referenced nodes and the global coordinator have committed the transaction, the global coordinator informs the commit point site of this fact.
2. The commit point site, which has been waiting for this message, erases the status information about this distributed transaction.
3. The commit point site informs the global coordinator that it is finished. In other words, the commit point site forgets about committing the distributed transaction. This action is permissible because all nodes involved in the two-phase commit have committed the transaction successfully, so they will never have to determine its status in the future.
4. The global coordinator finalizes the transaction by forgetting about the transaction itself.

After the completion of the `COMMIT` phase, the distributed transaction is itself complete. The steps described are accomplished automatically and in a fraction of a second.

34
Managing Distributed Transactions

Managing distributed transactions includes tasks such as specifying the comment point strength of a node, naming transactions, and managing in-doubt transactions.

- Specifying the Commit Point Strength of a Node
  The database with the highest commit point strength determines which node commits first in a distributed transaction.
- Naming Transactions
  You can name a transaction. This is useful for identifying a specific distributed transaction and replaces the use of the `COMMIT COMMENT` statement for this purpose.
- Viewing Information About Distributed Transactions
  The data dictionary of each database stores information about all open distributed transactions. You can use data dictionary tables and views to gain information about the transactions.
- Deciding How to Handle In-Doubt Transactions
  A transaction is in-doubt when there is a failure during any aspect of the two-phase commit. Distributed transactions become in-doubt in the following ways: a server system running Oracle Database software crashes, a network connection between two or more Oracle Databases involved in distributed processing is disconnected, or an unhandled software error occurs.
- Manually Overriding In-Doubt Transactions
  Use the `COMMIT` or `ROLLBACK` statement with the `FORCE` option and a text string that indicates either the local or global transaction ID of the in-doubt transaction to commit.
- Purging Pending Rows from the Data Dictionary
  You can purge pending rows from the data dictionary for in-doubt transactions.
- Manually Committing an In-Doubt Transaction: Example
  An example illustrates manually committing an in-doubt transaction.
- Data Access Failures Due to Locks
  When you issue a SQL statement, the database attempts to lock the resources needed to successfully execute the statement. If the requested data is currently held by statements of other uncommitted transactions, however, and remains locked for a long time, a timeout occurs.
- Simulating Distributed Transaction Failure
  You can force the failure of a distributed transaction to observe RECO automatically resolving the local portion of the transaction or to practice manually resolving in-doubt distributed transactions and observing the results.
- Managing Read Consistency
  An important restriction exists in the Oracle Database implementation of distributed read consistency.

34.1 Specifying the Commit Point Strength of a Node

The database with the highest commit point strength determines which node commits first in a distributed transaction.

When specifying a commit point strength for each node, ensure that the most critical server will be non-blocking if a failure occurs during a prepare or commit phase. The `COMMIT_POINT_STRENGTH` initialization parameter determines the commit point strength of a node.

The default value is operating system-dependent. The range of values is any integer from 0 to 255. For example, to set the commit point strength of a database to 200, include the following line in the database initialization parameter file:

```
COMMIT_POINT_STRENGTH = 200
```

The commit point strength is only used to determine the commit point site in a distributed transaction.

When setting the commit point strength for a database, note the following considerations:

- Because the commit point site stores information about the status of the transaction, the commit point site should not be a node that is frequently unreliable or unavailable in case other nodes need information about transaction status.
- Set the commit point strength for a database relative to the amount of critical shared data in the database. For example, a database on a mainframe computer usually shares more data among users than a database on a PC. Therefore, set the commit point strength of the mainframe to a higher value than the PC.

See Also:

"Commit Point Site " for a conceptual overview of commit points

34.2 Naming Transactions

You can name a transaction. This is useful for identifying a specific distributed transaction and replaces the use of the `COMMIT COMMENT` statement for this purpose.

To name a transaction, use the `SET TRANSACTION...NAME` statement. For example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
     NAME 'update inventory checkpoint 0';
```

This example shows that the user started a new transaction with isolation level equal to `SERIALIZABLE` and named it `'update inventory checkpoint 0'`.

For distributed transactions, the name is sent to participating sites when a transaction is committed. If a `COMMIT COMMENT` exists, it is ignored when a transaction name exists.

The transaction name is displayed in the `NAME` column of the `V$TRANSACTION` view, and in the `TRAN_COMMENT` field of the `DBA_2PC_PENDING` view when the transaction is committed.

34.3 Viewing Information About Distributed Transactions

The data dictionary of each database stores information about all open distributed transactions. You can use data dictionary tables and views to gain information about the transactions.

- Determining the ID Number and Status of Prepared Transactions
  Use the `DBA_2PC_PENDING` view to determine the global commit number for a particular transaction ID. You can use this global commit number when manually resolving an in-doubt transaction.
- Tracing the Session Tree of In-Doubt Transactions
  The `DBA_2PC_NEIGHBORS` view shows which in-doubt transactions are incoming from a remote client and which are outgoing to a remote server.

34.3.1 Determining the ID Number and Status of Prepared Transactions

Use the `DBA_2PC_PENDING` view to determine the global commit number for a particular transaction ID. You can use this global commit number when manually resolving an in-doubt transaction.

The following view shows the database links that have been defined at the local database and stored in the data dictionary:

View	Purpose
`DBA_2PC_PENDING`	Lists all in-doubt distributed transactions. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

The following table shows the most relevant columns (for a description of all the columns in the view, see *Oracle Database Reference*):

Table 34-1   DBA_2PC_PENDING

Column	Description
`LOCAL_TRAN_ID`	Local transaction identifier in the format *integer.integer.integer*. **Note:** When the `LOCAL_TRAN_ID` and the `GLOBAL_TRAN_ID` for a connection are the same, the node is the global coordinator of the transaction.
`GLOBAL_TRAN_ID`	Global database identifier in the format *global_db_name.db_hex_id.local_tran_id*, where *db_hex_id* is an eight-character hexadecimal value used to uniquely identify the database. This common transaction ID is the same on every node for a distributed transaction. **Note:** When the `LOCAL_TRAN_ID` and the `GLOBAL_TRAN_ID` for a connection are the same, the node is the global coordinator of the transaction.

Table 34-1 (Cont.) DBA_2PC_PENDING

Column	Description
`STATE`	`STATE` can have the following values: • Collecting This category normally applies only to the global coordinator or local coordinators. The node is currently collecting information from other database servers before it can decide whether it can prepare. • Prepared The node has prepared and may or may not have acknowledged this to its local coordinator with a prepared message. However, no commit request has been received. The node remains prepared, holding any local resource locks necessary for the transaction to commit. • Committed The node (any type) has committed the transaction, but other nodes involved in the transaction may not have done the same. That is, the transaction is still pending at one or more nodes. • Forced Commit A pending transaction can be forced to commit at the discretion of a database administrator. This entry occurs if a transaction is manually committed at a local node. • Forced termination (rollback) A pending transaction can be forced to roll back at the discretion of a database administrator. This entry occurs if this transaction is manually rolled back at a local node.
`MIXED`	`YES` means that part of the transaction was committed on one node and rolled back on another node.
`TRAN_COMMENT`	Transaction comment or, if using transaction naming, the transaction name is placed here when the transaction is committed.
`HOST`	Name of the host system.
`COMMIT#`	Global commit number for committed transactions.

Execute the following script, named `pending_txn_script`, to query pertinent information in `DBA_2PC_PENDING` (sample output included):

```
COL LOCAL_TRAN_ID FORMAT A13
COL GLOBAL_TRAN_ID FORMAT A30
COL STATE FORMAT A8
COL MIXED FORMAT A3
COL HOST FORMAT A10
COL COMMIT# FORMAT A10

SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE, MIXED, HOST, COMMIT#
   FROM DBA_2PC_PENDING
/

SQL> @pending_txn_script

LOCAL_TRAN_ID GLOBAL_TRAN_ID                 STATE    MIX HOST       COMMIT#
------------- ------------------------------ -------- --- ---------- ----------
1.15.870      HQ.EXAMPLE.COM.ef192da4.1.15.870  commit   no  dlsun183   115499
```

This output indicates that local transaction `1.15.870` has been committed on this node, but it may be pending on one or more other nodes. Because `LOCAL_TRAN_ID` and the local part of `GLOBAL_TRAN_ID` are the same, the node is the global coordinator of the transaction.

34.3.2 Tracing the Session Tree of In-Doubt Transactions

The `DBA_2PC_NEIGHBORS` view shows which in-doubt transactions are incoming from a remote client and which are outgoing to a remote server.

View	Purpose
`DBA_2PC_NEIGHBORS`	Lists all incoming (from remote client) and outgoing (to remote server) in-doubt distributed transactions. It also indicates whether the local node is the commit point site in the transaction. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

When a transaction is in-doubt, you may need to determine which nodes performed which roles in the session tree. Use to this view to determine:

- All the incoming and outgoing connections for a given transaction
- Whether the node is the commit point site in a given transaction
- Whether the node is a global coordinator in a given transaction (because its local transaction ID and global transaction ID are the same)

The following table shows the most relevant columns (for an account of all the columns in the view, see *Oracle Database Reference*):

Table 34-2 DBA_2PC_NEIGHBORS

Column	Description
`LOCAL_TRAN_ID`	Local transaction identifier with the format *integer*.*integer*.*integer*. **Note:** When `LOCAL_TRAN_ID` and `GLOBAL_TRAN_ID.DBA_2PC_PENDING` for a connection are the same, the node is the global coordinator of the transaction.
`IN_OUT`	`IN` for incoming transactions; `OUT` for outgoing transactions.
`DATABASE`	For incoming transactions, the name of the client database that requested information from this local node; for outgoing transactions, the name of the database link used to access information on a remote server.
`DBUSER_OWNER`	For incoming transactions, the local account used to connect by the remote database link; for outgoing transactions, the owner of the database link.
`INTERFACE`	`C` is a commit message; `N` is either a message indicating a prepared state or a request for a read-only commit. When `IN_OUT` is `OUT`, `C` means that the child at the remote end of the connection is the commit point site and knows whether to commit or terminate. `N` means that the local node is informing the remote node that it is prepared. When `IN_OUT` is `IN`, `C` means that the local node or a database at the remote end of an outgoing connection is the commit point site. `N` means that the remote node is informing the local node that it is prepared.

Execute the following script, named `neighbors_script`, to query pertinent information in `DBA_2PC_PENDING` (sample output included):

```
COL LOCAL_TRAN_ID FORMAT A13
COL IN_OUT FORMAT A6
COL DATABASE FORMAT A25
COL DBUSER_OWNER FORMAT A15
COL INTERFACE FORMAT A3
SELECT LOCAL_TRAN_ID, IN_OUT, DATABASE, DBUSER_OWNER, INTERFACE
   FROM DBA_2PC_NEIGHBORS
/

SQL> CONNECT SYS@hq.example.com AS SYSDBA
SQL> @neighbors_script

LOCAL_TRAN_ID IN_OUT DATABASE                  DBUSER_OWNER    INT
------------- ------ ------------------------- --------------- ---
1.15.870      out    SALES.EXAMPLE.COM            SYS             C
```

This output indicates that the local node sent an outgoing request to remote server `sales` to commit transaction `1.15.870`. If `sales` committed the transaction but no other node did, then you know that `sales` is the commit point site, because the commit point site always commits first.

34.4 Deciding How to Handle In-Doubt Transactions

A transaction is in-doubt when there is a failure during any aspect of the two-phase commit. Distributed transactions become in-doubt in the following ways: a server system running Oracle Database software crashes, a network connection between two or more Oracle Databases involved in distributed processing is disconnected, or an unhandled software error occurs.

You can manually force the commit or rollback of a local, in-doubt distributed transaction. Because this operation can generate consistency problems, perform it only when specific conditions exist.

- Discovering Problems with a Two-Phase Commit
  Error messages inform applications when there are problems with distributed transactions.
- Determining Whether to Perform a Manual Override
  You should override an in-doubt transaction only under certain conditions.
- Analyzing the Transaction Data
  If you decide to force the transaction to complete, then analyze the available information.

See Also:

In-Doubt Transactions

34.4.1 Discovering Problems with a Two-Phase Commit

Error messages inform applications when there are problems with distributed transactions.

The user application that commits a distributed transaction is informed of a problem by one of the following error messages:

```
ORA-02050: transaction ID rolled back,
           some remote dbs may be in-doubt
ORA-02053: transaction ID committed,
           some remote dbs may be in-doubt
ORA-02054: transaction ID in-doubt
```

A robust application should save information about a transaction if it receives any of the preceding errors. This information can be used later if manual distributed transaction recovery is desired.

No action is required by the administrator of any node that has one or more in-doubt distributed transactions due to a network or system failure. The automatic recovery features of the database transparently complete any in-doubt transaction so that the same outcome occurs on all nodes of a session tree (that is, all commit or all roll back) after the network or system failure is resolved.

In extended outages, however, you can force the commit or rollback of a transaction to release any locked data. Applications must account for such possibilities.

34.4.2 Determining Whether to Perform a Manual Override

You should override an in-doubt transaction only under certain conditions.

Override a specific in-doubt transaction manually *only* when one of the following conditions exists:

- The in-doubt transaction locks data that is required by other transactions. This situation occurs when the `ORA-01591` error message interferes with user transactions.
- An in-doubt transaction prevents the extents of an undo segment from being used by other transactions. The first portion of the local transaction ID of an in-doubt distributed transaction corresponds to the ID of the undo segment, as listed by the data dictionary view `DBA_2PC_PENDING`.
- The failure preventing the two-phase commit phases to complete cannot be corrected in an acceptable time period. Examples of such cases include a telecommunication network that has been damaged or a damaged database that requires a long recovery time.

Normally, you should decide to locally force an in-doubt distributed transaction in consultation with administrators at other locations. A wrong decision can lead to database inconsistencies that can be difficult to trace and that you must manually correct.

If none of these conditions apply, *always* allow the automatic recovery features of the database to complete the transaction. If any of these conditions are met, however, consider a local override of the in-doubt transaction.

34.4.3 Analyzing the Transaction Data

If you decide to force the transaction to complete, then analyze the available information.

- Find a Node that Committed or Rolled Back
  Use the `DBA_2PC_PENDING` view to find a node that has either committed or rolled back the transaction.
- Look for Transaction Comments
  See if any information is given in the `TRAN_COMMENT` column of `DBA_2PC_PENDING` for the distributed transaction.

- Look for Transaction Advice
  See if any information is given in the ADVICE column of DBA_2PC_PENDING for the distributed transaction.

34.4.3.1 Find a Node that Committed or Rolled Back

Use the DBA_2PC_PENDING view to find a node that has either committed or rolled back the transaction.

If you can find a node that has already resolved the transaction, then you can follow the action taken at that node.

34.4.3.2 Look for Transaction Comments

See if any information is given in the TRAN_COMMENT column of DBA_2PC_PENDING for the distributed transaction.

Comments are included in the COMMENT clause of the COMMIT statement, or if transaction naming is used, the transaction name is placed in the TRAN_COMMENT field when the transaction is committed.

For example, the comment of an in-doubt distributed transaction can indicate the origin of the transaction and what type of transaction it is:

```
COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
```

The SET TRANSACTION...NAME statement could also have been used (and is preferable) to provide this information in a transaction name.

See Also:

"Naming Transactions"

34.4.3.3 Look for Transaction Advice

See if any information is given in the ADVICE column of DBA_2PC_PENDING for the distributed transaction.

An application can prescribe advice about whether to force the commit or force the rollback of separate parts of a distributed transaction with the ADVISE clause of the ALTER SESSION statement.

The advice sent during the prepare phase to each node is the advice in effect at the time the most recent DML statement executed at that database in the current transaction.

For example, consider a distributed transaction that moves an employee record from the emp table at one node to the emp table at another node. The transaction can protect the record--even when administrators independently force the in-doubt transaction at each node--by including the following sequence of SQL statements:

```
ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@hq ... ;    /*advice to commit at HQ */
ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@sales ... ; /*advice to roll back at SALES*/
```

```
ALTER SESSION ADVISE NOTHING;
```

If you manually force the in-doubt transaction following the given advice, the worst that can happen is that each node has a copy of the employee record; the record cannot disappear.

34.5 Manually Overriding In-Doubt Transactions

Use the `COMMIT` or `ROLLBACK` statement with the `FORCE` option and a text string that indicates either the local or global transaction ID of the in-doubt transaction to commit.

Note:

In all examples, the transaction is committed or rolled back on the local node, and the local pending transaction table records a value of forced commit or forced termination for the `STATE` column the row for this transaction.

- Manually Committing an In-Doubt Transaction
  You can manually commit an in-doubt transaction using a transaction ID or an SCN.
- Manually Rolling Back an In-Doubt Transaction
  You can roll back an in-doubt transaction using the transaction ID.

34.5.1 Manually Committing an In-Doubt Transaction

You can manually commit an in-doubt transaction using a transaction ID or an SCN.

- Privileges Required to Commit an In-Doubt Transaction
  Before attempting to commit the transaction, ensure that you have the proper privileges.
- Committing Using Only the Transaction ID
  You can commit an in-doubt transaction using the transaction ID.
- Committing Using an SCN
  You can commit an in-doubt transaction using an SCN.

34.5.1.1 Privileges Required to Commit an In-Doubt Transaction

Before attempting to commit the transaction, ensure that you have the proper privileges.

Note the following requirements:

User Committing the Transaction	Privilege Required
You	`FORCE TRANSACTION`
Another user	`FORCE ANY TRANSACTION`

34.5.1.2 Committing Using Only the Transaction ID

You can commit an in-doubt transaction using the transaction ID.

The following SQL statement commits an in-doubt transaction:

```
COMMIT FORCE 'transaction_id';
```

The variable *transaction_id* is the identifier of the transaction as specified in either the `LOCAL_TRAN_ID` or `GLOBAL_TRAN_ID` columns of the `DBA_2PC_PENDING` data dictionary view.

For example, assume that you query `DBA_2PC_PENDING` and determine that `LOCAL_TRAN_ID` for a distributed transaction is `1:45.13`.

You then issue the following SQL statement to force the commit of this in-doubt transaction:

```
COMMIT FORCE '1.45.13';
```

34.5.1.3 Committing Using an SCN

You can commit an in-doubt transaction using an SCN.

Optionally, you can specify the SCN for the transaction when forcing a transaction to commit. This feature lets you commit an in-doubt transaction with the SCN assigned when it was committed at other nodes.

Consequently, you maintain the synchronized commit time of the distributed transaction even if there is a failure. Specify an SCN only when you can determine the SCN of the same transaction already committed at another node.

For example, assume you want to manually commit a transaction with the following global transaction ID:

```
SALES.EXAMPLE.COM.55d1c563.1.93.29
```

First, query the `DBA_2PC_PENDING` view of a remote database also involved with the transaction in question. Note the SCN used for the commit of the transaction at that node. Specify the SCN when committing the transaction at the local node. For example, if the SCN is `829381993`, issue:

```
COMMIT FORCE 'SALES.EXAMPLE.COM.55d1c563.1.93.29', 829381993;
```

See Also:

Oracle Database SQL Language Reference for more information about using the `COMMIT` statement

34.5.2 Manually Rolling Back an In-Doubt Transaction

You can roll back an in-doubt transaction using the transaction ID.

Before attempting to roll back the in-doubt distributed transaction, ensure that you have the proper privileges. Note the following requirements:

User Committing the Transaction	Privilege Required
You	`FORCE TRANSACTION`
Another user	`FORCE ANY TRANSACTION`

The following SQL statement rolls back an in-doubt transaction:

```
ROLLBACK FORCE 'transaction_id';
```

The variable *transaction_id* is the identifier of the transaction as specified in either the `LOCAL_TRAN_ID` or `GLOBAL_TRAN_ID` columns of the `DBA_2PC_PENDING` data dictionary view.

For example, to roll back the in-doubt transaction with the local transaction ID of `2.9.4`, use the following statement:

```
ROLLBACK FORCE '2.9.4';
```

Note:

You cannot roll back an in-doubt transaction to a savepoint.

See Also:

Oracle Database SQL Language Reference for more information about using the `ROLLBACK` statement

34.6 Purging Pending Rows from the Data Dictionary

You can purge pending rows from the data dictionary for in-doubt transactions.

- About Purging Pending Rows from the Data Dictionary
  You can purge pending rows from the data dictionary for in-doubt transactions using the `DBMS_TRANSACTION.PURGE_MIXED` procedure or the `DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY` procedure.
- Executing the PURGE_LOST_DB_ENTRY Procedure
  Use the `DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY` procedure to clean up entries for in-doubt transactions in the data dictionary.
- Determining When to Use DBMS_TRANSACTION
  You typically should perform a specific action based on the state of a distributed transaction.

34.6.1 About Purging Pending Rows from the Data Dictionary

You can purge pending rows from the data dictionary for in-doubt transactions using the `DBMS_TRANSACTION.PURGE_MIXED` procedure or the `DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY` procedure.

Before RECO recovers an in-doubt transaction, the transaction appears in `DBA_2PC_PENDING.STATE` as `COLLECTING`, `COMMITTED`, or `PREPARED`. If you force an in-doubt transaction using `COMMIT FORCE` or `ROLLBACK FORCE`, then the states `FORCED COMMIT` or `FORCED ROLLBACK` may appear.

Automatic recovery normally deletes entries in these states. The only exception is when recovery discovers a forced transaction that is in a state inconsistent with other sites in the transaction. In this case, the entry can be left in the table, and the `MIXED` column in `DBA_2PC_PENDING` has a value of `YES`. These entries can be cleaned up with the `DBMS_TRANSACTION.PURGE_MIXED` procedure.

If automatic recovery is not possible because a remote database has been permanently lost, then recovery cannot identify the re-created database because it receives a new database ID when it is re-created. In this case, you must use the `PURGE_LOST_DB_ENTRY` procedure in the `DBMS_TRANSACTION` package to clean up the entries. The entries do not hold up database resources, so there is no urgency in cleaning them up.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_TRANSACTION` package

34.6.2 Executing the PURGE_LOST_DB_ENTRY Procedure

Use the `DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY` procedure to clean up entries for in-doubt transactions in the data dictionary.

To manually remove an entry from the data dictionary, use the following syntax (where *trans_id* is the identifier for the transaction):

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('trans_id');
```

For example, to purge pending distributed transaction `1.44.99`, enter the following statement in SQL*Plus:

```
EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('1.44.99');
```

Execute this procedure only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples include:

- Total loss of the remote database
- Reconfiguration in software resulting in loss of two-phase commit capability
- Loss of information from an external transaction coordinator such as a TPMonitor

34.6.3 Determining When to Use DBMS_TRANSACTION

You typically should perform a specific action based on the state of a distributed transaction.

The following tables indicates what the various states indicate about the distributed transaction and what the administrator's action should be:

STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Collecting	Rolled back	Rolled back	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Committed	Committed	Committed	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Prepared	Unknown	Prepared	None	Force commit or rollback

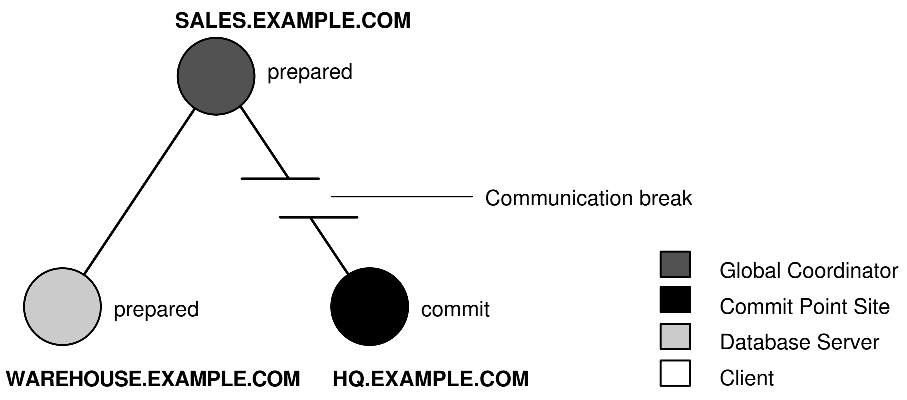
STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Forced commit	Unknown	Committed	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Forced rollback	Unknown	Rolled back	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Forced commit	Mixed	Committed	Manually remove inconsistencies then use `PURGE_MIXED`	-
Forced rollback	Mixed	Rolled back	Manually remove inconsistencies then use `PURGE_MIXED`	-

34.7 Manually Committing an In-Doubt Transaction: Example

An example illustrates manually committing an in-doubt transaction.

Figure 34-1, illustrates a failure during the commit of a distributed transaction. In this failure case, the prepare phase completes. During the commit phase, however, the commit confirmation of the commit point site never reaches the global coordinator, even though the commit point site committed the transaction. Inventory data is locked and cannot be accessed because the in-doubt transaction is critical to other transactions. Further, the locks must be held until the in-doubt transaction either commits or rolls back.

Figure 34-1 Example of an In-Doubt Distributed Transaction



You can manually force the local portion of the in-doubt transaction.

- Step 1: Record User Feedback
  An example illustrates recording user feedback for an in-doubt transaction.
- Step 2: Query DBA_2PC_PENDING
  An example illustrates querying the `DBA_2PC_PENDING` data dictionary view for information about in-doubt transactions.

- Step 3: Query DBA_2PC_NEIGHBORS on Local Node
  The purpose of this step is to climb the session tree so that you find coordinators, eventually reaching the global coordinator.
- Step 4: Querying Data Dictionary Views on All Nodes
  At this point, you can contact the administrator at the located nodes and ask each person to repeat Steps 2 and 3 using the global transaction ID.
- Step 5: Commit the In-Doubt Transaction
  Use the global ID to commit the in-doubt transaction.
- Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING
  After you manually force a transaction to commit or roll back, the corresponding row in the pending transaction table remains. The state of the transaction is changed depending on how you forced the transaction.

34.7.1 Step 1: Record User Feedback

An example illustrates recording user feedback for an in-doubt transaction.

The users of the local database system that conflict with the locks of the in-doubt transaction receive the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

In this case, `1.21.17` is the local transaction ID of the in-doubt distributed transaction. You should request and record this ID number from users that report problems to identify which in-doubt transactions should be forced.

34.7.2 Step 2: Query DBA_2PC_PENDING

An example illustrates querying the `DBA_2PC_PENDING` data dictionary view for information about in-doubt transactions.

After connecting with SQL*Plus to `warehouse`, query the local `DBA_2PC_PENDING` data dictionary view to gain information about the in-doubt transaction:

```
CONNECT SYS@warehouse.example.com AS SYSDBA
SELECT * FROM DBA_2PC_PENDING WHERE LOCAL_TRAN_ID = '1.21.17';
```

The database returns the following information:

```
Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.21.17
GLOBAL_TRAN_ID         SALES.EXAMPLE.COM.55d1c563.1.93.29
STATE                  prepared
MIXED                  no
ADVICE
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31-MAY-91
FORCE_TIME
RETRY_TIME             31-MAY-91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   system1
DB_USER                SWILLIAMS
COMMIT#
```

- Determining the Global Transaction ID
  The global transaction ID is the common transaction ID that is the same on every node for a distributed transaction.
- Determining the State of the Transaction
  The `STATE` column of the `DBA_2PC_PENDING` data dictionary view shows the state of the transaction.
- Looking for Comments or Advice
  The `TRANS_COMMENT` column of the `DBA_2PC_PENDING` data dictionary view shows the comment included for the transaction, while the `ADVICE` column provides advice.

34.7.2.1 Determining the Global Transaction ID

The global transaction ID is the common transaction ID that is the same on every node for a distributed transaction.

It is of the form:

global_database_name.hhhhhhhh.local_transaction_id

where:

- *global_database_name* is the database name of the global coordinator.
- *hhhhhhhh* is the internal database identifier of the global coordinator (in hexadecimal).
- *local_transaction_id* is the corresponding local transaction ID assigned on the global coordinator.

Note that the last portion of the global transaction ID and the local transaction ID match at the global coordinator. In the example, you can tell that `warehouse` is *not* the global coordinator because these numbers do not match:

```
LOCAL_TRAN_ID           1.21.17
GLOBAL_TRAN_ID          ... 1.93.29
```

34.7.2.2 Determining the State of the Transaction

The `STATE` column of the `DBA_2PC_PENDING` data dictionary view shows the state of the transaction.

The transaction on this node is in a prepared state:

```
STATE           prepared
```

Therefore, `warehouse` waits for its coordinator to send either a commit or a rollback request.

34.7.2.3 Looking for Comments or Advice

The `TRANS_COMMENT` column of the `DBA_2PC_PENDING` data dictionary view shows the comment included for the transaction, while the `ADVICE` column provides advice.

The transaction comment or advice can include information about this transaction. If so, use this comment to your advantage. In this example, the origin and transaction type is in the transaction comment:

```
TRAN_COMMENT            Sales/New Order/Trans_type 10B
```

It could also be provided as a transaction name with a `SET TRANSACTION...NAME` statement.

This information can reveal something that helps you decide whether to commit or rollback the local portion of the transaction. If useful comments do not accompany an in-doubt transaction, you must complete some extra administrative work to trace the session tree and find a node that has resolved the transaction.

34.7.3 Step 3: Query DBA_2PC_NEIGHBORS on Local Node

The purpose of this step is to climb the session tree so that you find coordinators, eventually reaching the global coordinator.

Along the way, you may find a coordinator that has resolved the transaction. If not, you can eventually work your way to the commit point site, which will always have resolved the in-doubt transaction. To trace the session tree, query the `DBA_2PC_NEIGHBORS` view on each node.

In this case, you query this view on the `warehouse` database:

```
CONNECT SYS@warehouse.example.com AS SYSDBA
SELECT * FROM DBA_2PC_NEIGHBORS
  WHERE LOCAL_TRAN_ID = '1.21.17'
  ORDER BY SESS#, IN_OUT;

Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.21.17
IN_OUT                 in
DATABASE               SALES.EXAMPLE.COM
DBUSER_OWNER           SWILLIAMS
INTERFACE              N
DBID                   000003F4
SESS#                  1
BRANCH                 0100
```

- Obtaining Database Role and Database Link Information
  The `DBA_2PC_NEIGHBORS` view provides information about connections associated with an in-doubt transaction.
- Determining the Commit Point Site
  The `INTERFACE` column tells whether the local node or a subordinate node is the commit point site.

34.7.3.1 Obtaining Database Role and Database Link Information

The `DBA_2PC_NEIGHBORS` view provides information about connections associated with an in-doubt transaction.

Information for each connection is different, based on whether the connection is **inbound** (`IN_OUT = in`) or **outbound** (`IN_OUT = out`):

IN_OUT	Meaning	DATABASE	DBUSER_OWNER
in	Your node is a server of another node.	Lists the name of the client database that connected to your node.	Lists the local account for the database link connection that corresponds to the in-doubt transaction.
out	Your node is a client of other servers.	Lists the name of the database link that connects to the remote node.	Lists the owner of the database link for the in-doubt transaction.

In this example, the `IN_OUT` column reveals that the `warehouse` database is a server for the `sales` client, as specified in the DATABASE column:

```
IN_OUT                  in
DATABASE                SALES.EXAMPLE.COM
```

The connection to `warehouse` was established through a database link from the `swilliams` account, as shown by the `DBUSER_OWNER` column:

```
DBUSER_OWNER            SWILLIAMS
```

34.7.3.2 Determining the Commit Point Site

The `INTERFACE` column tells whether the local node or a subordinate node is the commit point site.

```
INTERFACE               N
```

Neither `warehouse` nor any of its descendants is the commit point site, as shown by the `INTERFACE` column.

34.7.4 Step 4: Querying Data Dictionary Views on All Nodes

At this point, you can contact the administrator at the located nodes and ask each person to repeat Steps 2 and 3 using the global transaction ID.

> **Note:**
>
> If you can directly connect to these nodes with another network, you can repeat Steps 2 and 3 yourself.

For example, the following results are returned when Steps 2 and 3 are performed at `sales` and `hq`.

- Checking the Status of Pending Transactions at sales
  At this stage, the `sales` administrator queries the `DBA_2PC_PENDING` data dictionary view.
- Determining the Coordinators and Commit Point Site at sales
  Next, the `sales` administrator queries `DBA_2PC_NEIGHBORS` to determine the global and local coordinators as well as the commit point site.
- Checking the Status of Pending Transactions at HQ
  At this stage, the `hq` administrator queries the `DBA_2PC_PENDING` data dictionary view.

34.7.4.1 Checking the Status of Pending Transactions at sales

At this stage, the `sales` administrator queries the `DBA_2PC_PENDING` data dictionary view.

```
SQL> CONNECT SYS@sales.example.com AS SYSDBA
SQL> SELECT * FROM DBA_2PC_PENDING
   > WHERE GLOBAL_TRAN_ID = 'SALES.EXAMPLE.COM.55d1c563.1.93.29';

Column Name             Value
---------------------- --------------------------------------
LOCAL_TRAN_ID           1.93.29
```

```
GLOBAL_TRAN_ID          SALES.EXAMPLE.COM.55d1c563.1.93.29
STATE                   prepared
MIXED                   no
ADVICE
TRAN_COMMENT            Sales/New Order/Trans_type 10B
FAIL_TIME               31-MAY-91
FORCE_TIME
RETRY_TIME              31-MAY-91
OS_USER                 SWILLIAMS
OS_TERMINAL             TWA139:
HOST                    system1
DB_USER                 SWILLIAMS
COMMIT#
```

34.7.4.2 Determining the Coordinators and Commit Point Site at sales

Next, the `sales` administrator queries `DBA_2PC_NEIGHBORS` to determine the global and local coordinators as well as the commit point site.

```
SELECT * FROM DBA_2PC_NEIGHBORS
   WHERE GLOBAL_TRAN_ID = 'SALES.EXAMPLE.COM.55d1c563.1.93.29'
   ORDER BY SESS#, IN_OUT;
```

This query returns three rows:

- The connection to `warehouse`
- The connection to `hq`
- The connection established by the user

Reformatted information corresponding to the rows for the `warehouse` connection appears below:

```
Column Name             Value
---------------------- --------------------------------------
LOCAL_TRAN_ID           1.93.29
IN_OUT                  OUT
DATABASE                WAREHOUSE.EXAMPLE.COM
DBUSER_OWNER            SWILLIAMS
INTERFACE               N
DBID                    55d1c563
SESS#                   1
BRANCH                  1
```

Reformatted information corresponding to the rows for the `hq` connection appears below:

```
Column Name             Value
---------------------- --------------------------------------
LOCAL_TRAN_ID           1.93.29
IN_OUT                  OUT
DATABASE                HQ.EXAMPLE.COM
DBUSER_OWNER            ALLEN
INTERFACE               C
DBID                    00000390
SESS#                   1
BRANCH                  1
```

The information from the previous queries reveal the following:

- `sales` is the global coordinator because the local transaction ID and global transaction ID match.

- Two outbound connections are established from this node, but no inbound connections. `sales` is not the server of another node.
- `hq` or one of its servers is the commit point site.

34.7.4.3 Checking the Status of Pending Transactions at HQ

At this stage, the `hq` administrator queries the `DBA_2PC_PENDING` data dictionary view.

```
SELECT * FROM DBA_2PC_PENDING@hq.example.com
   WHERE GLOBAL_TRAN_ID = 'SALES.EXAMPLE.COM.55d1c563.1.93.29';

Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.45.13
GLOBAL_TRAN_ID         SALES.EXAMPLE.COM.55d1c563.1.93.29
STATE                  COMMIT
MIXED                  NO
ACTION
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31-MAY-91
FORCE_TIME
RETRY_TIME             31-MAY-91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   SYSTEM1
DB_USER                SWILLIAMS
COMMIT#                129314
```

At this point, you have found a node that resolved the transaction. As the view reveals, it has been committed and assigned a commit ID number:

```
STATE                  COMMIT
COMMIT#                129314
```

Therefore, you can force the in-doubt transaction to commit at your local database. It is a good idea to contact any other administrators you know that could also benefit from your investigation.

34.7.5 Step 5: Commit the In-Doubt Transaction

Use the global ID to commit the in-doubt transaction.

You contact the administrator of the `sales` database, who manually commits the in-doubt transaction using the global ID:

```
SQL> CONNECT SYS@sales.example.com AS SYSDBA
SQL> COMMIT FORCE 'SALES.EXAMPLE.COM.55d1c563.1.93.29';
```

As administrator of the `warehouse` database, you manually commit the in-doubt transaction using the global ID:

```
SQL> CONNECT SYS@warehouse.example.com AS SYSDBA
SQL> COMMIT FORCE 'SALES.EXAMPLE.COM.55d1c563.1.93.29';
```

34.7.6 Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING

After you manually force a transaction to commit or roll back, the corresponding row in the pending transaction table remains. The state of the transaction is changed depending on how you forced the transaction.

Every Oracle Database has a **pending transaction table**. This is a special table that stores information about distributed transactions as they proceed through the two-phase commit phases. You can query the pending transaction table of a database through the `DBA_2PC_PENDING` data dictionary view (see "Determining the ID Number and Status of Prepared Transactions").

Also of particular interest in the pending transaction table is the mixed outcome flag as indicated in `DBA_2PC_PENDING.MIXED`. You can make the wrong choice if a pending transaction is forced to commit or roll back. For example, the local administrator rolls back the transaction, but the other nodes commit it. Incorrect decisions are detected automatically, and the damage flag for the corresponding pending transaction record is set (`MIXED=yes`).

The RECO (Recoverer) background process uses the information in the pending transaction table to finalize the status of in-doubt transactions. You can also use the information in the pending transaction table to manually override the automatic recovery procedures for pending distributed transactions.

All transactions automatically resolved by RECO are removed from the pending transaction table. Additionally, all information about in-doubt transactions correctly resolved by an administrator (as checked when RECO reestablishes communication) are automatically removed from the pending transaction table. However, all rows resolved by an administrator that result in a mixed outcome across nodes remain in the pending transaction table of all involved nodes until they are manually deleted using `DBMS_TRANSACTIONS.PURGE_MIXED`.

34.8 Data Access Failures Due to Locks

When you issue a SQL statement, the database attempts to lock the resources needed to successfully execute the statement. If the requested data is currently held by statements of other uncommitted transactions, however, and remains locked for a long time, a timeout occurs.

- Transaction Timeouts
  A DML statement that requires locks on a remote database can be blocked if another transaction own locks on the requested data.
- Locks from In-Doubt Transactions
  A query or DML statement that requires locks on a local database can be blocked indefinitely due to the locked resources of an in-doubt distributed transaction.

34.8.1 Transaction Timeouts

A DML statement that requires locks on a remote database can be blocked if another transaction own locks on the requested data.

If these locks continue to block the requesting SQL statement, then the following sequence of events occurs:

1. A timeout occurs.
2. The database rolls back the statement.

3. The database returns this error message to the user:

```
ORA-02049: time-out: distributed transaction waiting for lock
```

Because the transaction did not modify data, no actions are necessary as a result of the timeout. Applications should proceed as if a deadlock has been encountered. The user who executed the statement can try to reexecute the statement later. If the lock persists, then the user should contact an administrator to report the problem.

34.8.2 Locks from In-Doubt Transactions

A query or DML statement that requires locks on a local database can be blocked indefinitely due to the locked resources of an in-doubt distributed transaction.

In this case, the database issues the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction identifier
```

In this case, the database rolls back the SQL statement immediately. The user who executed the statement can try to reexecute the statement later. If the lock persists, the user should contact an administrator to report the problem, *including* the ID of the in-doubt distributed transaction.

The chances of these situations occurring are rare considering the low probability of failures during the critical portions of the two-phase commit. Even if such a failure occurs, and assuming quick recovery from a network or system failure, problems are automatically resolved without manual intervention. Thus, problems usually resolve before they can be detected by users or database administrators.

34.9 Simulating Distributed Transaction Failure

You can force the failure of a distributed transaction to observe RECO automatically resolving the local portion of the transaction or to practice manually resolving in-doubt distributed transactions and observing the results.

- Forcing a Distributed Transaction to Fail
  You can include comments in the `COMMENT` parameter of the `COMMIT` statement.
- Disabling and Enabling RECO
  The RECO background process of an Oracle Database instance automatically resolves failures involving distributed transactions. At exponentially growing time intervals, the RECO background process of a node attempts to recover the local portion of an in-doubt distributed transaction.

34.9.1 Forcing a Distributed Transaction to Fail

You can include comments in the `COMMENT` parameter of the `COMMIT` statement.

To intentionally induce a failure during the two-phase commit phases of a distributed transaction, include the following comment in the `COMMENT` parameter:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

where *n* is one of the following integers:

n	Effect
1	Crash commit point after collect

n	Effect
2	Crash non-commit-point site after collect
3	Crash before prepare (non-commit-point site)
4	Crash after prepare (non-commit-point site)
5	Crash commit point site before commit
6	Crash commit point site after commit
7	Crash non-commit-point site before commit
8	Crash non-commit-point site after commit
9	Crash commit point site before forget
10	Crash non-commit-point site before forget

For example, the following statement returns the following messages if the local commit point strength is greater than the remote commit point strength and both nodes are updated:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-7';

ORA-02054: transaction 1.93.29 in-doubt
ORA-02059: ORA_CRASH_TEST_7 in commit comment
```

At this point, the in-doubt distributed transaction appears in the `DBA_2PC_PENDING` view. If enabled, RECO automatically resolves the transaction.

34.9.2 Disabling and Enabling RECO

The RECO background process of an Oracle Database instance automatically resolves failures involving distributed transactions. At exponentially growing time intervals, the RECO background process of a node attempts to recover the local portion of an in-doubt distributed transaction.

RECO can use an existing connection or establish a new connection to other nodes involved in the failed transaction. When a connection is established, RECO automatically resolves all in-doubt transactions. Rows corresponding to any resolved in-doubt transactions are automatically removed from the pending transaction table of each database.

You can enable and disable RECO using the `ALTER SYSTEM` statement with the `ENABLE`/`DISABLE DISTRIBUTED RECOVERY` options. For example, you can temporarily disable RECO to force the failure of a two-phase commit and manually resolve the in-doubt transaction.

The following statement disables RECO:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Alternatively, the following statement enables RECO so that in-doubt transactions are automatically resolved:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

34.10 Managing Read Consistency

An important restriction exists in the Oracle Database implementation of distributed read consistency.

The problem arises because each system has its own SCN, which you can view as the database internal timestamp. The Oracle Database server uses the SCN to decide which version of data is returned from a query.

The SCNs in a distributed transaction are synchronized at the end of each remote SQL statement and at the start and end of each transaction. Between two nodes that have heavy traffic and especially distributed updates, the synchronization is frequent. Nevertheless, no practical way exists to keep SCNs in a distributed system absolutely synchronized: a window always exists in which one node may have an SCN that is somewhat in the past with respect to the SCN of another node.

Because of the SCN gap, you can execute a query that uses a slightly old snapshot, so that the most recent changes to the remote database are not seen. In accordance with read consistency, a query can therefore retrieve consistent, but out-of-date data. Note that all data retrieved by the query will be from the old SCN, so that if a locally executed update transaction updates two tables at a remote node, then data selected from both tables in the next remote access contain data before the update.

One consequence of the SCN gap is that two consecutive `SELECT` statements can retrieve different data even though no DML has been executed between the two statements. For example, you can issue an update statement and then commit the update on the remote database. When you issue a `SELECT` statement on a view based on this remote table, the view does not show the update to the row. The next time that you issue the `SELECT` statement, the update is present.

You can use the following techniques to ensure that the SCNs of the two systems are synchronized just before a query:

- Because SCNs are synchronized at the end of a remote query, precede each remote query with a dummy remote query to the same site, for example, `SELECT * FROM DUAL@REMOTE`.
- Because SCNs are synchronized at the start of every remote transaction, commit or roll back the current transaction before issuing the remote query.

Part VI

Managing Read-Only Materialized Views

Read-only materialized views can provide read-only access to the master table's data. You can create and manage read-only materialized views.

35 Read-Only Materialized View Concepts

Understand the concepts related to read-only materialized views.

- Replication Databases
  Replication is the process of copying and maintaining database objects, such as tables, in multiple databases that comprise a distributed database system. One method that supports replication is read-only materialized views.
- Read-Only Materialized Views
  A **read-only materialized view** contains a complete or partial copy of a target master table from a single point in time. A partial copy can include a subset of row, a subset of columns, or both.
- The Uses of Materialized Views
  You can use materialized views to achieve goals such as easing network loads, enabling data subsetting, and enabling disconnected computing.
- Available Materialized Views
  Available materialized views include primary key materialized views, object materialized views, `ROWID` materialized views, and complex materialized views.
- Users and Privileges Related to Materialized Views
  The users related to materialized views include the creator, the refresher, and the owner. The privileges required to perform operations on materialized views depend on the type of user performing the operation.
- Data Subsetting with Materialized Views
  You can use row subsetting and column subsetting to configure materialized views reflect a subset of the data in the master table.
- Materialized View Refresh
  To ensure that a materialized view is consistent with its master table, you must **refresh** the materialized view periodically.
- Refresh Groups
  When it is important for materialized views to be transactionally consistent with each other, you can organize them into **refresh groups**.
- Materialized View Log
  A **materialized view log** is a table at the database that contains materialized view's master table. It records all of the DML changes to the master table.
- Materialized Views and User-Defined Data Types
  There are special considerations for materialized views with user-defined data types.
- Materialized View Registration at a Master Database
  At the master database, an Oracle Database automatically registers information about a materialized view based on its master table(s).

35.1 Replication Databases

Replication is the process of copying and maintaining database objects, such as tables, in multiple databases that comprise a distributed database system. One method that supports replication is read-only materialized views.

Replication environments support two basic types of databases: **master databases** and **materialized view databases**. Materialized view databases contain an image, or materialized view, of the table data from a certain point in time. The table on which a materialized view is defined is the **master table** for the materialized view. Typically, a materialized view is refreshed periodically to synchronize it with its master table.

You can organize materialized views into **refresh groups**. Materialized views in a refresh group are refreshed at the same time to ensure that the data in all materialized views in the refresh group correspond to the same transactionally consistent point in time.

Note:

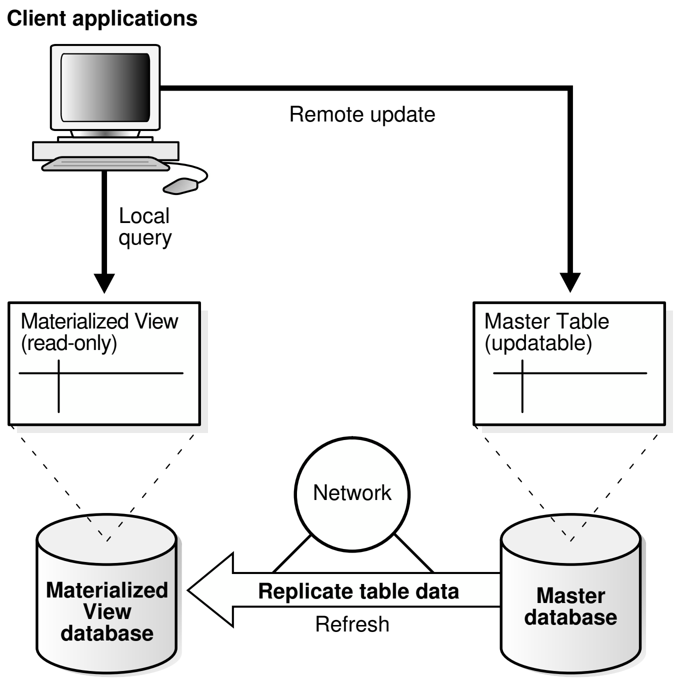
Oracle GoldenGate is Oracle's full-featured solution for replication. See the Oracle GoldenGate documentation for more information.

35.2 Read-Only Materialized Views

A **read-only materialized view** contains a complete or partial copy of a target master table from a single point in time. A partial copy can include a subset of row, a subset of columns, or both.

Read-only materialized views can provide read-only access to the master table's data. Applications can query data from read-only materialized views to avoid network access to the master database, regardless of network availability. However, applications throughout the system must access data at the master database to perform data manipulation language changes (DML). Figure 35-1 illustrates basic, read-only replication.

Figure 35-1 Read-Only Materialized View



Materialized views provide the following benefits:

- Enable local access, which improves response times and availability
- When the materialized view is in a different database than its source, offload queries from the master database, because users can query the local materialized view instead
- Increase data security by enabling you to replicate only a selected subset of the target master's data set

Users can synchronize (refresh) read-only materialized views on demand. When users refresh read-only materialized views, they receive any changes that happened on the master table since the last refresh.

35.3 The Uses of Materialized Views

You can use materialized views to achieve goals such as easing network loads, enabling data subsetting, and enabling disconnected computing.

- Ease Network Loads
  If one of your goals is to reduce network loads, then you can use materialized views to distribute your corporate database to regional databases.
- Enable Data Subsetting
  Materialized views enable you to replicate data based on column- and row-level subsetting.
- Enable Disconnected Computing
  Materialized views do not require a dedicated network connection.

35.3.1 Ease Network Loads

If one of your goals is to reduce network loads, then you can use materialized views to distribute your corporate database to regional databases.

Instead of the entire company accessing a single database server, user load is distributed across multiple database servers. To decrease the amount of data that is replicated, a materialized view can be a subset of a master table.

35.3.2 Enable Data Subsetting

Materialized views enable you to replicate data based on column- and row-level subsetting.

Data subsetting enables you to replicate information that pertains only to a particular database. For example, if you have a regional sales office, then you might replicate only the data that is needed in that region, thereby cutting down on unnecessary network traffic.

Both row and column subsetting enable you to create materialized views that contain a partial copy of the data at a master table. Row subsetting enables you to include only the rows that are needed from the masters in the materialized views by using a `WHERE` clause. Column subsetting enables you to include only the columns that are needed from the masters in the materialized views. You do this by specifying particular columns in the `SELECT` statement during materialized view creation.

35.3.3 Enable Disconnected Computing

Materialized views do not require a dedicated network connection.

Though you have the option of automating the refresh process by scheduling a job, you can manually refresh your materialized view on-demand, which is an ideal solution for queries running on a laptop computer.

35.4 Available Materialized Views

Available materialized views include primary key materialized views, object materialized views, `ROWID` materialized views, and complex materialized views.

- About the Available Materialized Views
  Oracle offers several types of read-only materialized views to meet the needs of many different replication (and nonreplication) situations.
- Primary Key Materialized Views
  Primary key materialized views are the default type of materialized view.
- Object Materialized Views
  If a materialized view is based on an object table and is created using the `OF` *type* clause, then the materialized view is called an object materialized view.
- ROWID Materialized Views
  A `ROWID` materialized view is based on the physical row identifiers (rowids) of the rows in a master table.
- Complex Materialized Views
  Complex materialized views cannot be fast refreshed.

35.4.1 About the Available Materialized Views

Oracle offers several types of read-only materialized views to meet the needs of many different replication (and nonreplication) situations.

Whenever you create a materialized view, regardless of its type, always specify the schema name of the table owner in the query for the materialized view. For example, consider the following `CREATE MATERIALIZED VIEW` statement:

```
CREATE MATERIALIZED VIEW hr.employees
  AS SELECT * FROM hr.employees@orc1.example.com;
```

Here, the schema `hr` is specified in the query.

35.4.2 Primary Key Materialized Views

Primary key materialized views are the default type of materialized view.

The following is an example of a SQL statement for creating a primary key materialized view:

```
CREATE MATERIALIZED VIEW oe.customers WITH PRIMARY KEY
  AS SELECT * FROM oe.customers@orc1.example.com;
```

Because primary key materialized views are the default, the following statement also results in a primary key materialized view:

```
CREATE MATERIALIZED VIEW oe.customers
  AS SELECT * FROM oe.customers@orc1.example.com;
```

Primary key materialized views can contain a subquery so that you can create a subset of rows at the remote materialized view database. A subquery is a query imbedded within the primary query, so that you have multiple `SELECT` statements in the `CREATE MATERIALIZED VIEW` statement. This subquery can be as simple as a basic `WHERE` clause or as complex as a multilevel `WHERE EXISTS` clause. Primary key materialized views that contain a selected class of subqueries can still be incrementally (or fast) refreshed, if each master referenced has a materialized view log. A fast refresh uses materialized view logs to update only the rows that have changed since the last refresh.

The following materialized view is created with a `WHERE` clause containing a subquery:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
 SELECT * FROM oe.orders@orc1.example.com o
 WHERE EXISTS
   (SELECT * FROM oe.customers@orc1.example.com c
    WHERE o.customer_id = c.customer_id AND c.credit_limit > 10000);
```

This type of materialized view is called a subquery materialized view.

Note:

To create this `oe.orders` materialized view, `credit_limit` must be logged in the master table's materialized view log. See "Logging Columns in a Materialized View Log" for more information.

See Also:

- "Materialized Views with Subqueries" for more information about materialized views with subqueries
- "Refresh Types" for more information about fast refresh
- "Materialized View Log" for more information about materialized view logs
- *Oracle Database SQL Language Reference* for more information about subqueries

35.4.3 Object Materialized Views

If a materialized view is based on an object table and is created using the `OF` *type* clause, then the materialized view is called an object materialized view.

An object materialized view is structured in the same way as an object table. That is, an object materialized view is composed of row objects, and each row object is identified by an object identifier (OID) column.

See Also:

- "Materialized Views Based on Object Tables"
- "Creating Read-Only Materialized Views" for an example that creates an object materialized view

35.4.4 ROWID Materialized Views

A `ROWID` materialized view is based on the physical row identifiers (rowids) of the rows in a master table.

`ROWID` materialized views can be used for materialized views based on master tables that do not have a primary key, or for materialized views that do not include all primary key columns of the master tables.

The following is an example of a `CREATE MATERIALIZED VIEW` statement that creates a `ROWID` materialized view:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH WITH ROWID AS
 SELECT * FROM oe.orders@orc1.example.com;
```

See Also:

- "Materialized View Log" for more information about the differences between a `ROWID` and primary key materialized view
- *Oracle Database SQL Language Reference* for more information about the `WITH ROWID` clause in the `CREATE MATERIALIZED VIEW` statement

35.4.5 Complex Materialized Views

Complex materialized views cannot be fast refreshed.

- About Complex Materialized Views
  To be fast refreshed, the defining query for a materialized view must observe certain restrictions.
- A Comparison of Simple and Complex Materialized Views
  For certain applications, you might want to consider using a complex materialized view.

35.4.5.1 About Complex Materialized Views

To be fast refreshed, the defining query for a materialized view must observe certain restrictions.

If you require a materialized view whose defining query is more general and cannot observe the restrictions, then the materialized view is complex and cannot be fast refreshed.

Specifically, a materialized view is considered complex when the defining query of the materialized view contains any of the following:

- A CONNECT BY clause

  For example, the following statement creates a complex materialized view:

  ```
  CREATE MATERIALIZED VIEW hr.emp_hierarchy AS
    SELECT LPAD(' ', 4*(LEVEL-1))||email USERNAME
      FROM hr.employees@orc1.example.com START WITH manager_id IS NULL
      CONNECT BY PRIOR employee_id = manager_id;
  ```

- An INTERSECT, MINUS, or UNION ALL set operation

  For example, the following statement creates a complex materialized view because it has a UNION ALL set operation:

  ```
  CREATE MATERIALIZED VIEW hr.mview_employees AS
    SELECT employees.employee_id, employees.email
    FROM hr.employees@orc1.example.com
  UNION ALL
    SELECT new_employees.employee_id, new_employees.email
    FROM hr.new_employees@orc1.example.com;
  ```

- The DISTINCT or UNIQUE keyword

  For example, the following statement creates a complex materialized view:

  ```
  CREATE MATERIALIZED VIEW hr.employee_depts AS
    SELECT DISTINCT department_id FROM hr.employees@orc1.example.com
    ORDER BY department_id;
  ```

- In some cases, an aggregate function, although it is possible to have an aggregate function in the defining query and still have a simple materialized view

  For example, the following statement creates a complex materialized view:

  ```
  CREATE MATERIALIZED VIEW hr.average_sal AS
    SELECT AVG(salary) "Average" FROM hr.employees@orc1.example.com;
  ```

- In some cases, joins other than those in a subquery, although it is possible to have joins in the defining query and still have a simple materialized view

  For example, the following statement creates a complex materialized view:

  ```
  CREATE MATERIALIZED VIEW hr.emp_join_dep AS
    SELECT last_name
    FROM hr.employees@orc1.example.com e, hr.departments@orc1.example.com d
    WHERE e.department_id = d.department_id;
  ```

- In some cases, a UNION operation

  Specifically, a materialized view with a UNION operation is complex if any one of these conditions is true:

  – Any query within the UNION is complex. The previous bullet items specify when a query makes a materialized view complex.

  – The outermost SELECT list columns do not match for the queries in the UNION. In the following example, the first query only has order_total in the outermost SELECT list while the second query has customer_id in the outermost SELECT list. Therefore, the materialized view is complex.

    ```
    CREATE MATERIALIZED VIEW oe.orders AS
      SELECT order_total
      FROM oe.orders@orc1.example.com o
      WHERE EXISTS
        (SELECT cust_first_name, cust_last_name
           FROM oe.customers@orc1.example.com c
           WHERE o.customer_id = c.customer_id
    ```

```
       AND c.credit_limit > 50)
UNION
  SELECT customer_id
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT cust_first_name, cust_last_name
       FROM oe.customers@orc1.example.com c
       WHERE o.customer_id = c.customer_id
       AND c.account_mgr_id = 30);
```

The innermost `SELECT` list has no bearing on whether a materialized view is complex. In the previous example, the innermost `SELECT` list is `cust_first_name` and `cust_last_name` for both queries in the `UNION`.

- Clauses that do not follow the requirements detailed in "Restrictions for Materialized Views with Subqueries"

Note:

If possible, you should avoid using complex materialized views because they cannot be fast refreshed, which might degrade performance.

See Also:

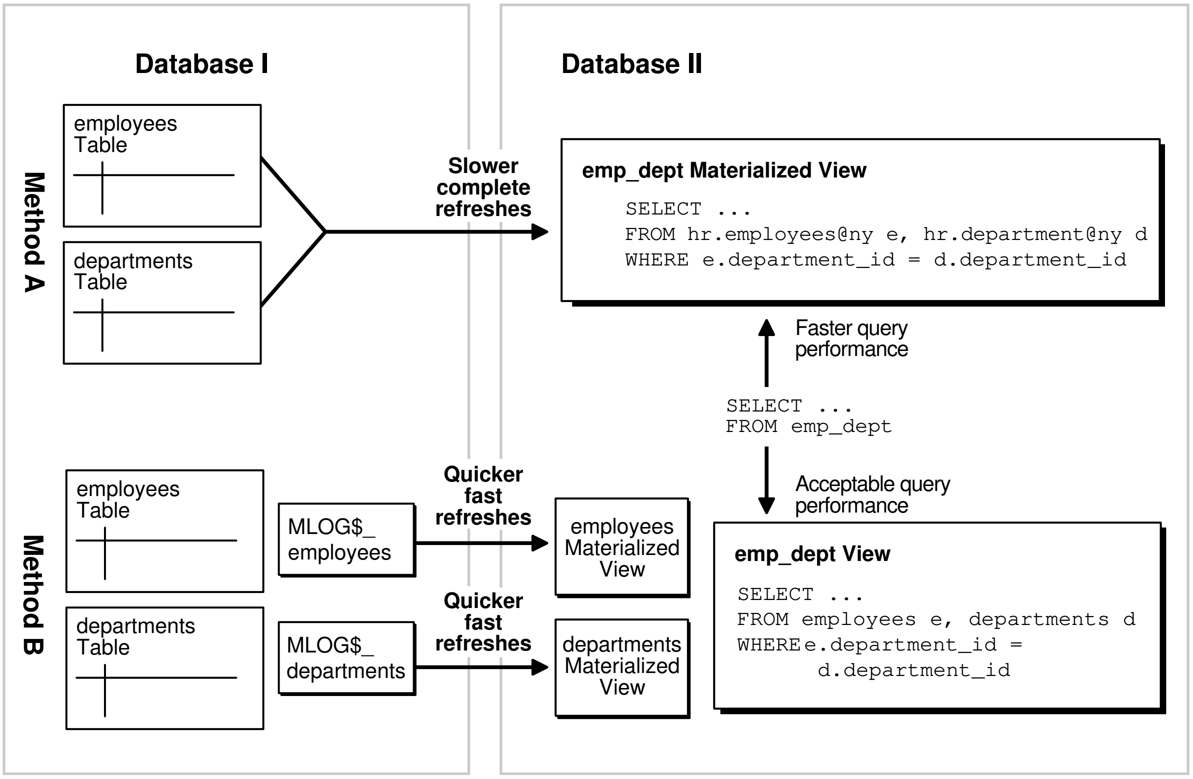
- "Refresh Process"
- *Oracle Database Data Warehousing Guide* for information about materialized views with aggregate functions and joins
- *Oracle Database SQL Language Reference* for more information about the `CONNECT BY` clause, set operations, the `DISTINCT` keyword, and aggregate functions

35.4.5.2 A Comparison of Simple and Complex Materialized Views

For certain applications, you might want to consider using a complex materialized view.

Figure 35-2 and the following text discuss some issues that you should consider.

Figure 35-2 Comparison of Simple and Complex Materialized Views



- **Complex Materialized View**: Method A in Figure 35-2 shows a complex materialized view. The materialized view in Database II exhibits efficient query performance because the join operation was completed during the materialized view's refresh. However, complete refreshes must be performed because the materialized view is complex, and these refreshes will probably be slower than fast refreshes.
- **Simple Materialized Views with a Joined View**: Method B in Figure 35-2 shows two simple materialized views in Database II, as well as a view that performs the join in the materialized view's database. Query performance against the view would not be as good as the query performance against the complex materialized view in Method A. However, the simple materialized views can be refreshed more efficiently using fast refresh and materialized view logs.

In summary, to decide which method to use:

- If you refresh rarely and want faster query performance, then use Method A (complex materialized view).
- If you refresh regularly and can sacrifice query performance, then use Method B (simple materialized view).

35.5 Users and Privileges Related to Materialized Views

The users related to materialized views include the creator, the refresher, and the owner. The privileges required to perform operations on materialized views depend on the type of user performing the operation.

- Required Privileges for Materialized View Operations
  Three distinct types of users perform operations on materialized views.

- Creator Is Owner
  If the creator of a materialized view also owns the materialized view, then this user must have the required privileges to create a materialized view.
- Creator Is Not Owner
  If the creator of a materialized view is not the owner, then certain privileges must be granted to the creator and to the owner to create a materialized view.
- Refresher Is Owner
  If the refresher of a materialized view also owns the materialized view, then this user must have the required privileges to create the materialized view.
- Refresher Is Not Owner
  If the refresher of a materialized view is not the owner, certain privileges must be granted to the refresher and to the owner.

35.5.1 Required Privileges for Materialized View Operations

Three distinct types of users perform operations on materialized views.

These users are:

- **Creator:** The user who creates the materialized view.
- **Refresher:** The user who refreshes the materialized view.
- **Owner:** The user who owns the materialized view. The materialized view resides in this user's schema.

One user can perform all of these operations on a particular materialized view. However, in some replication environments, different users perform these operations on a particular materialized view. The privileges required to perform these operations depend on whether the same user performs them or different users perform them.

If the owner of a materialized view at the materialized view database has a private database link to the master database, then the database link connects to the owner of the master table at the master database. Otherwise, the normal rules for connections through database links apply.

Note:

The following sections do not cover the requirements necessary to create materialized views with query rewrite enabled. See the *Oracle Database SQL Language Reference* for information.

See Also:

The following sections discuss database links. See Distributed Database Concepts for more information about using database links.

35.5.2 Creator Is Owner

If the creator of a materialized view also owns the materialized view, then this user must have the required privileges to create a materialized view.

The following privileges must be granted explicitly rather than through a role:

- `CREATE MATERIALIZED VIEW` or `CREATE ANY MATERIALIZED VIEW`
- `CREATE TABLE` or `CREATE ANY TABLE`
- `READ` or `SELECT` object privilege on the master table and the master table's materialized view log or either `READ ANY TABLE` or `SELECT ANY TABLE` system privilege

  If the master database is remote, then the `READ` or `SELECT` object privilege must be granted to the user at the master database to which the user at the materialized view database connects through a database link.

35.5.3 Creator Is Not Owner

If the creator of a materialized view is not the owner, then certain privileges must be granted to the creator and to the owner to create a materialized view.

Both the creator's privileges and the owner's privileges must be granted explicitly rather than through a role.

Table 35-1 shows the required privileges when the creator of the materialized view is not the owner.

Table 35-1 Required Privileges for Creating Materialized Views (Creator != Owner)

Creator	Owner
`CREATE ANY MATERIALIZED VIEW`	`CREATE TABLE` or `CREATE ANY TABLE` `READ` or `SELECT` object privilege on the master table and the master table's materialized view log or either `READ ANY TABLE` or `SELECT ANY TABLE` system privilege If the master database is remote, then the `READ` or `SELECT` object privilege must be granted to the user at the master database to which the user at the materialized view database connects through a database link.

35.5.4 Refresher Is Owner

If the refresher of a materialized view also owns the materialized view, then this user must have the required privileges to create the materialized view.

Specifically, this user must have `READ` or `SELECT` object privilege on the master table and the master table's materialized view log or either `READ ANY TABLE` or `SELECT ANY TABLE` system privilege. If the master database is remote, then the `READ` or `SELECT` object privilege must be granted to the user at the master database to which the user at the materialized view database connects through a database link. This privilege can be granted either explicitly or through a role.

35.5.5 Refresher Is Not Owner

If the refresher of a materialized view is not the owner, certain privileges must be granted to the refresher and to the owner.

These privileges can be granted either explicitly or through a role.

Table 35-2 shows the required privileges when the refresher of the materialized view is not the owner.

Table 35-2 Required Privileges for Refreshing Materialized Views (Refresher != Owner)

Refresher	Owner
`ALTER ANY MATERIALIZED VIEW`	If the master database is local, then `READ` or `SELECT` object privilege must be granted on the master table and master table's materialized view log or either `READ ANY TABLE` or `SELECT ANY TABLE` system privilege. If the master database is remote, then the `READ` or `SELECT` object privilege must be granted to the user at the master database to which the user at the materialized view database connects through a database link.

35.6 Data Subsetting with Materialized Views

You can use row subsetting and column subsetting to configure materialized views reflect a subset of the data in the master table.

- About Data Subsetting with Materialized Views
  In certain situations, you might want your materialized view to reflect a subset of the data in the master table.
- Materialized Views with Subqueries
  If you want to replicate data based on the information in multiple tables, then maintaining and defining these materialized views can be difficult.
- Restrictions for Materialized Views with Subqueries
  The defining query of a materialized view with a subquery is subject to several restrictions to preserve the materialized view's fast refresh capability.
- Restrictions for Materialized Views with Unions Containing Subqueries
  There are restrictions for fast refresh materialized views with unions containing subqueries.

35.6.1 About Data Subsetting with Materialized Views

In certain situations, you might want your materialized view to reflect a subset of the data in the master table.

Row subsetting enables you to include only the rows that are needed from the master table in the materialized views by using a `WHERE` clause. Column subsetting enables you to include only the columns that are needed from the master table in the materialized views. You do this by specifying certain select columns in the `SELECT` statement during materialized view creation.

Some reasons to use data subsetting are to:

- **Reduce Network Traffic**: In a column-subsetted materialized view, only changes that satisfy the `WHERE` clause of the materialized view's defining query are applied to the materialized view database, thereby reducing the amount of data transferred and reducing network traffic.

- **Secure Sensitive Data**: Users can only view data that satisfies the defining query for the materialized view.
- **Reduce Resource Requirements**: If the materialized view is located on a laptop, then hard disks are generally significantly smaller than the hard disks on a corporate server. Subsetted materialized views might require significantly less storage space.
- **Improve Refresh Times**: Because less data is applied to the materialized view database, the refresh process is faster, which might be essential for those who need to refresh materialized views using a network connection from a laptop.

For example, the following statement creates a materialized view based on the `oe.orders@orc1.example.com` master table and includes only the rows for the sales representative with a `sales_rep_id` number of `173`:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
 SELECT * FROM oe.orders@orc1.example.com
 WHERE sales_rep_id = 173;
```

Rows of the orders table with a `sales_rep_id` number other than `173` are excluded from this materialized view.

35.6.2 Materialized Views with Subqueries

If you want to replicate data based on the information in multiple tables, then maintaining and defining these materialized views can be difficult.

- Many to One Subqueries
  You can create a materialized view with a subquery with a many to one relationship.
- One to Many Subqueries
  You can create a materialized view with a subquery with a one to many relationship.
- Many to Many Subqueries
  You can create a materialized view with a subquery with a many to many relationship.
- Materialized Views with Subqueries and Unions
  You can create a materialized view with subqueries and unions.

35.6.2.1 Many to One Subqueries

You can create a materialized view with a subquery with a many to one relationship.

Consider a scenario where you have the `customers` table and `orders` table in the `oe` schema, and you want to create a materialized view of the `orders` table based on data in both the `orders` table and the `customers` table. For example, suppose a salesperson wants to see all of the orders for the customers with a credit limit greater than $10,000. In this case, the `CREATE MATERIALIZED VIEW` statement that creates the `orders` materialized view has a subquery with a many to one relationship, because there can be many orders for each customer.

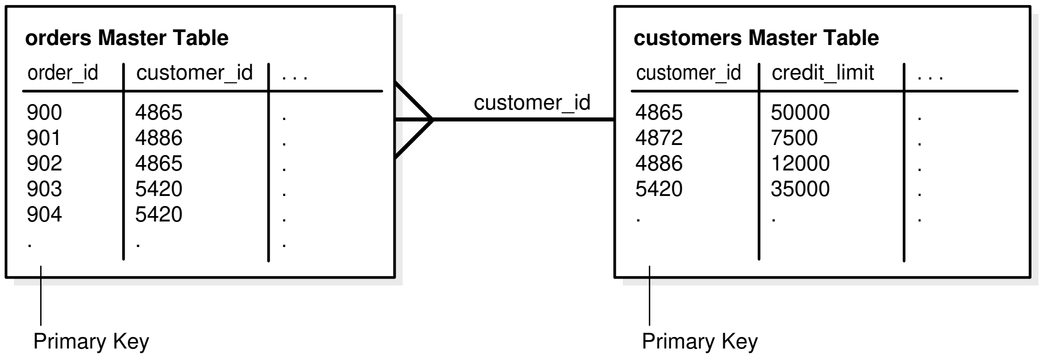
Look at the relationships in Figure 35-3, and notice that the `customers` and `orders` tables are related through the `customer_id` column. The following statement satisfies the original goal of the salesperson. That is, the following statement creates a materialized view that contains orders for customers whose credit limit is greater than $10,000:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
  SELECT * FROM oe.orders@orc1.example.com o
  WHERE EXISTS
```

```
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id AND c.credit_limit > 10000);
```

> **Note:**
>
> To create this `oe.orders` materialized view, `credit_limit` must be logged in the master table's materialized view log. See "Logging Columns in a Materialized View Log" for more information.

Figure 35-3 Row Subsetting with Many to One Subqueries



As you can see, the materialized view created by this statement is fast refreshable. If new customers are identified that have a credit limit greater than $10,000, then the new data will be propagated to the materialized view database during the subsequent refresh process. Similarly, if a customer's credit limit drops to less than $10,000, then the customer's data will be removed from the materialized view during the subsequent refresh process.

35.6.2.2 One to Many Subqueries

You can create a materialized view with a subquery with a one to many relationship.

Consider a scenario where you have the `customers` table and `orders` table in the `oe` schema, and you want to create a materialized view of the `customers` table based on data in both the `customers` table and the `orders` table. For example, suppose a salesperson wants to see all of the customers who have an order with an order total greater than $20,000. In this case, the most efficient method is to create a materialized view with a one to many subquery in the defining query of a materialized view.

Here, the defining query in the `CREATE MATERIALIZED VIEW` statement on the `customers` table has a subquery with a one to many relationship. That is, one customer can have many orders.

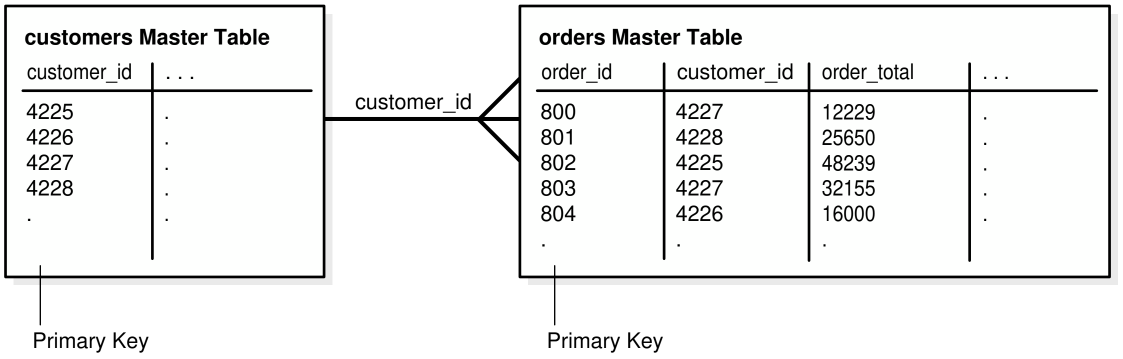
Look at the relationships in Figure 35-4, and notice that the `orders` table and `customers` table are related through the `customer_id` column. The following statement satisfies the original goal of the salesperson. That is, this statement creates a materialized view that contains customers who have an order with an order total greater than $20,000:

```
CREATE MATERIALIZED VIEW oe.customers REFRESH FAST AS
  SELECT * FROM oe.customers@orc1.example.com c
  WHERE EXISTS
    (SELECT * FROM oe.orders@orc1.example.com o
     WHERE c.customer_id = o.customer_id AND o.order_total > 20000);
```

Note:

To create this `oe.customers` materialized view, `customer_id` and `order_total` must be logged in the materialized view log for the `orders` master table. See "Logging Columns in a Materialized View Log" for more information.

Figure 35-4 Row Subsetting with One to Many Subqueries



The materialized view created by this statement is fast refreshable. If new customers are identified that have an order total greater than $20,000, then the new data will be propagated to the materialized view database during the subsequent refresh process. Similarly, if a customer cancels an order with an order total greater than $20,000 and has no other order totals greater than $20,000, then the customer's data will be removed from the materialized view during the subsequent refresh process.

35.6.2.3 Many to Many Subqueries

You can create a materialized view with a subquery with a many to many relationship.

Consider a scenario where you have the `order_items` table and `inventories` table in the `oe` schema, and you want to create a materialized view of the `inventories` table based on data in both the `inventories` table and the `order_items` table. For example, suppose a salesperson wants to see all of the inventories with a quantity on hand greater than 0 (zero) for each product whose `product_id` is in the `order_items` table. In other words, the salesperson wants to see the inventories that are greater than zero for all of the products that customers have ordered. Here, an inventory is a certain quantity of a product at a particular warehouse. So, a certain product can be in many order items and in many inventories.

To accomplish the salesperson's goal, you can create a materialized view with a subquery on the many to many relationship between the `order_items` table and the `inventories` table.

When you create the `inventories` materialized view, you want to retrieve the inventories with the quantity on hand greater than zero for the products that appear in the `order_items` table. Look at the relationships in Figure 35-5, and note that the `inventories` table and `order_items` table are related through the `product_id` column. The following statement creates the materialized view:

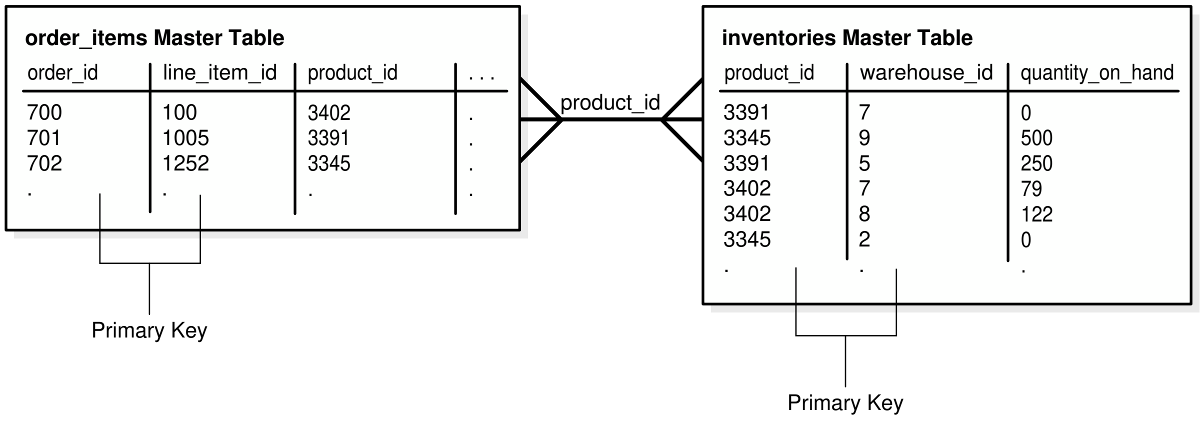
```
CREATE MATERIALIZED VIEW oe.inventories REFRESH FAST AS
  SELECT * FROM oe.inventories@orc1.example.com i
  WHERE i.quantity_on_hand > 0 AND EXISTS
```

```
(SELECT * FROM oe.order_items@orc1.example.com o
 WHERE i.product_id = o.product_id);
```

Note:

To create this `oe.inventories` materialized view, the `product_id` column in the `order_items` master table must be logged in the master table's materialized view log. See "Logging Columns in a Materialized View Log" for more information.

Figure 35-5 Row Subsetting with Many to Many Subqueries



The materialized view created by this statement is fast refreshable. If new inventories that are greater than zero are identified for products in the `order_items` table, then the new data will be propagated to the materialized view database during the subsequent refresh process. Similarly, if a customer cancels an order for a product and there are no other orders for the product in the `order_items` table, then the inventories for the product will be removed from the materialized view during the subsequent refresh process.

35.6.2.4 Materialized Views with Subqueries and Unions

You can create a materialized view with subqueries and unions.

In situations where you want a single materialized view to contain data that matches the complete results of two or more different queries, you can use the `UNION` operator. When you use the `UNION` operator to create a materialized view, you have two `SELECT` statements around each `UNION` operator; one is above it and one is below it. The resulting materialized view contains rows selected by either query.

You can use the `UNION` operator as a way to create fast refreshable materialized views that satisfy "or" conditions without using the `OR` expression in the `WHERE` clause of a subquery. Under some conditions, using an `OR` expression in the `WHERE` clause of a subquery causes the resulting materialized view to be complex, and therefore not fast refreshable.

See Also:

"Restrictions for Materialized Views with Subqueries" for more information about the `OR` expressions in subqueries

For example, suppose a salesperson wants the product information for the products in a particular `category_id` that are *either* in a warehouse in California *or* contain the word "Rouge" in their translated product descriptions (for the French translation). The following statement uses the `UNION` operator and subqueries to capture this data in a materialized view for products in `category_id` 29:

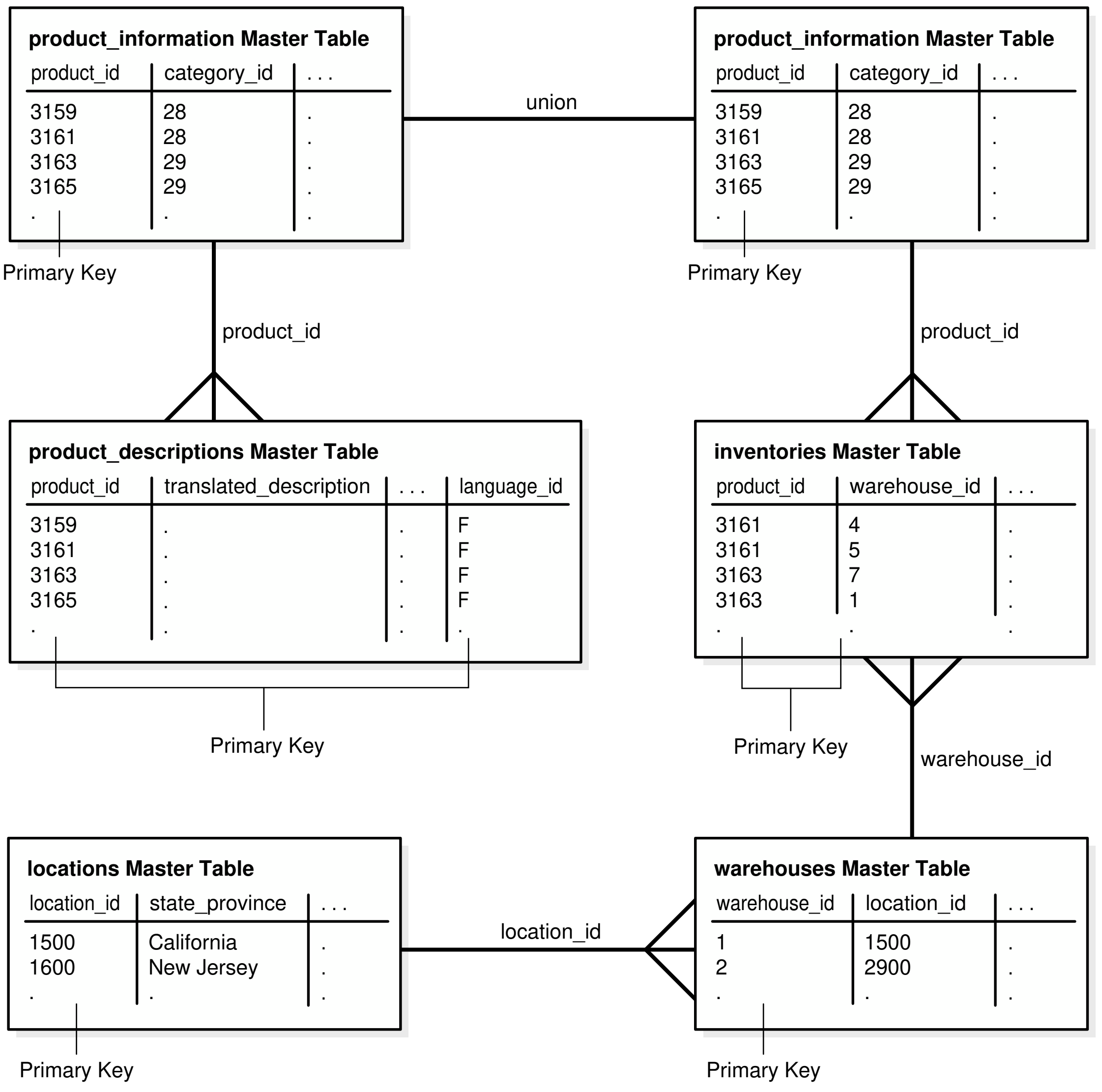
```
CREATE MATERIALIZED VIEW oe.product_information REFRESH FAST AS
 SELECT * FROM oe.product_information@orc1.example.com pi
 WHERE pi.category_id = 29 AND EXISTS
  (SELECT * FROM oe.product_descriptions@orc1.example.com pd
  WHERE pi.product_id = pd.product_id AND
        pd.translated_description LIKE '%Rouge%')
UNION
 SELECT * FROM oe.product_information@orc1.example.com pi
 WHERE pi.category_id = 29 AND EXISTS
  (SELECT * FROM oe.inventories@orc1.example.com i
  WHERE pi.product_id = i.product_id AND EXISTS
    (SELECT * FROM oe.warehouses@orc1.example.com w
    WHERE i.warehouse_id = w.warehouse_id AND EXISTS
      (SELECT * FROM hr.locations@orc1.example.com l
       WHERE w.location_id = l.location_id
       AND l.state_province = 'California')));
```

Note:

To create the `oe.product_information` materialized view, `translated_description` in the `oe.product_descriptions` master table, the `state_province` in the `hr.locations` master table, and the `location_id` column in the `oe.warehouses` master table must be logged in each master's materialized view log. See "Logging Columns in a Materialized View Log" for more information.

Figure 35-6 shows the relationships of the master tables involved in this statement.

Figure 35-6 Row Subsetting with Subqueries and Unions



In addition to the `UNION` operation, this statement contains the following subqueries:

- A subquery referencing the `product_information` table and the `product_descriptions` table. This subquery is one to many because one product can have multiple product descriptions (for different languages).
- A subquery referencing the `product_information` table and the `inventories` table. This subquery is one to many because a product can be in many inventories.
- A subquery referencing the `inventories` table and the `warehouses` table. This subquery is many to one because many inventories can be stored in one warehouse.
- A subquery referencing the `warehouses` table and the `locations` table. This subquery is many to one because many warehouses can be in one location.

The materialized view created by this statement is fast refreshable. If a new product is added that is stored in a warehouse in California or that has the string "Rouge" in the translated product description, then the new data will be propagated to the `product_information` materialized view during the subsequent refresh process.

35.6.3 Restrictions for Materialized Views with Subqueries

The defining query of a materialized view with a subquery is subject to several restrictions to preserve the materialized view's fast refresh capability.

The following are restrictions for fast refresh materialized views with subqueries:

- Materialized views must be primary key materialized views.
- The master table's materialized view log must include certain columns referenced in the subquery. For information about which columns must be included, see "Logging Columns in a Materialized View Log".
- If the subquery is many to many or one to many, then join columns that are not part of a primary key must be included in the materialized view log of the master table. This restriction does not apply to many to one subqueries.
- The subquery must be a positive subquery. For example, you can use the `EXISTS` condition, but not the `NOT EXISTS` condition.
- The subquery must use `EXISTS` to connect each nested level (`IN` is not allowed).
- Each table can be in only one `EXISTS` expression.
- The join expression must use exact match or equality comparisons (that is, equi-joins).
- Each table can be joined only once within the subquery.
- A primary key must exist for each table at each nested level.
- Each nested level can only reference the table in the level above it.
- Subqueries can include `AND` conditions, but each `OR` condition can only reference columns contained within one row. Multiple `OR` conditions within a subquery can be connected with an `AND` condition.
- All tables referenced in a subquery must reside in the same master database.

Note:

If the `CREATE MATERIALIZED VIEW` statement includes an `ON PREBUILT TABLE` clause and a subquery, then the subquery is treated as many to many. Therefore, in this case, the join columns must be recorded in the materialized view log. See the *Oracle Database SQL Language Reference* for more information about the `ON PREBUILT TABLE` clause in the `CREATE MATERIALIZED VIEW` statement.

See Also:

- "Primary Key Materialized Views" for more information about primary key materialized views
- "Determining the Fast Refresh Capabilities of a Materialized View"

35.6.4 Restrictions for Materialized Views with Unions Containing Subqueries

There are restrictions for fast refresh materialized views with unions containing subqueries.

The following are restrictions for fast refresh materialized views with unions containing subqueries:

- All of the restrictions described in "Restrictions for Materialized Views with Subqueries" apply to the subqueries in each union block.
- All join columns must be included in the materialized view log of the master table, even if the subquery is many to one.
- All of the restrictions described in "Complex Materialized Views" apply for clauses with `UNIONS`.

- Examples of Materialized Views with Unions Containing Subqueries
  Examples illustrate creating materialized views with unions containing subqueries.

35.6.4.1 Examples of Materialized Views with Unions Containing Subqueries

Examples illustrate creating materialized views with unions containing subqueries.

The following statement creates the `oe.orders` materialized view. This materialized view is fast refreshable because the subquery in each union block satisfies the restrictions for subqueries described in "Restrictions for Materialized Views with Subqueries".

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
  SELECT * FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id
     AND c.credit_limit > 50)
UNION
  SELECT *
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id
     AND c.account_mgr_id = 30);
```

Notice that one of the restrictions for subqueries states that each table can be in only one `EXISTS` expression. Here, the `customers` table appears in two `EXISTS` expressions, but the `EXISTS` expressions are in separate `UNION` blocks. Because the restrictions described in "Restrictions for Materialized Views with Subqueries" only apply to each `UNION` block, not to the entire `CREATE MATERIALIZED VIEW` statement, the materialized view is fast refreshable.

In contrast, the materialized view created with the following statement cannot be fast refreshed because the `orders` table is referenced in two different `EXISTS` expressions within the same `UNION` block:

```
CREATE MATERIALIZED VIEW oe.orders AS
  SELECT * FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id  -- first reference to orders table
     AND c.credit_limit > 50
     AND EXISTS
```

```
        (SELECT * FROM oe.orders@orc1.example.com o
         WHERE order_total > 5000
         AND o.customer_id = c.customer_id)) -- second reference to orders table
UNION
  SELECT *
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id
     AND c.account_mgr_id = 30);
```

See Also:

"Determining the Fast Refresh Capabilities of a Materialized View"

35.7 Materialized View Refresh

To ensure that a materialized view is consistent with its master table, you must **refresh** the materialized view periodically.

Oracle provides the following three methods to refresh materialized views:

- **Fast refresh** uses materialized view logs to update only the rows that have changed since the last refresh.
- **Complete refresh** updates the entire materialized view.
- **Force refresh** performs a fast refresh when possible. When a fast refresh is not possible, force refresh performs a complete refresh.

35.8 Refresh Groups

When it is important for materialized views to be transactionally consistent with each other, you can organize them into **refresh groups**.

By refreshing the refresh group, you can ensure that the data in all of the materialized views in the refresh group correspond to the same transactionally consistent point in time. A materialized view in a refresh group still can be refreshed individually, but doing so nullifies the benefits of the refresh group because refreshing the materialized view individually does not refresh the other materialized views in the refresh group.

35.9 Materialized View Log

A **materialized view log** is a table at the database that contains materialized view's master table. It records all of the DML changes to the master table.

A materialized view log is associated with a single master table, and each of those has only one materialized view log, regardless of how many materialized views refresh from the master table. A fast refresh of a materialized view is possible only if the materialized view's master table has a materialized view log. When a materialized view is fast refreshed, entries in the materialized view's associated materialized view log that have appeared since the materialized view was last refreshed are applied to the materialized view.

35.10 Materialized Views and User-Defined Data Types

There are special considerations for materialized views with user-defined data types.

- How Materialized Views Work with Object Types and Collections
  You can replicate object types and objects between master databases and materialized view databases in a replication environment.
- Type Agreement at Replication Databases
  User-defined types include all types created using the `CREATE TYPE` statement, including object, nested table, `VARRAY`, and indextype. To replicate schema objects based on user-defined types, the user-defined types themselves must exist, and must be the same, at the master database and the materialized view database.
- Column Subsetting of Masters with Column Objects
  A read-only materialized view can replicate specific attributes of a column object without replicating other attributes.
- Materialized Views Based on Object Tables
  You can create a materialized view based on an object table.
- Materialized Views with Collection Columns
  Collection columns are columns based on varray and nested table data types. Oracle supports the creation of materialized views with collection columns.
- Materialized Views with REF Columns
  Materialized views can contain `REF` columns.

35.10.1 How Materialized Views Work with Object Types and Collections

You can replicate object types and objects between master databases and materialized view databases in a replication environment.

Oracle **object types** are user-defined data types that make it possible to model complex real-world entities such as customers and orders as single entities, called **objects**, in the database. You create object types using the `CREATE TYPE ... AS OBJECT` statement.

An Oracle object that occupies a single column in a table is called a **column object**. Typically, tables that contain column objects also contain other columns, which can be built-in data types, such as `VARCHAR2` and `NUMBER`. An **object table** is a special kind of table in which each row represents an object. Each row in an object table is a **row object**.

You can also replicate **collections**. Collections are user-defined data types that are based on `VARRAY` and nested table data types. You create varrays with the `CREATE TYPE ... AS VARRAY` statement, and you create nested tables with the `CREATE TYPE ... AS TABLE` statement.

Note:

- You cannot create refresh-on-commit materialized views based on a master table with user-defined types or Oracle-supplied types. Refresh-on-commit materialized views are those created using the `ON COMMIT REFRESH` clause in the `CREATE MATERIALIZED VIEW` statement.
- Type inheritance and types created with the `NOT FINAL` clause are not supported.

See Also:

- *Oracle Database Object-Relational Developer's Guide* for detailed information about user-defined types, Oracle objects, and collections. This section assumes a basic understanding of those concepts.
- *Oracle Database SQL Language Reference* for more information about user-defined types and Oracle-supplied types

35.10.2 Type Agreement at Replication Databases

User-defined types include all types created using the `CREATE TYPE` statement, including object, nested table, `VARRAY`, and indextype. To replicate schema objects based on user-defined types, the user-defined types themselves must exist, and must be the same, at the master database and the materialized view database.

When replicating user-defined types and the schema objects on which they are based, the following conditions apply:

- The user-defined types replicated at the master database and materialized view database must be created at the materialized view database before you create any materialized views that depend on these types.
- All of the master tables on which a materialized view is based must be at the same master database to create a materialized view with user-defined types.
- A user-defined type must be the same at all databases:
  - All replication databases must have the same object identifier (OID), schema owner, and type name for each replicated user-defined type.
  - If the user-defined type is an object type, then all databases must agree on the order and data type of the attributes in the object type. You establish the order and data types of the attributes when you create the object type. For example, consider the following object type:

    ```
    CREATE TYPE cust_address_typ AS OBJECT
           (street_address     VARCHAR2(40),
            postal_code        VARCHAR2(10),
            city               VARCHAR2(30),
            state_province     VARCHAR2(10),
            country_id         CHAR(2));
    /
    ```

    At all databases, `street_address` must be the first attribute for this type and must be `VARCHAR2(40)`, `postal_code` must be the second attribute and must be `VARCHAR2(10)`, `city` must be the third attribute and must be `VARCHAR2(30)`, and so on.
  - All databases must agree on the hashcode of the user-defined type. Oracle examines a user-defined type and assigns the hashcode. This examination includes the type attributes, order of attributes, and type name. When all of these items are the same for two or more types, the types have the same hashcode. You can view the hashcode for a type by querying the `DBA_TYPE_VERSIONS` data dictionary view.

You can use a `CREATE TYPE` statement at the materialized view database to create the type. It might be necessary to do this to create a read-only materialized view that uses the type.

If you choose this option, then you must ensure the following:

- The type is in the same schema at both the materialized view database and the master database.
- The type has the same attributes in the same order at both the materialized view database and the master database.
- The type has the same data type for each attribute at both the materialized view database and the master database.
- The type has the same object identifier at both the materialized view database and the master database.

You can find the object identifier for a type by querying the `DBA_TYPES` data dictionary view. For example, to find the object identifier (OID) for the `cust_address_typ`, enter the following query:

```
SELECT TYPE_OID FROM DBA_TYPES WHERE TYPE_NAME = 'CUST_ADDRESS_TYP';

TYPE_OID
--------------------------------
6F9BC33653681B7CE03400400B40A607
```

For example, now that you know the OID for the type at the master database, you can complete the following steps to create the type at the materialized view database:

1. Log in to the materialized view database as the user who owns the type at the master database. If this user does not exist at the materialized view database, then create the user.
2. Issue the `CREATE TYPE` statement and specify the OID:

```
CREATE TYPE oe.cust_address_typ OID '6F9BC33653681B7CE03400400B40A607'
     AS OBJECT (
     street_address      VARCHAR2(40),
     postal_code         VARCHAR2(10),
     city                VARCHAR2(30),
     state_province      VARCHAR2(10),
     country_id          CHAR(2));
/
```

The type is now ready for use at the materialized view database.

35.10.3 Column Subsetting of Masters with Column Objects

A read-only materialized view can replicate specific attributes of a column object without replicating other attributes.

For example, using the `cust_address_typ` user-defined data type described in the previous section, suppose a `customers_sub` master table is created at master database `orc1.example.com`:

```
CREATE TABLE oe.customers_sub (
      customer_id        NUMBER(6)  PRIMARY KEY,
      cust_first_name    VARCHAR2(20),
      cust_last_name     VARCHAR2(20),
      cust_address       oe.cust_address_typ);
```

You can create the following read-only materialized view at a remote materialized view database:

```
CREATE MATERIALIZED VIEW oe.customers_mv1 AS
   SELECT customer_id, cust_last_name, c.cust_address.postal_code
   FROM oe.customers_sub@orc1.example.com c;
```

Notice that the `postal_code` attribute is specified in the `cust_address` column object.

35.10.4 Materialized Views Based on Object Tables

You can create a materialized view based on an object table.

- About Materialized Views Based on Object Tables
  If a materialized view is based on an object table and is created using the `OF` *type* clause, then the materialized view is called an **object materialized view**.
- Materialized Views Based on Object Tables Created Without Using the OF type Clause
  If you create a materialized view based on an object table without using the `OF` *type* clause, then the materialized view loses the object properties of the object table on which it is based.
- OID Preservation in Object Materialized Views
  An object materialized view inherits the object identifier (OID) specifications of its master.

35.10.4.1 About Materialized Views Based on Object Tables

If a materialized view is based on an object table and is created using the `OF` *type* clause, then the materialized view is called an **object materialized view**.

An object materialized view is structured in the same way as an object table. That is, an object materialized view is composed of row objects.

If a materialized view that is based on an object table is created without using the `OF` *type* clause, then the materialized view is not an object materialized view. That is, such a materialized view has regular rows, not row objects.

To create a materialized view based on an object table, the types on which the materialized view depends must exist at the materialized view database, and each type must have the same object identifier as it does at the master database.

See Also:

"Creating Read-Only Materialized Views" for an example that creates an object materialized view

35.10.4.2 Materialized Views Based on Object Tables Created Without Using the OF *type* Clause

If you create a materialized view based on an object table without using the `OF` *type* clause, then the materialized view loses the object properties of the object table on which it is based.

That is, the resulting read-only materialized view contains one or more of the columns of the master table, but each row functions as a row in a relational table. The rows are not row objects.

For example, you can create a materialized view based on the `categories_tab` master by using the following SQL statement:

```
CREATE MATERIALIZED VIEW oe.categories_relmv
   AS SELECT * FROM oe.categories_tab@orc1.example.com;
```

In this case, the rows in this materialized view function in the same way as rows in a relational table.

35.10.4.3 OID Preservation in Object Materialized Views

An object materialized view inherits the object identifier (OID) specifications of its master.

If the master table has a primary key-based OID, then the OIDs of row objects in the materialized view are primary key-based. If the master table has a system generated OID, then the OIDs of row objects in the materialized view are system generated. Also, the OID of each row in the object materialized view matches the OID of the same row in the master table, and the OIDs are preserved during refresh of the materialized view. Consequently, `REF`s to the rows in the object table remain valid at the materialized view database.

35.10.5 Materialized Views with Collection Columns

Collection columns are columns based on varray and nested table data types. Oracle supports the creation of materialized views with collection columns.

If the collection column is a nested table, then you can optionally specify the *nested_table_storage_clause* during materialized view creation. The *nested_table_storage_clause* lets you specify the name of the storage table for the nested table in the materialized view.

For example, suppose you create the master table `people_reltab` at the master database `orc1.example.com` that contains the nested table `phones_ntab`:

```
CREATE TYPE oe.phone_typ AS OBJECT (
   location     VARCHAR2(15),
   num          VARCHAR2(14));
/

CREATE TYPE oe.phone_ntabtyp AS TABLE OF oe.phone_typ;
/

CREATE TABLE oe.people_reltab (
   id               NUMBER(4) CONSTRAINT pk_people_reltab PRIMARY KEY,
   first_name       VARCHAR2(20),
   last_name        VARCHAR2(20),
   phones_ntab      oe.phone_ntabtyp)
   NESTED TABLE phones_ntab STORE AS phone_store_ntab
   ((PRIMARY KEY (NESTED_TABLE_ID, location)));
```

Notice the `PRIMARY KEY` specification in the last line of the preceding SQL statement. You must specify a primary key for the storage table if you plan to create materialized views based on its parent table. In this case, the storage table is `phone_store_ntab` and the parent table is `people_reltab`.

To create materialized views that can be fast refreshed, create a materialized view log on both the parent table and the storage table, specifying the nested table column as a filter column for the parent table's materialized view log:

```
CREATE MATERIALIZED VIEW LOG ON oe.people_reltab;

ALTER MATERIALIZED VIEW LOG ON oe.people_reltab ADD(phones_ntab);

CREATE MATERIALIZED VIEW LOG ON oe.phone_store_ntab WITH PRIMARY KEY;
```

At the materialized view database, create the required types, ensuring that the object identifier for each type is the same as the object identifier at the master database. Then, you can create a materialized view based on `people_reltab` and specify its storage table using the following statement:

```
CREATE MATERIALIZED VIEW oe.people_reltab_mv
   NESTED TABLE phones_ntab STORE AS phone_store_ntab_mv
   REFRESH FAST AS SELECT * FROM oe.people_reltab@orc1.example.com;
```

In this case, the *nested_table_storage_clause* is the line that begins with "`NESTED TABLE`" in the previous example, and it specifies that the storage table's name is `phone_store_ntab_mv`. The *nested_table_storage_clause* is optional. If you do not specify this clause, then Oracle Database automatically names the storage table. To view the name of a storage table, query the `DBA_NESTED_TABLES` data dictionary table.

The storage table:

- Is a separate, secondary materialized view
- Is refreshed automatically when you refresh the materialized view containing the nested table
- Is dropped automatically when you drop the materialized view containing the nested table
- Inherits the primary key constraint of the master's storage table

Because the storage table inherits the primary key constraint of the master table's storage table, do not specify `PRIMARY KEY` in the `STORE AS` clause.

The following actions are not allowed directly on the storage table of a nested table in a materialized view:

- Refreshing the storage table
- Altering the storage table
- Dropping the storage table
- Generating replication support on the storage table

These actions can occur indirectly when they are performed on the materialized view that contains the nested table. In addition, you cannot replicate a subset of the columns in a storage table.

- Restrictions for Materialized Views with Collection Columns
  Restrictions apply to materialized views with collection columns.

See Also:

Oracle Database SQL Language Reference for more information about the *nested_table_col_properties*, which is fully documented in the `CREATE TABLE` statement

35.10.5.1 Restrictions for Materialized Views with Collection Columns

Restrictions apply to materialized views with collection columns.

The following restrictions apply:

- Row subsetting of collection columns is not allowed. However, you can use row subsetting on the parent table of a nested table and doing so can result in a subset of the nested tables in the materialized view.
- Column subsetting of collection columns is not allowed.
- A nested table's storage table must have a primary key.
- For the parent table of a nested table to be fast refreshed, both the parent table and the nested table's storage table must have a materialized view log.

35.10.6 Materialized Views with REF Columns

Materialized views can contain `REF` columns.

- About Materialized Views with REF Columns
  You can create materialized views with `REF` columns. A `REF` is an Oracle built-in data type that is a logical "pointer" to a row object in an object table.
- Scoped REF Columns
  If you are creating a materialized view based on a master table that has a scoped `REF` column, then you can rescope the `REF` to a different object table or object materialized view at the materialized view database.
- Unscoped REF Columns
  If you create a materialized view based on a remote master table with an unscoped `REF` column, then the `REF` column is created in the materialized view, but the `REF`s are considered dangling because they point to a remote database.
- Logging REF Columns in the Materialized View Log
  If necessary, you can log `REF` columns in the materialized view log.
- REFs Created Using the WITH ROWID Clause
  If the `WITH ROWID` clause is specified for a `REF` column, then Oracle Database maintains the rowid of the object referenced in the `REF`.

35.10.6.1 About Materialized Views with REF Columns

You can create materialized views with `REF` columns. A `REF` is an Oracle built-in data type that is a logical "pointer" to a row object in an object table.

A scoped `REF` is a `REF` that can contain references only to a specified object table, while an unscoped `REF` can contain references to any object table in the database that is based on the corresponding object type. A scoped `REF` requires less storage space and provides more efficient access than an unscoped `REF`.

You can rescope a `REF` column to a local materialized view or table at the materialized view database during creation of the materialized view. If you do not rescope the `REF` column, then it continues to point to the remote master table. Unscoped `REF` columns always continue to point to the master table. When a `REF` column at a materialized view database points to a remote master table, the `REF`s are considered dangling. In SQL, dereferencing a dangling `REF` returns a

NULL. Also, PL/SQL only supports dereferencing REFs by using the UTL_OBJECT package and raises an exception for dangling REFs.

35.10.6.2 Scoped REF Columns

If you are creating a materialized view based on a master table that has a scoped REF column, then you can rescope the REF to a different object table or object materialized view at the materialized view database.

Typically, you would rescope the REF column to the local object materialized view instead of the original remote object table. To rescope a materialized view, you can either use the SCOPE FOR clause in the CREATE MATERIALIZED VIEW statement, or you can use the ALTER MATERIALIZED VIEW statement after creating the materialized view. If you do not rescope the REF column, then the materialized view retains the REF scope of the master table.

For example, suppose you create the customers_with_ref master table at the orc1.example.com master database using the following statements:

```
-- Create the user-defined data type cust_address_typ.
CREATE TYPE oe.cust_address_typ AS OBJECT
   (street_address     VARCHAR2(40),
    postal_code        VARCHAR2(10),
    city               VARCHAR2(30),
    state_province     VARCHAR2(10),
    country_id         CHAR(2));
/

-- Create the object table cust_address_objtab.
CREATE TABLE oe.cust_address_objtab OF oe.cust_address_typ;

-- Create table with REF to cust_address_typ.
CREATE TABLE oe.customers_with_ref (
         customer_id        NUMBER(6) PRIMARY KEY,
     cust_first_name    VARCHAR2(20),
     cust_last_name     VARCHAR2(20),
     cust_address       REF oe.cust_address_typ
                          SCOPE IS oe.cust_address_objtab);
```

Assuming the cust_address_typ exists at the materialized view database with the same object identifier as the type at the master database, you can create a cust_address_objtab_mv object materialized view using the following statement:

```
CREATE MATERIALIZED VIEW oe.cust_address_objtab_mv OF oe.cust_address_typ AS
   SELECT * FROM oe.cust_address_objtab@orc1.example.com;
```

Now, you can create a materialized view of the customers_with_ref master table and rescope the REF to the cust_address_objtab_mv materialized view using the following statement:

```
CREATE MATERIALIZED VIEW oe.customers_with_ref_mv
   (SCOPE FOR (cust_address) IS oe.cust_address_objtab_mv)
   AS SELECT * FROM oe.customers_with_ref@orc1.example.com;
```

To use the SCOPE FOR clause when you create a materialized view, remember to create the materialized view or table specified in the SCOPE FOR clause first. Otherwise, you cannot specify the SCOPE FOR clause during materialized view creation. For example, if you had created the customers_with_ref_mv materialized view before you created the cust_address_objtab_mv materialized view, then you could not use the SCOPE FOR clause when you created the customers_with_ref_mv materialized view. In this case, the REFs are considered dangling because they point back to the object table at the remote master database.

However, even if you do not use the `SCOPE FOR` clause when you are creating a materialized view, you can alter the materialized view to specify a `SCOPE FOR` clause. For example, you can alter the `customers_with_ref_mv` materialized view with the following statement:

```
ALTER MATERIALIZED VIEW oe.customers_with_ref_mv
   MODIFY SCOPE FOR (cust_address) IS oe.cust_address_objtab_mv;
```

35.10.6.3 Unscoped REF Columns

If you create a materialized view based on a remote master table with an unscoped `REF` column, then the `REF` column is created in the materialized view, but the `REF`s are considered dangling because they point to a remote database.

35.10.6.4 Logging REF Columns in the Materialized View Log

If necessary, you can log `REF` columns in the materialized view log.

See Also:

"Logging Columns in a Materialized View Log"

35.10.6.5 REFs Created Using the WITH ROWID Clause

If the `WITH ROWID` clause is specified for a `REF` column, then Oracle Database maintains the rowid of the object referenced in the `REF`.

Oracle Database can find the object referenced directly using the rowid contained in the `REF`, without the need to fetch the rowid from the OID index. Therefore, you use the `WITH ROWID` clause to specify a rowid hint. The `WITH ROWID` clause is not supported for scoped `REF`s.

Replicating a `REF` created using the `WITH ROWID` clause results in an incorrect rowid hint at each replication database except the database where the `REF` was first created or modified. The `ROWID` information in the `REF` is meaningless at the other databases, and Oracle Database does not correct the rowid hint automatically. Invalid rowid hints can cause performance problems. In this case, you can use the `VALIDATE STRUCTURE` option of the `ANALYZE TABLE` statement to determine which rowid hints at each replication database are incorrect.

See Also:

Oracle Database SQL Language Reference for more information about the `ANALYZE TABLE` statement

35.11 Materialized View Registration at a Master Database

At the master database, an Oracle Database automatically registers information about a materialized view based on its master table(s).

- Viewing Information about Registered Materialized Views
  A materialized view is registered at its master database.
- Internal Mechanisms
  Oracle Database automatically registers a materialized view at its master database when you create the materialized view, and unregisters the materialized view when you drop it.
- Manual Materialized View Registration
  If necessary, you can maintain registration manually.

35.11.1 Viewing Information about Registered Materialized Views

A materialized view is registered at its master database.

You can query the `DBA_REGISTERED_MVIEWS` data dictionary view at a master database to list the following information about a remote materialized view:

- The owner, name, and database that contains the materialized view
- The materialized view's defining query
- Other materialized view characteristics, such as its refresh method

You can also query the `DBA_MVIEW_REFRESH_TIMES` view at a master database to obtain the last refresh times for each materialized view. Administrators can use this information to monitor materialized view activity and coordinate changes to materialized view databases if a master table must be dropped, altered, or relocated.

35.11.2 Internal Mechanisms

Oracle Database automatically registers a materialized view at its master database when you create the materialized view, and unregisters the materialized view when you drop it.

Note:

Oracle Database cannot guarantee the registration or unregistration of a materialized view at its master database during the creation or drop of the materialized view, respectively. If Oracle Database cannot successfully register a materialized view during creation, then you must complete the registration manually using the `REGISTER_MVIEW` procedure in the `DBMS_MVIEW` package. If Oracle Database cannot successfully unregister a materialized view when you drop the materialized view, then the registration information for the materialized view persists in the master database until it is manually unregistered. It is possible that complex materialized views might not be registered.

35.11.3 Manual Materialized View Registration

If necessary, you can maintain registration manually.

Use the `REGISTER_MVIEW` and `UNREGISTER_MVIEW` procedures of the `DBMS_MVIEW` package at the master database to add, modify, or remove materialized view registration information.

See Also:

The `REGISTER_MVIEW` and `UNREGISTER_MVIEW` procedures are described in the *Oracle Database PL/SQL Packages and Types Reference*

36 Read-Only Materialized View Architecture

Several objects are used in materialized view replication. Some of these objects are optional and are used only as needed to support the created materialized view environment. For example, if you have a complex materialized view that cannot be fast refreshed, then you might not have a materialized view log at the master database.

- Master Database Mechanisms
  There are mechanisms that support materialized views at the master database.
- Materialized View Database Mechanisms
  When a materialized view is created, additional mechanisms are created at the materialized view database to support the materialized view. Specifically, at least one index is created.
- Organizational Mechanisms
  Several mechanisms organize the materialized views at the materialized view database. These mechanisms maintain organizational consistency between the materialized view database and its master database.
- Refresh Process
  To ensure that a materialized view is consistent with its master table, you must refresh the materialized view periodically.

36.1 Master Database Mechanisms

There are mechanisms that support materialized views at the master database.

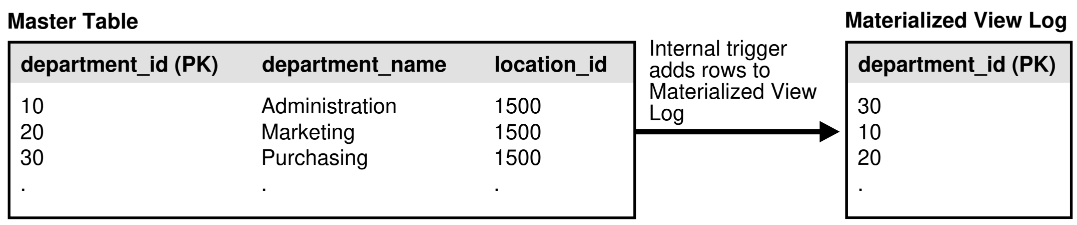
- Master Database Objects
  Specific database objects are required at a master database to support fast refreshing of materialized views.
- Master Table
  The master table is the basis for the materialized view.
- Internal Trigger for the Materialized View Log
  When changes are made to the master table using DML, an internal trigger records information about the affected rows in the materialized view log.
- Materialized View Logs
  Materialized view logs enable fast refreshes of materialized views.

36.1.1 Master Database Objects

Specific database objects are required at a master database to support fast refreshing of materialized views.

The three database objects displayed in Figure 36-1 are required at a master database to support fast refreshing of materialized views.

Figure 36-1 Master Database Objects



Master Table

department_id (PK)	department_name	location_id
10	Administration	1500
20	Marketing	1500
30	Purchasing	1500
.	.	.

Materialized View Log

department_id (PK)
30
10
20
.

36.1.2 Master Table

The master table is the basis for the materialized view.

A master table is located at the target master database. Remember that a materialized view points to only one master database. Data manipulation language (DML) changes made to the master table, as recorded by the materialized view log, are propagated to the materialized view during the refresh process.

Note:

Fast refreshable materialized views must be based on master tables or synonyms of master tables. Complete refresh must be used for a materialized view based on a view.

36.1.3 Internal Trigger for the Materialized View Log

When changes are made to the master table using DML, an internal trigger records information about the affected rows in the materialized view log.

This information includes the values of the primary key, rowid, or object id, or both, as well as the values of the other columns logged in the materialized view log. This is an internal `AFTER ROW` trigger that is automatically activated when you create a materialized view log for the target master table. It inserts a row into the materialized view log whenever an `INSERT`, `UPDATE`, or `DELETE` statement modifies the table's data. This trigger is always the last trigger to fire.

Note:

When the materialized view contains a subquery, you might need to log columns referenced in a subquery. See "Data Subsetting with Materialized Views" for information about subquery materialized views and "Logging Columns in a Materialized View Log" for more information about the columns that must be logged.

36.1.4 Materialized View Logs

Materialized view logs enable fast refreshes of materialized views.

- About Materialized View Logs
  A materialized view log is required on a master table to perform a fast refresh on materialized views based on the master table.
- Columns Logged in the Materialized View Log
  When you create a materialized view log, you can add columns to the log when necessary.
- Restriction on Import of Materialized View Logs to a Different Schema
  Materialized view logs are exported with the schema name explicitly given in the DDL statements. Therefore, materialized view logs cannot be imported into a schema that is different than the schema from which they were exported.

36.1.4.1 About Materialized View Logs

A materialized view log is required on a master table to perform a fast refresh on materialized views based on the master table.

When you create a materialized view log for a master table, Oracle Database creates an underlying table as the materialized view log. A materialized view log can hold the primary keys, rowids, or object identifiers of rows that have been updated in the master table. A materialized view log can also contain other columns to support fast refreshes of materialized views with subqueries.

The name of a materialized view log's table is `MLOG$_`*master_table_name*. The materialized view log is created in the same schema as the target master table. One materialized view log can support multiple materialized views on its master table. As described in the previous section, the internal trigger adds change information to the materialized view log whenever a DML transaction has taken place on the target master table.

Following are the types of materialized view logs:

- **Primary Key**: The materialized view records changes to the master table based on the primary key of the affected rows.
- **Row ID**: The materialized view records changes to the master table based on the rowid of the affected rows.
- **Object ID**: The materialized view records changes to the master object table based on the object identifier of the affected row objects.
- **Combination**: The materialized view records changes to the master table based any combination of the three options. It is possible to record changes based on the primary key, the `ROWID`, and the object identifier of the affected rows. Such a materialized view log supports primary key, `ROWID`, and object materialized views, which is helpful for environments that have all three types of materialized views based on a master table.

A combination materialized view log works in the same manner as a materialized view log that tracks only one type of value, except that more than one type of value is recorded. For example, a combination materialized view log can track both the primary key and the rowid of the affected row.

Though the difference between materialized view logs based on primary keys and rowids is small (one records affected rows using the primary key, while the other records affected rows using the physical rowid), the practical impact is large. Using rowid materialized views and materialized view logs makes reorganizing and truncating your master tables difficult because it prevents your `ROWID` materialized views from being fast refreshed. If you reorganize or truncate your master table, then your rowid materialized view must be `COMPLETE` refreshed because the rowids of the master table have changed.

Note:

- You use the `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION` procedures in the `DBMS_MVIEW` package to reorganize a master table. See the *Oracle Database PL/SQL Packages and Types Reference* for more information.
- Online redefinition of tables is another possible way to reorganize master tables, but online redefinition is not allowed on master tables with materialized view logs and materialized views. Online redefinition is allowed on master tables that do not have materialized view logs. See "Redefining Tables Online".

See Also:

"Creating Materialized View Logs"

36.1.4.2 Columns Logged in the Materialized View Log

When you create a materialized view log, you can add columns to the log when necessary.

To perform a fast refresh on a materialized view, the following types of columns must be added to the materialized view log:

- A column referenced in the `WHERE` clause of a subquery that is not part of an equi-join and is not a primary key column. These columns are called filter columns.
- A column in an equi-join that is not a primary key column, if the subquery is either many to many or one to many. If the subquery is many to one, then you do not need to add the join column to the materialized view log.

A collection column cannot be added to a materialized view log. Also, materialized view logs are not required for materialized views that use complete refresh.

For example, consider the following DDL:

```
1) CREATE MATERIALIZED VIEW oe.customers REFRESH FAST AS
2)  SELECT * FROM oe.customers@orc1.example.com c
3)  WHERE EXISTS
4)    (SELECT * FROM oe.orders@orc1.example.com o
5)     WHERE c.customer_id = o.customer_id AND o.order_total > 20000);
```

Notice in line 5 of the preceding DDL that three columns are referenced in the `WHERE` clause. Columns `orders.customer_id` and `customers.customer_id` are referenced as part of the equi-join clause. Because `customers.customer_id` is a primary key column, it is logged by default, but `orders.customer_id` is not a primary key column and so must be added to the materialized view log. Also, the column `orders.order_total` is an additional filter column and so must be logged.

Therefore, add `orders.customer_id` and `orders.order_total` the materialized view log for the `oe.orders` table.

You are encouraged to analyze the defining queries of your planned materialized views and identify which columns must be added to your materialized view logs. If you try to create or refresh a materialized view that requires an added column without adding the column to the materialized view log, then your materialized view creation or refresh might fail.

Note:

To perform a fast refresh on a materialized view, you must add join columns in subqueries to the materialized view log if the join column is not a primary key and the subquery is either many to many or one to many. If the subquery is many to one, then you do not need to add the join column to the materialized view log.

See Also:

- "Creating Materialized View Logs" for information about creating a materialized view log
- "Logging Columns in a Materialized View Log"
- "Data Subsetting with Materialized Views" for information about materialized views with subqueries
- "Restrictions for Materialized Views with Subqueries" for additional information about materialized views with subqueries

36.1.4.3 Restriction on Import of Materialized View Logs to a Different Schema

Materialized view logs are exported with the schema name explicitly given in the DDL statements. Therefore, materialized view logs cannot be imported into a schema that is different than the schema from which they were exported.

An error is written to the import log file and the items are not imported if you attempt an import using the Data Pump Import utility that specifies the `REMAP_SCHEMA` import parameter to import an export dump file that contains materialized view logs in the specified schema.

36.2 Materialized View Database Mechanisms

When a materialized view is created, additional mechanisms are created at the materialized view database to support the materialized view. Specifically, at least one index is created.

Note:

The size limit for a materialized view name is 30 bytes. If you try to create a materialized view with a name larger than 30 bytes, Oracle Database returns an error.

- Indexes for Materialized Views
  At least one index is created at the remote materialized view database for each primary key and `ROWID` materialized view.

36.2.1 Indexes for Materialized Views

At least one index is created at the remote materialized view database for each primary key and `ROWID` materialized view.

For a primary key materialized view, the index corresponds to the primary key of the target master table and includes `_PK` in its name. A number is appended if an index with the same name already exists at the materialized view database. For a `ROWID` materialized view, the index is on the `ROWID` column and includes `I_SNAP$_` in its name. Additional indexes can be created by Oracle Database at the remote materialized view database to support fast refreshing of materialized views with subqueries.

36.3 Organizational Mechanisms

Several mechanisms organize the materialized views at the materialized view database. These mechanisms maintain organizational consistency between the materialized view database and its master database.

- Refresh Groups
  To preserve referential integrity and transactional (read) consistency among multiple materialized views, Oracle Database can refresh individual materialized views as part of a refresh group.
- Refresh Group Size
  There are a few trade-offs to consider when you are deciding on the size of your refresh groups.

36.3.1 Refresh Groups

To preserve referential integrity and transactional (read) consistency among multiple materialized views, Oracle Database can refresh individual materialized views as part of a refresh group.

After refreshing all of the materialized views in a refresh group, the data of all materialized views in the group correspond to the same transactionally consistent point in time.

36.3.2 Refresh Group Size

There are a few trade-offs to consider when you are deciding on the size of your refresh groups.

Oracle Database is optimized for large refresh groups. So, large refresh groups refresh faster than an equal number of materialized views in small refresh groups, if the materialized views in the groups are similar. For example, refreshing a refresh group with 100 materialized views is faster than refreshing five refresh groups with 20 materialized views each. Also, large refresh groups enable you to refresh a greater number of materialized views with only one call to a PL/SQL subprogram.

Network connectivity must be maintained while performing a refresh. If the connectivity is lost or interrupted during the refresh, then all changes are rolled back so that the database remains consistent. Therefore, in cases where the network connectivity is difficult to maintain, consider using smaller refresh groups.

There is also an optimization for null refresh. That is, if there were no changes to the master tables since the last refresh for a particular materialized view, then almost no extra time is required for the materialized view during refresh.

36.4 Refresh Process

To ensure that a materialized view is consistent with its master table, you must refresh the materialized view periodically.

- About the Refresh Process
  A **materialized view refresh** is an efficient batch operation that makes a materialized view reflect a more current state of its master table.
- Refresh Types
  Oracle Database can refresh a materialized view using either a fast, complete, or force refresh.
- Initiating a Refresh
  When creating a refresh group, you can configure the group so that Oracle Database automatically refreshes the group's materialized views at scheduled intervals. Conversely, you can omit scheduling information so that the refresh group must be refreshed manually or "on-demand." Manual refresh is an ideal solution when the refresh is performed on a system that does not always have a network connection.
- Constraints and Refresh
  To avoid any integrity constraint violations during refresh of materialized views, make non primary key integrity constraints on each materialized view deferrable.

36.4.1 About the Refresh Process

A **materialized view refresh** is an efficient batch operation that makes a materialized view reflect a more current state of its master table.

A materialized view's data does not necessarily match the current data of its master table at all times. A materialized view is a transactionally (read) consistent reflection of its master table as the data existed at a specific point in time (that is, at creation or when a refresh occurs). To keep a materialized view's data relatively current with the data of its master table, the materialized view must be refreshed periodically.

A row in a master table can be updated many times between refreshes of a materialized view, but the refresh updates the row in the materialized view only once with the current data. For example, a row in a master table might be updated 10 times since the last refresh of a materialized view, but the result is still only one update of the corresponding row in the materialized view during the next refresh.

Decide how and when to refresh each materialized view to make it more current. For example, materialized views based on master tables that applications update often might require frequent refreshes. In contrast, materialized views based on relatively static master tables usually require infrequent refreshes. In summary, analyze application characteristics and requirements to determine appropriate materialized view refresh intervals.

To refresh materialized views, Oracle Database supports several refresh types and methods of initiating a refresh.

36.4.2 Refresh Types

Oracle Database can refresh a materialized view using either a fast, complete, or force refresh.

- Complete Refresh
  To perform a **complete refresh** of a materialized view, the server that manages the materialized view executes the materialized view's defining query, which essentially re-creates the materialized view.
- Fast Refresh
  To perform a **fast refresh**, the master table that manages the materialized view first identifies the changes that occurred in the master table since the most recent refresh of the materialized view and then applies these changes to the materialized view.
- Force Refresh
  To perform a **force refresh** of a materialized view, the server that manages the materialized view attempts to perform a fast refresh. If a fast refresh is not possible, then Oracle Database performs a complete refresh.

36.4.2.1 Complete Refresh

To perform a **complete refresh** of a materialized view, the server that manages the materialized view executes the materialized view's defining query, which essentially re-creates the materialized view.

To refresh the materialized view, the result set of the query replaces the existing materialized view data. Oracle Database can perform a complete refresh for any materialized view. Depending on the amount of data that satisfies the defining query, a complete refresh can take a substantially longer amount of time to perform than a fast refresh.

Note:

If complete refresh is used for a materialized view, then set its `PCTFREE` to `0` and `PCTUSED` to `99` for maximum efficiency.

36.4.2.2 Fast Refresh

To perform a **fast refresh**, the master table that manages the materialized view first identifies the changes that occurred in the master table since the most recent refresh of the materialized view and then applies these changes to the materialized view.

Fast refreshes are more efficient than complete refreshes when there are few changes to the master table because the participating server and network replicate a smaller amount of data. You can perform fast refreshes of materialized views only when the master table has a materialized view log. Also, for fast refreshes to be faster than complete refreshes, each join column in the `CREATE MATERIALIZED VIEW` statement must have an index on it.

After a direct path load on a master table using SQL*Loader, a fast refresh does not apply the changes that occurred during the direct path load. Also, fast refresh does not apply changes that result from other types of bulk load operations on master tables. Examples of these operations include `INSERT` statements with an `APPEND` hint and `INSERT ... SELECT * FROM` statements.

If you have materialized views based on partitioned master tables, then you might be able to use Partition Change Tracking (PCT) to identify which materialized view rows correspond to a particular partition. PCT is also used to support fast refresh after partition maintenance operations on a materialized view's master table. PCT-based refresh on a materialized view is possible only if several conditions are satisfied.

See Also:

Oracle Database Data Warehousing Guide for information about PCT and about PCT-based refresh

36.4.2.3 Force Refresh

To perform a **force refresh** of a materialized view, the server that manages the materialized view attempts to perform a fast refresh. If a fast refresh is not possible, then Oracle Database performs a complete refresh.

Use the force setting when you want a materialized view to refresh if a fast refresh is not possible.

36.4.3 Initiating a Refresh

When creating a refresh group, you can configure the group so that Oracle Database automatically refreshes the group's materialized views at scheduled intervals. Conversely, you can omit scheduling information so that the refresh group must be refreshed manually or "on-demand." Manual refresh is an ideal solution when the refresh is performed on a system that does not always have a network connection.

- Scheduled Refresh
  When you create a refresh group for automatic refreshing, you must specify a scheduled refresh interval for the group during the creation process.
- On-Demand Refresh
  On-demand refresh means that the materialized view is refreshed with an explicit procedure call.

36.4.3.1 Scheduled Refresh

When you create a refresh group for automatic refreshing, you must specify a scheduled refresh interval for the group during the creation process.

When setting a group's refresh interval, consider the following characteristics:

- The dates or date expressions specifying the refresh interval must evaluate to a future point in time.
- The refresh interval must be greater than the length of time necessary to perform a refresh.
- Relative date expressions evaluate to a point in time relative to the most recent refresh date. If a network or system failure interferes with a scheduled group refresh, then the evaluation of a relative date expression could change accordingly.
- Explicit date expressions evaluate to specific points in time, regardless of the most recent refresh date.
- Consider your environment's tolerance for stale data: if there is a low tolerance, then refresh often; whereas if there is a high tolerance, then refresh less often.

The following are examples of simple date expressions that you can use to specify an interval:

- An interval of one hour is specifies as:

  ```
  SYSDATE + 1/24
  ```

- An interval of seven days is specifies as:

  ```
  SYSDATE + 7
  ```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about date arithmetic

36.4.3.2 On-Demand Refresh

On-demand refresh means that the materialized view is refreshed with an explicit procedure call.

Scheduled materialized view refreshes might not always be the appropriate solution for your environment. For example, immediately following a bulk data load into a master table, dependent materialized views no longer represent the master table's data.

You might also want to refresh your materialized views on-demand when your materialized views are integrated with a sales force automation system located on a disconnected laptop. Developers designing the sales force automation software can create an application control, such as a button, that a salesperson can use to refresh the materialized views when they are ready to transfer the day's orders to the server after establishing a network connection.

The following example illustrates an on-demand refresh of the `hr_refg` refresh group:

```
EXECUTE DBMS_REFRESH.REFRESH('hr_refg');
```

> **Note:**
>
> Do not use the `DBMS_MVIEW.REFRESH_ALL_MVIEWS` or the `DBMS_MVIEW.REFRESH_DEPENDENT` procedure to refresh materialized views used in a replication environment. Instead, use the `DBMS_REFRESH.REFRESH` or the `DBMS_MVIEW.REFRESH` procedure to refresh materialized views in a replication environment.

36.4.4 Constraints and Refresh

To avoid any integrity constraint violations during refresh of materialized views, make non primary key integrity constraints on each materialized view deferrable.

This requirement includes LOB columns with `NOT NULL` constraints. In addition, all materialized views that are related by foreign key constraints should be refreshed together or in the same refresh group.

> **Note:**
>
> Primary key constraints on materialized views might or might not be deferrable.

See Also:

Oracle Database SQL Language Reference for information about making constraints deferrable

37

Planning for Read-Only Materialized Views

Before you begin to plan your read-only materialized view environment, it is important to understand the concepts and architecture related to materialized views. After you understand concepts and architecture of read-only materialized views, there are important considerations for planning a read-only materialized view environment.

- Considerations for Master Tables
  For master tables, you must consider primary keys, foreign keys, and data types.
- Planning for Master Databases and Materialized View Databases
  Planning databases in a read-only materialized view environment includes preparing for materialized views and configuring materialized view logs.

37.1 Considerations for Master Tables

For master tables, you must consider primary keys, foreign keys, and data types.

- Primary Keys and Master Tables
  If possible, each master table should have a primary key.
- Foreign Keys and Master Tables
  When replicating tables with foreign key referential constraints, Oracle recommends that you always index foreign key columns and replicate these indexes, unless no updates and deletes are allowed in the parent table. Indexes are not replicated automatically.
- Data Type Considerations for Master Tables
  There are several considerations for data types and master tables.
- Unsupported Table Types
  Materialized views cannot be based on certain types of tables.

37.1.1 Primary Keys and Master Tables

If possible, each master table should have a primary key.

Where a primary key is not possible, each master table must have a set of columns that can be used as a unique identifier for each row of the table. If the tables that you plan to use in your replication environment do not have a primary key or a set of unique columns, then alter these tables accordingly. In addition, if you plan to create any primary key materialized views based on a master table, then that master must have a primary key.

37.1.2 Foreign Keys and Master Tables

When replicating tables with foreign key referential constraints, Oracle recommends that you always index foreign key columns and replicate these indexes, unless no updates and deletes are allowed in the parent table. Indexes are not replicated automatically.

37.1.3 Data Type Considerations for Master Tables

There are several considerations for data types and master tables.

You can create read-only materialized views based on master tables with columns that use the following data types:

- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `DATE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `ROWID`
- `CHAR`
- `NCHAR`
- `CLOB` with `BASICFILE` storage
- `NCLOB` with `BASICFILE` storage
- `BLOB` with `BASICFILE` storage
- `XMLType` stored as `CLOB`
- User-defined types that do not use type inheritance or type evolution
- Oracle-supplied types that do not use type inheritance or type evolution

Note:

`XMLType` stored as a `CLOB` is deprecated.

You cannot reference LOB columns in a `WHERE` clause of a materialized view's defining query.

You can create materialized views that use user-defined types, including column objects, object tables, `REF`s, varrays, and nested tables.

You cannot create materialized views based on master tables with columns that use the following data types:

- `FLOAT`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `LONG`
- `LONG RAW`
- `CLOB` with `SECUREFILE` storage

- `NCLOB` with `SECUREFILE` storage
- `BLOB` with `SECUREFILE` storage
- `BFILE`
- `XMLType` stored object relationally or as binary XML
- `Expression` type
- User-defined types that use type inheritance or type evolution
- Oracle-supplied types that use type inheritance or type evolution

You should convert `LONG` data types to LOBs with `BASICFILE` storage.

See Also:

Oracle Database SQL Language Reference for information about data types

37.1.4 Unsupported Table Types

Materialized views cannot be based on certain types of tables.

You cannot create materialized views based on these types of tables:

- Tables that have been compressed with the table compression feature
- Tables with virtual columns
- Temporary tables
- Tables in a flashback data archive

37.2 Planning for Master Databases and Materialized View Databases

Planning databases in a read-only materialized view environment includes preparing for materialized views and configuring materialized view logs.

- Characteristics of Master Databases and Materialized View Databases
  When you are planning your replication environment, you must decide whether the databases participating in the replication environment will be master databases or materialized view databases.
- Advantages of Master Databases
  Master databases have several advantages.
- Advantages of Materialized View Databases
  Materialized view databases have certain advantages.
- Preparing for Materialized Views
  Most problems encountered with materialized view replication result from not preparing the environment properly.
- Creating Materialized View Logs
  Create a materialized view log on a master table so that materialized views based on the master table can be fast refreshed.

- Logging Columns in a Materialized View Log
  When you create a materialized view log, you can add columns to the log to enable fast refreshes of materialized views.

37.2.1 Characteristics of Master Databases and Materialized View Databases

When you are planning your replication environment, you must decide whether the databases participating in the replication environment will be master databases or materialized view databases.

Consider the characteristics and advantages of both types of databases when you are deciding whether a particular database in your environment should be a master database or a materialized view database. One replication environment can support both master databases and materialized view databases.

Table 37-1 Characteristics of Master Databases and Materialized View Databases

Master Databases	Materialized View Databases
Might communicate with a large number of materialized view databases	Communicate with one master database
Contain large amounts of data	Contain small amounts of data that can be subsets of the master database's data

37.2.2 Advantages of Master Databases

Master databases have several advantages.

Master databases have the following advantages:

- Support for highly available data access by remote databases
- Provide support data manipulation language (DML) changes
- Can provide failover protection

37.2.3 Advantages of Materialized View Databases

Materialized view databases have certain advantages.

Materialized view databases have the following advantages:

- Support disconnected computing
- Can contain a subset of its master database's data

37.2.4 Preparing for Materialized Views

Most problems encountered with materialized view replication result from not preparing the environment properly.

Ensure that the following prerequisites are met before creating your materialized view environment:

- Ensure that the required schemas exist.

- Ensure that the required database links exist.
- Ensure that the required privileges are granted.
- Ensure that the sufficient job processes exit.

- Required Schemas at Materialized View Database
  A schema containing a materialized view in a remote database must correspond to the schema that contains the master table in the master database.
- Required Database Links for Materialized Views
  The defining query of a materialized view can use one or more database links to reference remote table data.
- Required Privileges
  Both the creator and the owner of the materialized view must be able to issue the defining `SELECT` statement of the materialized view.
- Sufficient Job Processes
  It is important that you have allocated sufficient job processes to handle the automation of your replication environment. The job processes automatically refresh materialized views.

37.2.4.1 Required Schemas at Materialized View Database

A schema containing a materialized view in a remote database must correspond to the schema that contains the master table in the master database.

Therefore, identify the schemas that contain the master tables that you want to replicate with materialized views. After you have identified the target schemas at the master database, create the corresponding accounts with the same names at the remote database. For example, if all master tables are in the `sales` schema of the `ny.example.com` database, then create a corresponding `sales` schema in the materialized view database `sf.example.com`.

See Also:

"Required Privileges for Materialized View Operations"

37.2.4.2 Required Database Links for Materialized Views

The defining query of a materialized view can use one or more database links to reference remote table data.

Before creating materialized views, the database links you plan to use must be available. Furthermore, the account that a database link uses to access a remote database defines the security context under which Oracle Database creates and subsequently refreshes a materialized view.

To ensure proper behavior, a materialized view's defining query must use a database link that includes an embedded user name and password in its definition; you cannot use a public database link when creating a materialized view. A database link with an embedded name and password always establishes connections to the remote database using the specified account. Additionally, the remote account that the link uses must have the `SELECT` privileges necessary to access the data referenced in the materialized view's defining query.

Before creating your materialized views, you must create several administrative database links. Specifically, you should create a `PUBLIC` database link from the materialized view database to the master database. Doing so makes defining your private database links easier because you do not need to include the `USING` clause in each link.

For example, the following statement creates a public database link from a materialized view database to a master database:

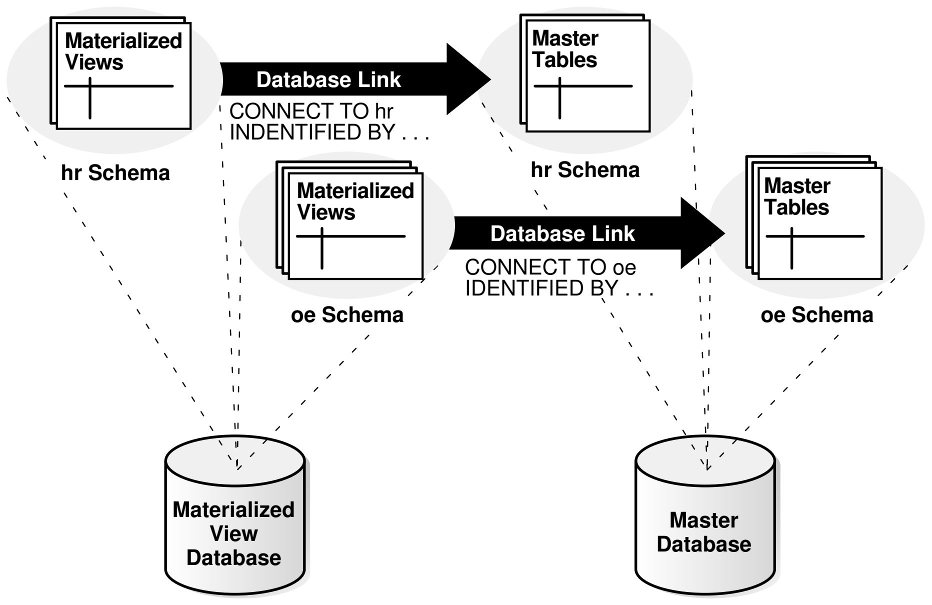
```
CREATE PUBLIC DATABASE LINK orc1.example.com USING 'orc1.example.com';
```

After the administrative database links have been created, a private database link must be created connecting each replicated materialized view schema at the materialized view database to the corresponding schema at the master database. Be sure to embed the associated master database account information in each private database link at the materialized view database. For example, the `hr` schema at a materialized view database should have a private database link to the master database that connects using the `hr` user name and password.

For example, the following statement creates a private database link from a materialized view database to a master database:

```
CREATE DATABASE LINK orc1.example.com
   CONNECT TO myuser IDENTIFIED BY password;
```

Figure 37-1 Recommended Schema and Database Link Configuration



The defining query for the materialized view cannot be modified by Virtual Private Database (VPD). VPD must return a `NULL` policy for the schema that performs both the create and refresh of the materialized view. Creating a materialized view with a non-`NULL` VPD policy is allowed when the `USING TRUSTED CONSTRAINTS` clause is specified. In this case, ensure that the materialized view behaves correctly. Materialized view results are computed based on the rows and columns filtered by VPD policy. Therefore, you must coordinate the materialized view definition with the VPD policy to ensure the correct results.

See Also:

- Distributed Database Concepts for more information about database links
- *Oracle Database Security Guide* for more information about VPD
- *Oracle Label Security Administrator's Guide* for information about Oracle Label Security

37.2.4.3 Required Privileges

Both the creator and the owner of the materialized view must be able to issue the defining `SELECT` statement of the materialized view.

The owner is the schema that contains the materialized view.

See Also:

"Required Privileges for Materialized View Operations"

37.2.4.4 Sufficient Job Processes

It is important that you have allocated sufficient job processes to handle the automation of your replication environment. The job processes automatically refresh materialized views.

By the nature of materialized view replication, each materialized view database typically has one scheduled link to the master database and requires at least one job process. Materialized view databases typically require between one and three job processes, depending on user-defined jobs and the scheduled link. Also, you need at least one job process for each degree of parallelism.

Alternatively, if your users are responsible for manually refreshing the materialized view through an application interface, then you do not need to create a scheduled link and your materialized view database requires one less job process.

The job processes are defined using the `JOB_QUEUE_PROCESSES` initialization parameter. This initialization parameter is modifiable. Therefore, you can modify it while an instance is running. Oracle Database automatically determines the interval for job processes. That is, Oracle Database determines when the job processes should "wake up" to execute jobs.

See Also:

Oracle Database Reference

37.2.5 Creating Materialized View Logs

Create a materialized view log on a master table so that materialized views based on the master table can be fast refreshed.

Before creating materialized views for a remote materialized view database, ensure that you create the necessary materialized view logs at the master database. A materialized view log is necessary for every master table that supports at least one materialized view with fast refreshes.

To create a materialized view log, you need the following privileges:

- `CREATE ANY TABLE`
- `CREATE ANY TRIGGER`
- `SELECT` (on the materialized view log's master table)
- `COMMENT ANY TABLE`

To create a materialized view log:

1. Connect to the master database as a user with the required privileges to create a materialized view log on the intended table.
2. Run the `CREATE MATERIALIZED VIEW LOG` statement.

   When you create a materialized view log on an object table, you must log the object identifier by specifying the `WITH OBJECT ID` clause, but you can also specify that the primary key is logged if the object identifier is primary key-based.

Example 37-1 Creating a Materialized View Log

```
CREATE MATERIALIZED VIEW LOG ON hr.employees;
```

Example 37-2 Creating a Materialized View Log on an Object Table

The following SQL statement creates the `categories_typ` user-defined type:

```
CREATE TYPE oe.category_typ AS OBJECT
   (category_name           VARCHAR2(50),
    category_description    VARCHAR2(1000),
    category_id             NUMBER(2));
/
```

When you create an object table based on this type, you can either specify that the object identifier should be system-generated or primary key-based:

```
CREATE TABLE oe.categories_tab_sys OF oe.category_typ
    (category_id    PRIMARY KEY)
    OBJECT ID SYSTEM GENERATED;

CREATE TABLE oe.categories_tab_pkbased OF oe.category_typ
    (category_id    PRIMARY KEY)
    OBJECT ID PRIMARY KEY;
```

For example, the following statement creates a materialized view log for the `categories_tab_sys` object table and specifies that the object identifier column be logged:

```
CREATE MATERIALIZED VIEW LOG ON oe.categories_tab_sys
   WITH OBJECT ID;
```

The following statement creates a materialized view log for the `categories_tab_pkbased` object table and specifies that the primary key column be logged along with the object identifier column:

```
CREATE MATERIALIZED VIEW LOG ON oe.categories_tab_pkbased
   WITH OBJECT ID, PRIMARY KEY;
```

See Also:

- "Materialized View Log"
- *Oracle Database SQL Language Reference*

37.2.6 Logging Columns in a Materialized View Log

When you create a materialized view log, you can add columns to the log to enable fast refreshes of materialized views.

1. Connect to the master database as a user with the required privileges to create or alter a materialized view log on the intended table.
2. Do one of the following:
   - Run the `CREATE MATERIALIZED VIEW LOG` statement and specify the columns to log.
   - Run the `ALTER MATERIALIZED VIEW LOG` statement and specify the columns to log.

Example 37-3 Logging Columns When Creating a Materialized View

To create the materialized view log on the `oe.orders` table with the `orders.customer_id` and `orders.order_total` columns added, issue the following statement:

```
CREATE MATERIALIZED VIEW LOG ON oe.orders
  WITH PRIMARY KEY (customer_id,order_total);
```

Example 37-4 Logging Columns of an Existing Materialized View

You can add the `orders.customer_id` and `orders.order_total` columns to the materialized view log on the `oe.orders` table by issuing the following statement:

```
ALTER MATERIALIZED VIEW LOG ON oe.orders ADD (customer_id,order_total);
```

Example 37-5 Logging the Attributes of Column Objects

If you are using user-defined data types, then the attributes of column objects can be logged in the materialized view log. For example, the `oe.customers` table has the `cust_address.postal_code` attribute, which can be logged in the materialized view log by issuing the following statement:

```
ALTER MATERIALIZED VIEW LOG ON oe.customers ADD (cust_address.postal_code);
```

See Also:

- "Columns Logged in the Materialized View Log"
- *Oracle Database SQL Language Reference*

38

Creating and Managing Read-Only Materialized Views

You can create and manage read-only materialized views and refresh groups. You can also refresh materialized views.

- Creating Read-Only Materialized Views
  Create a read-only materialized view to replicate a master table's data in a materialized view database.
- Creating Refresh Groups
  Add materialized views to a refresh group to ensure transactional consistency between the related materialized views in the refresh group.
- Refreshing Materialized Views
  Refreshing a materialized view synchronizes the data in the materialized view's master(s) and the data in the materialized view.
- Determining the Fast Refresh Capabilities of a Materialized View
  You can determine whether a materialized view is fast refreshable by attempting to create the materialized view with the `REFRESH FAST` clause or by using the `DBMS_MVIEW.EXPLAIN_MVIEW` procedure.
- Adding a New Materialized View Database
  After you have created a materialized view environment with one or more materialized view databases, you might need to add new materialized view databases.
- Monitoring Materialized View Logs
  You can run queries to display information about the materialized view logs at a master database.
- Monitoring Materialized Views
  You can run queries to display information about the materialized views and refresh groups.

38.1 Creating Read-Only Materialized Views

Create a read-only materialized view to replicate a master table's data in a materialized view database.

Before creating a materialized view to replicate data between a master database and a materialized view database, the database links you plan to use must be available.

1. Connect to the database as a user with the required privileges to create a materialized view.
2. Run the `CREATE MATERIALIZED VIEW` statement.

Example 38-1 Creating a Primary Key Materialized View

```
CREATE MATERIALIZED VIEW hr.employees_mv1 WITH PRIMARY KEY
  AS SELECT * FROM hr.employees@orc1.example.com;
```

Example 38-2   Creating a ROWID Materialized View

```
CREATE MATERIALIZED VIEW oe.orders REFRESH WITH ROWID AS
  SELECT * FROM oe.orders@orc1.example.com;
```

Example 38-3   Creating an Object Materialized View

After the required types are created at the materialized view database, you can create an object materialized view by specifying the `OF` *type* clause.

For example, suppose the following SQL statements create the `oe.categories_tab` object table at the `orc1.example.com` master database:

```
CREATE TYPE oe.category_typ AS OBJECT
   (category_name           VARCHAR2(50),
    category_description    VARCHAR2(1000),
    category_id             NUMBER(2));
/

CREATE TABLE oe.categories_tab OF oe.category_typ
    (category_id    PRIMARY KEY);
```

To create materialized views that can be fast refreshed based on the `oe.categories_tab` master table, create a materialized view log for this table:

```
CREATE MATERIALIZED VIEW LOG ON oe.categories_tab WITH OBJECT ID;
```

The `WITH OBJECT ID` clause is required when you create a materialized view log on an object table.

After you create the `oe.category_typ` type at the materialized view database with the same object identifier as the same type at the master database, you can create an object materialized view based on the `oe.categories_tab` object table using the `OF` *type* clause, as in the following SQL statement:

```
CREATE MATERIALIZED VIEW oe.categories_objmv OF oe.category_typ
   REFRESH FAST
   AS SELECT * FROM oe.categories_tab@orc1.example.com;
```

Here, *type* is `oe.category_typ`.

Note:

The types must be the same at the materialized view database and master database. See "Type Agreement at Replication Databases" for more information.

See Also:

- Read-Only Materialized View Concepts for several examples that create materialized views
- "Required Privileges for Materialized View Operations" for information about the privileges required to create materialized views
- "Required Database Links for Materialized Views"
- "Materialized Views Based on Object Tables"
- *Oracle Database SQL Language Reference*

38.2 Creating Refresh Groups

Add materialized views to a refresh group to ensure transactional consistency between the related materialized views in the refresh group.

When a refresh group is refreshed, all materialized views that are added to a particular refresh group are refreshed at the same time.

1. Connect to the materialized view database as an administrative user with the required privileges to create a refresh group and add materialized views to it.
2. Run the `DBMS_REFRESH.MAKE` procedure to create the refresh group.
3. Run the `DBMS_REFRESH.ADD` procedure one or more times to add materialized views to the refresh group.

Example 38-4 Creating a Refresh Group

This example creates a refresh group and adds two materialized views to it.

```
BEGIN
   DBMS_REFRESH.MAKE (
      name => 'mviewadmin.hr_refg',
      list => '',
      next_date => SYSDATE,
      interval => 'SYSDATE + 1/24',
      implicit_destroy => FALSE,
      rollback_seg => '',
      push_deferred_rpc => TRUE,
      refresh_after_errors => FALSE);
END;
/

BEGIN
    DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.countries_mv1',
      lax => TRUE);
END;
/

BEGIN
    DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.departments_mv1',
      lax => TRUE);
```

```
END;
/
```

See Also:

"Refresh Groups"

38.3 Refreshing Materialized Views

Refreshing a materialized view synchronizes the data in the materialized view's master(s) and the data in the materialized view.

You can either refresh all of the materialized views in a refresh group at once, or you can refresh materialized views individually. If you have applications that depend on multiple materialized views at a materialized view database, then Oracle recommends using refresh groups so that the data is transactionally consistent in all of the materialized views used by the application.

1. Connect to the materialized view database as a user with the required privileges to refresh a refresh group or an individual materialized view.
2. Do one of the following:
   - Run the `DBMS_REFRESH.REFRESH` procedure to refresh a refresh group.
   - Run the `DBMS_MVIEW.REFRESH` procedure to refresh an individual materialized view.

Example 38-5 Refreshing a Refresh Group

The following example refreshes the `hr_refg` refresh group:

```
EXECUTE DBMS_REFRESH.REFRESH ('hr_refg');
```

Example 38-6 Refreshing an Individual Materialized View

The following example refreshes the `hr.departments_mv` materialized view:

```
BEGIN
   DBMS_MVIEW.REFRESH (
     list   =>  'hr.departments_mv',
     method =>  '?');
END;
/
```

Note:

Do not use the `DBMS_MVIEW.REFRESH_ALL_MVIEWS` or the `DBMS_MVIEW.REFRESH_DEPENDENT` procedure to refresh materialized views. Instead, use the `DBMS_REFRESH.REFRESH` or the `DBMS_MVIEW.REFRESH` procedure to refresh materialized views in a replication environment.

See Also:

- "Required Privileges for Materialized View Operations" for information about the privileges required to create materialized views
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_MVIEW package

38.4 Determining the Fast Refresh Capabilities of a Materialized View

You can determine whether a materialized view is fast refreshable by attempting to create the materialized view with the REFRESH FAST clause or by using the DBMS_MVIEW.EXPLAIN_MVIEW procedure.

A fast refresh uses materialized view logs to update only the rows that have changed since the last refresh. To determine whether a materialized view is fast refreshable, create the materialized view with the REFRESH FAST clause. Oracle Database returns errors if the materialized view violates any restrictions for subquery materialized views. If you specify force refresh, then you might not receive any errors because, when a force refresh is requested, Oracle Database automatically performs a complete refresh if it cannot perform a fast refresh.

You can also use the EXPLAIN_MVIEW procedure in the DBMS_MVIEW package to determine the following information about an existing materialized view or a proposed materialized view that does not yet exist:

- The capabilities of a materialized view
- Whether each capability is possible
- If a capability is not possible, then why it is not possible

This information can be stored in a varray or in the MV_CAPABILITIES_TABLE. To store the information in the table, before you run the EXPLAIN_MVIEW procedure, you must build this table by running the utlxmv.sql script in the *Oracle_home*/rdbms/admin directory.

To determine the fast refresh capabilities of a materialized view:

1. Connect to the materialized view database as an administrative user.
2. Do one of the following:
   - Create the materialized view with the REFRESH FAST clause.
   - Run the DBMS_MVIEW.EXPLAIN_MVIEW procedure.

Example 38-7 Creating a Materialized View with the FAST REFRESH Clause

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
 SELECT * FROM oe.orders@orc1.example.com o
 WHERE EXISTS
   (SELECT * FROM oe.customers@orc1.example.com c
    WHERE o.customer_id = c.customer_id AND c.credit_limit > 10000);
```

Example 38-8 Determining the Refresh Capabilities of an Existing Materialized View

For example, to determine the capabilities of the oe.orders materialized view, enter:

```
EXECUTE DBMS_MVIEW.EXPLAIN_MVIEW ('oe.orders');
```

Example 38-9 Determining the Refresh Capabilities of a Materialized View That Does Not Yet Exist

Or, if the materialized view does not yet exist, then you can supply the query that you want to use to create it:

```
BEGIN
  DBMS_MVIEW.EXPLAIN_MVIEW ('SELECT * FROM oe.orders@orc1.example.com o
    WHERE EXISTS (SELECT * FROM oe.customers@orc1.example.com c
    WHERE o.customer_id = c.customer_id AND c.credit_limit > 500)');
END;
/
```

Query the `MV_CAPABILITIES_TABLE` to see the results.

Query the `MV_CAPABILITIES_TABLE` to see the results.

Note:

The `MV_CAPABILITIES_TABLE` does not show materialized view refresh capabilities that depend on prebuilt container tables. For example, complete refresh is required after a partition maintenance operation on a prebuilt container table, but the `MV_CAPABILITIES_TABLE` does not show this limitation.

See Also:

- "Restrictions for Materialized Views with Subqueries"
- "Materialized View Log"
- *Oracle Database Data Warehousing Guide* for more information about the `EXPLAIN_MVIEW` procedure

38.5 Adding a New Materialized View Database

After you have created a materialized view environment with one or more materialized view databases, you might need to add new materialized view databases.

You might encounter problems when you try to perform a fast refresh on the materialized views you create at a new materialized view database if both of the following conditions are true:

- Materialized views at the new materialized view database and existing materialized views at other materialized view databases are based on the same master table.
- Existing materialized views can be refreshed while you create the new materialized views at the new materialized view database.

The problem arises when the materialized view logs for the master tables are purged before a new materialized view can perform its first fast refresh. If this happens and you try to perform a fast refresh on the materialized views at the new materialized view database, then you might encounter the following errors:

```
ORA-12004 REFRESH FAST cannot be used for materialized view materialized_view_name
ORA-12034 materialized view log on materialized_view_name younger than last refresh
```

If you receive these errors, then the only solution is to perform a complete refresh of the new materialized view. To avoid this problem, create a dummy materialized view at the new materialized view database before you create your production materialized views. The dummy materialized view ensures that the materialized view log will not be purged while your production materialized views are being created.

If you choose to create a dummy materialized view at the materialized view database, complete the following steps:

1. Create a dummy materialized view called `dummy_mview` based on the master table. For example, to create a dummy materialized view based on a master table named `sales`, issue the following statement at the new materialized view database:

   ```
   CREATE MATERIALIZED VIEW dummy_mview REFRESH FAST AS
     SELECT * FROM pr.sales@orc1.example.com WHERE 1=0;
   ```

2. Create your production materialized views at the new materialized view database.
3. Perform fast refresh of your production materialized views at the new materialized view database.
4. Drop the dummy materialized view.

38.6 Monitoring Materialized View Logs

You can run queries to display information about the materialized view logs at a master database.

- Listing Information About the Materialized View Logs at a Master Database
  A materialized view log enables you to perform a fast refresh on materialized views based on a master. A master can be a master table or a master materialized view.
- Listing the Materialized Views that Use a Materialized View Log
  More than one materialized view can use a materialized view log.

38.6.1 Listing Information About the Materialized View Logs at a Master Database

A materialized view log enables you to perform a fast refresh on materialized views based on a master. A master can be a master table or a master materialized view.

If you have materialized view logs based at a master, then you can use the query in this section to list the following information about them:

- The name of each log table that stores the materialized view log data
- The owner of each materialized view log
- The master on which each materialized view log is based
- Whether a materialized view log is a row id materialized view log
- Whether a materialized view log is a primary key materialized view log
- Whether the materialized view log is an object id materialized view log
- Whether a materialized view log has filter columns

To view this information, complete the following steps:

1. Connect to the master database as an administrative user.
2. Run the following query:

```
COLUMN LOG_TABLE HEADING 'Log Table' FORMAT A20
COLUMN LOG_OWNER HEADING 'Log|Owner' FORMAT A5
COLUMN MASTER HEADING 'Master' FORMAT A15
COLUMN ROWIDS HEADING 'Row|ID?' FORMAT A3
COLUMN PRIMARY_KEY HEADING 'Primary|Key?' FORMAT A7
COLUMN OBJECT_ID HEADING 'Object|ID?' FORMAT A6
COLUMN FILTER_COLUMNS HEADING 'Filter|Columns?' FORMAT A8

SELECT DISTINCT LOG_TABLE,
       LOG_OWNER,
       MASTER,
       ROWIDS,
       PRIMARY_KEY,
       OBJECT_ID,
       FILTER_COLUMNS
    FROM DBA_MVIEW_LOGS
    ORDER BY 1;
```

Your output looks similar to the following:

```
                     Log                        Row Primary Object Filter
Log Table            Owner Master               ID? Key?    ID?    Columns?
-------------------- ----- --------------- --- ------- ------ --------
MLOG$_COUNTRIES      HR    COUNTRIES       NO  YES     NO     NO
MLOG$_DEPARTMENTS    HR    DEPARTMENTS     NO  YES     NO     NO
MLOG$_EMPLOYEES      HR    EMPLOYEES       NO  YES     NO     NO
MLOG$_JOBS           HR    JOBS            NO  YES     NO     NO
MLOG$_JOB_HISTORY    HR    JOB_HISTORY     NO  YES     NO     NO
MLOG$_LOCATIONS      HR    LOCATIONS       NO  YES     NO     NO
MLOG$_REGIONS        HR    REGIONS         NO  YES     NO     NO
```

38.6.2 Listing the Materialized Views that Use a Materialized View Log

More than one materialized view can use a materialized view log.

If you have materialized view logs based at a master, then you can use the query in this section to list the following the materialized views that use each log:

- The name of each log table that stores the materialized view log data
- The owner of each materialized view log
- The master on which each materialized view log is based
- The materialized view identification number of each materialized view that uses the materialized view log
- The name of each materialized view that uses the materialized view log

To view this information, complete the following steps:

1. Connect to the master database as an administrative user.
2. Run the following query:

```
COLUMN LOG_TABLE HEADING 'Mview|Log Table' FORMAT A20
COLUMN LOG_OWNER HEADING 'Mview|Log Owner' FORMAT A10
COLUMN MASTER HEADING 'Master' FORMAT A20
COLUMN MVIEW_ID HEADING 'Mview|ID' FORMAT 9999
COLUMN NAME HEADING 'Mview Name' FORMAT A20
```

```
SELECT L.LOG_TABLE, L.LOG_OWNER, B.MASTER, B.MVIEW_ID, R.NAME
FROM ALL_MVIEW_LOGS L, ALL_BASE_TABLE_MVIEWS B, ALL_REGISTERED_MVIEWS R
WHERE B.MVIEW_ID = R.MVIEW_ID
AND B.OWNER = L.LOG_OWNER
AND B.MASTER = L.MASTER;
```

Your output looks similar to the following:

```
Mview                Mview                                Mview
Log Table            Log Owner  Master                       ID Mview Name
-------------------- ---------- -------------------- ----- --------------------
MLOG$_COUNTRIES      HR         COUNTRIES                21 COUNTRIES_MV1
MLOG$_DEPARTMENTS    HR         DEPARTMENTS              22 DEPARTMENTS_MV1
MLOG$_EMPLOYEES      HR         EMPLOYEES                23 EMPLOYEES_MV1
MLOG$_JOBS           HR         JOBS                     24 JOBS_MV1
MLOG$_JOB_HISTORY    HR         JOB_HISTORY              25 JOB_HISTORY_MV1
MLOG$_LOCATIONS      HR         LOCATIONS                26 LOCATIONS_MV1
MLOG$_REGIONS        HR         REGIONS                  27 REGIONS_MV1
```

38.7 Monitoring Materialized Views

You can run queries to display information about the materialized views and refresh groups.

- Listing Information About Materialized Views
  You can run queries to display information about the materialized views.
- Listing Information About the Refresh Groups at a Materialized View Database
  Each refresh group at a materialized view database is associated with a refresh job that refreshes the materialized views in the refresh group at a set interval.
- Determining the Job ID for Each Refresh Job at a Materialized View Database
  Query the `DBA_REFRESH` and `DBA_JOBS` views to determine the job identification number for each refresh job at a materialized view database.
- Determining Which Materialized Views Are Currently Refreshing
  Query the `V$MVREFRESH` view to determine which materialized views are currently refreshing.

38.7.1 Listing Information About Materialized Views

You can run queries to display information about the materialized views.

- Listing Master Database Information For Materialized Views
  Query the `DBA_MVIEWS` view to list the master database information for materialized views.
- Listing the Properties of Materialized Views
  Query the `DBA_MVIEWS` view to list the properties of materialized views.

38.7.1.1 Listing Master Database Information For Materialized Views

Query the `DBA_MVIEWS` view to list the master database information for materialized views.

Complete the following steps to show the master database for each materialized view at a replication database and whether the materialized view can be fast refreshed:

1. Connect to the materialized view database as an administrative user.
2. Run the following query:

```
COLUMN MVIEW_NAME HEADING 'Materialized|View Name' FORMAT A15
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN MASTER_LINK HEADING 'Master Link' FORMAT A30
COLUMN Fast_Refresh HEADING 'Fast|Refreshable?' FORMAT A16

SELECT MVIEW_NAME,
       OWNER,
       MASTER_LINK,
       DECODE(FAST_REFRESHABLE,
              'NO', 'NO',
              'DML', 'YES',
              'DIRLOAD', 'DIRECT LOAD ONLY',
              'DIRLOAD_DML', 'YES',
              'DIRLOAD_LIMITEDDML', 'LIMITED') Fast_Refresh
    FROM DBA_MVIEWS;
```

Your output looks similar to the following:

```
Materialized                                              Fast
View Name       Owner      Master Link                    Refreshable?
--------------- ---------- ------------------------------ ----------------
COUNTRIES_MV1   HR         @ORC1.EXAMPLE.COM              YES
DEPARTMENTS_MV1 HR         @ORC1.EXAMPLE.COM              YES
EMPLOYEES_MV1   HR         @ORC1.EXAMPLE.COM              YES
JOBS_MV1        HR         @ORC1.EXAMPLE.COM              YES
JOB_HISTORY_MV1 HR         @ORC1.EXAMPLE.COM              YES
LOCATIONS_MV1   HR         @ORC1.EXAMPLE.COM              YES
REGIONS_MV1     HR         @ORC1.EXAMPLE.COM              YES
```

38.7.1.2 Listing the Properties of Materialized Views

Query the `DBA_MVIEWS` view to list the properties of materialized views.

You can use the query in this section to list the following information about the materialized views at the current replication database:

- The name of each materialized view
- The owner of each materialized view
- The refresh method used by each materialized view: `COMPLETE`, `FORCE`, `FAST`, or `NEVER`
- The last date on which each materialized view was refreshed

To view this information, complete the following steps:

1. Connect to the materialized view database as an administrative user.
2. Run the following query to list this information:

To view this information, complete the following steps:

1. Connect to the materialized view database as an administrative user.
2. Run the following query:

```
COLUMN MVIEW_NAME HEADING 'Materialized|View Name' FORMAT A15
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN REFRESH_METHOD HEADING 'Refresh|Method' FORMAT A10
COLUMN LAST_REFRESH_DATE HEADING 'Last|Refresh|Date'
COLUMN LAST_REFRESH_TYPE HEADING 'Last|Refresh|Type' FORMAT A15

SELECT MVIEW_NAME,
       OWNER,
       REFRESH_METHOD,
       LAST_REFRESH_DATE,
       LAST_REFRESH_TYPE
    FROM DBA_MVIEWS;
```

Your output looks similar to the following:

```
                                        Last      Last
Materialized               Refresh    Refresh   Refresh
View Name       Owner      Method     Date      Type
--------------- ---------- ---------- --------- ---------------
COUNTRIES_MV1   HR         FAST       21-OCT-03 FAST
DEPARTMENTS_MV1 HR         FAST       21-OCT-03 FAST
EMPLOYEES_MV1   HR         FAST       21-OCT-03 FAST
JOBS_MV1        HR         FAST       21-OCT-03 FAST
JOB_HISTORY_MV1 HR         FAST       21-OCT-03 FAST
LOCATIONS_MV1   HR         FAST       21-OCT-03 FAST
REGIONS_MV1     HR         FAST       21-OCT-03 FAST
```

38.7.2 Listing Information About the Refresh Groups at a Materialized View Database

Each refresh group at a materialized view database is associated with a refresh job that refreshes the materialized views in the refresh group at a set interval.

You can query the DBA_REFRESH data dictionary view to list the following information about the refresh jobs at a materialized view database:

- The name of the refresh group.
- The owner of the refresh group.
- Whether the refresh job is broken.
- The next date and time when the refresh job will run.
- The current interval setting for the refresh job. The interval setting specifies the amount of time between the start of a job and the next start of the same job.

To view this information, complete the following steps:

1. Connect to the materialized view database as an administrative user.
2. Run the following query:

```
COLUMN RNAME HEADING 'Refresh|Group|Name' FORMAT A10
COLUMN ROWNER HEADING 'Refresh|Group|Owner' FORMAT A10
COLUMN BROKEN HEADING 'Broken?' FORMAT A7
COLUMN next_refresh HEADING 'Next Refresh'
COLUMN INTERVAL HEADING 'Interval' FORMAT A20

SELECT RNAME,
           ROWNER,
       BROKEN,
       TO_CHAR(NEXT_DATE, 'DD-MON-YYYY HH:MI:SS AM') next_refresh,
       INTERVAL
    FROM DBA_REFRESH
    ORDER BY 1;
```

Your output looks similar to the following:

```
Refresh    Refresh
Group      Group
Name       Owner      Broken? Next Refresh            Interval
---------- ---------- ------- ----------------------- --------------------
HR_REFG    MVIEWADMIN N       24-OCT-2003 07:18:44 AM SYSDATE + 1/24
```

The N in the Broken? column means that the job is not broken. Therefore, the refresh job will run at the next start time. A Y in this column means that the job is broken.

38.7.3 Determining the Job ID for Each Refresh Job at a Materialized View Database

Query the DBA_REFRESH and DBA_JOBS views to determine the job identification number for each refresh job at a materialized view database.

You can run a query to list the following information about the refresh jobs at a materialized view database:

- The job identification number of each refresh job. Each job created by Oracle Scheduler is assigned a unique identification number.
- The privilege schema, which is the schema whose default privileges apply to the job.
- The schema that owns each refresh job.
- The name of the refresh group that the job refreshes.
- The status of the refresh job, either normal or broken.

To view this information, complete the following steps:

1. Connect to the materialized view database as an administrative user.
2. Run the following query:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN PRIV_USER HEADING 'Privilege|Schema' FORMAT A10
COLUMN RNAME HEADING 'Refresh|Group|Name' FORMAT A10
COLUMN ROWNER HEADING 'Refresh|Group|Owner' FORMAT A10
COLUMN BROKEN HEADING 'Broken?' FORMAT A7

SELECT J.JOB,
       J.PRIV_USER,
       R.ROWNER,
       R.RNAME,
       J.BROKEN
    FROM DBA_REFRESH R, DBA_JOBS J
    WHERE R.JOB = J.JOB
    ORDER BY 1;
```

Your output looks similar to the following:

```
                   Refresh    Refresh
        Privilege  Group      Group
 Job ID Schema     Owner      Name       Broken?
------- ---------- ---------- ---------- -------
     21 MVIEWADMIN MVIEWADMIN HR_REFG    N
```

The N in the Broken? column means that the job is not broken. Therefore, the job will run at the next start time. A Y in this column means that the job is broken.

38.7.4 Determining Which Materialized Views Are Currently Refreshing

Query the V$MVREFRESH view to determine which materialized views are currently refreshing.

Complete the following steps to show the materialized views that are currently refreshing:

1. Connect to the materialized view database as an administrative user.

2. Run the following query:

```
COLUMN SID HEADING 'Session|Identifier' FORMAT 9999
COLUMN SERIAL# HEADING 'Serial|Number' FORMAT 999999
COLUMN CURRMVOWNER HEADING 'Owner' FORMAT A15
COLUMN CURRMVNAME HEADING 'Materialized|View' FORMAT A25

SELECT * FROM V$MVREFRESH;
```

Your output looks similar to the following:

```
   Session  Serial                 Materialized
Identifier  Number Owner           View
---------- ------- --------------- -------------------------
        19     233 HR              COUNTRIES_MV
         5     647 HR              EMPLOYEES_MV
```

39 Troubleshooting Problems with Read-Only Materialized Views

You can diagnose and solve problems with database links, materialized view creation, and materialized view refresh.

- Diagnosing Problems with Database Links
  If you think a database link is not functioning properly, then you can drop and re-create it using Oracle Enterprise Manager Cloud Control, SQL*Plus, or another tool.
- Problems Creating Materialized Views
  There are items to check if you have problems creating a materialized view.
- Refresh Problems
  You can diagnose and solve common refresh problems.
- Advanced Troubleshooting of Refresh Problems
  There are several items you can check if you have problems with refreshing a materialized view.

39.1 Diagnosing Problems with Database Links

If you think a database link is not functioning properly, then you can drop and re-create it using Oracle Enterprise Manager Cloud Control, SQL*Plus, or another tool.

- Ensure that the database link name is the same as the global name of the target database.
- Ensure that the scheduled interval is what you want.
- Ensure that the scheduled interval is not shorter than the required execution time.

If you used a connection qualifier in a database link to a given database, then the other databases that link to that database must have the same connection qualifier. For example, suppose you create a database link as follows:

```
CREATE DATABASE LINK dbs1.example.com@myethernet CONNECT TO myadmin
  IDENTIFIED BY password USING 'connect_string_myethernet';
```

All the databases, whether master databases or materialized view databases, associated with `dbs1.example.com@myethernet` must include `myethernet` as the connection qualifier.

See Also:

- "Using Connection Qualifiers to Specify Service Names Within Link Names" for more information database links and connection qualifiers
- "Required Database Links for Materialized Views"

39.2 Problems Creating Materialized Views

There are items to check if you have problems creating a materialized view.

If you unsuccessfully attempt to create a materialized view, then try the following:

- Ensure that you have the necessary privileges to create the materialized view. You need `SELECT` privilege on the master table and its materialized view log. See "Required Privileges" for more information.
- If you are trying to create a fast refresh primary key or subquery materialized view, then ensure that the materialized view log on the master table logs primary keys.
- If you are trying to create a fast refresh rowid materialized view, then ensure that the materialized view log on the master table logs rowids.
- Check if the materialized view log has the required columns added for subquery materialized views. See "Logging Columns in a Materialized View Log" for information.
- Check if the materialized view log exists for all tables that are involved in a fast refresh materialized view. If the materialized view contains a subquery, then each table referenced in the subquery should have a materialized view log.

39.3 Refresh Problems

You can diagnose and solve common refresh problems.

- Common Refresh Problems
  Several common factors can prevent the automatic refresh of a group of materialized views.
- Automatic Refresh Retries
  When Oracle Database fails to refresh a refresh group automatically, the refresh group remains due for its refresh to complete.
- Fast Refresh Errors at New Materialized View Databases
  In some cases, a materialized view log for a master table might be purged during the creation of a materialized view at a new materialized view database.
- Materialized Views Continually Refreshing
  If you encounter a situation where Oracle Database continually refreshes a group of materialized views, then check the group's refresh interval.
- Materialized View Logs Growing Too Large
  If a materialized view log at a master database is growing too large, then check to see whether a network or database failure has prevented the master database from becoming aware that a materialized view has been dropped.

39.3.1 Common Refresh Problems

Several common factors can prevent the automatic refresh of a group of materialized views.

These factors include the following:

- The lack of a job slave at the materialized view database
- An intervening network or server failure
- An intervening server shutdown

When a refresh group is experiencing problems, ensure that none of the preceding situations is preventing Oracle Database from completing group refreshes.

39.3.2 Automatic Refresh Retries

When Oracle Database fails to refresh a refresh group automatically, the refresh group remains due for its refresh to complete.

Oracle Database will retry an automatic refresh of a group with the following behavior:

- Oracle Database retries the refresh group refresh first one minute later, then two minutes later, four minutes later, and so on, with the retry interval doubling with each failed attempt to refresh the group.
- Oracle Database does not allow the retry interval to exceed the refresh interval itself.
- Oracle Database retries the automatic refresh up to sixteen times.

If after 16 attempts to refresh a refresh group Oracle Database continues to encounter errors, then Oracle Database considers the group broken. You can query the `BROKEN` column of the `USER_REFRESH` and `USER_REFRESH_CHILDREN` data dictionary views to see the current status of a refresh group.

The errors causing Oracle Database to consider a refresh group broken are recorded in a trace file. After you correct the problems preventing a refresh group from refreshing successfully, you must refresh the refresh group manually. Oracle Database then resets the broken flag so that automatic refreshes can happen again.

See Also:

The name of the materialized view trace file is of the form j*n*, where *n* is operating system specific. See the Oracle documentation for your operating system for the name on your system.

39.3.3 Fast Refresh Errors at New Materialized View Databases

In some cases, a materialized view log for a master table might be purged during the creation of a materialized view at a new materialized view database.

When this happens, you might encounter the following errors:

```
ORA-12004 REFRESH FAST cannot be used for materialized view materialized_view_name
ORA-12034 materialized view log on materialized_view_name younger than last refresh
```

See Also:

"Adding a New Materialized View Database" for a complete description of how to avoid this problem.

39.3.4 Materialized Views Continually Refreshing

If you encounter a situation where Oracle Database continually refreshes a group of materialized views, then check the group's refresh interval.

Oracle Database evaluates a refresh group's automatic refresh interval before starting the refresh. If a refresh group's refresh interval is less than the amount of time it takes to refresh all materialized views in the group, then Oracle Database continually starts a refresh group refresh each time the job slave checks the queue of outstanding jobs.

39.3.5 Materialized View Logs Growing Too Large

If a materialized view log at a master database is growing too large, then check to see whether a network or database failure has prevented the master database from becoming aware that a materialized view has been dropped.

You might need to purge part of the materialized view log or unregister the unused materialized view database.

39.4 Advanced Troubleshooting of Refresh Problems

There are several items you can check if you have problems with refreshing a materialized view.

If you have a problem refreshing a materialized view, then try the following:

- Check the `NEXT_DATE` value in the `DBA_REFRESH_CHILDREN` view to determine if the refresh has been scheduled.
- If the refresh interval has passed, then check the `DBA_REFRESH` view for the associated job number for the materialized view refresh and then diagnose the problem with job queues.
- Check if there are job slaves running. Check the `JOB_QUEUE_PROCESSES` initialization parameter, query the `DBA_JOBS_RUNNING` view, and use your operating system to check if the job slaves are still running.
- You also might encounter an error if you attempt to define a master detail relationship between two materialized views. You should define master detail relationships only on the master tables by using declarative referential integrity constraints. The related materialized views should then be placed in the same refresh group to preserve this relationship. However, you can define deferred (or deferrable) constraints on materialized views.
- Materialized views in the same refresh groups have their rows updated in a single transaction. Such a transaction can be very large, requiring either a large rollback segment at the materialized view database, with the rollback segment specified to be used during refresh, or more frequent refreshes to reduce the transaction size.
- If Oracle error `ORA-12004` occurs, then the master database might have run out of rollback segments when trying to maintain the materialized view log, or the materialized view log might be out of date. For example, the materialized view log might have been purged or re-created.
- Complete refreshes of a single materialized view internally use the `TRUNCATE` feature to increase speed and reduce rollback segment requirements. However, until the materialized view refresh is complete, users might temporarily see no data in the materialized view. Refreshes of multiple materialized views (for example, refresh groups) do not use the `TRUNCATE` feature.

- Reorganization of the master table (for example, to reclaim system resources) should `TRUNCATE` the master table to force rowid materialized views to do complete refreshes. Otherwise, the materialized views have incorrect references to master table rowids. You use the `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION` procedures in the `DBMS_MVIEW` package to reorganize a master table.
- If while refreshing you see an `ORA-00942` (table or view does not exist), then check your database links and ensure that you still have the required privileges on the master table and the materialized view log.
- If a fast refresh was succeeding but then fails, then check whether:
  - The materialized view log was truncated, purged, or dropped.
  - You still have the required privileges on the materialized view log.
- If a force refresh takes an inordinately long time, then check if the materialized view log used by the refresh has been dropped.
- If the materialized view was created with `BUILD DEFERRED`, and its first fast refresh fails, then ensure that a previous complete refresh was done successfully before checking for other problems.

Part VII
Appendixes

Appendixes contain supplemental material for this document.

- Support for DBMS_JOB
  Oracle continues to support the `DBMS_JOB` package. However, you must grant the `CREATE JOB` privilege to the database schemas that submit `DBMS_JOB` jobs.
- Blockchain Tables Reference
  You can independently verify the hash value and signature of a row by using its row content.

A Support for DBMS_JOB

Oracle continues to support the `DBMS_JOB` package. However, you must grant the `CREATE JOB` privilege to the database schemas that submit `DBMS_JOB` jobs.

Oracle Scheduler replaces the `DBMS_JOB` package. Although `DBMS_JOB` is still supported for backward compatibility, Oracle strongly recommends that you switch from `DBMS_JOB` to Oracle Scheduler.

In upgrades of Oracle Database 19c and later releases, if the upgrade can recreate existing `DBMS_JOB` jobs using `DBMS_SCHEDULER`, then for backward compatibility, after the upgrade, `DBMS_JOB` continues to act as a legacy interface to the `DBMS_SCHEDULER` job. If existing jobs cannot be recreated using `DBMS_SCHEDULER` because of issues with the metadata, then you receive a `JOB_TABLE_INTEGRITY` warning when you run upgrade prechecks. In that case, you have three options:

- Fix the metadata. After the upgrade continue to run after the upgrade using `DBMS_JOBS` as an interface, and run as `DBMS_SCHEDULER` jobs.
- Drop the jobs, if no longer required.
- Drop `DBMS_JOBS` jobs, and recreate the jobs manually using `DBMS_SCHEDULER`.

For existing jobs created with `DBMS_JOB` that are recreated during the upgrade, the legacy `DBMS_JOB` job is still present as an interface, but using it always creates a `DBMS_SCHEDULER` entry. Apart from the interface, the job is run as a `DBMS_SCHEDULER` job. If you subsequently disable the `DBMS_JOB` job created before the upgrade, then the `DBMS_SCHEDULER` job is also disabled. To avoid this behavior,drop the legacy job, and replace it with a `DBMS_SCHEDULER` job.

For all new jobs, use `DBMS_SCHEDULER`.

- Oracle Scheduler Replaces DBMS_JOB
  Starting with Oracle Database 11*g* Release 2 (11.2), Oracle Scheduler replaces `DBMS_JOB`. Oracle Scheduler is more powerful and flexible than `DBMS_JOB`, which is a package used to schedule jobs. Although `DBMS_JOB` is still supported for backward compatibility, Oracle strongly recommends that you switch from `DBMS_JOB` to Oracle Scheduler.
- Moving from DBMS_JOB to Oracle Scheduler
  This section illustrates some examples of how you can take jobs created with the `DBMS_JOB` package and rewrite them using Oracle Scheduler, which you configure and control with the `DBMS_SCHEDULER` package.

A.1 Oracle Scheduler Replaces DBMS_JOB

Starting with Oracle Database 11*g* Release 2 (11.2), Oracle Scheduler replaces `DBMS_JOB`. Oracle Scheduler is more powerful and flexible than `DBMS_JOB`, which is a package used to schedule jobs. Although `DBMS_JOB` is still supported for backward compatibility, Oracle strongly recommends that you switch from `DBMS_JOB` to Oracle Scheduler.

- Configuring DBMS_JOB
  The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of processes that can be created for the execution of jobs.

- Using Both DBMS_JOB and Oracle Scheduler
  `DBMS_JOB` and Oracle Scheduler (the Scheduler) use the same job coordinator to start job slaves.

A.1.1 Configuring DBMS_JOB

The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of processes that can be created for the execution of jobs.

Starting with Oracle Database Release 21c, the default value for `JOB_QUEUE_PROCESSES` across all containers is automatically derived from the number of sessions and CPUs configured in the system. The job coordinator process starts only as many job queue processes as are required, based on the number of jobs to run and available resources. You can set `JOB_QUEUE_PROCESSES` to a lower number to limit the number of job queue processes.

Setting `JOB_QUEUE_PROCESSES` to 0 disables `DBMS_JOB` jobs and `DBMS_SCHEDULER` jobs.

See Also:

Oracle Database Reference for more information about the `JOB_QUEUE_PROCESSES` initialization parameter

A.1.2 Using Both DBMS_JOB and Oracle Scheduler

`DBMS_JOB` and Oracle Scheduler (the Scheduler) use the same job coordinator to start job slaves.

You can use the `JOB_QUEUE_PROCESSES` initialization parameter to limit the number job slaves for both `DBMS_JOB` and the Scheduler.

If `JOB_QUEUE_PROCESSES` is 0, both `DBMS_JOB` and Oracle Scheduler jobs are disabled.

See Also:

- Scheduling Jobs with Oracle Scheduler
- "Setting Scheduler Preferences"
- *Oracle Database Reference* for more information about the `JOB_QUEUE_PROCESSES` initialization parameter

A.2 Moving from DBMS_JOB to Oracle Scheduler

This section illustrates some examples of how you can take jobs created with the `DBMS_JOB` package and rewrite them using Oracle Scheduler, which you configure and control with the `DBMS_SCHEDULER` package.

- Creating a Job
  An example illustrates creating a job using the DBMS_JOB package and the DBMS_SCHEDULER package.
- Altering a Job
  An example illustrates altering a job using the DBMS_JOB package and the DBMS_SCHEDULER package.
- Removing a Job from the Job Queue
  An example illustrates removing a job using the DBMS_JOB package and the DBMS_SCHEDULER package.

A.2.1 Creating a Job

An example illustrates creating a job using the DBMS_JOB package and the DBMS_SCHEDULER package.

The following example creates a job using DBMS_JOB:

```
VARIABLE jobno NUMBER;
BEGIN
 DBMS_JOB.SUBMIT(:jobno, 'INSERT INTO employees VALUES (7935, ''SALLY'',
   ''DOGAN'', ''sally.dogan@examplecorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
    NULL, NULL, NULL);', SYSDATE, 'SYSDATE+1');
 COMMIT;
END;
/
```

The following is an equivalent statement using DBMS_SCHEDULER:

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
   job_name          =>  'job1',
   job_type          =>  'PLSQL_BLOCK',
   job_action        =>  'INSERT INTO employees VALUES (7935, ''SALLY'',
     ''DOGAN'', ''sally.dogan@examplecorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
      NULL, NULL, NULL);',
   start_date        =>  SYSDATE,
   repeat_interval   =>  'FREQ = DAILY; INTERVAL = 1');
END;
/
```

A.2.2 Altering a Job

An example illustrates altering a job using the DBMS_JOB package and the DBMS_SCHEDULER package.

The following example alters a job using DBMS_JOB:

```
BEGIN
 DBMS_JOB.WHAT(31, 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
   ''tom.dogan@examplecorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
   NULL, NULL, NULL);');
 COMMIT;
END;
/
```

This changes the action for JOB1 to insert a different value.

The following is an equivalent statement using DBMS_SCHEDULER:

```
BEGIN
 DBMS_SCHEDULER.SET_ATTRIBUTE(
   name          => 'JOB1',
   attribute     => 'job_action',
   value         => 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
      ''tom.dogan@examplecorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
      NULL, NULL, NULL);');
END;
/
```

A.2.3 Removing a Job from the Job Queue

An example illustrates removing a job using the `DBMS_JOB` package and the `DBMS_SCHEDULER` package.

The following example removes a job using `DBMS_JOB`, where 14144 is the number of the job being run:

```
BEGIN
   DBMS_JOB.REMOVE(14144);
COMMIT;
END;
/
```

Using `DBMS_SCHEDULER`, you would issue the following statement instead:

```
BEGIN
   DBMS_SCHEDULER.DROP_JOB('myjob1');
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SCHEDULER` package
- Scheduling Jobs with Oracle Scheduler

B

Blockchain Tables Reference

You can independently verify the hash value and signature of a row by using its row content.

You can use the data format for the row content and column content to create procedures or functions that verify the hash value and user signature for a row.

- Blockchain Tables Column Content
  The column content of a row consists of the column metadata and the column data.
- Blockchain Tables Row Content
  A predefined format is used to compute the row content of rows in the blockchain table.
- Format of the Signed Digest in Blockchain Tables
  The signed digest consists of metadata and data about the last row in each chain of a blockchain table.

B.1 Blockchain Tables Column Content

The column content of a row consists of the column metadata and the column data.

The **data format** for column content is a platform-neutral sequence of bytes, in a predefined format, that is based on the column metadata and the column data. To understand how the data format for column content is computed, consider a row in the `bctab` table. The table contains two columns `bank` and `amount`. A row in this table contains the values 'Chase' and 1000 respectively.

Data Format for Column Data

Data format is computed for both user-defined and hidden columns. The column data is platform-neutral and can be obtained by using the SQL DUMP function.

The following example, assuming that the database character set is ANSI ASCII, gets the data format for column data as:

```
SELECT REGEXP_REPLACE(REGEXP_SUBSTR(DUMP(bank_name, 16), '[^ ]+',1, 3), ',',
'') "Data Value"
    FROM examples.bank_ledger WHERE bank_name = 'MyBank';

Data Value
------------------------------------------------------------------------------
--
4368617365
```

Data Format for Column Metadata

The column metadata is a 20-byte structure in the following format:

```
typedef struct col_meta_data
{
     ub2  version;                  /* VALUE IS ALWAYS 1/
     ub2  col_position;
     ub2  col_type;
     ub1  is_col_null;
     ub1  reserved1;               /*VALUE IS ALWAYS 0*/
     ub8  col_len;
     ub4  spare;
} col_meta_data;
```

where:

- `ub1` is an unsigned single byte value
- `ub2` is an unsigned short, 2 bytes value
- `ub4` is an unsigned long, 4 bytes value
- `ub8` is unsigned long long, 8 bytes value

`ub2`, `ub4`, and `ub8` use the little-endian format.

The attributes of the `col_meta_data` structure describe information about the column, such as its position in the table, data type, whether the column value is `NULL` or not, and the length of the column data (in bytes).

In the example, the column metadata for the value 'Chase' in ASCII and using the structure defined above is:

```
01 00
01 00
01 00
00
00
05 00 00 00 00 00 00 00
00 00 00 00
```

The first line, 01 00, is a 2-byte representation of the data format version, which is 1 in this release. The second line, 01 00, is a 2-byte representation of the column position of `BANK_NAME`, which is 1. The third line, 01 00, is a 2-byte representation of the internal code for the data type of the `BANK_NAME` column. The data type is `VARCHAR2`, which has an internal code of 1. The fourth line, 00, has a byte that specifies whether the column value is NULL (01) or not NULL (00). The column value is "Chase" and is hence not NULL. The fifth line, 00, is a reserved byte and must be zero in this release. The sixth line, 05 00 00 00 00 00 00 00, is an 8-byte length. The value "Chase" is 5 bytes in the database character set. The seventh line, 00 00 00 00, is four reserved bytes, each of which must be zero in this release.

The data format for column content for the column value 'Chase' is the concatenation of bytes from its column metadata and its column data. Therefore, the data format for the column content is the following value:

```
01 00
01 00
01 00
00
00
05 00 00 00 00 00 00 00
00 00 00 00
43 68 61 73 65
```

Example B-1   Creating a User-Defined Function to Construct a Column Metadata Value

The procedure `little_endian_ubx` is a useful utility procedure. It converts a number to a `ub2`, `ub4`, or a `ub8` value in little-endian format. The parameter `x` represents the data type of the column. Use the value 2 for `ub2` input, 4 for `ub4` input, and 8 for `ub8` input.

```
CREATE OR REPLACE FUNCTION little_endian_ubx(value IN NUMBER, x IN NUMBER)
RETURN RAW IS
  format VARCHAR2(16);
  string VARCHAR2(16);
  result RAW(8);
BEGIN
  format := RPAD('0', 2*x-1, '0') || 'X';        -- conversion format is all
zeroes and a final X
  string := SUBSTR(TO_CHAR(value, format), 2);  -- use SUBSTR to strip
leading space
  dbms_output.put_line('string is ' || string);
  result := utl_raw.reverse(HEXTORAW(string));
  dbms_output.put_line('result is ' || RAWTOHEX(result));
  RETURN result;
END little_endian_ubx;
```

B.2 Blockchain Tables Row Content

A predefined format is used to compute the row content of rows in the blockchain table.

The **row content** of a row is a contiguous sequence of bytes that is based on the row data and the hash value of the previous row in the chain. The ***data format*** for the row content defines the order and sequence of bytes.

The data format for row content is different when computing the hash value and when computing the row signature.

Data Format for Row Content When Computing Hash Value

The data format for row content of a hash value is computed based on the user columns, some hidden columns, and the hash value of the previous row in the chain.

To understand the data format for row content when computing a hash value, consider a blockchain table `bctab` that contains two columns `bank` and `amount`. The second row in this table contains the values 'Chase' and 1000 respectively. The data format for the row content, when computing the row hash value, is obtained by concatenating the byte values of the following, in the order listed:

- Data format for bank column, containing value 'Chase'
- Data format for amount column, containing the value 1000
- Data format for hidden column `ORABCTAB_INST_ID$`
- Data format for hidden column `ORABCTAB_CHAIN_ID$`
- Data format for hidden column `ORABCTAB_SEQ_NUM$`
- Data format for hidden column `ORABCTAB_CREATION_TIME$`
- Data format for hidden column `ORABCTAB_USER_NUMBER$`
- Data format for hash value of the first row in the table (assuming both rows are in the same chain)

The order of columns is dictated by the `INTERNAL_COLUMN_ID` column in the view `ALL_TAB_COLS`.

Data Format for Row Content When Computing Row Signature

The data format for row content of a row signature is computed based on the hash value of the row.

B.3 Format of the Signed Digest in Blockchain Tables

The signed digest consists of metadata and data about the last row in each chain of a blockchain table.

The data format for the signed digest contains a header and an array of row information.

The structure of the header is as follows:

```
{
    ub1    version;
                 /* 1 in 21.1 */
    ub1    reserved_1;
    ub1    reserved_2;
    ub1    reserved_3;
    ub4    reserved_4;
    ub8    total_length;
                 /* total length of signature content buffer, except version,
reserved_% and total_length fields */
    ub1    pdb_guid[16];
                 /* 16 bytes long PDB GUID */
    ub4    owner_schema_objn;
    ub4    blockchain_table_objn;
    ub4    signature_algorithm;
    ub4    number_of_rows;
 }
```

The structure of the row information is as follows:

```
{
    ub4   instance_id ;
    ub4   chain_id
    ub8   sequence_number;
    ub4   user_number;
    ub1   row_creation_time[16];
                /* UTC format that Oracle uses has 13 bytes; padded 3 bytes */
    ub4   crypto_hash_len;
    ub1  *crypto_hash;
                /* padded to 4 byte boundary */
    ub4   user_columns_count;
                /* always 0 in 21.1
                 * padded to 8 byte boundary */
    ub8   user_columns_data_len;
                /* always 0 in 21.1 */
}
```

where:

- `ub1` is an unsigned single byte value
- `ub4` is an unsigned long, 4 bytes value
- `ub8` is an unsigned long, 8 bytes value

`ub4` and `ub8` use the little-endian format.

Padding is done by appending binary zeros.

Index