Oracle® Database

Concepts

11*g* Release 1 (11.1)

B28318-01

July 2007

ORACLE®

Oracle Database Concepts, 11g Release 1 (11.1)

B28318-01

Primary Author: Richard Strohm

Contributing Authors: Lance Ashdown, Mark Bauer, Michele Cyran, Steve Fogel, Janis Greenberg, Sumit Jeloka, Paul Lane, Diana Lorentz, Jack Melnick, Sheila Moore, Antonio Romero, Viv Schupmann, Cathy Shea, Douglas Williams

Contributors: Omar Alonso, Penny Avril, Hermann Baer, Sandeepan Banerjee, Bill Bridge, Sandra Cheevers, Carol Colrain, Vira Goorah, Mike Hartstein, John Haydu, Wei Hu, Ramkumar Krishnan, Vasudha Krishnaswamy, Bill Lee, Bryn Llewellyn, Rich Long, Paul Manning, Mughees Minhas, Valarie Moore, Gopal Mulagund, Muthu Olagappan, Jennifer Polk, Kathy Rich, John Russell, Bob Thome, Randy Urbano, Mark Van de Wiel, Michael Verheij, Ron Weiss, Steve Wertheimer

Contents

Part I What Is Oracle?

1 Introduction to Oracle Database

Part II   Oracle Database Architecture

2   Data Blocks, Extents, and Segments

3   Tablespaces, Datafiles, and Control Files

4   Transaction Management

5   Schema Objects

6   Schema Object Dependencies

7   The Data Dictionary

10   Application Architecture

11   Oracle Database Utilities

12 Database and Instance Startup and Shutdown

Part III   Oracle Database Features

13 Data Concurrency and Consistency

14   Manageability

15 Backup and Recovery

16 Business Intelligence

17 High Availability

18 Very Large Databases (VLDB)

19 Content Management

20 Database Security

21   Data Integrity

22   Triggers

23   Information Integration

Part IV Oracle Database Application Development

24 SQL

25 Supported Application Development Languages

26   Oracle Data Types

Glossary

Index

Preface

This manual describes all features of the Oracle database server, an object-relational database management system. It describes how the Oracle database server functions, and it lays a conceptual foundation for much of the practical information contained in other manuals. Information in this manual applies to the Oracle database server running on all operating systems.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documentation
- Conventions

Audience

Oracle Database Concepts is intended for database administrators, system administrators, and database application developers.

To use this document, you need to know the following:

- Relational database concepts in general
- Concepts and terminology in **Chapter 1, "Introduction to Oracle Database"**
- The operating system environment under which you are running Oracle

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Database Upgrade Guide* for information about upgrading a previous release of Oracle

- *Oracle Database Administrator's Guide* for information about how to administer the Oracle database server

- *Oracle Database Advanced Application Developer's Guide* for information about developing Oracle database applications

- *Oracle Database Performance Tuning Guide* for information about optimizing performance of an Oracle database

- *Oracle Database Data Warehousing Guide* for information about data warehousing and business intelligence

- *Oracle Database Utilities* for information about the utilities mentioned in this document

Many books in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

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

What Is Oracle?

Part I provides an overview of Oracle Database concepts and terminology. It contains the following chapter:

- Chapter 1, "Introduction to Oracle Database"

1

Introduction to Oracle Database

This chapter provides an overview of the Oracle database server. The topics include:

- Oracle Database Architecture
- Oracle Database Features
- Oracle Database Application Development

Oracle Database Architecture

A database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to information management. In general, a **server** reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost-effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need to provide extra hardware to support peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

The database has **physical structures** and **logical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures.

The section contains the following topics:

- Overview of Grid Architecture
- Overview of Application Architecture
- Overview of Physical Database Structures
- Overview of Logical Database Structures
- Overview of Schemas and Common Schema Objects
- Overview of the Oracle Database Data Dictionary
- Overview of the Oracle Database Instance
- Overview of Accessing the Database
- Overview of Oracle Database Utilities

Overview of Grid Architecture

Grid computing is an information technology (IT) architecture that produces more resilient and lower cost enterprise information systems. With grid computing, groups of independent, modular hardware and software components can be connected and rejoined on demand to meet the changing needs of businesses.

The grid style of computing solves some common problems with enterprise IT:

- Application silos that lead to underutilized, dedicated hardware resources

- Monolithic, unwieldy systems that are expensive to maintain and difficult to change

- Fragmented and disintegrated information that cannot be fully exploited by the enterprise as a whole.

Compared with other models of computing, IT systems designed and implemented in the grid style deliver higher quality of service, lower cost, and greater flexibility. Higher quality of service is achieved because there is no single point of failure, there is a robust security infrastructure, and management is centralized and policy-driven. Lower costs derive from increasing the utilization of resources and dramatically reducing management and maintenance costs. Rather than dedicating a stack of software and hardware to a specific task, all resources are pooled and allocated on demand, thus eliminating underutilized capacity and redundant capabilities. Greater flexibility is achieved because grid computing also enables the use of smaller individual hardware components, thus reducing the cost of each individual component and enabling the enterprise to devote resources in accordance with changing needs.

Overview of Application Architecture

The two most common database architectures are client/server and multitier. As internet computing becomes more prevalent in computing environments, many database management systems are moving to a multitier environment.

This section includes the following topics:

- Client/Server Architecture

- Multitier Architecture: Application Servers

- Multitier Architecture: Service-Oriented Architecture

Client/Server Architecture

An Oracle database system can easily take advantage of distributed processing by using its **client/server architecture**. In this architecture, the database system has two parts: a front-end or a **client**, and a back-end or a **server**.

The Client The client is a database application that initiates a request for an operation to be performed on the database server. It requests, processes, and presents data managed by the server. The client workstation can be optimized for its job. For example, the client might not need large disk capacity, or it might benefit from graphic capabilities. Often, the client runs on a different computer than the database server. Many clients can simultaneously run against one server.

The Server The server runs Oracle Database software and handles the functions required for concurrent, shared data access. The server receives and processes requests that originate from client applications. The computer that manages the server can be

optimized for its duties. For example, the server computer can have large disk capacity and fast processors.

Multitier Architecture: Application Servers

A traditional **multitier architecture** has the following components:

- A client or initiator process that starts an operation

- One or more application servers that perform parts of the operation. An **application server** contains a large part of the application logic, provides access to the data for the client, and performs some query processing, thus removing some of the load from the database server. The application server can serve as an interface between clients and multiple database servers and can provide an additional level of security.

- An end server or database server that stores most of the data used in the operation

This architecture enables use of an application server to do the following:

- Validate the credentials of a client, such as a Web browser

- Connect to an Oracle Database server

- Perform the requested operation on behalf of the client

If proxy authentication is being used, then the identity of the client is maintained throughout all tiers of the connection.

Multitier Architecture: Service-Oriented Architecture

Service-oriented architecture (SOA) is a multitier architecture in which application functionality is encapsulated in *services*. SOA services are usually implemented as Web services. Web services can be accessed with the HTTP protocol and are based on a set of XML-based open standards, such as WSDL and SOAP.

Beginning with Oracle Database 11*g*, Oracle Database can act as a Web service provider in a traditional multitier or SOA environment.

> **See Also:**
>
> - "Oracle Database as a Web Service Provider" on page 10-5 for more information about Oracle Database as a Web service provider
>
> - *Oracle XML DB Developer's Guide* for more information about using Web services with the database

Overview of Physical Database Structures

The following sections explain the physical database structures of an Oracle database, including datafiles, control files, redo log filesarchive log files, parameter files, alert and trace log files, and backup files.

This section includes the following topics:

- Datafiles

- Control Files

- Online Redo Log Files

- Archived Redo Log Files

- [Parameter Files](#)
- [Alert and Trace Log Files](#)
- [Backup Files](#)

Datafiles

Every Oracle database has one or more physical **datafiles**, which contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the datafiles allocated for a database.

Datafiles have the characteristics:

- A datafile can be associated with only one database.
- Datafiles can be defined to extend automatically when they are full.
- One or more datafiles form a logical unit of database storage called a tablespace.

Data in a datafile is read, as needed, during normal database operation and stored in the memory cache of Oracle Database. For example, if a user wants to access some data in a table of a database, and if the requested information is not already in the memory cache for the database, then it is read from the appropriate datafiles and stored in memory.

Modified or new data is not necessarily written to a datafile immediately. To reduce the amount of disk access and to increase performance, data is pooled in memory and written to the appropriate datafiles all at once, as determined by the background process **database writer process (DBWn)**.

Datafiles that are stored in temporary tablespaces are called **tempfile**s. Tempfiles are subject to some restrictions, as described in "Temporary Datafiles" on page 3-16.

> **See Also:** "Overview of the Oracle Database Instance" on page 1-9 for more information about the Oracle Database memory and process structures

Control Files

Every Oracle database has a **control file**. A control file contains entries that specify the physical structure of the database, including the following information:

- Database name
- Names and locations of datafiles and redo log files
- Timestamp of database creation

Oracle Database can **multiplex** the control file, that is, simultaneously maintain a number of identical control file copies, to protect against a failure involving the control file.

Every time an **instance** of an Oracle database is started, its control file identifies the datafiles, tempfiles, and redo log files that must be opened for database operation to proceed. If the physical makeup of the database is altered (for example, if a new datafile or redo log file is created), then the control file is automatically modified by Oracle Database to reflect the change. A control file is also used in database recovery.

> **See Also:** Chapter 3, "Tablespaces, Datafiles, and Control Files"

Online Redo Log Files

Every Oracle Database has a set of two or more online **redo log files**. These online redo log files, together with archived copies of redo log files, are collectively known as the redo log for the database. A **redo log** is made up of redo entries (also called **redo records**), which record all changes made to data. If a failure prevents modified data from being permanently written to the datafiles, then the changes can be obtained from the redo log, so work is never lost.

To protect against a failure involving the redo log itself, Oracle Database lets you create a **multiplexed redo log** so that two or more copies of the redo log can be maintained on different disks.

> **See Also:** "Overview of Database Backup and Recovery Features" on page 1-20

Archived Redo Log Files

When online redo log files are written to disk, they become archived redo log files. Oracle recommends that you enable automatic archiving of the redo log. Oracle Database automatically archives redo log files when the database is in `ARCHIVELOG` mode.

Parameter Files

Parameter files contain a list of configuration parameters for that instance and database. Both parameter files (pfiles) and server parameter files (spfiles) let you store and manage your initialization parameters persistently in a server-side disk file. A server parameter file has these additional advantages:

- The file is concurrently updated when some parameter values are changed in the active instance.

- The file is centrally located for access by all instance in a Real Application Services database.

Oracle recommends that you create a server parameter file as a dynamic means of maintaining initialization parameters.

> **See Also:**
>
> - "Initialization Parameter Files and Server Parameter Files" on page 12-3
>
> - *Oracle Database Administrator's Guide* for information about creating and changing parameter files

Alert and Trace Log Files

Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, while other information is for Oracle Support Services. Trace file information is also used to tune applications and instances. The alert file, or alert log, is a special trace file. The alert log of a database is a chronological log of messages and errors.

The following features provide automation and assistance in the collection and interpretation of trace and alert file information:

- The **Automatic Diagnostic Repository** (ADR) is a system-managed repository for storing and organizing trace files and other error diagnostic data. ADR provides a

comprehensive view of all the critical errors encountered by the database and maintains all relevant data needed for problem diagnosis and eventual resolution. When the same type of incident occurs too frequently, ADR performs flood control to avoid excessive dumping of diagnostic information.

- The Incident Packaging Service (IPS) extracts diagnostic and test case data associated with critical errors from the ADR and packages the data for transport to Oracle.

> **See Also:** *Oracle Database Administrator's Guide* for more information

Backup Files

To restore a file is to replace it with a backup file. Typically, you restore a file when a media failure or user error has damaged or deleted the original file.

User-managed backup and recovery requires you to actually restore backup files before you can perform a trial recovery of the backups.

Server-managed backup and recovery manages the backup process, such as scheduling of backups, as well as the recovery process, such as applying the correct backup file when recovery is needed.

> **See Also:**
>
> - Chapter 15, "Backup and Recovery"
> - *Oracle Database Backup and Recovery User's Guide*

Overview of Logical Database Structures

This section discusses logical storage structures: data blocks, extents, segments, and tablespaces. These logical storage structures enable Oracle Database to have fine-grained control of disk space use.

This section includes the following topics:

- Oracle Database Data Blocks
- Extents
- Segments
- Tablespaces

Oracle Database Data Blocks

At the finest level of granularity, Oracle Database data is stored in **data blocks**. One data block corresponds to a specific number of bytes of physical database space on disk. The standard block size is specified by the DB_BLOCK_SIZE initialization parameter. In addition, you can specify up to four other block sizes. A database uses and allocates free database space in Oracle Database data blocks.

Extents

The next level of logical database space is an **extent**. An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

Segments

Above extents, the level of logical database storage is a **segment**. A segment is a set of extents allocated for a table, index, rollback segment, or for temporary use by a session, transaction, or SQL parser. In relation to physical database structures, all extents belonging to a segment exist in the same tablespace, but they may be in different data files.

When the extents of a segment are full, Oracle Database dynamically allocates another extent for that segment. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

Tablespaces

A database is divided into logical storage units called **tablespaces**, which group related data blocks, extents, and segments. For example, tablespaces commonly group together all application objects to simplify some administrative operations.

Each database is logically divided into two or more tablespaces. One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. The combined size of the datafiles in a tablespace is the total storage capacity of the tablespace.

Every Oracle database contains a SYSTEM tablespace and a SYSAUX tablespace. Oracle Database creates them automatically when the database is created. The system default is to create a **smallfile tablespace**, which is the traditional type of Oracle tablespace. The SYSTEM and SYSAUX tablespaces are created as smallfile tablespaces.

Oracle Database also lets you create **bigfile tablespaces**, which are made up of single large file rather than numerous smaller ones. Bigfile tablespaces let Oracle Database utilize the ability of 64-bit systems to create and manage ultralarge files. As a result, Oracle Database can scale up to 8 exabytes in size. With Oracle-Managed Files, bigfile tablespaces make datafiles completely transparent for users. In other words, you can perform operations on tablespaces, rather than the underlying datafiles.

> **See Also:**
>
> - Chapter 2, "Data Blocks, Extents, and Segments"
> - Chapter 3, "Tablespaces, Datafiles, and Control Files"
> - "Introduction to Undo Segments and Automatic Undo Management" on page 2-16
> - "Read Consistency" on page 1-15
> - "Overview of Database Backup and Recovery Features" on page 1-20

Online and Offline Tablespaces A tablespace can be **online** or **offline**. A tablespace is generally online, so that users can access the information in the tablespace. However, to simplify administration, sometimes a tablespace is taken offline to make a portion of the database unavailable while allowing normal access to the remainder of the database.

Read-only Tablespaces A tablespace can be **read only**, which means that data in the tablespace cannot be modified. The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Oracle Database never updates the files of a read-only tablespace, and therefore the files can reside on read-only media such as CD-ROMs or WORM drives.

Overview of Schemas and Common Schema Objects

A **schema** is a collection of database objects. A schema is owned by a database user and has the same name as that user. **Schema objects** are the logical structures that directly refer to the database's data. There is no relationship between a tablespace and a schema. Objects in the same schema can be in different tablespaces, and a tablespace can hold objects from different schemas. Schema objects include structures such as **table**s, **view**s, and **index**es. Some of the most common schema objects are defined in the sections that follow.

This section includes the following topics:

- Tables
- Indexes
- Views
- Clusters
- Synonyms

Tables

Tables are the basic unit of data storage in an Oracle database. Database tables hold all user-accessible data. Each table has **column**s and **row**s. A table that has employee information, for example, can have a column called `employee_number`, and each row in that column is an employee number.

Indexes

Indexes are optional structures associated with tables. You can create indexes to increase the performance of data retrieval. Just as the index in this manual helps you quickly locate specific information, an Oracle database index provides an access path to table data.

When processing a request, Oracle Database can use some or all of the available indexes to locate the requested rows efficiently. Indexes are useful when applications frequently query a table for a range of rows (for example, all employees with a salary greater than 1000) or a specific row (for example, the employee with the highest salary).

You create an index on one or more columns of a table. Thereafter, Oracle Database automatically uses and maintains the index. Changes to table data (such as adding new rows, updating rows, or deleting rows) are automatically incorporated into all relevant indexes.

Views

Views are customized presentations of data in one or more tables or other views. A view can also be considered a stored query. Views do not contain actual data. Rather, they derive their data from the tables on which they are based, referred to as the **base tables** of the views.

You can query, update, insert into, and delete views as you can with tables, with some restrictions. If the view is updatable, then all operations performed on the view actually affect the base tables of the view.

Views can provide table security by restricting access to a predetermined set of rows and columns of a table. They also hide data complexity and store complex queries.

Clusters

Clusters are groups of one or more tables physically stored together because they share common columns and are often used together. Because related rows are physically stored together, disk access time improves.

Like indexes, clusters do not affect application design. Whether a table is part of a cluster is transparent to users and to applications. SQL statements access data stored in a clustered table in the same way that they access data stored in a nonclustered table.

Synonyms

A **synonym** is an alias for any table, view, materialized view, sequence, operator, procedure, function, package, Java class schema object, user-defined object type, or another synonym. A synonym is simply an alias, so it requires no storage other than its definition in the data dictionary.

> **See Also:** Chapter 5, "Schema Objects" for more information on these and other schema objects

Overview of the Oracle Database Data Dictionary

Each Oracle database has a **data dictionary**, which is a set of tables and views that serve as a reference about the database. For example, a data dictionary stores information about both the logical and physical structure of the database. A data dictionary also stores the valid users of an Oracle database, information about integrity constraints defined for tables in the database, and the amount of space allocated for a schema object and how much of that space is in use, among much other information.

A data dictionary is created when a database is created. To accurately reflect the status of the database at all times, the data dictionary is automatically updated by Oracle Database in response to specific actions, such as when the structure of the database is altered. Database users cannot modify the data dictionary. Various database processes rely on the data dictionary to record, verify, and conduct ongoing work. For example, during database operation, Oracle Database reads the data dictionary to verify that schema objects exist and that users have proper access to them.

> **See Also:** Chapter 7, "The Data Dictionary" for more information

Overview of the Oracle Database Instance

An Oracle Database server consists of an Oracle Database and one or more Oracle Database instances. Every time a database is started, a shared memory area called the system global area (SGA) is allocated and Oracle Database background processes are started. The combination of the background processes and the SGA is called an Oracle Database **instance**.

Oracle Real Application Clusters   Some hardware architectures (for example, shared disk systems) enable multiple computers to share access to data, software, or peripheral devices. Oracle Real Application Clusters (Oracle RAC) comprises two or more Oracle Database instances running on multiple clustered computers that communicate with each other by means of an interconnect. Oracle RAC uses Oracle Clusterware to access a shared database that resides on shared disks. Oracle RAC combines the processing power of these multiple interconnected computers to provide system redundancy, near linear scalability, and high availability. Oracle RAC also offers significant advantages for both OLTP and data warehouse systems, and all systems and applications can efficiently exploit clustered environments.

You can scale applications in Oracle RAC environments to meet increasing data processing demands without changing the application code. When you add resources such as nodes or storage, Oracle RAC extends the processing powers of these resources beyond the limits of the individual components.

> **See Also:** *Oracle Real Application Clusters Administration and Deployment Guide*

When users connect to an Oracle Database server, they are connected to an Oracle Database instance. The database instance services those users by allocating other memory areas in addition to the SGA, and starting other processes in addition to the Oracle Database background processes. The following sections describe the various Oracle Database memory areas and processes:

- Oracle Database Background Processes
- Instance Memory Structures

Oracle Database Background Processes

An Oracle database uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of the computers that constitute the database system. A **process** is a mechanism in an operating system that can run a series of steps. Some operating systems use the terms *job* or *task*. Oracle Database server uses three types of processes: Oracle processes—which include server processes and background processes—and user processes. On almost all systems, the Oracle processes and the user processes are on separate computers.

- Oracle Database creates a set of **background processes** for each instance. The background processes consolidate functions that would otherwise be handled by multiple Oracle Database programs running for each user process. They asynchronously perform I/O and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.

  > **See Also:** "Oracle Database Background Processes" on page 9-4 for more information on some of the most common background processes

- **User processes**—sometimes called client processes—are created and maintained to run the software code of an application program (such as an OCI or OCCI program) or an Oracle tool (such as Oracle Enterprise Manager). Most environments have separate machines (laptops, desktops, and so forth) for the client processes. User processes also manage communication with the server process through the program interface, which is described in a later section.

- Oracle Database creates **server process**es to handle requests from connected user processes. A server process communicates with the user process and interacts with Oracle Database to carry out requests from the associated user process. For example, if a user queries some data not already in the **database buffer**s of the SGA, then the associated server process reads the proper **data block**s from the datafiles into the SGA.

  Oracle Database can be configured to vary the number of user processes for each server process. In a **dedicated server configuration**, a server process handles requests for a single user process. A **shared server configuration** lets many user processes share a small number of server processes, minimizing the number of server processes and maximizing the use of available system resources.

  > **See Also:** Chapter 9, "Process Architecture"

Instance Memory Structures

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data shared among users, and private data areas for each connected user. Two basic memory structures are associated with an Oracle Database:

- The System Global Area (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.

- The Program Global Areas (PGA) are memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the process. Each server process and background process has its own PGA.

> **See Also:** Chapter 8, "Memory Architecture" for more information

Overview of Accessing the Database

This section describes Oracle Net Services, as well as how to start up the database, in the following sections:

- Network Connections
- Starting Up the Database
- How Oracle Database Works

Network Connections

Oracle Net Services is the interface between Oracle Database and the network communication protocols that facilitate distributed processing and distributed databases. Communication protocols define the way that data is transmitted and received on a network. Oracle Net Services supports communications on all major network protocols, including TCP/IP, HTTP, FTP, and WebDAV.

Using Oracle Net Services, application developers do not need to be concerned with supporting network communications in a database application. If a new protocol is used, then the database administrator makes some minor changes, and the application requires no modifications and continues to function.

Oracle Net, a component of Oracle Net Services, establishes and maintains a network session from a client application to an Oracle Database server. Once a network session is established, Oracle Net acts as the data courier for both the client application and the database server, exchanging messages between them. Oracle Net can perform these jobs because it is located on each computer in the network.

> **See Also:**
>
> - *Oracle Database Net Services Administrator's Guide* for more information about network connections
>
> - *Oracle XML DB Developer's Guide* for information about using WebDAV with the database

Starting Up the Database

The three steps to starting an Oracle database and making it available for system-wide use are:

1. Start an instance.

2. Mount the database.

3. Open the database.

A database administrator can perform these steps using Oracle Enterprise Manager, the SQL*Plus STARTUP statement, the srvctl command-line tool, or the Express Edition START command. When Oracle Database starts an instance, it reads the server parameter file (spfile) or initialization parameter file (pfile) to determine the values of initialization parameters. Then, it allocates an SGA and creates background processes.

> **See Also:** Chapter 12, "Database and Instance Startup and Shutdown"

How Oracle Database Works

The following example describes Oracle Database operations at the most basic level. This illustrates an Oracle Database configuration where the user and associated server process are on separate computers, connected through a network.

1. An **instance** has started on the computer running Oracle Database, often called the **host** or **database server**.

2. A computer running an application (a **local computer** or **client workstation**) runs an application in a **user process**. The client application attempts to establish a **connection** to the server using the proper Oracle Net Services driver.

3. The server is running the proper Oracle Net Services driver. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.

4. The user runs a SQL statement and commits the transaction. For example, the user changes a name in a row of a table.

5. The server process receives the statement and checks the **shared pool** (an SGA component) for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, then the server process checks the user's access privileges to the requested data, and the existing shared SQL area is used to process the statement. If not, then a new shared SQL area is allocated for the statement, so it can be parsed and processed.

6. The server process retrieves any necessary data values, either from the actual datafile (table) or those stored in the SGA.

7. The server process modifies data in the system global area. The database writer process (DBWn) writes modified blocks permanently to disk when doing so is efficient. Because the transaction is committed, the log writer process (LGWR) immediately records the transaction in the redo log file.

8. If the transaction is successful, then the server process sends a message across the network to the application. If it is not successful, then an error message is transmitted.

9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

> **See Also:** Chapter 9, "Process Architecture" for more information background processes

Overview of Oracle Database Utilities

Oracle Database provides several utilities for data transfer, data maintenance, and database administration They are described briefly in Chapter 11, "Oracle Database Utilities" and more fully in *Oracle Database Utilities*.

Oracle Database Features

This section contains the following topics:

- Overview of Oracle Real Application Testing
- Overview of Concurrency Features
- Overview of Manageability Features
- Overview of Diagnosability Features
- Overview of Database Backup and Recovery Features
- Overview of High Availability Features
- Overview of Business Intelligence Features
- Overview of Content Management Features
- Overview of Security Features
- Overview of Data Integrity and Triggers
- Overview of Information Integration Features

Overview of Oracle Real Application Testing

System changes, such as hardware and software upgrades and patch application, are essential for businesses for compliance and security purposes or to maintain their competitive edge. Oracle Real Application Testing helps you fully assess the effect of system changes on real-world applications in test environments before deploying them in production. Oracle Real Application Testing consists of two features:

- Database Replay
- SQL Performance Analyzer

Database Replay

Database Replay enables realistic testing of system changes by essentially recreating the production workload environment on a test system. It does this by capturing a workload on the production system and then replaying it on a test system with the exact timing, concurrency, and transaction characteristics of the original workload. This makes possible complete assessment of the impact of the change including undesired results, new contention points, and performance regressions. Extensive analysis and reporting is provided to help identify any potential problems, such as new errors encountered and performance divergences.

With Database Replay, businesses can rapidly test changes and adopt new technologies with a high degree of confidence in the overall success of the effort and at significantly lower risk.

Database Replay can be used to assess the impact of the following types of system changes:

- Database upgrades, patches, parameter, and schema changes
- Configuration changes, such as conversion from a single instance to Oracle Real Application Clusters and Automatic Storage Management
- Storage, network, and interconnect changes
- Operating system patches, upgrades, and parameter changes and hardware migrations

SQL Performance Analyzer

Changes that affect SQL execution plans can severely impact system performance and availability. As a result, DBAs spend considerable time in identifying and fixing SQL statements that have regressed due to a change.

SQL Performance Analyzer automates the process of assessing the overall effect of a change on the full SQL workload by identifying performance divergence for each statement. A report that shows the net impact on the workload performance due to the change is provided. For regressed SQL statements, appropriate execution plan details, along with recommendations to tune them, is also provided. As a result, DBAs can remedy any negative outcome before their end users are affected and can confirm, with significant time and cost savings, that the system change to the production environment will, in fact, result in net improvement.

You can use the SQL Performance Analyzer to analyze the SQL performance impact of any type of system change. Examples of common system changes include:

- Database upgrades
- Configuration changes to the operating system, hardware, or database
- Database initialization parameter changes
- Schema changes, such as adding new indexes or materialized views
- Gathering optimizer statistics
- SQL tuning actions, such as creating SQL profiles

> **See Also:** *Oracle Database Performance Tuning Guide* to learn how to use the SQL Performance Analyzer

Overview of Concurrency Features

All information management systems have these important requirements:

- Data **concurrency** of a multiuser system must be maximized.
- Data must be read and modified in a consistent fashion. The data a user is viewing or changing must not changed (by other users) until the first user is finished with the data.
- High performance is required for maximum productivity from the many users of the database system.

Oracle Database contains several software mechanisms that satisfy these requirements. This contains the following sections:

- Concurrency

- Read Consistency

- Caching Mechanisms

- Locking Mechanisms

Concurrency

A primary feature of a multiuser database management system is **concurrency**, which is the simultaneous access of the same data by many users. Without adequate concurrency controls, data could be updated or changed improperly, compromising data integrity.

One way to manage data concurrency is to make each user wait for a turn. The goal of a database management system is to reduce that wait so it is either nonexistent or not noticeable to users. Data manipulation language operations (inserts, updates, and deletes) should proceed with as little interference as possible, and destructive interactions between concurrent transactions must be prevented. A destructive interaction is one that incorrectly updates data or incorrectly alters underlying data structures. Neither performance nor data integrity can be sacrificed.

Oracle Database resolves these issues by using various types of locks and a multiversion consistency model. These features are based on the concept of a **transaction**.

The transaction is key to the Oracle Database strategy for providing read consistency. This unit of committed (or uncommitted) SQL statements:

- Dictates the start point for read-consistent views generated on behalf of readers

- Controls when modified data can be seen by other transactions of the database for reading or updating

It is the application designer's responsibility to ensure that transactions fully exploit these concurrency and consistency features.

> **See Also:** Chapter 4, "Transaction Management"

Read Consistency

Read consistency, as provided by Oracle Database, achieves the following goals:

- Guarantees that the set of data seen by a statement is consistent with respect to a single point in time and does not change during statement execution (statement-level read consistency)

- Ensures that readers of database data do not wait for writers or other readers of the same data

- Ensures that writers of database data do not wait for readers of the same data

- Ensures that writers only wait for other writers if they attempt to update identical rows in concurrent transactions

In the Oracle Database implementation of read consistency, it is as if each user operates a private copy of the database. This is sometimes called a multiversion consistency model.

> **See Also:** Chapter 13, "Data Concurrency and Consistency"

Read Consistency, Undo Records, and Transactions To manage the multiversion consistency model, Oracle Database uses current information in the System Global Area and information in the undo records to construct a **read-consistent view** of a

table's data for a query. When an update occurs, the original data values are recorded in the database undo records. As long as this update remains part of an uncommitted transaction, any user that later queries the modified data views the original data values. Only when a transaction is committed are the changes of the transaction made permanent. Queries that are initiated *after* the transaction is committed see the changes made by the committed transaction.

Read-Only Transactions By default, Oracle Database guarantees statement-level read consistency. The set of data returned by a single query is consistent with respect to a single point in time. However, in some situations, you might also require transaction-level read consistency. This is the ability to run multiple queries within a single transaction, all of which are read-consistent with respect to the same point in time, so that queries in this transaction do not see the effects of intervening committed transactions. If you want to run a number of queries against multiple tables and if you are not doing any updating, you can initiate the transaction with commands that define it as a **read-only transaction**.

> **See Also:** *Oracle Database Concepts* for more information on transaction-level read consistency

Caching Mechanisms

Oracle Database optimizes database performance by caching in memory user data, log data, dictionary data, and other types of data.

Oracle Database also caches query results, so that if a query is repeated, the database can return results from the cache instead of reprocessing the query and reading data from storage. The cached results are stored in a dedicated portion of the shared pool. Query retrieval from the query result cache is faster than rerunning the query. The query result cache enables explicit caching of results in database memory. Frequently executed queries especially see performance improvements when using the query result cache.

Locking Mechanisms

Oracle Database also uses **locks** to control concurrent access to data. When updating information, the data server holds that information with a lock until the update is submitted or committed. Until that happens, no one else can make changes to the locked information. This ensures the data integrity of the system.

Oracle Database provides unique nonescalating row-level locking. Unlike other data servers that escalate locks to cover entire groups of rows or even the entire table, Oracle Database always locks only the row of information being updated. Because the database includes the locking information with the actual rows themselves, it can lock an unlimited number of rows so users can work concurrently without unnecessary delays.

Automatic Locking Oracle Database locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested.

The Oracle Database lock manager maintains several different types of row locks, depending on what type of operation established the lock. The two general types of locks are **exclusive locks** and **share locks**. Only one exclusive lock can be placed on a resource (such as a row or a table); however, many share locks can be placed on a single resource. Both exclusive and share locks always permit queries on the locked resource but prohibit other activity on the resource (such as updates and deletes).

Manual Locking Under some circumstances, you might want to override default locking. With Oracle Database, you can manually override automatic locking features at both the row level (by first querying for the rows that will be updated in a subsequent statement) and the table level.

Overview of Manageability Features

People who administer the operation of an Oracle database system, known as database administrators (DBAs), are responsible for creating Oracle databases, ensuring their smooth operation, and monitoring their use. In addition to the many alerts and advisors Oracle provides, Oracle Database also offers features described in the following sections:

- Self-Managing Database
- Automatic Maintenance Tasks
- Oracle Enterprise Manager
- SQL Developer and SQL*Plus
- Automatic Memory Management
- Automatic Storage Management
- Automatic Database Diagnostic Monitor
- SQL Tuning Advisor
- SQL Access Advisor
- Streams Tuning Advisor
- The Scheduler
- Database Resource Manager

Self-Managing Database

Oracle Database provides a high degree of self-management by automating routine DBA tasks and reducing complexity of space, memory, and resource administration. Oracle Database self-managing features include the following: automatic undo management, automatic server memory management, Oracle-managed files, free space management, and Recovery Manager (RMAN).

Automatic Maintenance Tasks

Oracle Database automatically schedules periodic maintenance tasks such as statistics collection and space recovery. These tasks run in a set of Oracle Scheduler windows known as *maintenance windows*. You can control the start time and duration of these maintenance windows, and limit the amount of CPU and I/O resources that they consume.

Oracle Enterprise Manager

Oracle Enterprise Manager is a system management tool that provides central management of your database environment. Combining a graphical console, Oracle Management Servers, Oracle Intelligent Agents, common services, and administrative tools, Oracle Enterprise Manager provides a comprehensive systems management platform for managing Oracle products.

From the client interface, the Oracle Enterprise Manager Console, you can perform the following tasks:

- Administer the entire Oracle environment, including databases, Oracle Application Server servers, applications, and services

- Diagnose, modify, and tune multiple databases

- Schedule tasks on multiple systems at varying time intervals

- Monitor database conditions throughout the network

- Administer multiple network nodes and services from many locations

- Share tasks with other administrators

- Group related targets together to facilitate administration tasks

- Launch integrated Oracle and third-party tools

- Customize the display of an Oracle Enterprise Manager administrator

SQL Developer and SQL*Plus

Oracle SQL Developer is a graphical development tool that provides a convenient way to perform these tasks:

- Browse, create, edit, and delete (drop) database objects

- Edit and debug PL/SQL code

- Run SQL statements and scripts

- Manipulate and export data

- Create and view reports

With SQL Developer, you can connect to any target Oracle database schema using standard Oracle database authentication. Once connected, you can perform operations on objects in the database. You can also connect to schemas for selected third-party (non-Oracle) databases, such as MySQL, Microsoft SQL Server, and Microsoft Access, view metadata and data in these databases, and migrate these databases to Oracle.

SQL*Plus is a basic command-line tool for entering and running ad hoc database statements. It lets you run SQL statements and PL/SQL blocks, and perform many additional tasks as well.

> **See Also:** *Oracle Database SQL Developer User's Guide* and *SQL*Plus User's Guide and Reference* for more information on these tools

Automatic Memory Management

Beginning with Oracle Database 11*g*, Release 1, Oracle Database can manage the System Global Area (SGA) memory and instance Program Global Area (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. In this memory management mode, the database also dynamically tunes the sizes of the individual SGA components and the sizes of the individual PGAs.

> **See Also:** *Oracle Database 2 Day DBA* for more information

Automatic Storage Management

Automatic Storage Management automates and simplifies the management of all types of database files. Database files are automatically distributed across all available

disks, and database storage is rebalanced automatically whenever the storage configuration changes. Automatic Storage Management also provides redundancy through the mirroring of database files.

Oracle Database has built-in support for the network file system (NFS) and does not depend on OS support for NFS. This improves manageability and diagnosability of network attached storage accessed with NFS.

> **See Also:** *Oracle Database Storage Administrator's Guide* for more information about NFS

Automatic Database Diagnostic Monitor

The Automatic Database Diagnostic Monitor (ADDM) lets you conduct performance analyzes over any time period defined by a pair of Automatic Workload Repository (AWR) snapshots taken on a particular instance. Analysis is performed top down, first identifying symptoms and then refining them to reach the root causes of performance problems. ADDM also documents non-problem areas of the system. For example, wait event classes that are not significantly affecting the performance of the system are identified and removed from the tuning consideration at an early stage, saving time and effort that would be spent on items with little or no impact on overall system performance.

In addition to problem diagnostics, ADDM recommends possible solutions. When appropriate, ADDM recommends multiple solutions for the DBA to choose from. ADDM considers a variety of changes to a system while generating its recommendations, which include hardware changes, database configuration changes, modification of schema objects, modification of applications, and referrals to other advisors.

> **See Also:** *Oracle Database 2 Day DBA* for more information about Automatic Database Diagnostic Monitor and *Oracle Database Performance Tuning Guide* for more information about Automatic Workload Repository

SQL Tuning Advisor

Oracle Database provides a server utility called the SQL Tuning Advisor. The SQL Tuning Advisor takes one or more SQL statements as input and invokes the Automatic SQL Tuning Advisor to perform SQL tuning on the statements. The output of the SQL Tuning Advisor is in the form of an advice or recommendation, along with a rationale for each recommendation and its expected benefit. The recommendation relates to collection of statistics on objects, creation of new indexes, restructuring of the SQL statement, or creation of SQL Profile. Users can choose whether or not to accept the recommendation to complete the tuning of the SQL statements.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information

SQL Access Advisor

The SQL Access Advisor makes schema modification recommendations. It can recommend that you create access structures such as indexes and materialized views to optimize SQL queries. It can also recommend that you partition tables, indexes, or materialized views to improve query performance.

The SQL Access Advisor takes a SQL workload as input. You can select your workload from various sources, including current and recent SQL activity, a SQL repository, or a

user-defined workload such as from a development environment. The advisor then recommends changes to improve the performance of the workload as a whole.

> **See Also:** *Oracle Database 2 Day + Performance Tuning Guide* for more information

Streams Tuning Advisor

A Streams topology is a representation of the databases in a Streams environment, the Streams components configured in these databases, and the flow of messages between these components. The Streams Performance Advisor reports performance measurements for a Streams topology, including throughput and latency measurements. The Streams Performance Advisor also identifies bottlenecks in a Streams topology so that they can be corrected. In addition, the Streams Performance advisor examines the Streams components in a Streams topology and recommends ways to improve their performance.

> **See Also:** *Oracle Streams Concepts and Administration* for more information

The Scheduler

To help simplify management tasks, as well as providing a rich set of functionality for complex scheduling needs, Oracle Database provides a collection of functions and procedures in the DBMS_SCHEDULER package. Collectively, these functions are called the Scheduler, and they are callable from any PL/SQL program.

The Scheduler lets database administrators and application developers control when and where various tasks take place in the database environment. For example, database administrators can schedule and monitor database maintenance jobs such as backups or nightly data warehousing loads and extracts.

Database Resource Manager

Traditionally, operating systems regulated resource management among various applications, including Oracle databases, that run on a system. The Database Resource Manager controls the distribution of resources among various sessions by controlling the execution schedule inside the database. By controlling which sessions run and for how long, the Database Resource Manager can ensure that resource distribution matches the plan directive and hence, the business objectives.

> **See Also:** Chapter 14, "Manageability" for more information on Database Resource Manager

Overview of Diagnosability Features

Beginning with Oracle Database 11*g*, Oracle Database includes an advanced *fault diagnosability infrastructure* for preventing, detecting, diagnosing, and resolving problems. The problems that are targeted are critical errors such as those caused by database code bugs, metadata corruption, and customer data corruption. For information on the goals of this infrastructure and the Oracle technologies that achieve these goals, see "Fault Diagnosability Infrastructure" on page 14-5.

Overview of Database Backup and Recovery Features

The possibility of a system or hardware failure exists in every database system. The purpose of a backup and recovery strategy is to protect the database against data loss caused by failures and reconstruct the database after data loss.

RMAN and User-Managed Backup and Recovery   Database backups are the cornerstone of any backup and recovery strategy. A backup is a copy of data. This copy can include important parts of the database such as datafiles, the control file, and the server parameter file. Media recovery is the application of redo logs or incremental backups to a restored backup datafile or individual data block. By reapplying the lost changes, recovery rolls the backup forward in time.

When implementing a backup and recovery strategy, you have the following solutions available:

- Recovery Manager (RMAN). This tool integrates with sessions running on an Oracle database to perform a range of backup and recovery activities, including maintaining an RMAN repository of historical data about backups. You can access RMAN through the command line or through Enterprise Manager.

- User-managed backup and recovery. In this solution, you perform backup and recovery with a mixture of host operating system commands and SQL*Plus recovery commands.

Both of the preceding solutions are supported by Oracle and are fully documented, but RMAN is the preferred solution for database backup and recovery. RMAN provides access to several backup and recovery techniques and features not available with user-managed backup and recovery. The most noteworthy are the following:

- Incremental backups

- Block media recovery

- Unused block compression

- Binary compression

- Encrypted backups

Whether you use RMAN or user-managed methods, you can supplement physical backups with logical backups of schema objects made with Data Pump Export utility. You can later use Data Pump Import to re-create data after restore and recovery.

> **See Also:**   "RMAN and User-Managed Backups" on page 15-5 for more information about these backup methods and *Oracle Database Utilities* for more information about Data Pump

Oracle Flashback Technology   Most Oracle flashback features operate at the logical level, enabling you to view and manipulate database objects. The logical-level flashback features of Oracle do not depend on RMAN and are available whether or not RMAN is part of your backup strategy. With the exception of Flashback Drop, the logical flashback features rely on undo data, which are records of the effects of each database update and the values overwritten in the update. Oracle Database includes the following logical flashback features:

- Oracle Flashback Query

- Oracle Flashback Version Query

- Oracle Flashback Transaction Query

- Oracle Flashback Transaction Backout

- Oracle Flashback Table

- Oracle Flashback Drop

- Flashback Data Archive

> **See Also:** "Oracle Flashback Technology" on page 15-9 for more information about these features

Data Recovery Advisor   Oracle Database includes a Data Recovery Advisor tool that automatically diagnoses persistent data failures, presents appropriate repair options, and executes repairs at your request. The Data Recovery Advisor provides a single point of entry for Oracle backup and recovery solutions. You can use Data Recovery Advisor through the Enterprise Manager Database Control or Grid Control console or through the RMAN command-line client.

> **See Also:**
>
> - Chapter 15, "Backup and Recovery" for more information about Oracle backup and recovery methods
>
> - "Data Recovery Advisor" on page 15-9 for more information about this tool

Overview of High Availability Features

Computing environments configured to provide nearly full-time availability are known as high availability systems. Such systems typically have redundant hardware and software that makes the system available despite failures. Well-designed high availability systems avoid having single points of failure.

Oracle Database includes a number of products and features that provide high availability in cases of unplanned downtime or planned downtime. These features, which are described in the sections that follow, can be used in various combinations to meet specific high availability needs.

Oracle Real Application Clusters   Oracle Real Application Clusters (Oracle RAC) allows Oracle Database to run any packaged or custom application unchanged across a set of clustered servers. This capability provides the highest levels of availability and the most flexible scalability. If a clustered server fails, Oracle Database continues running on the surviving servers. When more processing power is needed, you can add another server without interrupting access to data.

Oracle Data Guard   Oracle Data Guard provides a comprehensive set of services that create, maintain, manage, and monitor one or more standby databases to enable production Oracle databases to survive failures, disasters, errors, and data corruption. Data Guard maintains these standby databases as transactionally consistent copies of the production database. If the production database becomes unavailable due to a planned or an unplanned outage, Data Guard can switch any standby database to the production role, thus greatly reducing the downtime caused by the outage.

Oracle Streams   Oracle Streams enables the propagation and management of data, transactions, and events in a data stream, either within a database or from one database to another. Streams provides a set of elements that enables you to control what information is put into a data stream, how the stream is routed from node to node, what happens to events in the stream as they flow into each node, and how the stream terminates.

Oracle Flashback Technology   Flashback technology provides a set of features that let you switch between views of the data as it existed at different points in time. Using flashback features you can query past versions of schema objects and historical data. You can also perform change analysis and self-service repair to recover from logical

corruption while the database is online. Flashback technology is unique to Oracle Database and supports recovery at all levels including row, transaction, table, tablespace, and database.

Online Table Reorganization   Oracle provides a Reorganize Objects wizard in Oracle Enterprise Manager that can automatically generate a script and perform online table reorganization. The entire reorganization process occurs while users have full access to the table.

Automatic Storage Management   Automatic Storage Management (ASM) provides a vertically integrated file system and volume manager directly in the Oracle kernel. ASM spreads files across all available storage. To protect against data loss, ASM extends the concept of SAME (stripe and mirror everything) and adds more flexibility in that it can mirror at the database file level rather than the entire disk level. DBAs using ASM create and administer a large-grained object called a **disk group**. The disk group identifies the set of disks that are managed as a logical unit. Automation of file naming and placement of the underlying database files save DBAs time and ensures adherence to standard best practices.

Recovery Manager   is an Oracle Database utility to manage the backup and recovery of the database. RMAN determines the most efficient method of executing the requested backup, restoration, or recovery operation and then submits these operations to the Oracle Database server for processing. RMAN and the server automatically identify modifications to the structure of the database and dynamically adjust the required operation to adapt to the changes.

Flash Recovery Area   The flash recovery area is a unified storage location for all recovery-related files and activities in Oracle Database. When this feature is enabled, all RMAN backups, archive logs, control file autobackups, and datafile copies are automatically written to a specified file system or to an Automatic Storage Management disk group. The management of this disk space is handled by RMAN and the database server. The flash recovery area eliminates the bottleneck of writing to tape. Further, if database media recovery is required, then datafile backups are readily available.

> **See Also:**   Chapter 17, "High Availability"

Overview of Business Intelligence Features

This section describes the following business intelligence features:

- Data Warehousing
- Materialized Views
- Table Compression
- Parallel Execution
- Analytic SQL
- OLAP Capabilities
- Data Mining
- Very Large Databases (VLDB)

Data Warehousing

A data warehouse is a relational database designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources.

In addition to a relational database, a data warehouse environment includes an extraction, transformation, and loading (ETL) solution, an online analytical processing (OLAP) engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users.

Extraction, Transformation, and Loading (ETL)   You must load your data warehouse regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational systems must be extracted and copied into the warehouse. The process of extracting data from source systems and bringing it into the data warehouse is commonly called **ETL**, which stands for extraction, transformation, and loading.

Bitmap Indexes in Data Warehousing   The purpose of an index is to provide pointers to the rows in a table that contain a given key value. In a regular index, this is achieved by storing a list of rowids for each key corresponding to the rows with that key value. Oracle Database stores each key value repeatedly with each stored rowid. Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of space, because the indexes can be several times larger than the data in the table.

In a bitmap index, the database stores a bitmap for each key value instead of a list of rowids. Bitmap indexes are typically only a fraction of the size of the indexed data in the table. Data warehousing environments typically have large amounts of data and ad hoc queries, but a low level of concurrent database manipulation language (DML) transactions. For such applications, bitmap indexing provides several advantages:

- Reduced response time for large classes of ad hoc queries

- Reduced storage requirements compared with other indexing techniques

- Dramatic performance gains even on hardware with a relatively small number of CPUs or a small amount of memory

- Efficient maintenance during parallel DML and loads

In addition, bitmap join indexes improve query performance for typical data warehouse queries—which often include dimension/fact table joins—with about the same space usage as regular bitmap indexes.

Materialized Views

A **materialized view** provides access to table data by storing the results of a query in a separate schema object. Unlike an ordinary view, which does not take up any storage space or contain any data, a materialized view contains the rows resulting from a query against one or more base tables or views. Query response time is improved because the query accesses the materialized view instead of executing against the base tables. A materialized view can be stored in the same database as its base tables or in a different database.

Materialized views stored in the same database as their base tables can further improve query performance through **query rewrite**. Query rewrite is a mechanism that automatically rewrites a SQL query to use a materialized view instead of its base

tables. With query rewrite, developers need not rewrite applications to take advantage of materialized views. Query rewrite is particularly useful in a data warehouse environment.

Table Compression

Oracle provides comprehensive data compression capabilities to compress all types of data, backups, and network traffic in an application transparent manner. These capabilities include table compression targeted at OLTP workloads, resulting in reduced storage consumption and improved query performance while incurring minimal write performance overhead. Table compression can be used to compress any relational data. To compress unstructured content use SecureFiles compression. Deduplication provides the ability to automatically eliminate redundant copies of SecureFiles data. A new faster compression algorithm is included to speed up RMAN backups. Data Pump exports can now be compressed to reduce disk space requirements. Finally, Data Guard can compress redo data resulting in reduced network traffic and faster gap resolution.

> **See Also:** "Table Compression" on page 5-7

Parallel Execution

When Oracle Database runs SQL statements in parallel, multiple processes work together simultaneously to run a single SQL statement. By dividing the work necessary to run a statement among multiple processes, Oracle Database can run the statement more quickly than if only a single process ran it. This is called **parallel execution** or **parallel processing**. Parallel execution dramatically reduces response time for data-intensive operations on large databases.

Analytic SQL

Oracle Database has many SQL operations for performing analytic operations in the database. These include ranking, moving averages, cumulative sums, ratio-to-reports, and period-over-period comparisons.

OLAP Capabilities

Oracle online analytical processing (OLAP) provides native multidimensional storage and speed-of-thought response times when analyzing data across multiple dimensions. The database provides rich support for analytics such as time series calculations, forecasting, advanced aggregation with additive and nonadditive operators, and allocation operators. These capabilities make the Oracle database a complete analytical platform, capable of supporting the entire spectrum of business intelligence and advanced analytical applications. Oracle OLAP is fully integrated in the database, so that you can use standard SQL administrative, querying, and reporting tools.

Data Mining

With Oracle Data Mining, data never leaves the database — the data, data preparation, model building, and model scoring results all remain in the database. This enables Oracle Database to provide an infrastructure for application developers to integrate data mining seamlessly with database applications. Some typical examples of the applications that data mining are used in are call centers, ATMs, E-business relational management (ERM), and business planning applications. Data mining functions such as model building, testing, and scoring are provided through a Java API.

> **See Also:** Chapter 16, "Business Intelligence" for more information about data mining

Very Large Databases (VLDB)

Partitioning is a critical feature for managing very large databases (VLDB). Growth is the basic challenge that partitioning addresses, and partitioning allows a database to scale for very large datasets while maintaining consistent performance, without unduly increasing administrative or hardware resources. Partitioning allows a table, index, or index-organized table to be subdivided into smaller pieces called **partitions**. No modifications to applications are necessary when accessing a partitioned table using SQL DML statements.

Partitioning can provide tremendous benefit to a wide variety of applications by improving availability, manageability, and performance.

Information Lifecycle Management (ILM) Information Lifecycle Management (ILM) is a set of processes and policies for managing data throughout its useful life. One of the benefits of implementing an ILM solution is to reduce costs, by leveraging appropriate storage tiers, while maintaining all of the data required for business or regulatory purposes. Partitioning is the capability that enables an ILM solution to be implemented within the database.

> **See Also:** Chapter 18, "Very Large Databases (VLDB)" for more information about VLDB topics

Overview of Content Management Features

Oracle Database includes datatypes to handle all the types of rich content such as XML, text, audio, video, image, medical image, and spatial. These datatypes appear as native types in the database. They can all be queried using SQL. A single SQL statement can include data belonging to any or all of these datatypes.

This section includes the following topics:

- XML in Oracle Database
- LOBs
- SecureFiles
- Oracle Text
- Oracle Ultra Search
- Oracle Multimedia
- Oracle Spatial

XML in Oracle Database

Oracle XML DB is a set of Oracle Database technologies related to high-performance XML storage and retrieval. It provides native XML support by encompassing both SQL and XML data models in an interoperable manner. Oracle XML DB includes the following features:

- Support for the World Wide Web Consortium (W3C) XML and XML Schema data models and standard access methods for navigating and querying XML. The data models are incorporated into Oracle Database.
- The ability to store, query, update, and transform XML data while accessing it using SQL.

- The ability to perform XML operations on SQL data.

- A simple, lightweight XML repository where you can organize and manage database content, including XML, using a file/folder/URL metaphor.

- An infrastructure independent of storage format, content, and programming language for storing and managing XML data. This infrastructure provides new ways of navigating and querying XML content stored in the database. For example, Oracle XML DB Repository facilitates this by managing XML document hierarchies.

- Industry-standard access to and update of XML. The standards include the W3C XPath recommendation and the ISO-ANSI SQL/XML standard. FTP, HTTP(S), and WebDAV can be used to move XML content into and out of Oracle Database. Industry-standard APIs provide programmatic access and manipulation of XML content using Java, C, and PL/SQL.

- XML-specific memory management and optimizations.

- Enterprise-level Oracle Database features for XML content: reliability, availability, scalability, and security.

Oracle XML DB can be used in conjunction with Oracle XML Developer's Kit (XDK) to build applications that run in the middle tier in either Oracle Application Server or Oracle Database.

LOBs

The LOB datatypes `BLOB`, `CLOB`, `NCLOB`, and `BFILE` enable you to store and manipulate large blocks of unstructured data (such as text, graphic images, video clips, and sound waveforms) in binary or character format. They provide efficient, random, piece-wise access to the data.

> **See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about SecureFiles LOBs

SecureFiles

SecureFiles is a new feature in Oracle Database 11*g* that offers the best solution for storing file content, such as images, audio, video, PDFs, and spreadsheets. Traditionally, relational data is stored in a database, while unstructured content—both semi-structured and unstructured—is stored as files in file systems. SecureFiles is a major paradigm shift in the choice of files storage. SecureFiles is specifically engineered to deliver high performance for file data comparable to that of traditional file systems, while retaining the advantages of Oracle Database. SecureFiles offers the best database and file system architecture attributes for storing unstructured content.

Key Technical Advantages   SecureFiles includes advanced features, typically found in high-end file systems, such as:

- **Deduplication**: Oracle Database automatically detects multiple, identical SecureFiles data and stores only one copy, thereby saving storage space. In addition to storing only one copy, SecureFiles maintains references to other duplicates. Deduplication is completely transparent to applications and, in addition to simplifying storage management, it also results in significantly better performance, especially for copy operations. Duplicate detection happens within a LOB segment. The `lob_storage_clause` allows for specifying deduplication at a partition level so that duplicate detection does not span across partitions or subpartitions for partitioned SecureFiles columns.

> **See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about deduplication

SecureFiles deduplication is part of the **Advanced Compression** option.

- **Compression**: SecureFiles data is compressed using industry standard compression algorithms. Compression not only results in significant savings in storage but also improved performance by reducing I/O, buffer cache requirements, redo generation, and encryption overhead. If the compression does not yield any savings or if the data is already compressed, SecureFiles automatically turns off compression for such columns. Compression is performed on the server side and allows for random reads and writes to SecureFiles data. SecureFiles provides for varying degrees of compression: MEDIUM (default) and HIGH, which represent a tradeoff between storage savings and latency.

  > **See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about compression

  SecureFiles compression is part of the **Advanced Compression** option.

- **Encryption**: In Oracle Database 11*g*, Oracle has extended the encryption capability to SecureFiles and uses the Transparent Data Encryption (TDE) syntax. Oracle Database supports automatic key management for all SecureFiles columns within a table and transparently encrypts and decrypts data, backups, and redo log files. Applications require no changes and can take advantage of Oracle Database 11*g* SecureFiles using TDE semantics. SecureFiles supports the following encryption algorithms:

  - 3DES168: Triple Data Encryption Standard with a 168-bit key size

  - AES128: Advanced Encryption Standard with a 128 bit key size

  - AES192: Advanced Encryption Standard with a 192-bit key size (default)

  - AES256: Advanced Encryption Standard with a 256-bit key size

  > **See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about encryption

  SecureFiles encryption is part of the **Advanced Security** option.

- **File System-like Logging**: Modern file systems have the ability to keep a running log of the file system metadata. Putting this metadata into a running log (called a journal) that is flushed in a lazy fashion increases performance and removes the need for file system checking operations like `fsck`. SecureFiles' file system-like logging provides this same high performance journaling. File system-like logging also allows for *soft corruptions*, so that if an error is found on a block, SecureFiles returns a block with the LOB fill character. This allows the application to detect the error by seeing known invalid data and to recover either through deletion of the LOB (something that is not possible with the original implementation of LOBs) or by other means.

In addition to the aforementioned advanced file system features, SecureFiles can take advantage of several advanced Oracle Database capabilities, including:

- Transactions, read consistency, and flashback

- 100% backward compatibility with LOB interfaces

- Readable standby, consistent backup, and point-in-time recovery

- Fine-grained auditing and label security

- XML indexing, XML queries, and XPath

- Oracle Real Application Clusters

- Automatic Storage Management

- Partitioning and ILM

- Search across metadata and file content

High Performance   SecureFiles is designed from the ground up for high performance and scalability. SecureFiles delivers comparable file system-like performance for basic read and write operations. The optimized algorithms with SecureFiles make it up to 10 times faster than LOBs. The scalability associated with SecureFiles goes far beyond what is offered in file systems. Organizations can scale-up using large SMP systems, or scale-out using Oracle Real Application Clusters to hundreds of computers while still preserving a single system image. Scaling of CPUs and disks can be done independently and transparently. With Oracle Database 11*g*, organizations can store all types of content and scale to store petabytes or exabytes of data.

Oracle Text

Oracle Text indexes any document or textual content to add fast, accurate retrieval of information. Oracle Text lets you combine text searches with regular database searches in a single SQL statement. The ability to find documents based on their textual content, metadata, or attributes, makes the Oracle Database the single point of integration for all data management.

The Oracle Text SQL API makes it simple and intuitive for application developers and DBAs to create and maintain Text indexes and run Text searches.

Oracle Ultra Search

Oracle Ultra Search lets you index and search Web sites, database tables, files, mailing lists, Oracle Application Server Portals, and user-defined data sources. This search capability lets you can use Oracle Ultra Search to build different kinds of search applications.

Oracle Multimedia

Oracle Multimedia provides an array of services to simplify the development of applications that include images, audio, and video. Oracle Multimedia objects are accessed as columns in tables, like other more typical relational data. Multimedia content can be stored and managed internally in the database, or externally by storing references to the content in the database. Java and PL/SQL APIs provide metadata extraction, image format conversion, and thumbnail image generation to greatly reduce application development and maintenance costs. Excellent integration with application development tools such as Oracle JDeveloper, Application Express, and Oracle Application Server Portal enable application developers to create and maintain media-rich applications with ease. In addition, Oracle Multimedia provides similar support for Digital Imaging and Communications in Medicine (DICOM) content such as single-frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports.

> **See Also:**   Chapter 19, "Content Management" for more information about Oracle Multimedia

Oracle Spatial

Oracle Database includes built-in spatial features that let you store, index, and manage location content—assets, buildings, roads, land parcels, sales regions, and so on—and query location relationships using the power of the database. The Oracle Spatial option adds advanced spatial features such as linear reference support and coordinate systems.

> **See Also:** Chapter 19, "Content Management" for more information about Oracle Spatial

Overview of Security Features

Oracle Database includes security features that control how a database is accessed and used. Security mechanisms are needed for several purposes:

- To prevent unauthorized database access

- To prevent unauthorized access to schema objects

- To audit user actions

Associated with each database user is a schema by the same name. By default, each database user creates and has access to all objects in the corresponding schema.

Database security can be classified into two categories: **system security** and **data security**.

System security lets you control access to and use of the database at the system level. System security mechanisms check whether a user is authorized to connect to the database, whether database auditing is active, and which system operations a user can perform. For example, system security includes:

- Valid user name/password combinations

- The amount of disk space available to a user's schema objects

- The resource limits for a user

Data security lets you control access to and use of the database at the schema object level. For example, data security determines:

- Which users have access to a specific schema object and the specific types of actions allowed for each user on the schema object (for example, user `SCOTT` can issue `SELECT` and `INSERT` statements but not `DELETE` statements using the `employees` table)

- The actions, if any, that are audited for each schema object

- Data encryption to prevent unauthorized users from bypassing Oracle Database and accessing data

Security Mechanisms

Oracle Database provides **discretionary access control**, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access a schema object. Appropriately privileged users can grant other users privileges at their discretion.

Oracle Database manages database security using several different facilities:

- Authentication to validate the identity of the entities using your networks, databases, and applications

- Authorization processes to limit access and actions, limits that are linked to user's identities and roles

- Access restrictions on objects such as tables or rows

- Security policies

- Database auditing

> **See Also:** Chapter 20, "Database Security" for more information on security mechanisms

Overview of Data Integrity and Triggers

Data must adhere to certain business rules, as determined by the database administrator or application developer. For example, assume that a business rule says that no row in the `inventory` table can contain a numeric value greater than nine in the `sale_discount` column. If an `INSERT` or `UPDATE` statement attempts to violate this integrity rule, Oracle Database must undo the invalid statement and return an error to the application. Oracle Database provides integrity constraints and database triggers to manage data integrity rules.

> **Note:** Database triggers let you define and enforce integrity rules, but a database trigger is not the same as an integrity constraint. Among other things, a database trigger does not check data already loaded into a table. Therefore, Oracle strongly recommends that you use database triggers only when the integrity rule cannot be enforced by integrity constraints.

This section includes the following topics:

- Integrity Constraints

- Triggers

Integrity Constraints

An **integrity constraint** is a declarative way to define a business rule for a column of a table. An integrity constraint is a statement about table data that is always true and that follows these rules:

- If an integrity constraint is created for a table and some existing table data does not satisfy the constraint, then the constraint cannot be enforced.

- After a constraint is defined, if any of the results of a DML statement violate the integrity constraint, then the statement is rolled back, and an error is returned.

Integrity constraints are stored as part of the table's definition in the data dictionary, so that all database applications adhere to the same set of rules. When a rule changes, you define it only once at the database level and not once for each application. A **key** is the column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database. Individual values in a key are called **key values**.

The following integrity constraints are supported by Oracle Database:

- A **not null** constraint disallows nulls (empty entries) in a table's column.

- A **unique** constraint disallows duplicate values in a column or set of columns. The **unique key** is the column or set of columns included in the definition of a unique constraint.

- A **primary key** constraint disallows duplicate values and nulls in a column or set of columns. The **primary key** is the column or set of columns included in the definition of a table's primary key constraint. The primary key values uniquely identify the rows in a table. You can define only one primary key for each table.

- A **foreign key** constraint—sometimes called a referential integrity constraint—requires each value in a column or set of columns to match a value in another table's unique key or primary key. Foreign key constraints also define referential integrity actions that dictate what Oracle Database should do with dependent data if the data it references is altered. The **foreign key** is the column or set of columns included in the definition of the foreign key constraint. The **referenced key** is the unique key or primary key of the same or a different table referenced by a foreign key.

- A check constraint disallows values that do not satisfy the logical expression of the constraint.

> **See Also:** Chapter 21, "Data Integrity" for more information about integrity constraints

Triggers

Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly whenever a table or view is modified or when some user actions or database system actions occur.

Triggers supplement the standard capabilities of Oracle Database to provide a highly customized database management system. For example, a trigger can restrict DML operations against a table to those issued during regular business hours.

> **See Also:** Chapter 22, "Triggers" for more information about triggers

Overview of Information Integration Features

A distributed environment is a network of disparate systems that seamlessly communicate with each other. Each system in the distributed environment is called a node. The system to which a user is directly connected is called the local system. Any additional systems accessed by this user are called remote systems. A distributed environment lets applications access and exchange data from the local and remote systems. All the data can be simultaneously accessed and modified.

This section includes the following topics:

- Distributed SQL

- Oracle Streams

- Oracle Database Gateways and Generic Connectivity

Distributed SQL

A homogeneous distributed database system is a network of two or more Oracle databases that reside on one or more computers. Distributed SQL enables applications and users to simultaneously access or modify the data in several databases as easily as they access or modify a single database.

A distributed Oracle database system can appear as though it is a single Oracle database. Companies can use this distributed SQL feature to make all its Oracle databases look like one and thus reduce some of the complexity of the distributed system.

Oracle Database uses database links to enable users on one database to access objects in a remote database. A local user can access a link to a remote database without having to be a user on the remote database.

Location Transparency   Location transparency occurs when the physical location of data is transparent to applications and users. For example, a view that joins table data from several databases provides location transparency because the user of the view does not need to know from where the data originates.

SQL and Transaction Transparency   Oracle Database provides query, update, and transaction transparency. For example, standard SQL statements like `SELECT`, `INSERT`, `UPDATE`, and `DELETE` manipulate data just as they do in a nondistributed database environment. Applications can control transactions using the standard SQL statements `COMMIT`, `SAVEPOINT`, and `ROLLBACK`. Oracle Database ensures the integrity of data in a distributed transaction using the two-phase commit mechanism, whereby all nodes in a distributed system are instructed to commit the transaction. If this is not possible, then all nodes roll back the transaction.

> **See Also:**   *Oracle Database Administrator's Guide* for more information on the two-phase commit mechanism

Distributed Query Optimization   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.

Oracle Streams

Oracle Streams enables the propagation and management of data, transactions, and events in a data stream either within a database or from one database to another. The stream conveys published information to subscribed destinations.

Oracle Streams lets users control what information is put into a stream, how the stream flows or is routed from node to node, what happens to events in the stream as they flow into each node, and how the stream terminates. By specifying the configuration of the elements acting on the stream, a user can address specific requirements, such as message queuing or data replication.

Capture   Oracle Streams implicitly and explicitly captures events and places them in the staging area. Database events, such as DML and DDL operations, are implicitly captured by mining the redo log files. Sophisticated subscription rules can determine what events should be captured.

Staging   The staging area is a queue that stores and manages captured events. Changes to database tables are formatted as logical change records (LCRs), and stored in a staging area until subscribers consume them. LCR staging provides a secure holding area and supports auditing and tracking of LCR data.

Consumption   Messages in a staging area are consumed by the apply engine, where changes are applied to a database or consumed by an application. A flexible apply

engine lets you use a standard or custom apply function. Support for explicit dequeue lets application developers use Oracle Streams to reliably exchange messages. They can also notify applications of changes to data.

Message Queuing Oracle Streams Advanced Queuing is built on the flexible Oracle Streams infrastructure. It provides a unified framework for processing events. Events generated in applications, in workflow, or implicitly captured from redo logs or database triggers can be captured in a queue. These events can be consumed in a variety of ways. They can be automatically applied with a user-defined function or database table operation, can be explicitly dequeued, or a notification can be sent to the consuming application. These events can be transformed at any stage. If the consuming application is on a different database, then the events are automatically propagated to the appropriate database. Operations on these events can be automatically audited, and the history can be retained for a user-specified duration.

Data Replication Replication is the maintenance of database objects in two or more databases. Oracle Streams provides powerful replication features that can be used to synchronize multiple copies of distributed objects.

Oracle Streams automatically determines what information is relevant and shares that information with those who need it. This active sharing of information includes capturing and managing events in the database, including data changed with DML operations, and propagating those events to other databases and applications. Data changes can be applied directly to the replica database, or can call a user-defined procedure to perform alternative work at the destination database, for example, populate a staging table used to load a data warehouse.

Oracle Streams is an open information sharing solution, supporting heterogeneous replication between Oracle and non-Oracle systems. Using a transparent gateway, DML changes initiated at Oracle databases can be applied on non-Oracle platforms.

Oracle Streams is fully interoperational with materialized views, which can maintain updatable or read-only, point-in-time copies of data. They can contain a full copy of a table or a defined subset of the rows in the master table that satisfy a value-based selection criterion. Materialized views can be multitier, where one materialized view is a subset of another materialized view. Materialized views are periodically updated, or refreshed, from their associated master tables through transactionally consistent batch updates.

Oracle Database Gateways and Generic Connectivity

Oracle Database Gateways and Generic Connectivity extend distributed Oracle database features to non-Oracle systems. Generic Connectivity is a generic solution. Oracle Database Gateways are tailored solutions, specifically coded for a particular non-Oracle system. Oracle Database can work with non-Oracle data sources, non-Oracle message queuing systems, and non-SQL applications, ensuring interoperability with other vendors' products and technologies.

Oracle Database Gateways and Generic Connectivity can be used for synchronous access, using distributed SQL, and for asynchronous access, using Oracle Streams. Introducing a Transparent Gateway into an Oracle Streams environment enables replication of data from an Oracle database to a non-Oracle database.

Oracle Database Gateways and Generic Connectivity translate third-party SQL dialects, data dictionaries, and datatypes into Oracle Database formats, thus making the non-Oracle data store appear as a remote Oracle database. These features enable companies to seamlessly integrate the different systems and provide a consolidated view of the company as a whole.

Oracle Database Application Development

SQL and PL/SQL form the core of the Oracle Database application development stack:

- Most enterprise back-ends run SQL

- Web applications accessing databases do so using SQL (wrapped by Java classes as JDBC)

- Enterprise Application Integration applications generate XML from SQL queries

- Content-repositories are built on top of SQL tables

SQL and PL/SQL provide a simple, widely understood, unified data model. They are used standalone in many applications, but are also invoked directly from Java (JDBC), Oracle Call Interface (OCI), Oracle C++ Call Interface (OCCI), or XSU (XML SQL Utility). Stored packages, procedures, and triggers can all be written in PL/SQL or in Java.

This section includes the following topics:

- Overview of Oracle SQL

- Overview of PL/SQL

- Overview of Java

- Overview of Application Programming Languages (APIs)

- Overview of Application Development Environments

- Overview of Datatypes

- Overview of Globalization

Overview of Oracle SQL

Structured query language (**SQL**—pronounced "sequel") is the programming language that defines and manipulates the database. SQL databases are relational databases, which means that data is stored in a set of simple relations.

SQL Statements

All operations on the information in an Oracle database are performed using SQL statements. A SQL statement is a string of SQL text. A statement must be the equivalent of a complete SQL sentence, as in:

```
SELECT last_name, department_id FROM employees;
```

> **Note:** The end of a SQL statement is indicated differently in different programming environments. This documentation set uses the default SQL*Plus character, the semicolon (;).

Only a complete SQL statement can run successfully. A sentence fragment like the following one generates an error indicating that more text is required:

```
SELECT last_name
```

A SQL statement can be thought of as a very simple but powerful computer program or instruction. SQL statements are divided into the following categories:

Data definition language (DDL) statements create, alter, maintain, and drop schema objects. DDL statements also include statements that permit a user to grant other users the privileges to access the database and specific objects within the database.

Data manipulation language (DML) statements manipulate data. Querying, inserting, updating, and deleting rows of a table are all DML operations. The most common SQL statement is the SELECT statement, which retrieves data from the database. Locking a table or view and examining the execution plan of a SQL statement are also DML operations.

Transaction control statements manage the changes made by DML statements. They enable a user to group changes into logical transactions. Examples include COMMIT, ROLLBACK, and SAVEPOINT.

Session control statements let a user control the properties of the current session, including enabling and disabling roles and changing language settings. The two session control statements are ALTER SESSION and SET ROLE.

System control statements changes the properties of the Oracle database instance. ALTER SYSTEM is the only system control statement. It lets you change settings, such as the minimum number of shared servers. It also lets you kill a session and perform other system-wide tasks.

Embedded SQL statements incorporate DDL, DML, and transaction control statements in a procedural language program, such as those used with the Oracle precompilers. Examples include OPEN, CLOSE, FETCH, and EXECUTE.

> **See Also:** Chapter 24, "SQL" for more information about SQL

Overview of PL/SQL

PL/SQL is the Oracle procedural language extension to SQL. PL/SQL combines the ease and flexibility of SQL with the procedural functionality of a structured programming language, including such routines as IF ... THEN, WHILE, and LOOP.

When designing a database application, consider the following advantages of using stored PL/SQL:

- PL/SQL code can be stored in a database. Network traffic between applications and the database is reduced, so application and system performance increases. Even when PL/SQL is not stored in the database, applications can send to the database blocks of PL/SQL rather than individual SQL statements, thereby reducing network traffic.

- Native compilation of PL/SQL code is very easy and offers significant performance advantages.

- Data access can be controlled by stored PL/SQL code. PL/SQL users can access data only as intended by application developers, unless another access route is granted.

- Oracle supports PL/SQL Server Pages, so your application logic can be invoked directly from your Web pages.

The following sections describe some of the PL/SQL program units that can be defined and stored centrally in a database.

Procedures and **functions** are sets of SQL and PL/SQL statements grouped together as a unit to solve a specific problem or to perform a set of related tasks. They are

created and stored in compiled form in the database and can be run by a user or a database application. Procedures and functions are identical, except that functions always return a single value to the user. Procedures do not return values.

Packages encapsulate and store related procedures, functions, variables, and other constructs together as a unit in the database. They offer increased functionality. For example, global package variables can be declared and used by any procedure in the package. Packages also improve performance, because all objects of the package are parsed, compiled, and loaded into memory once.

> **See Also:** Chapter 24, "SQL" for more information about PL/SQL

Overview of Java

Java is an object-oriented programming language efficient for application-level programs. Oracle Database provides all types of JDBC drivers and enhances database access from Java applications. Java Stored Procedures are portable and secure in terms of access control, and they let non-Java and legacy applications transparently invoke Java. In addition, native compilation of Java code is very easy and offers significant performance advantages.

> **See Also:** Chapter 24, "SQL" for more information about Java

Overview of Application Programming Languages (APIs)

Oracle Database developers have a choice of languages for developing applications—C, C++, Java, COBOL, PL/SQL, PHP, and Visual Basic. The entire functionality of the database is available in all of the languages. All language-specific standards are supported. Developers can choose the languages in which they are most proficient or one that is most suitable for a specific task. For example, an application might use Java on the server side to create dynamic Web pages, PL/SQL to implement stored procedures in the database, and C++ to implement computationally intensive logic in the middle tier.

> **See Also:** The following books describe the various Oracle APIs:
>
> - *Pro*C/C++ Programmer's Guide*
> - *Oracle Call Interface Programmer's Guide*
> - *Pro*COBOL Programmer's Guide*
> - *Oracle Database PL/SQL Language Reference*
> - *Oracle Database Data Cartridge Java API Reference*
>
> Also refer to Chapter 25, "Supported Application Development Languages" for more information.

Overview of Application Development Environments

Oracle provides different application development environments for different application developer needs.

- Oracle Application Express is a hosted declarative development environment for developing and deploying database-centric Web applications. Using only a Web browser and limited programming experience, you can develop and deploy professional applications that are both fast and secure. The Application Express engine lives completely within your Oracle database and is written in PL/SQL. It renders applications in real time from data stored in database tables. When you

create or extend an application, Oracle Application Express creates or modifies metadata stored in database tables. When the application is run, the Application Express engine then reads the metadata and displays the application. Oracle Application Express also transparently manages session state in the database. Application developers can get and set session state using simple substitutions as well as standard SQL bind variable syntax. Application Express is a tool to build Web-based applications and the application development environment is also conveniently Web based itself.

> **See Also:** *Oracle Database Express Edition 2 Day Developer Guide* for more information

■ PHP—a self-referencing acronym for PHP - Hypertext Preprocessor—is a popular scripting language commonly embedded with HTML to create dynamic web pages. PHP is perfect for rapidly developing Web 2.0 applications. PHP's oci8 extension is a stable, high-performance PHP database driver that is fully integrated with Oracle Database. Using PHP with Oracle Database, you can query and update data, execute stored procedures and functions, load images, and easily build scalable, global applications.

> **See Also:** *Oracle Database 2 Day + PHP Developer's Guide* for more information

■ In the Microsoft Windows environment, Oracle provides the following development environments:

    – The Oracle Data Provider for .NET (ODP.NET) features optimized data access to the Oracle database from a .NET environment. ODP.NET allows developers to take advantage of advanced Oracle database functionality, including Oracle Real Application Clusters, XML DB, and advanced security. The data provider can be used from any .NET language, including C# and Visual Basic .NET.

> **See Also:**

Overview of Datatypes

Each column value and constant in a SQL statement has a **datatype**, which is associated with a specific storage format, constraints, and a valid range of values. When you create a table, you must specify a datatype for each of its columns.

Oracle Database lets you use many datatypes, in several categories:

■ Scalar datatypes, such as character, numeric, and datetime datatypes

■ Collection types such as variable-length arrays (varrays) and nested tables for more fine-graine3d organization of and access to data in the database

■ ANSI-supported types, which facilitates working with data from non-Oracle databases

■ Supplied datatypes, which are SQL-based interfaces for defining new types when the built-in or ANSI-supported types are not sufficient. The behavior for these types can be implemented in C/C++, Java, or PL/ SQL.

In addition, user-defined object types can be created from any built-in datatypes or any previously created object types, object references, and collection types. Metadata for user-defined types is stored in a schema available to SQL, PL/SQL, Java, and other published interfaces.

A user-defined object type differs from native SQL datatypes in that it specifies both the underlying persistent data (attributes) and the related behaviors (methods). Object types are abstractions of the real-world entities and are sometimes called abstract datatypes (ADTs).

See Also:

- *Oracle Database SQL Language Reference* for a complete listing of the Oracle built-in and supplied datatypes

  Chapter 26, "Oracle Data Types"

- *Oracle Database Object-Relational Developer's Guide*

Overview of Globalization

Oracle databases can be deployed anywhere in the world, and a single instance of Oracle Database can be accessed by users across the globe. Information is presented to each user in the language and format specific to his or her location.

The Globalization Development Kit (GDK) simplifies the development process and reduces the cost of developing internet applications for a multilingual market. GDK lets a single program work with text in any language from anywhere in the world.

See Also: *Oracle Database Globalization Support Guide* for more information about globalization

Part II

Oracle Database Architecture

Part II describes the basic structural architecture of the Oracle database, including physical and logical storage structures. Part II contains the following chapters:

2

Data Blocks, Extents, and Segments

This chapter describes the nature of and relationships among the logical storage structures in the Oracle database server.

This chapter contains the following topics:

- Introduction to Data Blocks, Extents, and Segments
- Overview of Data Blocks
- Overview of Extents
- Overview of Segments

Introduction to Data Blocks, Extents, and Segments

Oracle Database allocates logical database space for all data in a database. The units of database space allocation are data blocks, extents, and segments. Figure 2–1 shows the relationships among these data structures.

Figure 2–1   The Relationships Among Segments, Extents, and Data Blocks
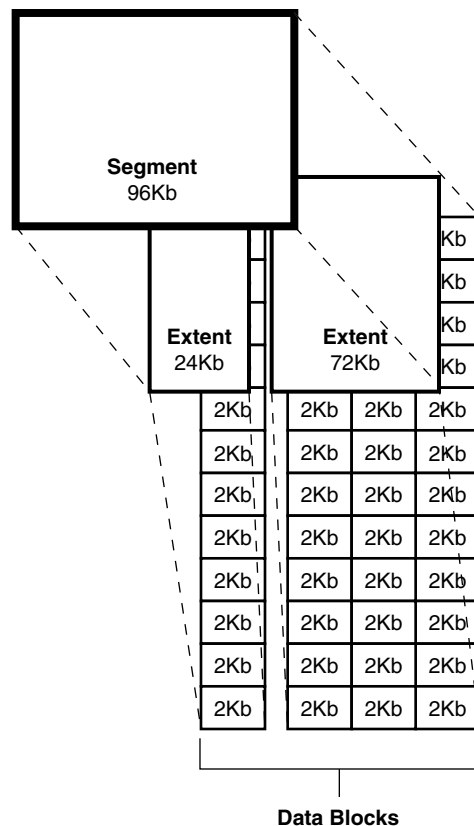
Data Blocks

At the finest level of granularity, Oracle Database stores data in **data blocks** (also called **logical blocks**, **Oracle blocks**, or **pages**). One data block corresponds to a specific number of bytes of physical database space on disk.

The next level of logical database space is an **extent**. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information.

The level of logical database storage greater than an extent is called a **segment**. A segment is a set of extents, each of which has been allocated for a specific data structure and all of which are stored in the same tablespace. For example, each table's data is stored in its own **data segment**, while each index's data is stored in its own **index segment**. If the table or index is partitioned, each partition is stored in its own segment.

Oracle Database allocates space for segments in units of one extent. When the existing extents of a segment are full, Oracle Database allocates another extent for that segment. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

A segment and all its extents are stored in one tablespace. Within a tablespace, a segment can include extents from more than one file; that is, the segment can span datafiles. However, each extent can contain data from only one datafile.

Although you can allocate additional extents, the blocks themselves are allocated separately. If you allocate an extent to a specific instance, the blocks are immediately allocated to the free list. However, if the extent is not allocated to a specific instance, then the blocks themselves are allocated only when the high water mark moves. The **high water mark** is the boundary between used and unused space in a segment.

> **Note:** Oracle recommends that you manage free space automatically. See "Free Space Management" on page 2-5.

Overview of Data Blocks

Oracle Database manages the storage space in the datafiles of a database in units called **data blocks**. A data block is the smallest unit of data used by a database. In contrast, at the physical, operating system level, all data is stored in bytes. Each operating system has a **block size**. Oracle Database requests data in multiples of Oracle Database data blocks, not operating system blocks.

The standard block size is specified by the DB_BLOCK_SIZE initialization parameter. In addition, you can specify of up to five nonstandard block sizes. The data block sizes should be a multiple of the operating system's block size within the maximum limit to avoid unnecessary I/O. Oracle Database data blocks are the smallest units of storage that Oracle Database can use or allocate.

This section includes the following topics:

- Data Block Format
- Free Space Management
- PCTFREE, PCTUSED, and Row Chaining

> **See Also:**
> - Your Oracle Database operating system-specific documentation for more information about data block sizes
> - Multiple Block Sizes on page 3-11

Data Block Format

The Oracle Database data block format is similar regardless of whether the data block contains table, index, or clustered data. Figure 2–2 illustrates the format of a data block.

Figure 2–2   Data Block Format
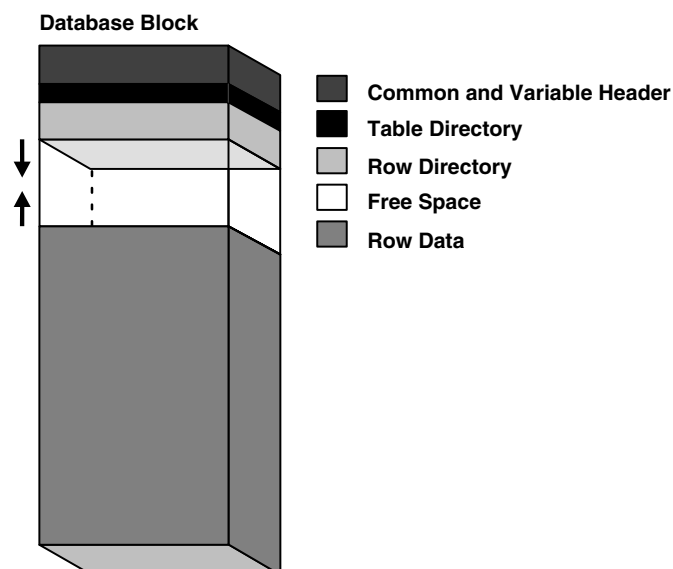
This section discusses the following components of the data block:

- Header (Common and Variable)
- Table Directory
- Row Directory
- Overhead
- Row Data
- Free Space

Header (Common and Variable)

The header contains general block information, such as the block address and the type of segment (for example, data or index).

Table Directory

This portion of the data block contains information about the table having rows in this block.

Row Directory

This portion of the data block contains information about the actual rows in the block (including addresses for each row piece in the row data area).

After the space has been allocated in the row directory of a data block's overhead, this space is not reclaimed when the row is deleted. Therefore, a block that is currently empty but had up to 50 rows at one time continues to have 100 bytes allocated in the header for the row directory. Oracle Database reuses this space only when new rows are inserted in the block.

Overhead

The data block header, table directory, and row directory are referred to collectively as **overhead**. Some block overhead is fixed in size; the total block overhead size is variable. On average, the fixed and variable portions of data block overhead total 84 to 107 bytes.

Row Data

This portion of the data block contains table or index data. Rows can span blocks.

> **See Also:** "Row Chaining and Migrating" on page 2-5

Free Space

Free space is allocated for insertion of new rows and for updates to rows that require additional space (for example, when a trailing null is updated to a nonnull value).

In data blocks allocated for the data segment of a table or cluster, or for the index segment of an index, free space can also hold transaction entries. A **transaction entry** is required in a block for each `INSERT`, `UPDATE`, `DELETE`, and `SELECT...FOR UPDATE` statement accessing one or more rows in the block. The space required for transaction entries is operating system dependent; however, transaction entries in most operating systems require approximately 23 bytes.

Free Space Management

Free space can be managed automatically or manually.

Free space can be managed automatically inside database segments. The in-segment free/used space is tracked using bitmaps, as opposed to free lists. Automatic segment-space management offers the following benefits:

- Ease of use

- Better space utilization, especially for the objects with highly varying row sizes

- Better run-time adjustment to variations in concurrent access

- Better multi-instance behavior in terms of performance/space utilization

You specify automatic segment-space management when you create a locally managed tablespace. The specification then applies to all segments subsequently created in this tablespace.

> **See Also:** *Oracle Database Administrator's Guide*

This section includes the following topics:

- Availability and Optimization of Free Space in a Data Block

- Row Chaining and Migrating

Availability and Optimization of Free Space in a Data Block

Two types of statements can increase the free space of one or more data blocks: `DELETE` statements, and `UPDATE` statements that update existing values to smaller values. The released space from these types of statements is available for subsequent `INSERT` statements under the following conditions:

- If the `INSERT` statement is in the same transaction and subsequent to the statement that frees space, then the `INSERT` statement can use the space made available.

- If the `INSERT` statement is in a separate transaction from the statement that frees space (perhaps being run by another user), then the `INSERT` statement can use the space made available only after the other transaction commits and only if the space is needed.

Released space may or may not be contiguous with the main area of free space in a data block. Oracle Database coalesces the free space of a data block *only* when (1) an `INSERT` or `UPDATE` statement attempts to use a block that contains enough free space to contain a new row piece, and (2) the free space is fragmented so the row piece cannot be inserted in a contiguous section of the block. Oracle Database does this compression only in such situations, because otherwise the performance of a database system decreases due to the continuous compression of the free space in data blocks.

Row Chaining and Migrating

In two circumstances, the data for a row in a table may be too large to fit into a single data block. In the first case, the row is too large to fit into one data block when it is first inserted. In this case, Oracle Database stores the data for the row in a **chain** of data blocks (one or more) reserved for that segment. Row chaining most often occurs with large rows, such as rows that contain a column of datatype `LONG` or `LONG RAW`. Row chaining in these cases is unavoidable.

However, in the second case, a row that originally fit into one data block is updated so that the overall row length increases, and the block's free space is already completely

filled. In this case, Oracle Database **migrates** the data for the entire row to a new data block, assuming the entire row can fit in a new block. Oracle Database preserves the original row piece of a migrated row to point to the new block containing the migrated row. The rowid of a migrated row does not change.

When a row is chained or migrated, I/O performance associated with this row decreases because Oracle Database must scan more than one data block to retrieve the information for the row.

> **See Also:**
>
> - "Row Format and Size" on page 5-5 for more information on the format of a row and a row piece
>
> - "Rowids of Row Pieces" on page 5-7 for more information on rowids
>
> - "Physical Rowids" on page 26-14 for information about rowids
>
> - *Oracle Database Performance Tuning Guide* for information about reducing chained and migrated rows and improving I/O performance

PCTFREE, PCTUSED, and Row Chaining

For manually managed tablespaces, two space management parameters, `PCTFREE` and `PCTUSED`, enable you to control the use of free space for inserts and updates to the rows in all the data blocks of a particular segment. Specify these parameters when you create or alter a table or cluster (which has its own data segment). You can also specify the storage parameter `PCTFREE` when creating or altering an index (which has its own index segment).

This section includes the following topics:

- The PCTFREE Parameter

- The PCTUSED Parameter

- How PCTFREE and PCTUSED Work Together

> **Note:** This discussion does not apply to LOB datatypes (`BLOB`, `CLOB`, `NCLOB`, and `BFILE`). They do not use the `PCTFREE` storage parameter or free lists.
>
> See "Overview of LOB Datatypes" on page 26-11 for information.

The PCTFREE Parameter

The `PCTFREE` parameter sets the minimum percentage of a data block to be **reserved** as free space for possible updates to rows that already exist in that block. For example, assume that you specify the following parameter within a `CREATE TABLE` statement:

```
PCTFREE 20
```

This states that 20% of each data block in this table's data segment be kept free and available for possible updates to the existing rows already within each block. New rows can be added to the row data area, and corresponding information can be added to the variable portions of the overhead area, until the row data and overhead total 80% of the total block size. Figure 2–3 illustrates `PCTFREE`.

Figure 2–3   PCTFREE

Data Block
PCTFREE = 20

20% Free Space

Block allows row inserts
until 80% is occupied,
leaving 20% free for updates
to existing rows in the block
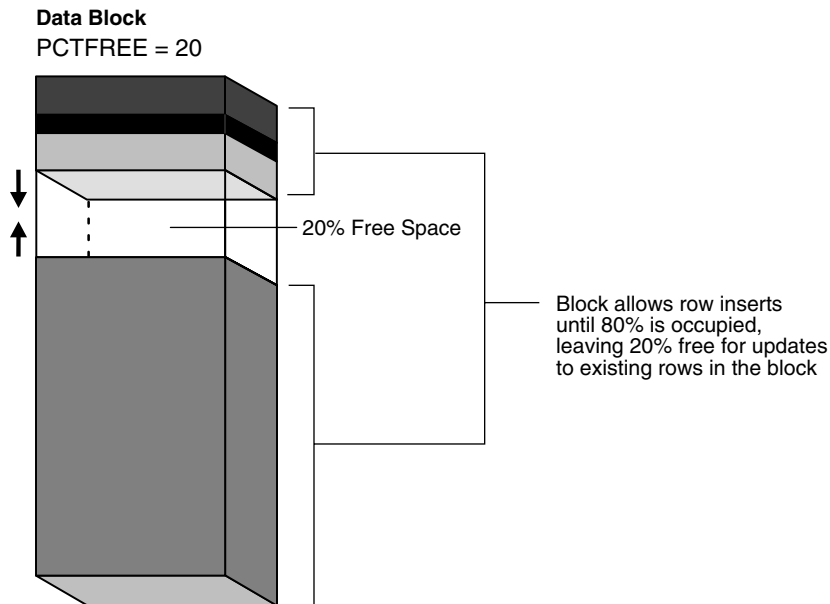
The PCTUSED Parameter

The PCTUSED parameter sets the minimum percentage of a block that can be used for row data plus overhead before new rows are added to the block. After a data block is filled to the limit determined by PCTFREE, Oracle Database considers the block unavailable for the insertion of new rows until the percentage of that block falls beneath the parameter PCTUSED. Until this value is achieved, Oracle Database uses the free space of the data block only for updates to rows already contained in the data block. For example, assume that you specify the following parameter in a CREATE TABLE statement:

```
PCTUSED 40
```

In this case, a data block used for this table's data segment is considered unavailable for the insertion of any new rows until the amount of used space in the block falls to 39% or less (assuming that the block's used space has previously reached PCTFREE). Figure 2–4 illustrates this.

Figure 2–4   PCTUSED

Data Block
PCTUSED = 40

61% Free
Space

No new rows are
inserted until amount
of used space falls
below 40%
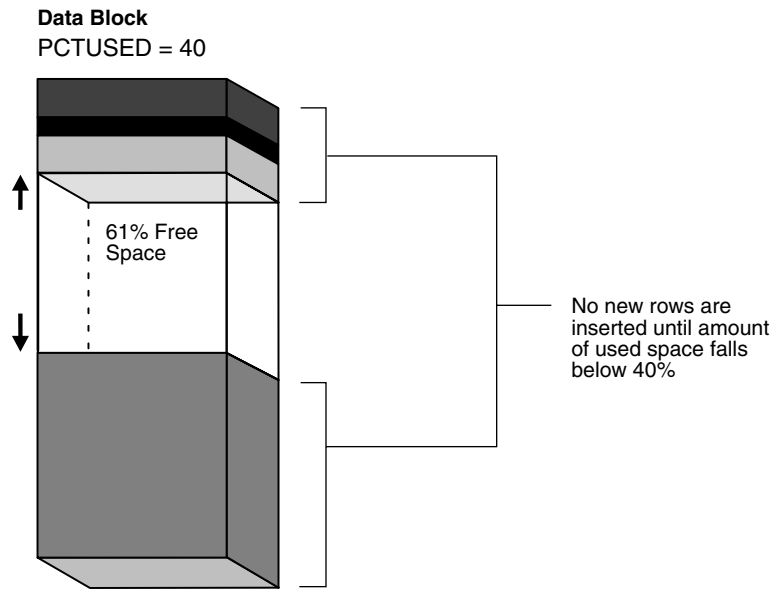
How PCTFREE and PCTUSED Work Together

PCTFREE and PCTUSED work together to optimize the use of space in the data blocks
of the extents within a data segment. Figure 2–5 illustrates the interaction of these two
parameters.

Figure 2–5 Maintaining the Free Space of Data Blocks with PCTFREE and PCTUSED

1 Rows are inserted up to 80% only, because PCTFREE specifies that 20% of the block must remain open for updates of existing rows.

2 Updates to exisiting rows use the free space reserved in the block. No new rows can be inserted into the block until the amount of used space is 39% or less.

3 After the amount of used space falls below 40%, new rows can again be inserted into this block.

4 Rows are inserted up to 80% only, because PCTFREE specifies that 20% of the block must remain open for updates of existing rows. This cycle continues . . .

In a newly allocated data block, the space available for inserts is the block size minus the sum of the block overhead and free space (PCTFREE). Updates to existing data can use any available space in the block. Therefore, updates can reduce the available space of a block to less than PCTFREE, the space reserved for updates but not accessible to inserts.

For each data and index segment, Oracle Database maintains one or more **free lists**—lists of data blocks that have been allocated for that segment's extents and have free space greater than PCTFREE. These blocks are available for inserts. When you issue an INSERT statement, Oracle Database checks a free list of the table for the first available data block and uses it if possible. If the free space in that block is not large enough to accommodate the INSERT statement, and the block is at least PCTUSED, then Oracle Database takes the block off the free list. Multiple free lists for each segment can reduce contention for free lists when concurrent inserts take place.

After you issue a DELETE or UPDATE statement, Oracle Database processes the statement and checks to see if the space being used in the block is now less than PCTUSED. If it is, then the block goes to the beginning of the transaction free list, and it is the first of the available blocks to be used in that transaction. When the transaction commits, free space in the block becomes available for other transactions.

Overview of Extents

An extent is a logical unit of database storage space allocation made up of a number of contiguous data blocks. One or more extents in turn make up a segment. When the existing space in a segment is completely used, Oracle Database allocates a new extent for the segment.

This section includes the following topics:

- When Extents Are Allocated
- Determine the Number and Size of Extents
- How Extents Are Allocated
- When Extents Are Deallocated

When Extents Are Allocated

When you create a table, Oracle Database allocates to the table's data segment an **initial extent** of a specified number of data blocks. Although no rows have been inserted yet, the Oracle Database data blocks that correspond to the initial extent are reserved for that table's rows.

If the data blocks of a segment's initial extent become full and more space is required to hold new data, Oracle Database automatically allocates an **incremental extent** for that segment. An incremental extent is a subsequent extent of the same or greater size than the previously allocated extent in that segment.

For maintenance purposes, the header block of each segment contains a directory of the extents in that segment.

> **Note:** This chapter applies to serial operations, in which one server process parses and runs a SQL statement. Extents are allocated somewhat differently in parallel SQL statements, which entail multiple server processes.

Determine the Number and Size of Extents

Storage parameters expressed in terms of extents define every segment. Storage parameters apply to all types of segments. They control how Oracle Database allocates free database space for a given segment. For example, you can determine how much space is initially reserved for a table's data segment or you can limit the number of extents the table can allocate by specifying the storage parameters of a table in the STORAGE clause of the CREATE TABLE statement. If you do not specify a table's storage parameters, then it uses the default storage parameters of the tablespace.

You can have dictionary managed tablespaces, which rely on data dictionary tables to track space utilization, or locally managed tablespaces, which use bitmaps (instead of data dictionary tables) to track used and free space. Because of the better performance and easier manageability of locally managed tablespaces, the default for non-SYSTEM permanent tablespaces is locally managed whenever the type of extent management is not explicitly specified.

A tablespace that manages its extents locally can have either uniform extent sizes or variable extent sizes that are determined automatically by the system. When you create the tablespace, the UNIFORM or AUTOALLOCATE (system-managed) clause specifies the type of allocation.

- For uniform extents, you can specify an extent size or use the default size, which is 1 MB. Ensure that each extent contains at least five database blocks, given the database block size. Temporary tablespaces that manage their extents locally can only use this type of allocation.

- For system-managed extents, Oracle Database determines the optimal size of additional extents, with a minimum extent size of 64 KB. If the tablespaces are created with 'segment space management auto', and if the database block size is 16K or higher, then Oracle Database manages segment size by creating extents with a minimum size of 1M. This is the default for permanent tablespaces.

The storage parameters INITIAL, NEXT, PCTINCREASE, and MINEXTENTS cannot be specified at the tablespace level for locally managed tablespaces. They can, however, be specified at the segment level. In this case, INITIAL, NEXT, PCTINCREASE, and MINEXTENTS are used together to compute the initial size of the segment. After the segment size is computed, internal algorithms determine the size of each extent.

> **See Also:**
>
> - "Managing Space in Tablespaces" on page 3-9
> - "Bigfile Tablespaces" on page 3-5
> - *Oracle Database Administrator's Guide*

How Extents Are Allocated

Oracle Database uses different algorithms to allocate extents, depending on whether they are locally managed or dictionary managed.

With locally managed tablespaces, Oracle Database looks for free space to allocate to a new extent by first determining a candidate datafile in the tablespace and then searching the datafile's bitmap for the required number of adjacent free blocks. If that datafile does not have enough adjacent free space, then Oracle Database looks in another datafile.

> **Note:** Oracle strongly recommends that you use locally managed tablespaces.

When Extents Are Deallocated

Oracle Database provides a Segment Advisor that helps you determine whether an object has space available for reclamation based on the level of space fragmentation within the object.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for guidelines on reclaiming segment space
> - *Oracle Database SQL Language Reference* for SQL syntax and semantics

In general, the extents of a segment do not return to the tablespace until you drop the schema object whose data is stored in the segment (using a DROP TABLE or DROP CLUSTER statement). Exceptions to this include the following:

- The owner of a table or cluster, or a user with the DELETE ANY privilege, can truncate the table or cluster with a TRUNCATE...DROP STORAGE statement.

- A database administrator (DBA) can deallocate unused extents using the following SQL syntax:

```
ALTER TABLE table_name DEALLOCATE UNUSED;
```

- Periodically, Oracle Database deallocates one or more extents of a rollback segment if it has the OPTIMAL size specified.

When extents are freed, Oracle Database modifies the bitmap in the datafile (for locally managed tablespaces) or updates the data dictionary (for dictionary managed tablespaces) to reflect the regained extents as available space. Any data in the blocks of freed extents becomes inaccessible.

This section includes the following topics:

- Extents in Nonclustered Tables
- Extents in Clustered Tables
- Extents in Materialized Views and Their Logs
- Extents in Indexes
- Extents in Temporary Segments
- Extents in Rollback Segments

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
> - *Oracle Database SQL Language Reference*

Extents in Nonclustered Tables

As long as a nonclustered table exists or until you truncate the table, any data block allocated to its data segment remains allocated for the table. Oracle Database inserts new rows into a block if there is enough room. Even if you delete all rows of a table, Oracle Database does not reclaim the data blocks for use by other objects in the tablespace.

After you drop a nonclustered table, this space can be reclaimed when other extents require free space. Oracle Database reclaims all the extents of the table's data and index segments for the tablespaces that they were in and makes the extents available for other schema objects in the same tablespace.

In dictionary managed tablespaces, when a segment requires an extent larger than the available extents, Oracle Database identifies and combines contiguous reclaimed extents to form a larger one. This is called **coalescing** extents. Coalescing extents is not necessary in locally managed tablespaces, because all contiguous free space is available for allocation to a new extent regardless of whether it was reclaimed from one or more extents.

Extents in Clustered Tables

Clustered tables store information in the data segment created for the cluster. Therefore, if you drop one table in a cluster, the data segment remains for the other tables in the cluster, and no extents are deallocated. You can also truncate clusters (except for hash clusters) to free extents.

Extents in Materialized Views and Their Logs

Oracle Database deallocates the extents of materialized views and materialized view logs in the same manner as for tables and clusters.

> **See Also:** "Overview of Materialized Views" on page 5-18

Extents in Indexes

All extents allocated to an index segment remain allocated as long as the index exists. When you drop the index or associated table or cluster, Oracle Database reclaims the extents for other uses within the tablespace.

Extents in Temporary Segments

When Oracle Database completes the execution of a statement requiring a temporary segment, Oracle Database automatically drops the temporary segment and returns the extents allocated for that segment to the associated tablespace. A single sort allocates its own temporary segment in a temporary tablespace of the user issuing the statement and then returns the extents to the tablespaces.

Multiple sorts, however, can use sort segments in temporary tablespaces designated exclusively for sorts. These sort segments are allocated only once for the instance, and they are not returned after the sort, but remain available for other multiple sorts.

A temporary segment in a temporary table contains data for multiple statements of a single transaction or session. Oracle Database drops the temporary segment at the end of the transaction or session, returning the extents allocated for that segment to the associated tablespace.

> **See Also:**
>
> - "Introduction to Temporary Segments" on page 2-14
> - "Temporary Tables" on page 5-10

Extents in Rollback Segments

Oracle Database periodically checks the rollback segments of the database to see if they have grown larger than their optimal size. If a rollback segment is larger than is optimal (that is, it has too many extents), then Oracle Database automatically deallocates one or more extents from the rollback segment.

Overview of Segments

A segment is a set of extents that contains all the data for a specific logical storage structure within a tablespace. For example, for each table, Oracle Database allocates one or more extents to form that table's data segment, and for each index, Oracle Database allocates one or more extents to form its index segment.

This section contains the following topics:

- Introduction to Data Segments
- Introduction to Index Segments
- Introduction to Temporary Segments
- Introduction to Undo Segments and Automatic Undo Management

Introduction to Data Segments

A single data segment in an Oracle Database database holds all of the data for one of the following:

- A table that is not partitioned or clustered
- A partition of a partitioned table
- A cluster of tables

Oracle Database creates this data segment when you create the table or cluster with the CREATE statement.

The storage parameters for a table or cluster determine how its data segment's extents are allocated. You can set these storage parameters directly with the appropriate CREATE or ALTER statement. These storage parameters affect the efficiency of data retrieval and storage for the data segment associated with the object.

> **Note:** Oracle Database creates segments for materialized views and materialized view logs in the same manner as for tables and clusters.

See Also:

- *Oracle Database Advanced Replication* for information on materialized views and materialized view logs
- *Oracle Database SQL Language Reference* for syntax

Introduction to Index Segments

Every nonpartitioned index in an Oracle database has a single index segment to hold all of its data. For a partitioned index, every partition has a single index segment to hold its data.

Oracle Database creates the index segment for an index or an index partition when you issue the CREATE INDEX statement. In this statement, you can specify storage parameters for the extents of the index segment and a tablespace in which to create the index segment. (The segments of a table and an index associated with it do not have to occupy the same tablespace.) Setting the storage parameters directly affects the efficiency of data retrieval and storage.

Introduction to Temporary Segments

When processing queries, Oracle Database often requires temporary workspace for intermediate stages of SQL statement parsing and execution. Oracle Database automatically allocates this disk space called a **temporary segment**. Typically, Oracle Database requires a temporary segment as a database area for sorting. Oracle Database does not create a segment if the sorting operation can be done in memory or if Oracle Database finds some other way to perform the operation using indexes.

This section includes the following topics:

- Operations that Require Temporary Segments
- Segments in Temporary Tables and Their Indexes
- How Temporary Segments Are Allocated

Operations that Require Temporary Segments

The following statements sometimes require the use of a temporary segment:

- `CREATE INDEX`

- `SELECT ... ORDER BY`

- `SELECT DISTINCT ...`

- `SELECT ... GROUP BY`

- `SELECT...UNION`

- `SELECT ... INTERSECT`

- `SELECT ... MINUS`

Some unindexed joins and correlated subqueries can require use of a temporary segment. For example, if a query contains a `DISTINCT` clause, a `GROUP BY`, and an `ORDER BY`, Oracle Database can require as many as two temporary segments.

Segments in Temporary Tables and Their Indexes

Oracle Database can also allocate temporary segments for temporary tables and indexes created on temporary tables. Temporary tables hold data that exists only for the duration of a transaction or session.

> **See Also:** "Temporary Tables" on page 5-10

How Temporary Segments Are Allocated

Oracle Database allocates temporary segments differently for queries and temporary tables.

This section includes the following topics:

- Allocation of Temporary Segments for Queries

- Allocation of Temporary Segments for Temporary Tables and Indexes

Allocation of Temporary Segments for Queries Oracle Database allocates temporary segments as needed during a user session in one of the temporary tablespaces of the user issuing the statement. Specify these tablespaces with a `CREATE USER` or an `ALTER USER` statement using the `TEMPORARY TABLESPACE` clause.

> **Note:** You cannot assign a permanent tablespace as a user's temporary tablespace.

If no temporary tablespace is defined for the user, then the default temporary tablespace is the `SYSTEM` tablespace. The default storage characteristics of the containing tablespace determine those of the extents of the temporary segment. Oracle Database drops temporary segments when the statement completes.

Because allocation and deallocation of temporary segments occur frequently, create at least one special tablespace for temporary segments. By doing so, you can distribute I/O across disk devices, and you can avoid fragmentation of the `SYSTEM` and other tablespaces that otherwise hold temporary segments.

> **Note:** When the SYSTEM tablespace is locally managed, you must define a default temporary tablespace when creating a database. A locally managed SYSTEM tablespace cannot be used for default temporary storage.

Entries for changes to temporary segments used for sort operations are not stored in the redo log, except for space management operations on the temporary segment.

See Also:

- "Bigfile Tablespaces" on page 3-5
- Chapter 20, "Database Security" for more information about assigning a user's temporary segment tablespace

Allocation of Temporary Segments for Temporary Tables and Indexes Oracle Database allocates segments for a temporary table when the first INSERT into that table is issued. (This can be an internal insert operation issued by CREATE TABLE AS SELECT.) The first INSERT into a temporary table allocates the segments for the table and its indexes, creates the root page for the indexes, and allocates any LOB segments.

Segments for a temporary table are allocated in a temporary tablespace of the user who created the temporary table.

Oracle Database drops segments for a transaction-specific temporary table at the end of the transaction and drops segments for a session-specific temporary table at the end of the session. If other transactions or sessions share the use of that temporary table, the segments containing their data remain in the table.

See Also: "Temporary Tables" on page 5-10

Introduction to Undo Segments and Automatic Undo Management

Oracle Database maintains information to reverse changes made to the database. This information consists of records of the actions of transactions, collectively known as **undo**. Undo is stored in undo segments in an *undo tablespace*. Oracle Database uses undo information to do the following:

- Rollback an active transaction
- Recover a terminated transaction
- Provide read consistency
- Recovery from logical corruptions

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 datafiles. 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 "How Oracle Database Manages Data Concurrency and Consistency" on page 13-3 for more information on read consistency.

Oracle Database provides a fully automated mechanism, referred to as **automatic undo management**, for managing undo information and space. In this management mode, for all current sessions, the server automatically manages undo segments and space in the undo tablespace.

Automatic undo management eliminates the complexities of managing rollback segment space. In addition, the system automatically tunes itself to provide the best possible retention of undo information to satisfy long-running queries that may require this undo information. Automatic undo management is the default for new installations of Oracle Database. The installation process automatically creates an undo tablespace.

Oracle Database contains an Undo Advisor that provides advice on and helps automate the establishment of your undo environment.

This section includes the following topics:

- Manual Undo Management

- Undo Quota

- Automatic Undo Retention

> **See Also:** *Oracle Database 2 Day DBA* for information on the Undo Advisor and on how to use advisors and see *Oracle Database Administrator's Guide* for more information on using automatic undo management

Manual Undo Management

A database system can also run in manual undo management mode. In manual undo management mode, undo space is managed through rollback segments, and no undo tablespace is used.

Earlier releases of Oracle Database defaulted to manual undo management mode. To change to automatic undo management, it was necessary to first create an undo tablespace and then change an initialization parameter. If your Oracle Database is release 9*i* or later and you want to change to automatic undo management, see *Oracle Database Upgrade Guide* for instructions.

> **Note:** Space management for rollback segments is complex. Oracle strongly recommends using automatic undo management.

Undo Quota

In automatic undo management mode, the system controls exclusively the assignment of transactions to undo segments, and controls space allocation for undo segments. An ill-behaved transaction can potentially consume much of the undo space, thus paralyzing the entire system. The Resource Manager directive `UNDO_POOL` is a more explicit way to control large transactions. This lets database administrators group users into consumer groups, with each group assigned a maximum undo space limit. When the total undo space consumed by a group exceeds the limit, its users cannot make further updates until undo space is freed up by other member transactions ending.

The default value of `UNDO_POOL` is `UNLIMITED`, where users are allowed to consume as much undo space as the undo tablespace has. Database administrators can limit a particular user by using the `UNDO_POOL` directive.

Automatic Undo Retention

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 undo information can be retained provided that the undo tablespace has space available for new transactions. When available space in the tablespace becomes short, the database begins to overwrite old undo information for transactions that have been committed.

Oracle Database automatically tunes the system to provide the best possible undo retention for the current undo tablespace. The database collects usage statistics and tunes the undo retention period based on these statistics and the undo tablespace size. If the undo tablespace is configured with the AUTOEXTEND option, with maximum size not specified, undo retention tuning is slightly different. In this case, the database tunes the undo retention period to be slightly longer than the longest-running query, if space allows.

> **See Also:** *Oracle Database Administrator's Guide* for more details on automatic tuning of undo retention

3

Tablespaces, Datafiles, and Control Files

This chapter describes tablespaces, the primary logical database structures of any Oracle database, and the physical datafiles that correspond to each tablespace.

This chapter contains the following topics:

- Introduction to Tablespaces, Datafiles, and Control Files
- Overview of Tablespaces
- Overview of Datafiles
- Overview of Control Files

Introduction to Tablespaces, Datafiles, and Control Files

Oracle Database stores data logically in **tablespaces** and physically in **datafiles** associated with the corresponding tablespace. Figure 3–1 illustrates this relationship.

Figure 3–1   Datafiles and Tablespaces
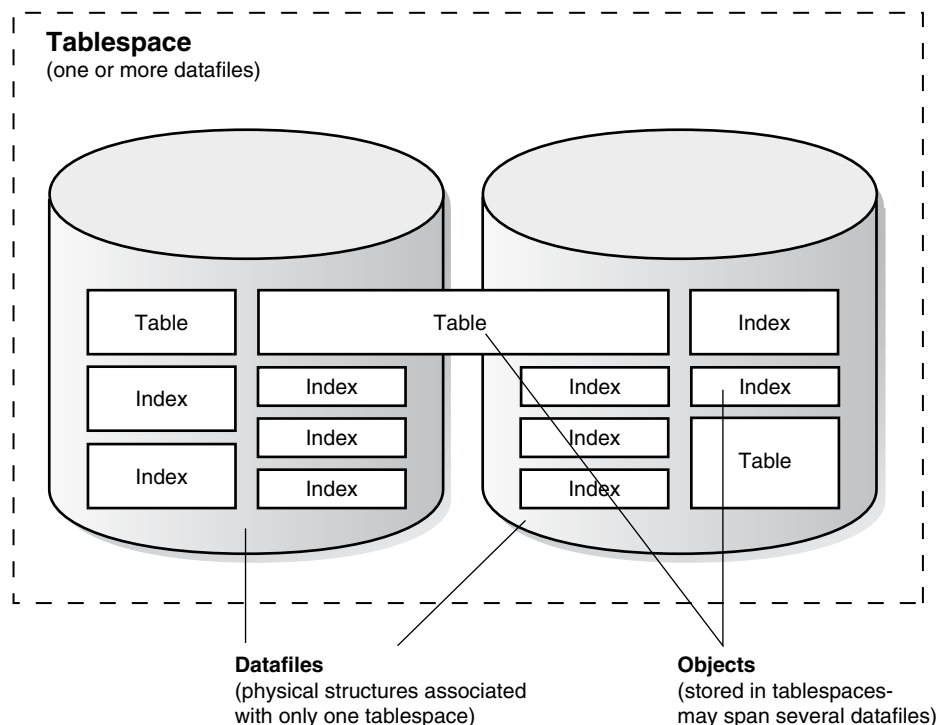
Databases, tablespaces, and datafiles are closely related, but they have important differences:

- An Oracle database consists of one or more logical storage units called tablespaces, which collectively store all of the database's data.

- Each tablespace in an Oracle database consists of one or more files called datafiles, which are physical structures that conform to the operating system in which Oracle Database is running.

- A database's data is collectively stored in the datafiles that constitute each tablespace of the database. For example, the simplest Oracle database would have one tablespace and one datafile. Another database can have three tablespaces, each consisting of two datafiles (for a total of six datafiles).

This section includes the following topics:

- Oracle-Managed Files
- Allocate More Space for a Database

Oracle-Managed Files

Oracle-managed files eliminate the need for you, the DBA, to directly manage the operating system files comprising an Oracle database. You specify operations in terms of database objects rather than filenames. Oracle 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

Through initialization parameters, you specify the file system directory to be used for a particular type of file. Oracle Database then ensures that a unique file, an Oracle-managed file, is created and deleted when no longer needed.

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
> - "Automatic Storage Management" on page 14-22

Allocate More Space for a Database

The size of a tablespace is the size of the datafiles that constitute the tablespace. The size of a database is the collective size of the tablespaces that constitute the database.

You can enlarge a database in three ways:

- Add a datafile to a tablespace
- Add a new tablespace
- Increase the size of a datafile

When you add another datafile to an existing tablespace, you increase the amount of disk space allocated for the corresponding tablespace. Figure 3–2 illustrates this kind of space increase.

Figure 3–2   Enlarging a Database by Adding a Datafile to a Tablespace
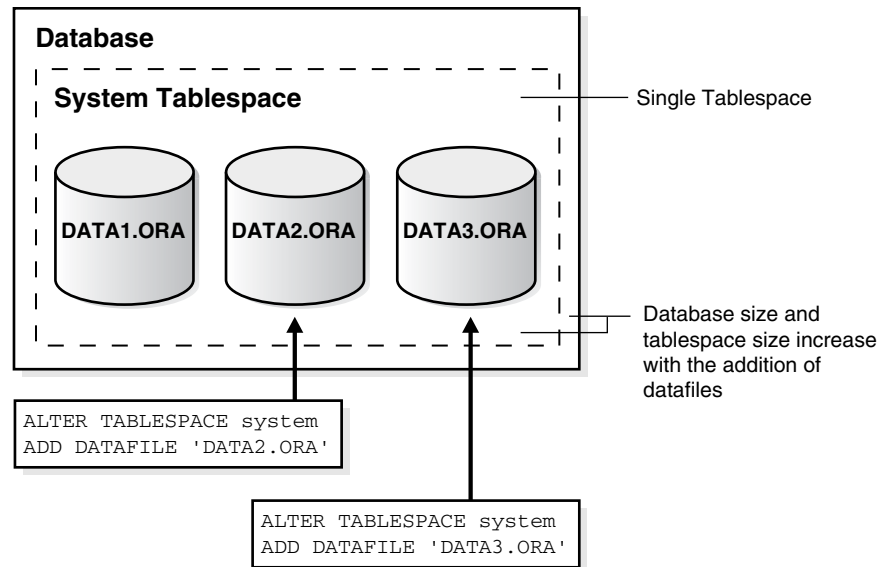
Alternatively, you can create a new tablespace (which contains at least one additional datafile) to increase the size of a database. Figure 3–3 illustrates this.

Figure 3–3   Enlarging a Database by Adding a New Tablespace
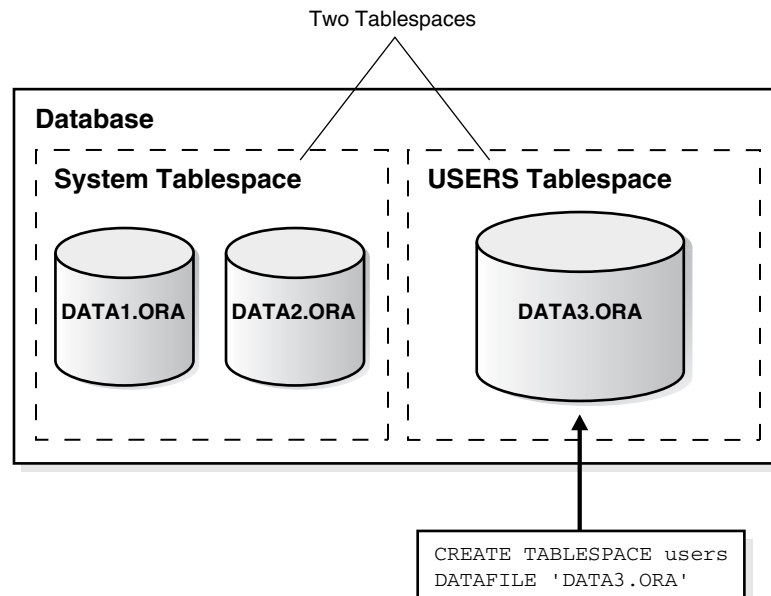
The third option for enlarging a database is to change a datafile's size or let datafiles in existing tablespaces grow dynamically as more space is needed. You accomplish this by altering existing files or by adding files with dynamic extension properties. Figure 3–4 illustrates this.

Figure 3–4 Enlarging a Database by Dynamically Sizing Datafiles
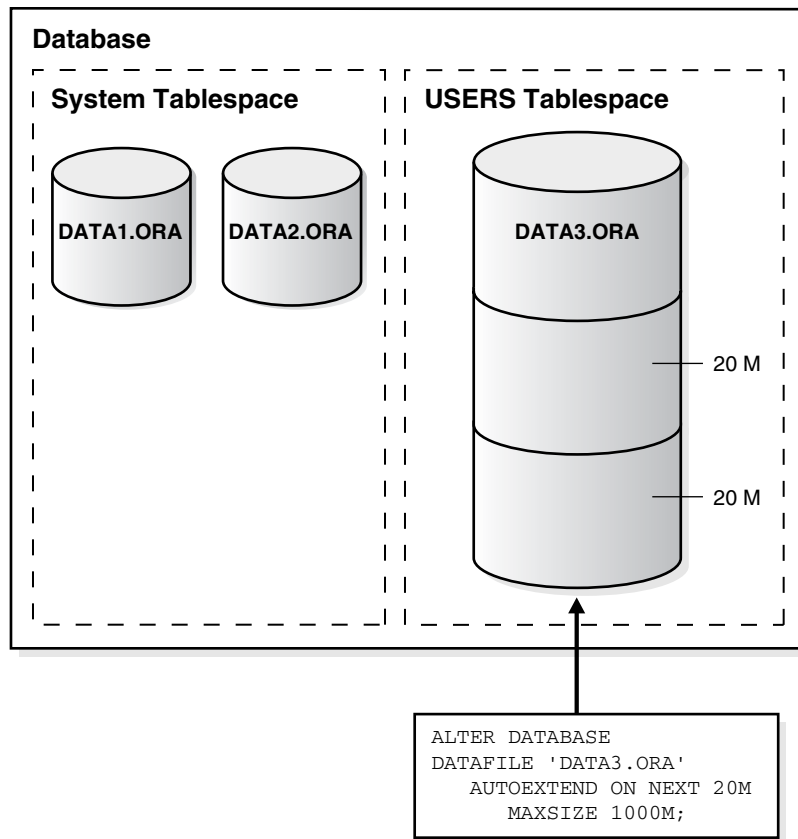
See Also: *Oracle Database Administrator's Guide* for more information
about increasing the amount of space in your database

Overview of Tablespaces

A database is divided into one or more logical storage units called tablespaces.
Tablespaces are divided into logical units of storage called **segments**, which are further
divided into **extents**. Extents are a collection of contiguous blocks.

This section includes the following topics about tablespaces:

- Bigfile Tablespaces

- The SYSTEM Tablespace

- The SYSAUX Tablespace

- Undo Tablespaces

- Default Temporary Tablespace

- Using Multiple Tablespaces

- Managing Space in Tablespaces

- Multiple Block Sizes

- Online and Offline Tablespaces

- Read-Only Tablespaces

- Temporary Tablespaces for Sort Operations

■ Transport of Tablespaces Between Databases

> **See Also:**
>
> ■ Chapter 2, "Data Blocks, Extents, and Segments" for more information about segments and extents
>
> ■ *Oracle Database Administrator's Guide* for detailed information on creating and configuring tablespaces

Bigfile Tablespaces

Oracle Database lets you create **bigfile tablespaces**. This allows Oracle Database to contain tablespaces made up of single large files rather than numerous smaller ones. This lets Oracle Database utilize the ability of 64-bit systems to create and manage ultralarge files. The consequence of this is that Oracle Database can now scale up to 8 exabytes in size.

With Oracle-managed files, bigfile tablespaces make datafiles completely transparent for users. In other words, you can perform operations on tablespaces, rather than the underlying datafile. Bigfile tablespaces make the tablespace the main unit of the disk space administration, backup and recovery, and so on. Bigfile tablespaces also simplify datafile management with Oracle-managed files and Automatic Storage Management by eliminating the need for adding new datafiles and dealing with multiple files.

The system default is to create a smallfile tablespace, which is the traditional type of Oracle Database tablespace. The SYSTEM and SYSAUX tablespace types are always created using the system default type.

Bigfile tablespaces are supported only for locally managed tablespaces with automatic segment-space management. There are two exceptions: locally managed undo and temporary tablespaces can be bigfile tablespaces, even though their segments are manually managed.

An Oracle database can contain both bigfile and smallfile tablespaces. Tablespaces of different types are indistinguishable in terms of execution of SQL statements that do not explicitly refer to datafiles.

You can create a group of temporary tablespaces that let a user consume temporary space from multiple tablespaces. A tablespace group can also be specified as the default temporary tablespace for the database. This is useful with bigfile tablespaces, where you could need a lot of temporary tablespace for sorts.

This section includes the following topics:

■ Benefits of Bigfile Tablespaces

■ Considerations with Bigfile Tablespaces

Benefits of Bigfile Tablespaces

■ Bigfile tablespaces can significantly increase the storage capacity of an Oracle database. Smallfile tablespaces can contain up to 1024 files, but bigfile tablespaces contain only one file that can be 1024 times larger than a smallfile tablespace. The total tablespace capacity is the same for smallfile tablespaces and bigfile tablespaces. However, because there is limit of 64K datafiles for each database, a database can contain 1024 times more bigfile tablespaces than smallfile tablespaces, so bigfile tablespaces increase the total database capacity by 3 orders of magnitude. In other words, 8 exabytes is the maximum size of the Oracle database when bigfile tablespaces are used with the maximum block size (32 k).

- Bigfile tablespaces simplify management of datafiles in ultra large databases by reducing the number of datafiles needed. You can also adjust parameters to reduce the SGA space required for datafile information and the size of the control file.

- They simplify database management by providing datafile transparency.

Considerations with Bigfile Tablespaces

- Bigfile tablespaces are intended to be used with Automatic Storage Management or other logical volume managers that support dynamically extensible logical volumes and striping or RAID.

- Avoid creating bigfile tablespaces on a system that does not support striping because of negative implications for parallel execution and RMAN backup parallelization.

- Avoid using bigfile tablespaces if there could possibly be no free space available on a disk group, and the only way to extend a tablespace is to add a new datafile on a different disk group.

- Using bigfile tablespaces on platforms that do not support large file sizes is not recommended and can limit tablespace capacity. Refer to your operating system specific documentation for information about maximum supported file sizes.

- Performance of database opens, checkpoints, and DBWR processes should improve if data is stored in bigfile tablespaces instead of traditional tablespaces. However, increasing the datafile size might increase time to restore a corrupted file or create a new datafile.

> **See Also:** *Oracle Database Administrator's Guide* for details on creating, altering, and administering bigfile tablespaces

The SYSTEM Tablespace

Every Oracle database contains a tablespace named SYSTEM, which Oracle Database creates automatically when the database is created. The SYSTEM tablespace is always online when the database is open.

To take advantage of the benefits of locally managed tablespaces, you can create a locally managed SYSTEM tablespace, or you can migrate an existing dictionary managed SYSTEM tablespace to a locally managed format.

In a database with a locally managed SYSTEM tablespace, dictionary managed tablespaces cannot be created. It is possible to plug in a dictionary managed tablespace using the transportable feature, but it cannot be made writable.

This section includes the following topics:

- The Data Dictionary

- PL/SQL Program Units Description

> **Note:** If a tablespace is locally managed, then it cannot be reverted back to being dictionary managed.

The Data Dictionary

The SYSTEM tablespace always contains the data dictionary tables for the entire database. The data dictionary tables are stored in datafile 1.

PL/SQL Program Units Description

All data stored on behalf of stored PL/SQL program units (that is, procedures, functions, packages, and triggers) resides in the `SYSTEM` tablespace. If the database contains many of these program units, then the database administrator must provide the space the units need in the `SYSTEM` tablespace.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about creating or migrating to a locally managed `SYSTEM` tablespace
>
> - "Online and Offline Tablespaces" on page 3-11 for information about the permanent online condition of the `SYSTEM` tablespace
>
> - Chapter 24, "SQL" and Chapter 22, "Triggers" for information about the space requirements of PL/SQL program units

The SYSAUX Tablespace

The `SYSAUX` tablespace is an auxiliary tablespace to the `SYSTEM` tablespace. Many database components use the `SYSAUX` tablespace as their default location to store data. Therefore, the `SYSAUX` tablespace is always created during database creation or database upgrade.

The `SYSAUX` tablespace provides a centralized location for database metadata that does not reside in the `SYSTEM` tablespace. It reduces the number of tablespaces created by default, both in the seed database and in user-defined databases.

During normal database operation, the Oracle database server does not allow the `SYSAUX` tablespace to be dropped or renamed. Transportable tablespaces for `SYSAUX` is not supported.

> **Note:** If the `SYSAUX` tablespace is unavailable, such as due to a media failure, then some database features might fail.

Undo Tablespaces

Undo tablespaces are special tablespaces used solely for storing undo information. You cannot create any other segment types (for example, tables or indexes) in undo tablespaces. Undo tablespaces are used only when the database is in automatic undo management mode (the default). A database can contain more than one undo tablespace, but only one can be in use at any time. Undo data is managed within an undo tablespace using undo segments that are automatically created and maintained by the database.

When the first DML operation is run within a transaction, the transaction is bound (assigned) to an undo segment (and therefore to a transaction table) in the current undo tablespace. In rare circumstances, if the instance does not have a designated undo tablespace, the transaction binds to the system undo segment.

Each undo tablespace is composed of a set of datafiles and is locally managed. Like other types of tablespaces, undo blocks are grouped in extents and the status of each extent is represented in the bitmap. At any point in time, an extent is either allocated to (and used by) a transaction table, or it is free.

You can create a bigfile undo tablespace.

See Also:

- *Oracle Database Administrator's Guide* for information on managing the undo tablespace

- "Bigfile Tablespaces" on page 3-5

Creation of Undo Tablespaces

An undo tablespace is automatically created with each new installation of Oracle Database. Earlier versions of Oracle Database may not include an undo tablespace and may instead use rollback segments. This is known as manual undo management mode. When upgrading to Oracle Database 11*g* you can migrate to automatic undo management by creating an undo tablespace and enabling automatic undo management mode. See *Oracle Database Upgrade Guide* for details.

Default Temporary Tablespace

When the SYSTEM tablespace is locally managed, you must define at least one default temporary tablespace when creating a database. A locally managed SYSTEM tablespace cannot be used for default temporary storage.

If SYSTEM is dictionary managed and if you do not define a default temporary tablespace when creating the database, then SYSTEM is still used for default temporary storage. However, you will receive a warning in ALERT.LOG saying that a default temporary tablespace is recommended and will be necessary in future releases.

How to Specify a Default Temporary Tablespace

Specify default temporary tablespaces when you create a database, using the DEFAULT TEMPORARY TABLESPACE extension to the CREATE DATABASE statement.

If you drop all default temporary tablespaces, then the SYSTEM tablespace is used as the default temporary tablespace.

You can create bigfile temporary tablespaces. A bigfile temporary tablespaces uses tempfiles instead of datafiles.

> **Note:** You cannot make a default temporary tablespace permanent or take it offline.

See Also:

- *Oracle Database SQL Language Reference* for information about defining and altering default temporary tablespaces

- "Bigfile Tablespaces" on page 3-5

Using Multiple Tablespaces

A very small database may need only the SYSTEM tablespace; however, Oracle recommends that you create at least one additional tablespace to store user data separate from data dictionary information. This gives you more flexibility in various database administration operations and reduces contention among dictionary objects and schema objects for the same datafiles.

You can use multiple tablespaces to perform the following tasks:

- Control disk space allocation for database data

- Assign specific space quotas for database users

- Control availability of data by taking individual tablespaces online or offline

- Perform partial database backup or recovery operations

- Allocate data storage across devices to improve performance

A database administrator can use tablespaces to do the following actions:

- Create new tablespaces

- Add datafiles to tablespaces

- Set and alter default segment storage settings for segments created in a tablespace

- Make a tablespace read only or read/write

- Make a tablespace temporary or permanent

- Rename tablespaces

- Drop tablespaces

Managing Space in Tablespaces

Tablespaces allocate space in extents. Tablespaces can use two different methods to keep track of their free and used space:

- **Locally managed tablespaces**: Extent management by the tablespace

- **Dictionary managed tablespaces:** Extent management by the data dictionary

When you create a tablespace, you choose one of these methods of space management. Later, you can change the management method with the DBMS_SPACE_ADMIN PL/SQL package.

This section includes the following topics:

- Locally Managed Tablespaces

- Segment Space Management in Locally Managed Tablespaces

- Dictionary Managed Tablespaces

> **Note:** If you do not specify extent management when you create a tablespace, then the default is locally managed.

See Also: "Overview of Extents" on page 2-10

Locally Managed Tablespaces

A tablespace that manages its own extents maintains a bitmap in each datafile to keep track of the free or used status of blocks in that datafile. Each bit in the bitmap corresponds to a block or a group of blocks. When an extent is allocated or freed for reuse, Oracle Database changes the bitmap values to show the new status of the blocks. These changes do not generate rollback information because they do not update tables in the data dictionary (except for special cases such as tablespace quota information).

Locally managed tablespaces have the following advantages over dictionary managed tablespaces:

- Local management of extents automatically tracks adjacent free space, eliminating the need to coalesce free extents.

- Local management of extents avoids recursive space management operations. Such recursive operations can occur in dictionary managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in a data dictionary table or rollback segment.

The sizes of extents that are managed locally can be determined automatically by the system. Alternatively, all extents can have the same size in a locally managed tablespace and override object storage options.

The LOCAL clause of the CREATE TABLESPACE or CREATE TEMPORARY TABLESPACE statement is specified to create locally managed permanent or temporary tablespaces, respectively.

Segment Space Management in Locally Managed Tablespaces

When you create a locally managed tablespace using the CREATE TABLESPACE statement, the SEGMENT SPACE MANAGEMENT clause lets you specify how free and used space within a segment is to be managed. Your choices are:

- AUTO

  This keyword tells Oracle Database that you want to use bitmaps to manage the free space within segments. A bitmap, in this case, is a map that describes the status of each data block within a segment with respect to the amount of space in the block available for inserting rows. As more or less space becomes available in a data block, its new state is reflected in the bitmap. Bitmaps enable Oracle Database to manage free space more automatically; thus, this form of space management is called automatic segment-space management.

  Locally managed tablespaces using automatic segment-space management can be created as smallfile (traditional) or bigfile tablespaces. AUTO is the default.

- MANUAL

  This keyword tells Oracle Database that you want to use free lists for managing free space within segments. Free lists are lists of data blocks that have space available for inserting rows.

  > **See Also:**
  >
  > - *Oracle Database SQL Language Reference* for syntax
  >
  > - *Oracle Database Administrator's Guide* for more information about automatic segment space management
  >
  > - "Determine the Number and Size of Extents" on page 2-10
  >
  > - "Temporary Tablespaces for Sort Operations" on page 3-13 for more information about temporary tablespaces

Dictionary Managed Tablespaces

If you created your database with an earlier version of Oracle Database, then you could be using dictionary managed tablespaces. For a tablespace that uses the data dictionary to manage its extents, Oracle Database updates the appropriate tables in the data dictionary whenever an extent is allocated or freed for reuse. Oracle Database also stores rollback information about each update of the dictionary tables. Because dictionary tables and rollback segments are part of the database, the space that they occupy is subject to the same space management operations as all other data.

Multiple Block Sizes

Oracle Database supports multiple block sizes in a database. The **standard block size** is used for the SYSTEM tablespace. This is set when the database is created and can be any valid size. You specify the standard block size by setting the initialization parameter DB_BLOCK_SIZE. Legitimate values are from 2K to 32K.

In the initialization parameter file or server parameter file, you can configure subcaches within the buffer cache for each of these block sizes. Subcaches can also be configured while an instance is running. You can create tablespaces having any of these block sizes. The standard block size is used for the system tablespace and most other tablespaces.

> **Note:** All partitions of a partitioned object must reside in tablespaces of a single block size.

Multiple block sizes are useful primarily when transporting a tablespace from an OLTP database to an enterprise data warehouse. This facilitates transport between databases of different block sizes.

See Also:

- "Transport of Tablespaces Between Databases" on page 3-14
- *Oracle Database Data Warehousing Guide* for information about transporting tablespaces in data warehousing environments

Online and Offline Tablespaces

A database administrator can bring any tablespace other than the SYSTEM tablespace **online** (accessible) or **offline** (not accessible) whenever the database is open. The SYSTEM tablespace is always online when the database is open because the data dictionary must always be available to Oracle Database.

A tablespace is usually online so that the data contained within it is available to database users. However, the database administrator can take a tablespace offline for maintenance or backup and recovery purposes.

This section includes the following topics:

- Bringing Tablespaces Offline
- Use of Tablespaces for Special Procedures

Bringing Tablespaces Offline

When a tablespace goes offline, Oracle Database does not permit any subsequent SQL statements to reference objects contained in that tablespace. Active transactions with completed statements that refer to data in that tablespace are not affected at the transaction level. Oracle Database saves rollback data corresponding to those completed statements in a deferred rollback segment in the SYSTEM tablespace. When the tablespace is brought back online, Oracle Database applies the rollback data to the tablespace, if needed.

When a tablespace goes offline or comes back online, this is recorded in the data dictionary in the SYSTEM tablespace. If a tablespace is offline when you shut down a database, the tablespace remains offline when the database is subsequently mounted and reopened.

You can bring a tablespace online only in the database in which it was created because the necessary data dictionary information is maintained in the SYSTEM tablespace of that database. An offline tablespace cannot be read or edited by any utility other than Oracle Database. Thus, offline tablespaces cannot be transposed to other databases.

Oracle Database automatically switches a tablespace from online to offline when certain errors are encountered. For example, Oracle Database switches a tablespace from online to offline when the database writer process, DBW*n*, fails in several attempts to write to a datafile of the tablespace. Users trying to access tables in the offline tablespace receive an error. If the problem that causes this disk I/O to fail is media failure, you must recover the tablespace after you correct the problem.

> **See Also:**
>
> - "Temporary Tablespaces for Sort Operations" on page 3-13 for more information about transferring online tablespaces between databases
>
> - *Oracle Database Utilities* for more information about tools for data transfer

Use of Tablespaces for Special Procedures

If you create multiple tablespaces to separate different types of data, you take specific tablespaces offline for various procedures. Other tablespaces remain online, and the information in them is still available for use. However, special circumstances can occur when tablespaces are taken offline. For example, if two tablespaces are used to separate table data from index data, the following is true:

- If the tablespace containing the indexes is offline, then queries can still access table data because queries do not require an index to access the table data.

- If the tablespace containing the tables is offline, then the table data in the database is not accessible because the tables are required to access the data.

If Oracle Database has enough information in the online tablespaces to run a statement, it does so. If it needs data in an offline tablespace, then it causes the statement to fail.

Read-Only Tablespaces

The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Oracle Database never updates the files of a read-only tablespace, and therefore the files can reside on read-only media such as CD-ROMs or WORM drives.

> **Note:** Because you can only bring a tablespace online in the database in which it was created, read-only tablespaces are not meant to satisfy archiving requirements.

Read-only tablespaces cannot be modified. To update a read-only tablespace, first make the tablespace read/write. After updating the tablespace, you can then reset it to be read only.

Because read-only tablespaces cannot be modified, and as long as they have not been made read/write at any point, they do not need repeated backup. Also, if you need to recover your database, you do not need to recover any read-only tablespaces, because they could not have been modified.

See Also:

- *Oracle Database Administrator's Guide* for information about changing a tablespace to read only or read/write mode

- *Oracle Database SQL Language Reference* for more information about the `ALTER TABLESPACE` statement

- *Oracle Database Backup and Recovery User's Guide* for more information about recovery

Temporary Tablespaces for Sort Operations

You can manage space for sort operations more efficiently by designating one or more temporary tablespaces exclusively for sorts. Doing so effectively eliminates serialization of space management operations involved in the allocation and deallocation of sort space. A single SQL operation can use more than one temporary tablespace for sorting. For example, you can create indexes on very large tables, and the sort operation during index creation can be distributed across multiple tablespaces.

All operations that use sorts, including joins, index builds, ordering, computing aggregates (`GROUP BY`), and collecting optimizer statistics, benefit from temporary tablespaces. The performance gains are significant with Oracle Real Application Clusters.

This section includes the following topics:

- Sort Segments

- Creation of Temporary Tablespaces

Sort Segments

One or more temporary tablespaces can be used only for sort segments. A temporary tablespace is not the same as a tablespace that a user designates for temporary segments, which can be any tablespace available to the user. No permanent schema objects can reside in a temporary tablespace.

Sort segments are used when a segment is shared by multiple sort operations. One sort segment exists for every instance that performs a sort operation in a given tablespace.

Temporary tablespaces provide performance improvements when you have multiple sorts that are too large to fit into memory. The sort segment of a given temporary tablespace is created at the time of the first sort operation. The sort segment expands by allocating extents until the segment size is equal to or greater than the total storage demands of all of the active sorts running on that instance.

> **See Also:** Chapter 2, "Data Blocks, Extents, and Segments" for more information about segments

Creation of Temporary Tablespaces

Create temporary tablespaces by using the `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE` statement.

See Also:

-

-

- *Oracle Database SQL Language Reference* for syntax

- *Oracle Database Performance Tuning Guide* for information about setting up temporary tablespaces for sorts and hash joins

Transport of Tablespaces Between Databases

A **transportable tablespace** lets you move a subset of an Oracle database from one Oracle database to another, even across different platforms. You can clone a tablespace and plug it into another database, copying the tablespace between databases, or you can unplug a tablespace from one Oracle database and plug it into another Oracle database, moving the tablespace between databases.

Moving data by transporting tablespaces can be orders of magnitude faster than either export/import or unload/load of the same data, because transporting a tablespace involves only copying datafiles and integrating the tablespace metadata. When you transport tablespaces you can also move index data, so you do not have to rebuild the indexes after importing or loading the table data.

You can transport tablespaces across platforms. (Many, but not all, platforms are supported for cross-platform tablespace transport.) This can be used for the following:

- Provide an easier and more efficient means for content providers to publish structured data and distribute it to customers running Oracle Database on a different platform

- 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 across a heterogeneous cluster

- Allow a database to be migrated from one platform to another

This section includes the following topics:

- Tablespace Repository

- How to Move or Copy a Tablespace to Another Database

Tablespace Repository

A tablespace repository is a collection of tablespace sets. Tablespace repositories are built on file group repositories, but tablespace repositories only contain the files required to move or copy tablespaces between databases. Different tablespace sets may be stored in a tablespace repository, and different versions of a particular tablespace set also may be stored. A version of a tablespace set in a tablespace repository consists of the following files:

- The Data Pump export dump file for the tablespace set

- The Data Pump log file for the export

- The datafiles that comprise the tablespace set

See Also: *Oracle Streams Concepts and Administration*

How to Move or Copy a Tablespace to Another Database

To move or copy a set of tablespaces, you must make the tablespaces read only, copy the datafiles of these tablespaces, and use export/import to move the database information (metadata) stored in the data dictionary. Both the datafiles and the metadata export file must be copied to the target database. The transport of these files can be done using any facility for copying flat files, such as the operating system copying facility, ftp, or publishing on CDs.

After copying the datafiles and importing the metadata, you can optionally put the tablespaces in read/write mode.

The first time a tablespace's datafiles are opened under Oracle Database with the COMPATIBLE initialization parameter set to 10 or higher, each file identifies the platform to which it belongs. These files have identical on disk formats for file header blocks, which are used for file identification and verification. Read only and offline files get the compatibility advanced after they are made read/write or are brought online. This implies that tablespaces that are read only prior to Oracle Database 10*g* must be made read/write at least once before they can use the cross platform transportable feature.

> **Note:** In a database with a locally managed SYSTEM tablespace, dictionary tablespaces cannot be created. It is possible to plug in a dictionary managed tablespace using the transportable feature, but it cannot be made writable.

See Also:

- *Oracle Database Administrator's Guide* for details about how to move or copy tablespaces to another database, including details about transporting tablespaces across platforms

- *Oracle Database Utilities* for import/export information

- *Oracle Database PL/SQL Packages and Types Reference* for information on the DBMS_FILE_TRANSFER package

- *Oracle Streams Concepts and Administration* for more information on ways to copy or transport files

Overview of Datafiles

A tablespace in an Oracle database consists of one or more physical **datafiles**. A datafile can be associated with only one tablespace and only one database.

Oracle Database creates a datafile for a tablespace by allocating the specified amount of disk space plus the overhead required for the file header. When a datafile is created, the operating system under which Oracle Database runs is responsible for clearing old information and authorizations from a file before allocating it to Oracle Database. If the file is large, this process can take a significant amount of time. The first tablespace in any database is always the SYSTEM tablespace, so Oracle Database automatically allocates the first datafiles of any database for the SYSTEM tablespace during database creation.

This section includes the following topics:

- Datafile Contents
- Size of Datafiles

- Offline Datafiles

- Temporary Datafiles

> **See Also:** Your Oracle Database operating system-specific documentation for information about the amount of space required for the file header of datafiles on your operating system

Datafile Contents

When a datafile is first created, the allocated disk space is formatted but does not contain any user data. However, Oracle Database reserves the space to hold the data for future segments of the associated tablespace—it is used exclusively by Oracle Database. As the data grows in a tablespace, Oracle Database uses the free space in the associated datafiles to allocate extents for the segment.

The data associated with schema objects in a tablespace is physically stored in one or more of the datafiles that constitute the tablespace. Note that a schema object does not correspond to a specific datafile; rather, a datafile is a repository for the data of any schema object within a specific tablespace. Oracle Database allocates space for the data associated with a schema object in one or more datafiles of a tablespace. Therefore, a schema object can span one or more datafiles. Unless table **striping** is used (where data is spread across more than one disk), the database administrator and end users cannot control which datafile stores a schema object.

> **See Also:** Chapter 2, "Data Blocks, Extents, and Segments" for more information about use of space

Size of Datafiles

You can alter the size of a datafile after its creation or you can specify that a datafile should dynamically grow as schema objects in the tablespace grow. This functionality enables you to have fewer datafiles for each tablespace and can simplify administration of datafiles.

> **Note:** You need sufficient space on the operating system for expansion.

> **See Also:** *Oracle Database Administrator's Guide* for more information about resizing datafiles

Offline Datafiles

You can take tablespaces offline or bring them online at any time, except for the SYSTEM tablespace. All of the datafiles of a tablespace are taken offline or brought online as a unit when you take the tablespace offline or bring it online, respectively.

You can take individual datafiles offline. However, this is usually done only during some database recovery procedures.

Temporary Datafiles

Locally managed temporary tablespaces have temporary datafiles (**tempfile**s), which are similar to ordinary datafiles, with the following exceptions:

- Tempfiles are always set to NOLOGGING mode.

- You cannot make a tempfile read only.

- You cannot create a tempfile with the ALTER DATABASE statement.

- Media recovery does not recognize tempfiles:

    - BACKUP CONTROLFILE does not generate any information for tempfiles.

    - CREATE CONTROLFILE cannot specify any information about tempfiles.

- When you create or resize tempfiles, they are not always guaranteed allocation of disk space for the file size specified. On certain file systems (for example, UNIX) disk blocks are allocated not at file creation or resizing, but before the blocks are accessed.

    > **Caution:** This enables fast tempfile creation and resizing; however, the disk could run of space later when the tempfiles are accessed.

- Tempfile information is shown in the dictionary view DBA_TEMP_FILES and the dynamic performance view V$TEMPFILE, but not in DBA_DATA_FILES or the V$DATAFILE view.

    > **See Also:** "Managing Space in Tablespaces" on page 3-9 for more information about locally managed tablespaces

Overview of Control Files

The database control file is a small binary file necessary for the database to start and operate successfully. A control file is updated continuously by Oracle Database during database use, so it must be available for writing whenever the database is open. If for some reason the control file is not accessible, then the database cannot function properly.

Each control file is associated with only one Oracle database.

This section includes the following topics:

- Control File Contents

- Multiplexed Control Files

Control File Contents

A control file contains information about the associated database that is required for access by an instance, both at startup and during normal operation. Control file information can be modified only by Oracle Database; no database administrator or user can edit a control file.

Among other things, a control file contains information such as:

- The database name

- The timestamp of database creation

- The names and locations of associated datafiles and redo log files

- Tablespace information

- Datafile offline ranges

- The log history

- Archived log information
- Backup set and backup piece information
- Backup datafile and redo log information
- Datafile copy information
- The current log sequence number
- Checkpoint information

The database name and timestamp originate at database creation. The database name is taken from either the name specified by the DB_NAME initialization parameter or the name used in the CREATE DATABASE statement.

Each time that a datafile or a redo log file is added to, renamed in, or dropped from the database, the control file is updated to reflect this physical structure change. These changes are recorded so that:

- Oracle Database can identify the datafiles and redo log files to open during database startup
- Oracle Database can identify files that are required or available in case database recovery is necessary

Therefore, if you make a change to the physical structure of your database (using ALTER DATABASE statements), then you should immediately make a backup of your control file.

Control files also record information about checkpoints. Every three seconds, the checkpoint process (CKPT) records information in the control file about the checkpoint position in the redo log. This information is used during database recovery to tell Oracle Database that all redo entries recorded before this point in the redo log group are not necessary for database recovery; they were already written to the datafiles.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for information about backing up a database's control file

Multiplexed Control Files

As with redo log files, Oracle Database enables multiple, identical control files to be open concurrently and written for the same database. By storing multiple control files for a single database on different disks, you can safeguard against a single point of failure with respect to control files. If a single disk that contained a control file crashes, then the current instance fails when Oracle Database attempts to access the damaged control file. However, when other copies of the current control file are available on different disks, an instance can be restarted without the need for database recovery.

If *all* control files of a database are permanently lost during operation, then the instance is aborted and media recovery is required. Media recovery is not straightforward if an older backup of a control file must be used because a current copy is not available. It is strongly recommended that you adhere to the following:

- Use multiplexed control files with each database
- Store each copy on a different physical disk
- Use operating system mirroring
- Monitor backups

4

Transaction Management

This chapter defines a transaction and describes how you can manage your work using transactions.

This chapter contains the following topics:

- Introduction to Transactions
- Overview of Transaction Management
- Overview of Autonomous Transactions

Introduction to Transactions

A **transaction** is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all **committed** (applied to the database) or all **rolled back** (undone from the database).

A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.

To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operations:

- Decrement the savings account
- Increment the checking account
- Record the transaction in the transaction journal

Oracle Database must allow for two situations. If all three SQL statements can be performed to maintain the accounts in proper balance, the effects of the transaction can be applied to the database. However, if a problem such as insufficient funds, invalid account number, or a hardware failure prevents one or two of the statements in the transaction from completing, the entire transaction must be rolled back so that the balance of all accounts is correct.

Figure 4–1 illustrates the banking transaction example.

Figure 4–1   A Banking Transaction

Transaction Begins

```
UPDATE savings_accounts
     SET balance = balance - 500
     WHERE account = 3209;
```
Decrement Savings Account

```
UPDATE checking_accounts
     SET balance = balance + 500
     WHERE account = 3208;
```
Increment Checking Account

```
INSERT INTO journal VALUES
     (journal_seq.NEXTVAL, '1B'
     3209, 3208, 500);
```
Record in Transaction Journal

```
COMMIT WORK;
```
End Transaction
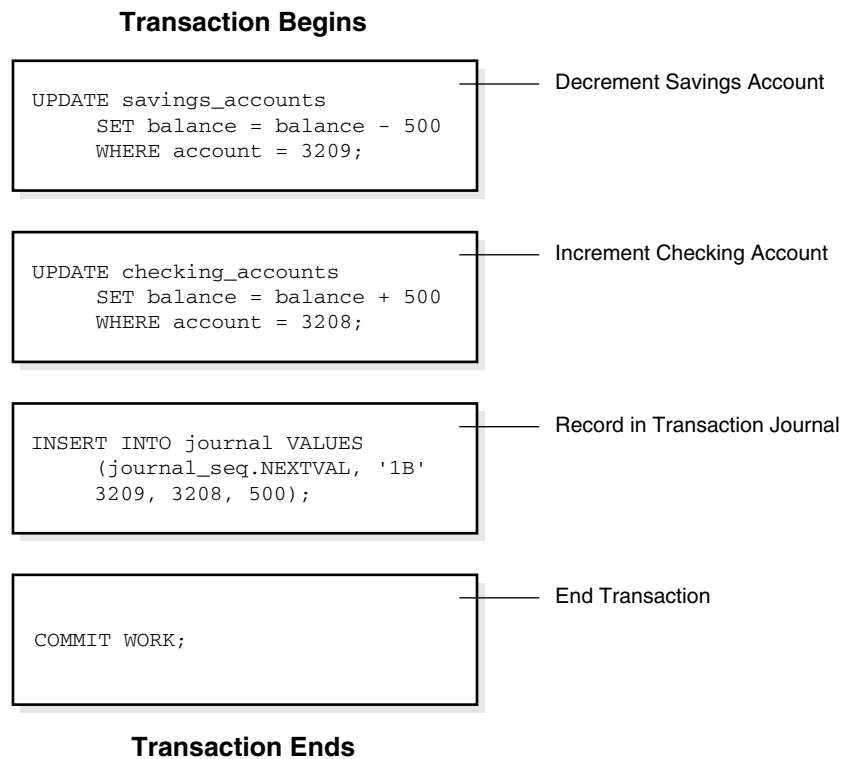
Transaction Ends

This section includes the following topics:

- Statement Execution and Transaction Control
- Statement-Level Rollback
- Resumable Space Allocation

Statement Execution and Transaction Control

A SQL statement that runs successfully is different from a committed transaction. Executing successfully means that a single statement was:

- Parsed
- Found to be a valid SQL construction
- Run without error as an atomic unit. For example, all rows of a multirow update are changed.

However, until the transaction that contains the statement is committed, the transaction can be rolled back, and all of the changes of the statement can be undone. A statement, rather than a transaction, runs successfully.

Committing means that a user has explicitly or implicitly requested that the changes in the transaction be made permanent. An explicit request occurs when the user issues a COMMIT statement. An implicit request occurs after normal termination of an application or completion of a data definition language (DDL) operation. The changes made by the SQL statement(s) of a transaction become permanent and visible to other users only after that transaction commits. Queries that are issued after the transaction commits will see the committed changes.

You can name a transaction using the SET TRANSACTION ... NAME statement before you start the transaction. This makes it easier to monitor long-running transactions and to resolve in-doubt distributed transactions.

> **See Also:** "Transaction Naming" on page 4-7

Statement-Level Rollback

If at any time during execution a SQL statement causes an error, all effects of the statement are rolled back. The effect of the rollback is as if that statement had never been run. This operation is a **statement-level rollback**.

Errors discovered during SQL statement **execution** cause statement-level rollbacks. An example of such an error is attempting to insert a duplicate value in a primary key. Single SQL statements involved in a **deadlock** (competition for the same data) can also cause a statement-level rollback. Errors discovered during SQL statement **parsing**, such as a syntax error, have not yet been run, so they do not cause a statement-level rollback.

A SQL statement that fails causes the loss only of any work it would have performed itself. *It does not cause the loss of any work that preceded it in the current transaction.* If the statement is a DDL statement, then the implicit commit that immediately preceded it is not undone.

> **Note:** Users cannot directly refer to implicit savepoints in rollback statements.

> **See Also:** "Deadlocks" on page 13-15

Resumable Space Allocation

Oracle Database provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. This enables an administrator to 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.

A statement runs in a resumable mode only when the client explicitly enables resumable semantics for the session using the ALTER SESSION statement.

Resumable space allocation is suspended when one of the following conditions occur:

- Out of space condition
- Maximum extents reached condition
- Space quota exceeded condition

For nonresumable space allocation, these conditions result in errors and the statement is rolled back.

Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held through a statement suspend and resume.

When the error condition disappears (for example, as a result of user intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution.

> **See Also:** *Oracle Database Administrator's Guide* for information about enabling resumable space allocation, what conditions are correctable, and what statements can be made resumable.

Overview of Transaction Management

A transaction in Oracle Database begins when the first executable SQL statement is encountered. An **executable SQL statement** is a SQL statement that generates calls to an instance, including DML and DDL statements.

When a transaction begins, Oracle Database assigns the transaction to an available undo tablespace to record the rollback entries for the new transaction.

A transaction ends when any of the following occurs:

- A user issues a COMMIT or ROLLBACK statement without a SAVEPOINT clause.

- A user runs a DDL statement such as CREATE, DROP, RENAME, or ALTER. If the current transaction contains any DML statements, Oracle Database first commits the transaction, and then runs and commits the DDL statement as a new, single statement transaction.

- A user disconnects from Oracle Database. The current transaction is committed.

- A user process terminates abnormally. The current transaction is rolled back.

After one transaction ends, the next executable SQL statement automatically starts the following transaction.

This section includes the following topics:

- Commit Transactions

- Rollback of Transactions

- Savepoints In Transactions

- Transaction Naming

- The Two-Phase Commit Mechanism

> **Note:** Applications should always explicitly commit or undo transactions before program termination.

Commit Transactions

Committing a transaction means making permanent the changes performed by the SQL statements within the transaction.

Before a transaction that modifies data is committed, the following has occurred:

- Oracle Database has generated undo information. The undo information contains the old data values changed by the SQL statements of the transaction.

- Oracle Database has generated redo log entries in the redo log buffer of the SGA. The redo log record contains the change to the data block and the change to the rollback block. These changes may go to disk before a transaction is committed.

- The changes have been made to the database buffers of the SGA. These changes may go to disk before a transaction is committed.

> **Note:** The data changes for a committed transaction, stored in the database buffers of the SGA, are not necessarily written immediately to the datafiles by the database writer (DBW*n*) background process. This writing takes place when it is most efficient for the database to do so. It can happen before the transaction commits or, alternatively, it can happen some time after the transaction commits.

When a transaction is committed, the following occurs:

1. The internal transaction table for the associated undo tablespace records that the transaction has committed, and the corresponding unique system change number (SCN) of the transaction is assigned and recorded in the table.

2. The log writer process (LGWR) writes redo log entries in the SGA's redo log buffers to the redo log file. It also writes the transaction's SCN to the redo log file. This atomic event constitutes the commit of the transaction.

3. Oracle Database releases locks held on rows and tables.

4. Oracle Database marks the transaction complete.

> **Note:** The default behavior is for LGWR to write redo to the online redo log files synchronously and for transactions to wait for the redo to go to disk before returning a commit to the user. However, for lower transaction commit latency application developers can specify that redo be written asynchronously and that transactions do not need to wait for the redo to be on disk.

See Also:

- *Oracle Database Advanced Application Developer's Guide* for more information on asynchronous commit
- "Overview of Locking Mechanisms" on page 13-2
- "Overview of Oracle Database Processes" on page 9-3 for more information about the background processes LGWR and DBW*n*

Rollback of Transactions

Rolling back means undoing any changes to data that have been performed by SQL statements within an uncommitted transaction. Oracle Database uses undo tablespaces (or rollback segments) to store old values. The redo log contains a record of changes.

Oracle Database lets you roll back an entire uncommitted transaction. Alternatively, you can roll back the trailing portion of an uncommitted transaction to a marker called a savepoint.

All types of rollbacks use the same procedures:

- Statement-level rollback (due to statement or deadlock execution error)
- Rollback to a savepoint
- Rollback of a transaction due to user request
- Rollback of a transaction due to abnormal process termination
- Rollback of all outstanding transactions when an instance terminates abnormally

■ Rollback of incomplete transactions during recovery

In rolling back **an entire transaction**, without referencing any savepoints, the following occurs:

1. Oracle Database undoes all changes made by all the SQL statements in the transaction by using the corresponding undo tablespace.

2. Oracle Database releases all the transaction's locks of data.

3. The transaction ends.

> **See Also:**
>
> ■ "Savepoints In Transactions" on page 4-6
>
> ■ "Overview of Locking Mechanisms" on page 13-2
>
> ■ *Oracle Database Backup and Recovery User's Guide* for information about what happens to committed and uncommitted changes during recovery

Savepoints In Transactions

You can declare intermediate markers called **savepoints** within the context of a transaction. Savepoints divide a long transaction into smaller parts.

Using savepoints, you can arbitrarily mark your work at any point within a long transaction. You then have the option later of rolling back work performed before the current point in the transaction but after a declared savepoint within the transaction. For example, you can use savepoints throughout a long complex series of updates, so if you make an error, you do not need to resubmit every statement.

Savepoints are similarly useful in application programs. If a procedure contains several functions, then you can create a savepoint before each function begins. Then, if a function fails, it is easy to return the data to its state before the function began and re-run the function with revised parameters or perform a recovery action.

After a rollback to a savepoint, Oracle Database releases the data locks obtained by rolled back statements. Other transactions that were waiting for the previously locked resources can proceed. Other transactions that want to update previously locked rows can do so.

When a transaction is rolled back to a savepoint, the following occurs:

1. Oracle Database rolls back only the statements run after the savepoint.

2. Oracle Database preserves the specified savepoint, but all savepoints that were established after the specified one are lost.

3. Oracle Database releases all table and row locks acquired since that savepoint but retains all data locks acquired previous to the savepoint.

The transaction remains active and can be continued.

Whenever a session is waiting on a transaction, a rollback to savepoint does not free row locks. To make sure a transaction does not hang if it cannot obtain a lock, use FOR UPDATE ... NOWAIT before issuing UPDATE or DELETE statements. (This refers to locks obtained before the savepoint to which has been rolled back. Row locks obtained after this savepoint are released, as the statements executed after the savepoint have been rolled back completely.)

Transaction Naming

You can name a transaction, using a simple and memorable text string. This name is a reminder of what the transaction is about. Transaction names replace commit comments for distributed transactions, with the following advantages:

- It is easier to monitor long-running transactions and to resolve in-doubt distributed transactions.

- You can view transaction names along with transaction IDs in applications. For example, a database administrator can view transaction names in Enterprise Manager when monitoring system activity.

- Transaction names are written to the transaction auditing redo record, if compatibility is set to Oracle9*i* or higher.

- LogMiner can use transaction names to search for a specific transaction from transaction auditing records in the redo log.

- You can use transaction names to find a specific transaction in data dictionary views, such as `V$TRANSACTION`.

This section includes the following topics:

- How Transactions Are Named

- Commit Comment

How Transactions Are Named

Name a transaction using the `SET TRANSACTION ... NAME` statement before you start the transaction.

When you name a transaction, you associate the transaction's name with its ID. Transaction names do not have to be unique; different transactions can have the same transaction name at the same time by the same owner. You can use any name that enables you to distinguish the transaction.

Commit Comment

In previous releases, you could associate a comment with a transaction by using a commit comment. However, a comment can be associated with a transaction only when a transaction is being committed.

Commit comments are still supported for backward compatibility. However, Oracle strongly recommends that you use transaction names. Commit comments are ignored in named transactions.

> **Note:** In a future release, commit comments will be deprecated.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about distributed transactions
>
> - *Oracle Database SQL Language Reference* for more information about transaction naming syntax

The Two-Phase Commit Mechanism

In a distributed database, Oracle Database must coordinate transaction control over a network and maintain data consistency, even if a network or system failure occurs.

A **distributed transaction** is a transaction that includes one or more statements that update data on two or more distinct nodes of a distributed database.

A **two-phase commit** mechanism guarantees that *all* database servers participating in a distributed transaction either all commit or all undo the statements in the transaction. A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

The Oracle Database two-phase commit mechanism is completely transparent to users who issue distributed transactions. In fact, users need not even know the transaction is distributed. A COMMIT statement denoting the end of a transaction automatically triggers the two-phase commit mechanism to commit the transaction. No coding or complex statement syntax is required to include distributed transactions within the body of a database application.

The recoverer (RECO) background process automatically resolves the outcome of **in-doubt distributed transactions**—distributed transactions in which the commit was interrupted by any type of system or network failure. After the failure is repaired and communication is reestablished, the RECO process of each local Oracle database automatically commits or rolls back any in-doubt distributed transactions consistently on all involved nodes.

In the event of a long-term failure, Oracle Database allows each local administrator to manually commit or undo any distributed transactions that are in doubt as a result of the failure. This option enables the local database administrator to free any locked resources that are held indefinitely as a result of the long-term failure.

If a database must be recovered to a point in the past, Oracle Database recovery facilities enable database administrators at other sites to return their databases to the earlier point in time also. This operation ensures that the global database remains consistent.

> **See Also:** *Oracle Database Heterogeneous Connectivity Administrator's Guide*

Overview of Autonomous Transactions

Autonomous transactions are independent transactions that can be called from within another transaction. An autonomous transaction lets you leave the context of the calling transaction, perform some SQL operations, commit or undo those operations, and then return to the calling transaction's context and continue with that transaction.

Once invoked, an autonomous transaction is totally independent of the main transaction that called it. It does not see any of the uncommitted changes made by the main transaction and does not share any locks or resources with the main transaction. Changes made by an autonomous transaction become visible to other transactions upon commit of the autonomous transactions.

One autonomous transaction can call another. There are no limits, other than resource limits, on how many levels of autonomous transactions can be called.

Deadlocks are possible between an autonomous transaction and its calling transaction. Oracle Database detects such deadlocks and returns an error. The application developer is responsible for avoiding deadlock situations.

Autonomous transactions are useful for implementing actions that need to be performed independently, regardless of whether the calling transaction commits or rolls back, such as transaction logging and retry counters.

Autonomous PL/SQL Blocks

You can call autonomous transactions from within a PL/SQL block. Use the pragma `AUTONOMOUS_TRANSACTION`. A **pragma** is a compiler directive. You can declare the following kinds of PL/SQL blocks to be autonomous:

- Stored procedure or function

- Local procedure or function

- Package

- Type method

- Top-level anonymous block

When an autonomous PL/SQL block is entered, the transaction context of the caller is suspended. This operation ensures that SQL operations performed in this block (or other blocks called from it) have no dependence or effect on the state of the caller's transaction context.

When an autonomous block invokes another autonomous block or itself, the called block does not share any transaction context with the calling block. However, when an autonomous block invokes a non-autonomous block (that is, one that is not declared to be autonomous), the called block inherits the transaction context of the calling autonomous block.

Transaction Control Statements in Autonomous Blocks

Transaction control statements in an autonomous PL/SQL block apply only to the currently active autonomous transaction. Examples of such statements are:

```
SET TRANSACTION
COMMIT
ROLLBACK
SAVEPOINT
ROLLBACK TO SAVEPOINT
```

Similarly, transaction control statements in the main transaction apply only to that transaction and not to any autonomous transaction that it calls. For example, rolling back the main transaction to a savepoint taken before the beginning of an autonomous transaction does not undo the autonomous transaction.

> **See Also:** *Oracle Database PL/SQL Language Reference*

5

Schema Objects

This chapter discusses the different types of database objects contained in a user's schema.

This chapter contains the following topics:

- Introduction to Schema Objects
- Overview of Tables
- Overview of Views
- Overview of Materialized Views
- Overview of Dimensions
- Overview of the Sequence Generator
- Overview of Synonyms
- Overview of Indexes
- Overview of Index-Organized Tables
- Overview of Application Domain Indexes
- Overview of Clusters
- Overview of Hash Clusters

Introduction to Schema Objects

A **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 objects can be created and manipulated with SQL and include the following types of objects:

- Clusters
- Constraints
- Database links
- Database triggers
- Dimensions
- External procedure libraries
- Indexes and indextypes
- Java classes, Java resources, and Java sources

- Materialized views and materialized view logs

- Object tables, object types, and object views

- Operators

- Sequences

- Stored functions, procedures, and packages

- Synonyms

- Tables and index-organized tables

- Views

Other types of objects are also stored in the database and can be created and manipulated with SQL but are not contained in a schema:

- Contexts

- Directories

- Parameter files (PFILEs) and server parameter files (SPFILEs)

- Profiles

- Roles

- Rollback segments

- Tablespaces

- Users

Schema objects are logical data storage structures. Schema objects do not have a one-to-one correspondence to physical files on disk that store their information. However, Oracle Database stores a schema object logically within a tablespace of the database. The data of each object is physically contained in one or more of the tablespace's datafiles. For some objects, such as tables, indexes, and clusters, you can specify how much disk space Oracle Database allocates for the object within the tablespace's datafiles.

There is no relationship between schemas and tablespaces: a tablespace can contain objects from different schemas, and the objects for a schema can be contained in different tablespaces.

Figure 5–1 illustrates the relationship among objects, tablespaces, and datafiles.

Figure 5–1   Schema Objects, Tablespaces, and Datafiles
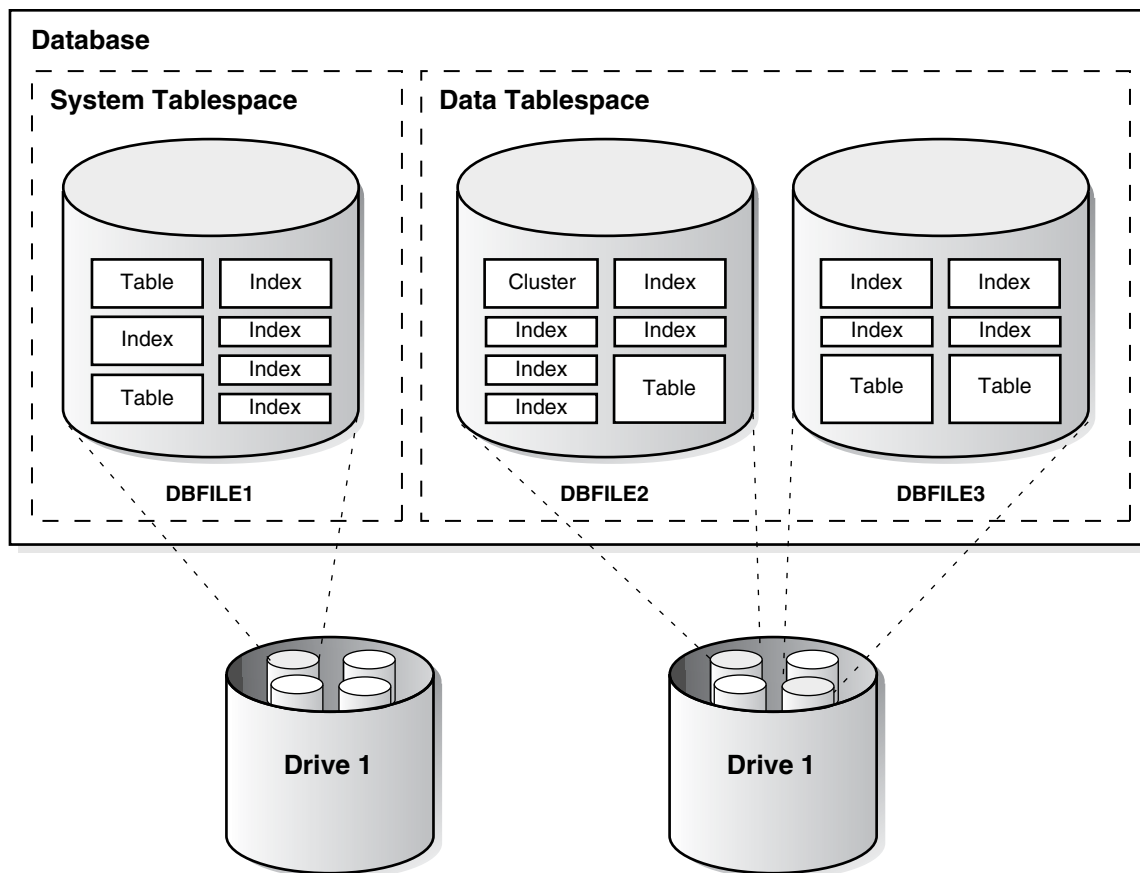
See Also: *Oracle Database Administrator's Guide*

Overview of 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 set of columns. You give each column a **column name** (such as employee_id, last_name, and job_id), a **datatype** (such as VARCHAR2, DATE, or NUMBER), and a **width**. The width can be predetermined by the datatype, as in DATE. If columns are of the NUMBER datatype, define **precision** and **scale** instead of width.

You can specify rules called **integrity constraints** for each column. An example is a NOT NULL integrity constraint, which forces the column to have a value in every row.

A table can contain a **virtual column**, which unlike normal columns does not consume space on disk. Rather, the database derives the values in a virtual column on demand by computing a set of user-specified expressions or functions. Virtual columns can be used in queries, DML, and DDL statements. You can index virtual columns, collect statistics on them, and create integrity constraints. Thus, they can be treated much as nonvirtual columns.

You can also specify table columns for which data is encrypted before being stored in the datafile. Encryption prevents users from circumventing database access control mechanisms by looking inside datafiles directly with operating system tools.

After you create a table, insert rows of data using SQL statements. A row is a collection of column information corresponding to a single record. Table data can then be queried, deleted, or updated using SQL.

Figure 5–2 shows a sample table.

Figure 5–2   The EMP Table

This section includes the following topics:

- How Table Data Is Stored
- Table Compression
- Nulls Indicate Absence of Value
- Default Values for Columns
- Partitioned Tables
- Nested Tables
- Temporary Tables
- External Tables

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information on managing tables
> - *Oracle Database Advanced Security Administrator's Guide* for information on transparent data encryption
> - *Oracle Database SQL Language Reference* for reference information about virtual columns
> - Chapter 26, "Oracle Data Types"
> - Chapter 21, "Data Integrity"

How Table Data Is Stored

When you create a table, Oracle Database automatically allocates a data segment in a tablespace to hold the table's future data. You can control the allocation and use of space for a table's data segment in the following ways:

- You can control the amount of space allocated to the data segment by setting the storage parameters for the data segment.

■ You can control the use of the free space in the data blocks that constitute the data segment's extents by setting the `PCTFREE` and `PCTUSED` parameters for the data segment.

Oracle Database stores data for a clustered table in the data segment created for the cluster instead of in a data segment in a tablespace. Storage parameters cannot be specified when a clustered table is created or altered. The storage parameters set for the cluster always control the storage of all tables in the cluster.

A table's data segment (or cluster data segment, when dealing with a clustered table) is created in either the table owner's default tablespace or in a tablespace specifically named in the `CREATE TABLE` statement.

> **See Also:** "PCTFREE, PCTUSED, and Row Chaining" on page 2-6

This section includes the following topics:

■ Row Format and Size

■ Rowids of Row Pieces

■ Column Order

Row Format and Size

Oracle Database stores each row of a database table containing data for less than 256 columns as one or more row pieces. If an entire row can be inserted into a single data block, Oracle Database stores the row as one row piece. However, if all of a row's data cannot be inserted into a single data block or if an update to an existing row causes the row to outgrow its data block, Oracle Database stores the row using multiple row pieces. A data block usually contains only one row piece for each row. When Oracle Database must store a row in more than one row piece, it is **chained** across multiple blocks.

When a table has more than 255 columns, rows that have data after the 255th column are likely to be chained within the same block. This is called **intra-block chaining**. A chained row's pieces are chained together using the rowids of the pieces. With intra-block chaining, users receive all the data in the same block. If the row fits in the block, users do not see an effect in I/O performance, because no extra I/O operation is required to retrieve the rest of the row.

Each row piece, chained or unchained, contains a **row header** and data for all or some of the row's columns. Individual columns can also span row pieces and, consequently, data blocks. Figure 5–3 shows the format of a row piece.

Figure 5–3   The Format of a Row Piece
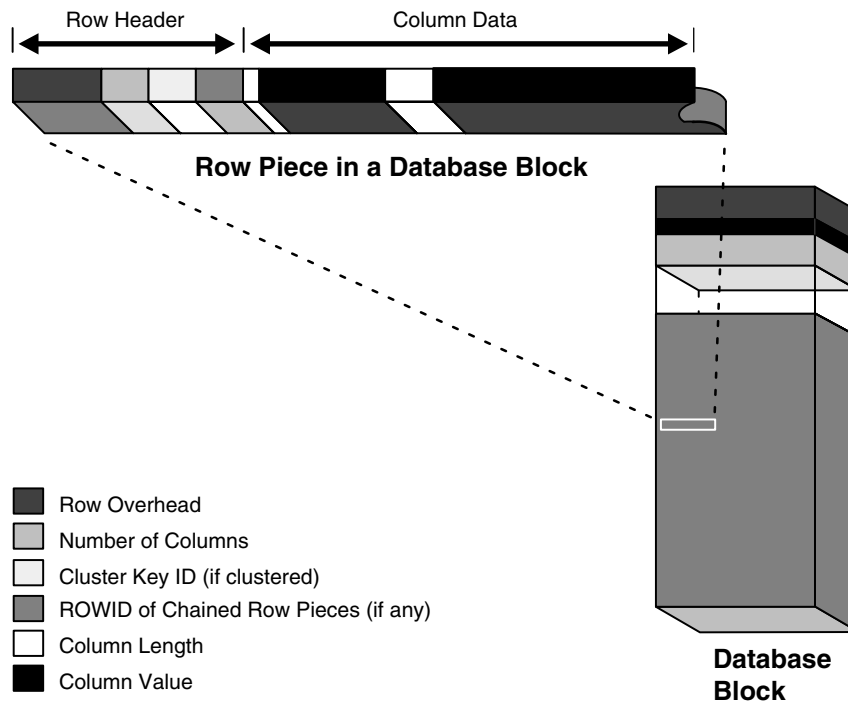
The **row header** precedes the data and contains information about:

- Row pieces
- Chaining (for chained row pieces only)
- Columns in the row piece
- Cluster keys (for clustered data only)

A row fully contained in one block has at least 3 bytes of row header. After the row header information, each row contains column length and data. The column length requires 1 byte for columns that store 250 bytes or less, or 3 bytes for columns that store more than 250 bytes, and precedes the column data. Space required for column data depends on the datatype. If the datatype of a column is variable length, then the space required to hold a value can grow and shrink with updates to the data.

To conserve space, a null in a column only stores the column length (zero). Oracle Database does not store data for the null column. Also, for trailing null columns, Oracle Database does not even store the column length.

> **Note:** Each row also uses 2 bytes in the data block header's row directory.

Clustered rows contain the same information as nonclustered rows. In addition, they contain information that references the cluster key to which they belong.

See Also:

- *Oracle Database Administrator's Guide* for more information about clustered rows and tables

-

-

-

-

Rowids of Row Pieces

The **rowid** identifies each row piece by its location or address. After they are assigned, a given row piece retains its rowid until the corresponding row is deleted or exported and imported using Oracle Database utilities. For clustered tables, if the cluster key values of a row change, then the row keeps the same rowid but also gets an additional pointer rowid for the new values.

Because rowids are constant for the lifetime of a row piece, it is useful to reference rowids in SQL statements such as `SELECT`, `UPDATE`, and `DELETE`.

> **See Also:**

Column Order

The column order is the same for all rows in a given table. Columns are usually stored in the order in which they were listed in the `CREATE TABLE` statement, but this is not guaranteed. For example, if a table has a column of datatype `LONG`, Oracle Database always stores this column last. Also, if a table is altered so that a new column is added, then the new column becomes the last column stored.

In general, try to place columns that frequently contain nulls last so that rows take less space. Note, though, that if the table you are creating includes a `LONG` column as well, then the benefits of placing frequently null columns last are lost.

Table Compression

The Oracle Database table compression feature compresses data by eliminating duplicate values in a database block. Compressed data stored in a database block (also known as disk page) is self-contained. That is, all the information needed to re-create the uncompressed data in a block is available within that block. Duplicate values in all the rows and columns in a block are stored once at the beginning of the block, in what is called a symbol table for that block. All occurrences of such values are replaced with a short reference to the symbol table.

With the exception of a symbol table at the beginning, compressed database blocks look very much like regular database blocks. All database features and functions that work on regular database blocks also work on compressed database blocks. Database objects that can be compressed include tables and materialized views. For partitioned tables, you can choose to compress some or all partitions. Compression attributes can be declared for a tablespace, a table, or a partition of a table. If declared at the tablespace level, then all tables created in that tablespace are compressed by default. You can alter the compression attribute for a table (or a partition or tablespace), and the change only applies to new data going into that table. As a result, a single table or partition may contain some compressed blocks and some regular blocks. This guarantees that data size will not increase as a result of compression; in cases where compression could increase the size of a block, it is not applied to that block.

Using Table Compression

Compression can occur while data is being inserted, updated, bulk inserted, or bulk loaded into a compressed table. These operations include:

- Direct path SQL*Loader

- CREATE TABLE and AS SELECT statements

- Parallel INSERT (or serial INSERT with an APPEND hint) statements

- Single-row or array inserts

- Single-row or array updates

Existing data in the database can also be compressed by moving it into compressed form through ALTER TABLE and MOVE statements. This operation takes an exclusive lock on the table, and therefore prevents any updates and loads until it completes. If this is not acceptable, the Oracle Database online redefinition utility (the DBMS_REDEFINITION PL/SQL package) can be used.

Data compression works for all datatypes except for all variants of LOBs and datatypes derived from LOBs, such as varrays stored out of line or the XML datatype stored in a CLOB.

Table compression is done as part of bulk loading data into the database or during single-row or array inserts and updates. The overhead associated with compression is most visible at that time. This overhead is the primary trade-off that needs to be taken into account when considering compression.

Compressed tables or partitions can be modified the same as other Oracle Database tables or partitions. Deleting compressed data is as fast as deleting uncompressed data. Inserting new data is also as fast. Updating compressed data can be slower in some cases. Because Oracle Database supports all DML operations (insert, update, delete) on compressed tables, table compression is suitable for OLTP applications as well as data warehousing applications. In both these environments, data should be organized so that read only or infrequently changing portions of the data (for example, historical data) are kept compressed.

Nulls Indicate Absence of Value

A **null** is the absence of a value in a column of a row. Nulls indicate missing, unknown, or inapplicable data. A null should not be used to imply any other value, such as zero. A column allows nulls unless a NOT NULL or PRIMARY KEY integrity constraint has been defined for the column, in which case no row can be inserted without a value for that column.

Nulls are stored in the database if they fall between columns with data values. In these cases they require 1 byte to store the length of the column (zero).

Trailing nulls in a row require no storage because a new row header signals that the remaining columns in the previous row are null. For example, if the last three columns of a table are null, no information is stored for those columns. In tables with many columns, the columns more likely to contain nulls should be defined last to conserve disk space.

Most comparisons between nulls and other values are by definition neither true nor false, but unknown. To identify nulls in SQL, use the IS NULL predicate. Use the SQL function NVL to convert nulls to non-null values.

Nulls are not indexed, except when the cluster key column value is null or the index is a bitmap index.

See Also:

- *Oracle Database SQL Language Reference* for comparisons using `IS NULL` and the `NVL` function

- "Indexes and Nulls" on page 5-25

- "Bitmap Indexes and Nulls" on page 5-35

Default Values for Columns

If a default value is not explicitly defined for a column, then the default for the column is implicitly `NULL`. You can also assign a default value to a column of a table so that when a new row is inserted and a value for the column is omitted or keyword `DEFAULT` is supplied, a default value is supplied automatically.

Default column values work as though an `INSERT` statement actually specifies the default value. The datatype of the default literal or expression must match or be convertible to the column datatype.

Integrity constraint checking occurs after the row with a default value is inserted. For example, in Figure 5–4, a row is inserted into the `emp` table that does not include a value for the department number of the employee. Because no value is supplied for the department number, Oracle Database inserts the `deptno` column's default value of 20. After inserting the default value, Oracle Database checks the `FOREIGN KEY` integrity constraint defined on the `deptno` column.

Figure 5–4   DEFAULT Column Values
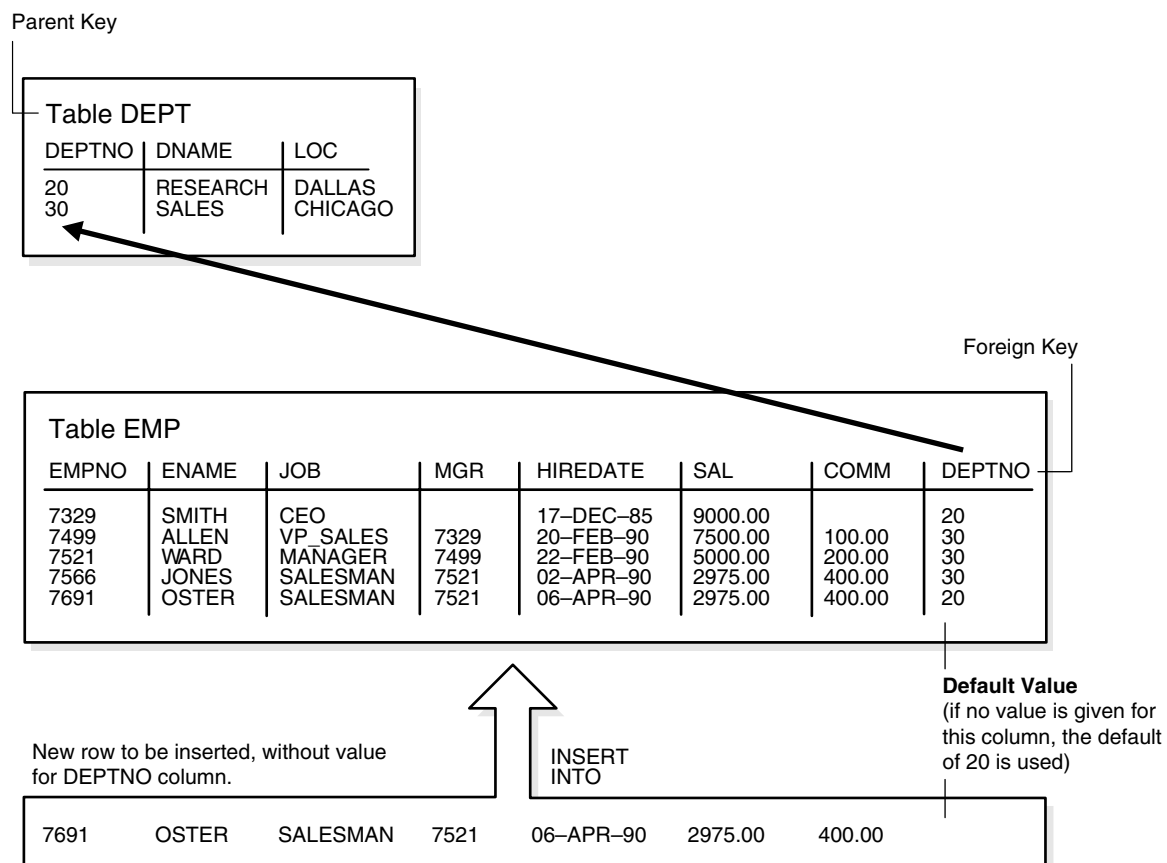
For certain types of tables of column datatypes, when adding a column that has both a `NOT NULL` constraint and a default value, the database can optimize the operation and reduce the amount of time that the table is locked for DML.

> **See Also:** Chapter 21, "Data Integrity" for more information about integrity constraints

Partitioned Tables

Partitioned tables allow your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Indexes can be partitioned in similar fashion. Each partition can be managed individually, and can operate independently of the other partitions, thus providing a structure that can be better tuned for availability and performance.

> **Note:** To reduce disk use and memory use (specifically, the buffer cache), you can store tables and partitioned tables in a compressed format inside the database. This often leads to a better scaleup for read-only operations. Table compression can also speed up query execution. There is, however, a slight cost in CPU overhead.

See Also:

- "Table Compression" on page 16-8
- *Oracle Database VLDB and Partitioning Guide*

Nested Tables

You can create a table with a column whose datatype is another table. That is, tables can be **nested** within other tables as values in a column. The Oracle database server stores nested table data out of line from the rows of the parent table, using a **store table** that is associated with the nested table column. The parent row contains a unique set identifier value associated with a nested table instance.

See Also:

- *Oracle Database Object-Relational Developer's Guide* for further information on nested tables
- *Oracle Database Advanced Application Developer's Guide*

Temporary Tables

In addition to permanent tables, Oracle Database can create **temporary tables** to hold session-private data that exists only for the duration of a transaction or session.

The `CREATE GLOBAL TEMPORARY TABLE` statement creates a temporary table that can be transaction-specific or session-specific. For transaction-specific temporary tables, data exists for the duration of the transaction. For session-specific temporary tables, data exists for the duration of the session. Data in a temporary table is private to the session. Each session can only see and modify its own data. DML locks are not acquired on the data of the temporary tables. The `LOCK` statement has no effect on a temporary table, because each session has its own private data.

A `TRUNCATE` statement issued on a session-specific temporary table truncates data in its own session. It does not truncate the data of other sessions that are using the same table.

DML statements on temporary tables do not generate redo logs for the data changes. However, undo logs for the data and redo logs for the undo logs are generated. Data from the temporary table is automatically dropped in the case of session termination, either when the user logs off or when the session terminates abnormally such as during a session or instance failure.

You can create indexes for temporary tables using the CREATE INDEX statement. Indexes created on temporary tables are also temporary, and the data in the index has the same session or transaction scope as the data in the temporary table.

You can create views that access both temporary and permanent tables. You can also create triggers on temporary tables.

Oracle Database utilities can export and import the definition of a temporary table. However, no data rows are exported even if you use the ROWS clause. Similarly, you can replicate the definition of a temporary table, but you cannot replicate its data.

This section includes the following topics:

- Segment Allocation
- Parent and Child Transactions

Segment Allocation

Temporary tables use temporary segments. Unlike permanent tables, temporary tables and their indexes 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. This means that if a SELECT, UPDATE, or DELETE is performed before the first INSERT, then the table appears to be empty.

You can perform DDL statements (ALTER TABLE, DROP TABLE, CREATE INDEX, and so on) on a temporary table only when no session is currently bound to it. A session gets bound to a temporary table when an INSERT is performed on it. The session gets unbound by a TRUNCATE, at session termination, or by doing a COMMIT or ROLLBACK for a transaction-specific temporary table.

Temporary segments are deallocated at the end of the transaction for transaction-specific temporary tables and at the end of the session for session-specific temporary tables.

> **See Also:** "Extents in Temporary Segments" on page 2-13

Parent and Child Transactions

Transaction-specific temporary tables are accessible by user transactions and their child transactions. However, a given transaction-specific temporary table cannot be used concurrently by two transactions in the same session, although it can be used by transactions in different sessions.

If a user transaction does an INSERT into the temporary table, then none of its child transactions can use the temporary table afterward.

If a child transaction does an INSERT into the temporary table, then at the end of the child transaction, the data associated with the temporary table goes away. After that, either the user transaction or any other child transaction can access the temporary table.

External Tables

External tables access data in external sources as if it were in a table in the database. You can connect to the database and create metadata for the external table using DDL. The DDL for an external table consists of two parts: one part that describes the Oracle Database column types, and another part (the access parameters) that describes the mapping of the external data to the Oracle Database data columns.

An external table does not describe any data that is stored in the database. Nor does it describe how data is stored in the external source. Instead, it describes how the external table layer needs to present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the datafile so that it matches the external table definition.

External tables are read only; therefore, no DML operations are possible, and no index can be created on them. Also, virtual columns are not supported.

This section includes the following topics:

- The Access Driver
- Data Loading with External Tables
- Parallel Access to External Tables

The Access Driver

When you create an external table, you specify its type. Each type of external table has its own access driver that provides access parameters unique to that type of external table. The access driver ensures that data from the data source is processed so that it matches the definition of the external table.

In the context of external tables, loading data refers to the act of reading data from an external table and loading it into a table in the database. Unloading data refers to the act of reading data from a table in the database and inserting it into an external table.

The default type for external tables is ORACLE_LOADER, which lets you read table data from an external table and load it into a database. Oracle Database also provides the ORACLE_DATAPUMP type, which lets you unload data (that is, read data from a table in the database and insert it into an external table) and then reload it into an Oracle database.

The definition of an external table is kept separately from the description of the data in the data source. This means that:

- The source file can contain more or fewer fields than there are columns in the external table
- The datatypes for fields in the data source can be different from the columns in the external table

Data Loading with External Tables

The main use for external tables is to use them as a row source for loading data into an actual table in the database. After you create an external table, you can then use a CREATE TABLE AS SELECT or INSERT INTO ... AS SELECT statement, using the external table as the source of the SELECT clause.

> **Note:** You cannot insert data into external tables or update records in them; external tables are read only.

When you access the external table through a SQL statement, the fields of the external table can be used just like any other field in a regular table. In particular, you can use the fields as arguments for any SQL built-in function, PL/SQL function, or Java function. This lets you manipulate data from the external source. For data warehousing, you can do more sophisticated transformations in this way than you can with simple datatype conversions. You can also use this mechanism in data warehousing to do data cleansing.

While external tables cannot contain a column object, constructor functions can be used to build a column object from attributes in the external table

Parallel Access to External Tables

After the metadata for an external table is created, you can query the external data directly and in parallel, using SQL. As a result, the external table acts as a view, which lets you run any SQL query against external data without loading the external data into the database.

The degree of parallel access to an external table is specified using standard parallel hints and with the PARALLEL clause. Using parallelism on an external table allows for concurrent access to the datafiles that comprise an external table. Whether a single file is accessed concurrently is dependent upon the access driver implementation, and attributes of the datafile(s) being accessed (for example, record formats).

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about managing external tables, external connections, and directories
>
> - *Oracle Database Performance Tuning Guide* for information about tuning loads from external tables
>
> - *Oracle Database Utilities* for information about external tables and import and export
>
> - *Oracle Database SQL Language Reference* for information about creating and querying external tables

Overview of Views

A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Therefore, a view can be thought of as a stored query or a virtual table. You can use views in most places where a table can be used.

For example, the employees table has several columns and numerous rows of information. If you want users to see only five of these columns or only specific rows, then you can create a view of that table for other users to access.

Figure 5–5 shows an example of a view called staff derived from the base table employees. Notice that the view shows only five of the columns in the base table.

Figure 5–5 An Example of a View

Because views are derived from tables, they have many similarities. For example, you can define views with up to 1000 columns, just like a table. You can query views, and with some restrictions you can update, insert into, and delete from views. All operations performed on a view actually affect data in some base table of the view and are subject to the integrity constraints and triggers of the base tables.

You cannot explicitly define triggers on views, but you can define them for the underlying base tables referenced by the view. Oracle Database does support definition of logical constraints on views.

> **See Also:** *Oracle Database SQL Language Reference*

This section includes the following topics:

- How Views are Stored
- How Views Are Used
- Mechanics of Views
- Dependencies and Views
- Updatable Join Views
- Object Views
- Inline Views

How Views are Stored

Unlike a table, a view is not allocated any storage space, nor does a view actually contain data. Rather, a view is defined by a query that extracts or derives data from the tables that the view references. These tables are called **base tables**. Base tables can in turn be actual tables or can be views themselves (including materialized views). Because a view is based on other objects, a view requires no storage other than storage for the definition of the view (the stored query) in the data dictionary.

How Views Are Used

Views provide a means to present a different representation of the data that resides within the base tables. Views are very powerful because they let you tailor the presentation of data to different types of users. Views are often used to:

- Provide an additional level of table security by restricting access to a predetermined set of rows or columns of a table

  For example, shows how the `STAFF` view does not show the `salary` or `commission_pct` columns of the base table `employees`.

- Hide data complexity

  For example, a single view can be defined with a **join**, which is a collection of related columns or rows in multiple tables. However, the view hides the fact that this information actually originates from several tables.

- Simplify statements for the user

  For example, views allow users to select information from multiple tables without actually knowing how to perform a join.

- Present the data in a different perspective from that of the base table

  For example, the columns of a view can be renamed without affecting the tables on which the view is based.

- Isolate applications from changes in definitions of base tables

  For example, if a view's defining query references three columns of a four column table, and a fifth column is added to the table, then the view's definition is not affected, and all applications using the view are not affected.

- Express a query that cannot be expressed without using a view

  For example, a view can be defined that joins a `GROUP BY` view with a table, or a view can be defined that joins a `UNION` view with a table.

- Save complex queries

  For example, a query can perform extensive calculations with table information. By saving this query as a view, you can perform the calculations each time the view is queried.

  > **See Also:** *Oracle Database SQL Language Reference* for information about the `GROUP BY` or `UNION` views

Mechanics of Views

Oracle Database stores a view's definition in the data dictionary as the text of the query that defines the view. When you reference a view in a SQL statement, Oracle Database:

1. Merges the statement that references the view with the query that defines the view

2. Parses the merged statement in a shared SQL area

3. Executes the statement

Oracle Database parses a statement that references a view in a new shared SQL area *only* if no existing shared SQL area contains a similar statement. Therefore, you get the benefit of reduced memory use associated with shared SQL when you use views.

This section includes the following topics:

- Globalization Support Parameters in Views
- Use of Indexes Against Views

Globalization Support Parameters in Views

When Oracle Database evaluates views containing string literals or SQL functions that have globalization support parameters as arguments (such as `TO_CHAR`, `TO_DATE`, and `TO_NUMBER`), Oracle Database takes default values for these parameters from the globalization support parameters for the session. You can override these default values by specifying globalization support parameters explicitly in the view definition.

> **See Also:** *Oracle Database Globalization Support Guide* for information about globalization support

Use of Indexes Against Views

Oracle Database determines whether to use indexes for a query against a view by transforming the original query when merging it with the view's defining query.

Consider the following view:

```
CREATE VIEW employees_view AS
  SELECT employee_id, last_name, salary, location_id
    FROM employees JOIN departments USING (department_id)
    WHERE departments.department_id = 10;
```

Now consider the following user-issued query:

```
SELECT last_name
  FROM employees_view
  WHERE employee_id = 9876;
```

The final query constructed by Oracle Database is:

```
SELECT last_name
  FROM employees, departments
  WHERE employees.department_id = departments.department_id AND
        departments.department_id = 10 AND
        employees.employee_id = 9876;
```

In all possible cases, Oracle Database merges a query against a view with the view's defining query and those of any underlying views. Oracle Database optimizes the merged query as if you issued the query without referencing the views. Therefore, Oracle Database can use indexes on any referenced base table columns, whether the columns are referenced in the view definition or in the user query against the view.

In some cases, Oracle Database cannot merge the view definition with the user-issued query. In such cases, Oracle Database may not use all indexes on referenced columns.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about query optimization

Dependencies and Views

Because a view is defined by a query that references other objects (tables, materialized views, or other views), a view depends on the referenced objects. Oracle Database automatically handles the dependencies for views. For example, if you drop a base table of a view and then create it again, Oracle Database determines whether the new base table is acceptable to the existing definition of the view.

See Also: Chapter 6, "Schema Object Dependencies"

Updatable Join Views

A **join view** is defined as a view that has more than one table or view in its FROM clause (a **join**) and that does not use any of these clauses: DISTINCT, aggregation, GROUP BY, START WITH, CONNECT BY, ROWNUM, and set operations (UNION ALL, INTERSECT, and so on).

An **updatable join view** is a join view that involves two or more base tables or views, where UPDATE, INSERT, and DELETE operations are permitted. The data dictionary views ALL_UPDATABLE_COLUMNS, DBA_UPDATABLE_COLUMNS, and USER_UPDATABLE_COLUMNS contain information that indicates which of the view columns are updatable. In order to be inherently updatable, a view cannot contain any of the following constructs:

- A set operator
- A DISTINCT operator
- An aggregate or analytic function
- A GROUP BY, ORDER BY, CONNECT BY, or START WITH clause
- A collection expression in a SELECT list
- A subquery in a SELECT list
- Joins (with some exceptions)

Views that are not updatable can be modified using INSTEAD OF triggers.

See Also:

- *Oracle Database Administrator's Guide*
- *Oracle Database SQL Language Reference* for more information about updatable views
- "INSTEAD OF Triggers" on page 22-8

Object Views

In the Oracle object-relational database, an **object view** let you retrieve, update, insert, and delete relational data as if it was stored as an object type. You can also define views with columns that are object datatypes, such as objects, REFs, and collections (nested tables and VARRAYs).

See Also:

- *Oracle Database Object-Relational Developer's Guide*
- *Oracle Database Advanced Application Developer's Guide*

Inline Views

An **inline view** is not a schema object. It is a subquery with an alias (correlation name) that you can use like a view within a SQL statement.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for information about subqueries
> - *Oracle Database Performance Tuning Guide* for an example of an inline query causing a view

Overview of Materialized Views

Materialized views are schema objects that can be used to summarize, compute, replicate, and distribute data. They are suitable in various computing environments such as data warehousing, decision support, and distributed or mobile computing:

- In data warehouses, materialized views are used to compute and store aggregated data such as sums and averages. Materialized views in these environments are typically referred to as **summaries** because they store summarized data. They can also be used to compute joins with or without aggregations. If compatibility is set to Oracle9*i* or higher, then materialized views can be used for queries that include filter selections.

    The optimizer can use materialized views to improve query performance by automatically recognizing when a materialized view can and should be used to satisfy a request. The optimizer transparently rewrites the request to use the materialized view. Queries are then directed to the materialized view and not to the underlying detail tables or views.

- In distributed environments, materialized views are used to replicate data at distributed sites and synchronize updates done at several sites with conflict resolution methods. The materialized views as replicas provide local access to data that otherwise has to be accessed from remote sites.

- In mobile computing environments, materialized views are used to download a subset of data from central servers to mobile clients, with periodic refreshes from the central servers and propagation of updates by clients back to the central servers.

Materialized views are similar to indexes in several ways:

- They consume storage space.
- They must be refreshed when the data in their master tables changes.
- They improve the performance of SQL execution when they are used for query rewrites.
- Their existence is transparent to SQL applications and users.

Unlike indexes, materialized views can be accessed directly using a SELECT statement. Depending on the types of refresh that are required, they can also be accessed directly in an INSERT, UPDATE, or DELETE statement.

A materialized view can be partitioned. You can define a materialized view on a partitioned table and one or more indexes on the materialized view.

This section includes the following topics:

- Define Constraints on Views
- Refresh Materialized Views
- Materialized View Logs

See Also:

- ["Overview of Indexes"](#) on page 5-23

- *Oracle Database VLDB and Partitioning Guide*

- *Oracle Database Data Warehousing Guide* for information about materialized views in a data warehousing environment

Define Constraints on Views

Data warehousing applications recognize multidimensional data in the Oracle database by identifying Referential Integrity (RI) constraints in the relational schema. RI constraints represent primary and foreign key relationships among tables. By querying the Oracle Database data dictionary, applications can recognize RI constraints and therefore recognize the multidimensional data in the database. In some environments, database administrators, for schema complexity or security reasons, define views on fact and dimension tables. Oracle Database provides the ability to constrain views. By allowing constraint definitions between views, database administrators can propagate base table constraints to the views, thereby allowing applications to recognize multidimensional data even in a restricted environment.

Only logical constraints, that is, constraints that are declarative and not enforced by Oracle Database, can be defined on views. The purpose of these constraints is not to enforce any business rules but to identify multidimensional data. The following constraints can be defined on views:

- Primary key constraint

- Unique constraint

- Referential Integrity constraint

Given that view constraints are declarative, `DISABLE`, `NOVALIDATE` is the only valid state for a view constraint. However, the `RELY` or `NORELY` state is also allowed, because constraints on views may be used to enable more sophisticated query rewrites; a view constraint in the `RELY` state allows query rewrites to occur when the rewrite integrity level is set to trusted mode.

> **Note:** Although view constraint definitions are declarative in nature, operations on views are subject to the integrity constraints defined on the underlying base tables, and constraints on views can be enforced through constraints on base tables.

Refresh Materialized Views

Oracle Database maintains the data in materialized views by refreshing them after changes are made to their master tables. The refresh method can be incremental (**fast refresh**) or complete. For materialized views that use the fast refresh method, a **materialized view log** or **direct loader log** keeps a record of changes to the master tables.

Materialized views can be refreshed either on demand or at regular time intervals. Alternatively, materialized views in the same database as their master tables can be refreshed whenever a transaction commits its changes to the master tables.

Materialized View Logs

A **materialized view log** is a schema object that records changes to a master table's data so that a materialized view defined on the master table can be refreshed incrementally.

Each materialized view log is associated with a single master table. The materialized view log resides in the same database and schema as its master table.

> **See Also:**
>
> - *Oracle Database Data Warehousing Guide* for information about materialized views and materialized view logs in a warehousing environment
>
> - *Oracle Database Advanced Replication* for information about materialized views used for replication

Overview of Dimensions

A dimension defines hierarchical (parent/child) relationships between pairs of columns or column sets. Each value at the child level is associated with one and only one value at the parent level. A hierarchical relationship is a **functional dependency** from one level of a hierarchy to the next level in the hierarchy. A dimension is a container of logical relationships between columns, and it does not have any data storage assigned to it.

The CREATE DIMENSION statement specifies:

- Multiple LEVEL clauses, each of which identifies a column or column set in the dimension

- One or more HIERARCHY clauses that specify the parent/child relationships between adjacent levels

- Optional ATTRIBUTE clauses, each of which identifies an additional column or column set associated with an individual level

The columns in a dimension can come either from the same table (**denormalized**) or from multiple tables (**fully** or **partially normalized**). To define a dimension over columns from multiple tables, connect the tables using the JOIN clause of the HIERARCHY clause.

For example, a normalized time dimension can include a date table, a month table, and a year table, with join conditions that connect each date row to a month row, and each month row to a year row. In a fully denormalized time dimension, the date, month, and year columns are all in the same table. Whether normalized or denormalized, the hierarchical relationships among the columns need to be specified in the CREATE DIMENSION statement.

> **See Also:**
>
> - *Oracle Database Data Warehousing Guide* for information about how dimensions are used in a warehousing environment
>
> - *Oracle Database SQL Language Reference* for information about creating dimensions

Overview of the Sequence Generator

The sequence generator provides a sequential series of numbers. The sequence generator is especially useful in multiuser environments for generating unique sequential numbers without the overhead of disk I/O or transaction locking. For example, assume two users are simultaneously inserting new employee rows into the `employees` table. By using a sequence to generate unique employee numbers for the `employee_id` column, neither user has to wait for the other to enter the next available employee number. The sequence automatically generates the correct values for each user.

Therefore, the sequence generator reduces serialization where the statements of two transactions must generate sequential numbers at the same time. By avoiding the serialization that results when multiple users wait for each other to generate and use a sequence number, the sequence generator improves transaction throughput, and a user's wait is considerably shorter.

Sequence numbers are integers of up to 38 digits defined in the database. A sequence definition indicates general information, such as the following:

- The name of the sequence

- Whether the sequence ascends or descends

- The interval between numbers

- Whether Oracle Database should cache sets of generated sequence numbers in memory

Oracle Database stores the definitions of all sequences for a particular database as rows in a single data dictionary table in the `SYSTEM` tablespace. Therefore, all sequence definitions are always available, because the `SYSTEM` tablespace is always online.

Sequence numbers are used by SQL statements that reference the sequence. You can issue a statement to generate a new sequence number or use the current sequence number. After a statement in a user's session generates a sequence number, the particular sequence number is available only to that session. Each user that references a sequence has access to the current sequence number.

Sequence numbers are generated independently of tables. Therefore, the same sequence generator can be used for more than one table. Sequence number generation is useful to generate unique primary keys for your data automatically and to coordinate keys across multiple rows or tables. Individual sequence numbers can be skipped if they were generated and used in a transaction that was ultimately rolled back. Applications can make provisions to catch and reuse these sequence numbers, if desired.

> **Caution:** If your application can never lose sequence numbers, then you cannot use Oracle Database sequences, and you may choose to store sequence numbers in database tables. Be careful when implementing sequence generators using database tables. Even in a single instance configuration, for a high rate of sequence values generation, a performance overhead is associated with the cost of locking the row that stores the sequence value.

See Also:

- *Oracle Database Advanced Application Developer's Guide* for performance implications when using sequences

- *Oracle Database SQL Language Reference* for information about the `CREATE SEQUENCE` statement

Overview of Synonyms

A **synonym** is an alias for any table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms are often used for security and convenience. For example, they can do the following:

- Mask the name and owner of an object

- Provide location transparency for remote objects of a distributed database

- Simplify SQL statements for database users

- Enable restricted access similar to specialized views when exercising fine-grained access control

You can create both public and private synonyms. A **public** synonym is owned by the special user group named `PUBLIC` and every user in a database can access it. A **private** synonym is in the schema of a specific user who has control over its availability to others.

Synonyms are very useful in both distributed and nondistributed database environments because they hide the identity of the underlying object, including its location in a distributed system. This is advantageous because if the underlying object must be renamed or moved, then only the synonym needs to be redefined. Applications based on the synonym continue to function without modification.

Synonyms can also simplify SQL statements for users in a distributed database system. The following example shows how and why public synonyms are often created by a database administrator to hide the identity of a base table and reduce the complexity of SQL statements. Assume the following:

- A table called `SALES_DATA` is in the schema owned by the user `JWARD`.

- The `SELECT` privilege for the `SALES_DATA` table is granted to `PUBLIC`.

At this point, you have to query the table `SALES_DATA` with a SQL statement similar to the following:

```
SELECT * FROM jward.sales_data;
```

Notice how you must include both the schema that contains the table along with the table name to perform the query.

Assume that the database administrator creates a public synonym with the following SQL statement:

```
CREATE PUBLIC SYNONYM sales FOR jward.sales_data;
```

After the public synonym is created, you can query the table `SALES_DATA` with a simple SQL statement:

```
SELECT * FROM sales;
```

Notice that the public synonym SALES hides the name of the table SALES_DATA and the name of the schema that contains the table.

Overview of Indexes

Indexes are optional structures associated with tables and clusters. You can create indexes on one or more columns of a table to speed SQL statement execution on that 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. Indexes are the primary means of reducing disk I/O when properly used.

You can create many indexes for a table as long as the combination of columns differs for each index. You can create more than one index using the same columns if you specify distinctly different combinations of the columns. For example, the following statements specify valid combinations:

```
CREATE INDEX employees_idx1 ON employees (last_name, job_id);
CREATE INDEX employees_idx2 ON employees (job_id, last_name);
```

Oracle Database provides several indexing schemes, which provide complementary performance functionality:

- B-tree indexes

- B-tree cluster indexes

- Hash cluster indexes

- Reverse key indexes

- Bitmap indexes

- Bitmap join indexes

Oracle Database also provides support for function-based indexes and domain indexes specific to an application or cartridge.

The absence or presence of an index does not require a change in the wording of any SQL statement. An index is merely a fast access path to the data. It affects only the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value.

Indexes are logically and physically independent of the data in the associated table. You can create or drop an index at any time without affecting the base tables or other indexes. If you drop an index, all applications continue to work. However, access of previously indexed data can be slower. Indexes, as independent structures, require storage space.

Oracle Database automatically maintains and uses indexes after they are created. Oracle Database automatically reflects changes to data, such as adding new rows, updating rows, or deleting rows, in all relevant indexes with no additional action by users.

Retrieval performance of indexed data remains almost constant, even as new rows are inserted. However, the presence of many indexes on a table decreases the performance of updates, deletes, and inserts, because Oracle Database must also update the indexes associated with the table.

The optimizer can use an existing index to build another index. This results in a much faster index build.

This section includes the following topics:

- [Unique and Nonunique Indexes](#)
- [Visible and Invisible Indexes](#)
- [Composite Indexes](#)
- [Indexes and Keys](#)
- [Indexes and Nulls](#)
- [Function-Based Indexes](#)
- [How Indexes Are Stored](#)
- [Index Unique Scan](#)
- [Index Range Scan](#)
- [Key Compression](#)
- [Reverse Key Indexes](#)
- [Bitmap Indexes](#)
- [Bitmap Join Indexes](#)

Unique and Nonunique Indexes

Indexes can be unique or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Nonunique indexes do not impose this restriction on the column values.

Oracle recommends that unique indexes be created explicitly, using `CREATE UNIQUE INDEX`. Creating unique indexes through a primary key or unique constraint is not guaranteed to create a new index, and the index they create is not guaranteed to be a unique index.

> **See Also:** *Oracle Database Administrator's Guide* for information about creating unique indexes explicitly

Visible and Invisible Indexes

Indexes can be visible or invisible. An invisible index is maintained by DML operations and cannot be used by the optimizer.

Making an index invisible is an alternative to making it unusable or dropping it.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about creating invisible indexes
> - *Oracle Database Administrator's Guide* for information about making indexes invisible

Composite Indexes

A **composite index** (also called a **concatenated index**) is an index that you create on multiple columns in a table. Columns in a composite index can appear in any order and need not be adjacent in the table.

Composite indexes can speed retrieval of data for `SELECT` statements in which the `WHERE` clause references all or the leading portion of the columns in the composite

index. Therefore, the order of the columns used in the definition is important. Generally, the most commonly accessed or most selective columns go first.

Figure 5–6 illustrates the VENDOR_PARTS table that has a composite index on the VENDOR_ID and PART_NO columns.

Figure 5–6   Composite Index Example

VENDOR_PARTS

VEND ID	PART NO	UNIT COST
1012	10–440	.25
1012	10–441	.39
1012	457	4.95
1010	10–440	.27
1010	457	5.10
1220	08–300	1.33
1012	08–300	1.19
1292	457	5.28

Concatenated Index
(index with multiple columns)

No more than 32 columns can form a regular composite index. For a bitmap index, the maximum number columns is 30. A key value cannot exceed roughly half (minus some overhead) the available data space in a data block.

> **See Also:**   *Oracle Database Performance Tuning Guide* for more information about using composite indexes

Indexes and Keys

Although the terms are often used interchangeably, indexes and keys are different. **Indexes** are structures actually stored in the database, which users create, alter, and drop using SQL statements. You create an index to provide a fast access path to table data. **Keys** are strictly a logical concept. Keys correspond to another feature of Oracle Database called integrity constraints, which enforce the business rules of a database.

Because Oracle Database uses indexes to enforce some integrity constraints, the terms key and index are often are used interchangeably. However, do not confuse them with each other.

> **See Also:**   Chapter 21, "Data Integrity"

Indexes and Nulls

NULL values in indexes are considered to be distinct except when all the non-NULL values in two or more rows of an index are identical, in which case the rows are considered to be identical. Therefore, UNIQUE indexes prevent rows containing NULL values from being treated as identical. This does not apply if there are no non-NULL values—in other words, if the rows are entirely NULL.

Oracle Database does not index table rows in which all key columns are NULL, except in the case of bitmap indexes or when the cluster key column value is NULL.

> **See Also:**   "Bitmap Indexes and Nulls" on page 5-35

Function-Based Indexes

You can create indexes on functions and expressions that involve one or more columns in the table being indexed. A **function-based index** computes the value of the function or expression and stores it in the index. You can create a function-based index as either a B-tree or a bitmap index.

The function used for building the index can be an arithmetic expression or an expression that contains a PL/SQL function, package function, C callout, or SQL function. The expression cannot contain any aggregate functions, and it must be DETERMINISTIC. For building an index on a column containing an object type, the function can be a method of that object, such as a map method. However, you cannot build a function-based index on a LOB column, REF, or nested table column, nor can you build a function-based index if the object type contains a LOB, REF, or nested table.

This section includes the following topics:

- Uses of Function-Based Indexes

- Optimization with Function-Based Indexes

- Dependencies of Function-Based Indexes

> **See Also:**
>
> - "Bitmap Indexes"
>
> - *Oracle Database Performance Tuning Guide* for more information about using function-based indexes

Uses of Function-Based Indexes

Function-based indexes provide an efficient mechanism for evaluating statements that contain functions in their WHERE clauses. The value of the expression is computed and stored in the index. When it processes INSERT and UPDATE statements, however, Oracle Database must still evaluate the function to process the statement.

For example, if you create the following index:

```
CREATE INDEX idx ON table_1 (a + b * (c - 1), a, b);
```

Oracle Database can use it when processing queries such as this:

```
SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;
```

Function-based indexes defined on UPPER(*column_name*) or LOWER(*column_name*) can facilitate case-insensitive searches. For example, the following index:

```
CREATE INDEX uppercase_idx ON employees (UPPER(first_name));
```

can facilitate processing queries such as this:

```
SELECT * FROM employees WHERE UPPER(first_name) = 'RICHARD';
```

A function-based index can also be used for a globalization support sort index that provides efficient linguistic collation in SQL statements.

> **See Also:** *Oracle Database Globalization Support Guide* for information about linguistic indexes

Optimization with Function-Based Indexes

You must gather statistics about function-based indexes for the optimizer. Otherwise, the indexes cannot be used to process SQL statements.

The optimizer can use an index range scan on a function-based index for queries with expressions in `WHERE` clause. For example, in this query:

```
SELECT * FROM t WHERE a + b < 10;
```

the optimizer can use index range scan if an index is built on `a+b`. The range scan access path is especially beneficial when the predicate (`WHERE` clause) has low selectivity. In addition, the optimizer can estimate the selectivity of predicates involving expressions more accurately if the expressions are materialized in a function-based index.

The optimizer performs expression matching by parsing the expression in a SQL statement and then comparing the expression trees of the statement and the function-based index. This comparison is case-insensitive and ignores blank spaces.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about gathering statistics

Dependencies of Function-Based Indexes

Function-based indexes depend on the function used in the expression that defines the index. If the function is a PL/SQL function or package function, the index is disabled by any changes to the function specification.

To create a function-based index, the user must be granted `CREATE INDEX` or `CREATE ANY INDEX`.

To use a function-based index:

- The table must be analyzed after the index is created.

- The query must be guaranteed not to need any `NULL` values from the indexed expression, because `NULL` values are not stored in indexes.

The following sections describe additional requirements.

DETERMINISTIC Functions Any user-written function used in a function-based index must have been declared with the `DETERMINISTIC` keyword to indicate that the function will always return the same output return value for any given set of input argument values, now and in the future.

> **See Also:** *Oracle Database Performance Tuning Guide*

Privileges on the Defining Function The index owner needs the `EXECUTE` privilege on the function used to define a function-based index. If the `EXECUTE` privilege is revoked, Oracle Database marks the index `DISABLED`. The index owner does not need the `EXECUTE WITH GRANT OPTION` privilege on this function to grant `SELECT` privileges on the underlying table.

Resolve Dependencies of Function-Based Indexes A function-based index depends on any function that it is using. If the function or the specification of a package containing the function is redefined (or if the index owner's `EXECUTE` privilege is revoked), then the following conditions hold:

- The index is marked as `DISABLED`.

- Queries on a `DISABLED` index fail if the optimizer chooses to use the index.

- DML operations on a `DISABLED` index fail unless the index is also marked `UNUSABLE` and the initialization parameter `SKIP_UNUSABLE_INDEXES` is set to true.

To re-enable the index after a change to the function, use the `ALTER INDEX ... ENABLE` statement.

How Indexes Are Stored

When you create an index, Oracle Database automatically allocates an index segment to hold the index's data in a tablespace. You can control allocation of space for an index's segment and use of this reserved space in the following ways:

- Set the storage parameters for the index segment to control the allocation of the index segment's extents.

- Set the `PCTFREE` parameter for the index segment to control the free space in the data blocks that constitute the index segment's extents.

The tablespace of an index's segment is either the owner's default tablespace or a tablespace specifically named in the `CREATE INDEX` statement. You do not have to place an index in the same tablespace as its associated table. Furthermore, you can improve performance of queries that use an index by storing an index and its table in different tablespaces located on different disk drives, because Oracle Database can retrieve both index and table data in parallel.

> **See Also:**

This section includes the following topics:

- Format of Index Blocks
- The Internal Structure of Indexes
- Index Properties
- Advantages of B-tree Structure

Format of Index Blocks

Space available for index data is the Oracle Database block size minus block overhead, entry overhead, rowid, and one length byte for each value indexed.

When you create an index, Oracle Database fetches and sorts the columns to be indexed and stores the rowid along with the index value for each row. Then Oracle Database loads the index from the bottom up. For example, consider the statement:

```
CREATE INDEX employees_last_name ON employees(last_name);
```

Oracle Database sorts the `employees` table on the `last_name` column. It then loads the index with the `last_name` and corresponding rowid values in this sorted order. When it uses the index, Oracle Database does a quick search through the sorted `last_name` values and then uses the associated rowid values to locate the rows having the sought `last_name` value.

The Internal Structure of Indexes

Oracle Database uses B-trees to store indexes to speed up data access. With no indexes, you have to do a sequential scan on the data to find a value. For n rows, the average number of rows searched is $n/2$. This does not scale very well as data volumes increase.

Consider an ordered list of the values divided into block-wide ranges (leaf blocks). The end points of the ranges along with pointers to the blocks can be stored in a search tree and a value in log(n) time for n entries could be found. This is the basic principle behind Oracle Database indexes.

Figure 5–7 illustrates the structure of a B-tree index.

Figure 5–7   Internal Structure of a B-tree Index
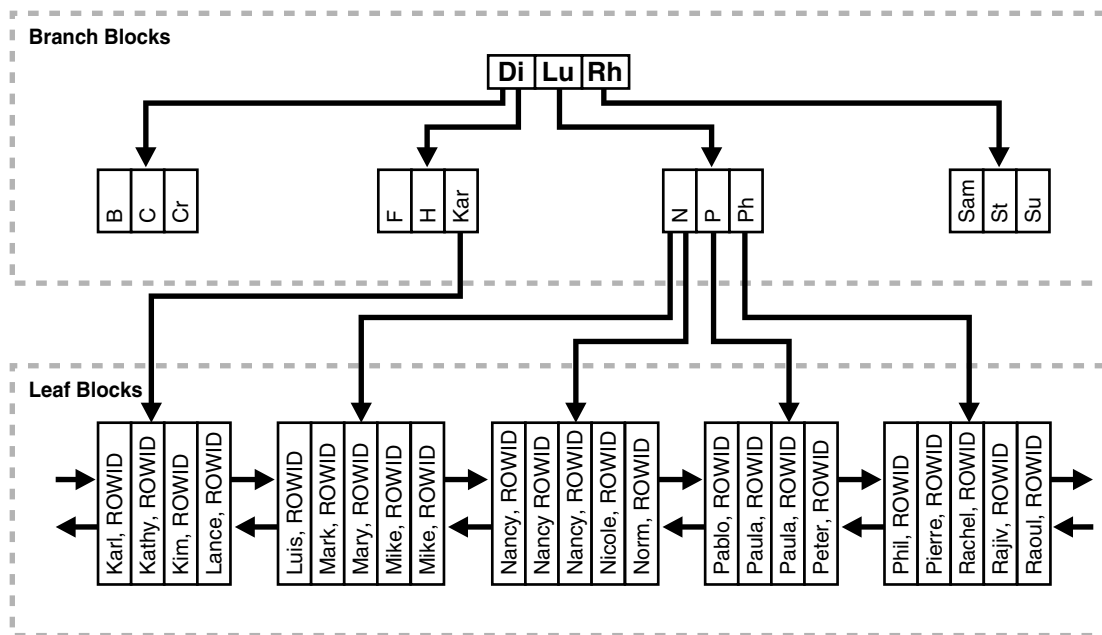
The upper blocks (**branch blocks**) of a B-tree index contain index data that points to lower-level index blocks. The lowest level index blocks (**leaf blocks**) contain every indexed data value and a corresponding rowid used to locate the actual row. The leaf blocks are doubly linked. Indexes in columns containing character data are based on the binary values of the characters in the database character set.

For a unique index, one rowid exists for each data value. For a nonunique index, the rowid is included in the key in sorted order, so nonunique indexes are sorted by the index key and rowid. Key values containing all nulls are not indexed, except for cluster indexes. Two rows can both contain all nulls without violating a unique index.

Index Properties

The two kinds of blocks:

- Branch blocks for searching
- Leaf blocks that store the values

Branch Blocks   Branch blocks store the following:

- The minimum key prefix needed to make a branching decision between two keys
- The pointer to the child block containing the key

If the blocks have n keys then they have n+1 pointers. The number of keys and pointers is limited by the block size.

Leaf Blocks All leaf blocks are at the same depth from the root branch block. Leaf blocks store the following:

- The complete key value for every row

- ROWIDs of the table rows

All key and ROWID pairs are linked to their left and right siblings. They are sorted by (key, ROWID).

Advantages of B-tree Structure

The B-tree structure has the following advantages:

- All leaf blocks of the tree are at the same depth, so retrieval of any record from anywhere in the index takes approximately the same amount of time.

- B-tree indexes automatically stay balanced.

- All blocks of the B-tree are three-quarters full on the average.

- B-trees provide excellent retrieval performance for a wide range of queries, including exact match and range searches.

- Inserts, updates, and deletes are efficient, maintaining key order for fast retrieval.

- B-tree performance is good for both small and large tables and does not degrade as the size of a table grows.

> **See Also:** Computer science texts for more information about B-tree indexes

Index Unique Scan

Index unique scan is one of the most efficient ways of accessing data. This access method is used for returning the data from B-tree indexes. The optimizer chooses a unique scan when all columns of a unique (B-tree) index are specified with equality conditions.

Index Range Scan

Index range scan is a common operation for accessing selective data. It can be bounded (bounded on both sides) or unbounded (on one or both sides). Data is returned in the ascending order of index columns. Multiple rows with identical values are sorted (in ascending order) by the ROWIDs.

Key Compression

Key compression lets you compress portions of the primary key column values in an index or index-organized table, which reduces the storage overhead of repeated values.

Generally, keys in an index have two pieces, a grouping piece and a unique piece. If the key is not defined to have a unique piece, Oracle Database provides one in the form of a rowid appended to the grouping piece. Key compression is a method of breaking off the grouping piece and storing it so it can be shared by multiple unique pieces.

This section includes the following topics:

- Prefix and Suffix Entries

- Performance and Storage Considerations

■   Uses of Key Compression

Prefix and Suffix Entries

Key compression breaks the index key into a prefix entry (the grouping piece) and a suffix entry (the unique piece). Compression is achieved by sharing the prefix entries among the suffix entries in an index block. Only keys in the leaf blocks of a B-tree index are compressed. In the branch blocks the key suffix can be truncated, but the key is not compressed.

Key compression is done within an index block but not across multiple index blocks. Suffix entries form the compressed version of index rows. Each suffix entry references a prefix entry, which is stored in the same index block as the suffix entry.

By default, the prefix consists of all key columns excluding the last one. For example, in a key made up of three columns (column1, column2, column3) the default prefix is (column1, column2). For a 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) in the prefix are compressed.

Alternatively, you can specify the prefix length, which is the number of columns in the prefix. For example, if you specify prefix length 1, then the prefix is column1 and the suffix is (column2, column3). 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 in the prefix are compressed.

The maximum prefix length for a nonunique index is the number of key columns, and the maximum prefix length for a unique index is the number of key columns minus one.

Prefix entries are written to the index block only if the index block does not already contain a prefix entry whose value is equal to the present prefix entry. Prefix entries are available for sharing immediately after being written to the index block and remain available until the last deleted referencing suffix entry is cleaned out of the index block.

Performance and Storage Considerations

Key compression can lead to a huge saving in space, letting you store more keys in each index block, which can lead to less I/O and better performance.

Although key compression reduces the storage requirements of an index, it can increase the CPU time required to reconstruct the key column values during an index scan. It also incurs some additional storage overhead, because every prefix entry has an overhead of 4 bytes associated with it.

Uses of Key Compression

Key compression is useful in many different scenarios, such as:

■   In a nonunique regular index, Oracle Database stores duplicate keys with the rowid appended to the key to break the duplicate rows. If key compression is used, Oracle Database stores the duplicate key as a prefix entry on the index block without the rowid. The rest of the rows are suffix entries that consist of only the rowid.

■   This same behavior can be seen in a unique index that has a key of the form (**item**, **time stamp**), for example (`stock_ticker`, `transaction_time`). Thousands of rows can have the same `stock_ticker` value, with `transaction_time` preserving uniqueness. On a particular index block a `stock_ticker` value is stored only once as a prefix entry. Other entries on the index block are `transaction_time` values stored as suffix entries that reference the common `stock_ticker` prefix entry.

■ In an index-organized table that contains a `VARRAY` or `NESTED TABLE` datatype, the object identifier is repeated for each element of the collection datatype. Key compression lets you compress the repeating object identifier values.

In some cases, however, key compression cannot be used. For example, in a unique index with a single attribute key, key compression is not possible, because even though there is a unique piece, there are no grouping pieces to share.

> **See Also:** "Overview of Index-Organized Tables" on page 5-36

Reverse Key Indexes

Creating a **reverse key index**, *c*ompared to a standard index, reverses the bytes of each column indexed (except the rowid) while keeping the column order. Such an arrangement can help avoid performance degradation with Oracle Real Application Clusters where modifications to the index are concentrated on a small set of leaf blocks. By reversing the keys of the index, the insertions become distributed across all leaf keys in the index.

Using the reverse key arrangement eliminates the ability to run an index range scanning query on the index. Because lexically adjacent keys are not stored next to each other in a reverse-key index, only fetch-by-key or full-index (table) scans can be performed.

Sometimes, using a reverse-key index can make an OLTP Oracle Real Application Clusters application faster. For example, keeping the index of mail messages in an e-mail application: some users keep old messages, and the index must maintain pointers to these as well as to the most recent.

The `REVERSE` keyword provides a simple mechanism for creating a reverse key index. You can specify the keyword `REVERSE` along with the optional index specifications in a `CREATE INDEX` statement:

```
CREATE INDEX i ON t (a,b,c) REVERSE;
```

You can specify the keyword `NOREVERSE` to `REBUILD` a reverse-key index into one that is not reverse keyed:

```
ALTER INDEX i REBUILD NOREVERSE;
```

Rebuilding a reverse-key index without the `NOREVERSE` keyword produces a rebuilt, reverse-key index.

Bitmap Indexes

The purpose of an index is to provide pointers to the rows in a table that contain a given key value. In a regular index, this is achieved by storing a list of rowids for each key corresponding to the rows with that key value. Oracle Database stores each key value repeatedly with each stored rowid. In a **bitmap index**, a bitmap for each key value is used instead of a list of rowids.

Each bit in the bitmap corresponds to a possible rowid. If the bit is set, then it means that the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid, so the bitmap index provides the same functionality as a regular index even though it uses a different representation internally. If the number of different key values is small, then bitmap indexes are very space efficient.

Bitmap indexing efficiently merges indexes that correspond to several conditions in a `WHERE` clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically.

This section includes the following topics:

- Benefits for Data Warehousing Applications

- Cardinality

- Bitmap Index Example

- Bitmap Indexes and Nulls

- Bitmap Indexes on Partitioned Tables

Benefits for Data Warehousing Applications

Bitmap indexing benefits data warehousing applications which have large amounts of data and ad hoc queries but a low level of concurrent transactions. For such applications, bitmap indexing provides:

- Reduced response time for large classes of ad hoc queries

- A substantial reduction of space use compared to other indexing techniques

- Dramatic performance gains even on very low end hardware

- Very efficient parallel DML and loads

Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of space, because the index can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

Bitmap indexes are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data. These indexes are primarily intended for decision support in data warehousing applications where users typically query the data rather than update it.

Bitmap indexes are also not suitable for columns that are primarily queried with less than or greater than comparisons. For example, a salary column that usually appears in `WHERE` clauses in a comparison to a certain value is better served with a B-tree index. Bitmapped indexes are only useful with equality queries, especially in combination with `AND`, `OR`, and `NOT` operators.

Bitmap indexes are integrated with the Oracle Database optimizer and execution engine. They can be used seamlessly in combination with other Oracle Database execution methods. For example, the optimizer can decide to perform a hash join between two tables using a bitmap index on one table and a regular B-tree index on the other. The optimizer considers bitmap indexes and other available access methods, such as regular B-tree indexes and full table scan, and chooses the most efficient method, taking parallelism into account where appropriate.

Parallel query and parallel DML work with bitmap indexes as with traditional indexes. Bitmap indexes on partitioned tables must be local indexes. Parallel create index and concatenated indexes are also supported.

Cardinality

The advantages of using bitmap indexes are greatest for low cardinality columns: that is, columns in which the number of distinct values is small compared to the number of rows in the table. If the number of distinct values of a column is less than 1% of the number of rows in the table, or if the values in a column are repeated more than 100

times, then the column is a candidate for a bitmap index. Even columns with a lower number of repetitions and thus higher cardinality can be candidates if they tend to be involved in complex conditions in the WHERE clauses of queries.

For example, on a table with 1 million rows, a column with 10,000 distinct values is a candidate for a bitmap index. A bitmap index on this column can out-perform a B-tree index, particularly when this column is often queried in conjunction with other columns.

B-tree indexes are most effective for high-cardinality data: that is, data with many possible values, such as CUSTOMER_NAME or PHONE_NUMBER. In some situations, a B-tree index can be larger than the indexed data. Used appropriately, bitmap indexes can be significantly smaller than a corresponding B-tree index.

In ad hoc queries and similar situations, bitmap indexes can dramatically improve query performance. AND and OR conditions in the WHERE clause of a query can be quickly resolved by performing the corresponding Boolean operations directly on the bitmaps before converting the resulting bitmap to rowids. If the resulting number of rows is small, the query can be answered very quickly without resorting to a full table scan of the table.

Bitmap Index Example

Table 5–1 shows a portion of a company's customer data.

Table 5–1   Bitmap Index Example

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

MARITAL_STATUS, REGION, GENDER, and INCOME_LEVEL are all low-cardinality columns. There are only three possible values for marital status and region, two possible values for gender, and four for income level. Therefore, it is appropriate to create bitmap indexes on these columns. A bitmap index should not be created on CUSTOMER# because this is a high-cardinality column. Instead, use a unique B-tree index on this column to provide the most efficient representation and retrieval.

Table 5–2 illustrates the bitmap index for the REGION column in this example. It consists of three separate bitmaps, one for each region.

Table 5–2   Sample Bitmap

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

Each entry or bit in the bitmap corresponds to a single row of the CUSTOMER table. The value of each bit depends upon the values of the corresponding row in the table. For instance, the bitmap REGION='east' contains a one as its first bit. This is because the region is east in the first row of the CUSTOMER table. The bitmap REGION='east' has a zero for its other bits because none of the other rows of the table contain east as their value for REGION.

An analyst investigating demographic trends of the company's customers can ask, "How many of our married customers live in the central or west regions?" This corresponds to the following SQL query:

```
SELECT COUNT(*) FROM CUSTOMER
    WHERE MARITAL_STATUS = 'married' AND REGION IN ('central','west');
```

Bitmap indexes can process this query with great efficiency by counting the number of ones in the resulting bitmap, as illustrated in Figure 5–8. To identify the specific customers who satisfy the criteria, the resulting bitmap can be used to access the table.

Figure 5–8   Running a Query Using Bitmap Indexes

Bitmap Indexes and Nulls

Bitmap indexes can include rows that have NULL values, unlike most other types of indexes. Indexing of nulls can be useful for some types of SQL statements, such as queries with the aggregate function COUNT.

Bitmap Indexes on Partitioned Tables

Like other indexes, you can create bitmap indexes on partitioned tables. The only restriction is that bitmap indexes must be local to the partitioned table—they cannot be global indexes. Global bitmap indexes are supported only on nonpartitioned tables.

See Also:

- *Oracle Database VLDB and Partitioning Guide* for information about partitioned tables and descriptions of local and global indexes

- *Oracle Database VLDB and Partitioning Guide*

- *Oracle Database Performance Tuning Guide* for more information about using bitmap indexes, including an example of indexing null values

Bitmap Join Indexes

In addition to a bitmap index on a single table, you can create a bitmap join index, which is a bitmap index for the join of two or more tables. A bitmap join index is a space efficient way of reducing the volume of data that must be joined by performing restrictions in advance. For each value in a column of a table, a bitmap join index stores the rowids of corresponding rows in one or more other tables. In a data warehousing environment, the join condition is an equi-inner join between the primary key column or columns of the dimension tables and the foreign key column or columns in the fact table.

Bitmap join indexes are much more efficient in storage than materialized join views, an alternative for materializing joins in advance. This is because the materialized join views do not compress the rowids of the fact tables.

See Also: *Oracle Database Data Warehousing Guide* for more information on bitmap join indexes

Overview of 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. 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.

As shown in Figure 5–9, the index-organized table is somewhat similar to a configuration consisting of an ordinary table and an index on one or more of the table columns, but instead of maintaining two separate storage structures, one for the table and one for the B-tree index, the database system maintains only a single B-tree index. Also, rather than having a row's rowid stored in the index entry, the nonkey column values are stored. Thus, each B-tree index entry contains `<primary_key_value, non_primary_key_column_values>`.

Figure 5–9   Structure of a Regular Table Compared with an Index-Organized Table

Regular Table and Index

Index
Finance ROWID
Invest ROWID

Table

| Finance | 5543 |
| Invest | 6879 |

Index-Organized Table

Index
Finance 5543
Invest 6879

Table Data Stored
in Index
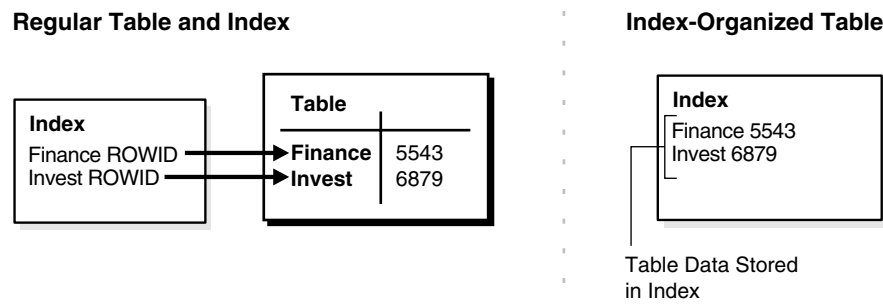
Applications manipulate the index-organized table just like an ordinary table, using SQL statements. However, the database system performs all operations by manipulating the corresponding B-tree index.

Table 5–3 summarizes the differences between index-organized tables and ordinary tables.

Table 5–3   Comparison of Index-Organized Tables with Ordinary Tables

Ordinary Table	Index-Organized Table
Rowid uniquely identifies a row. Primary key can be optionally specified	Primary key uniquely identifies a row. Primary key must be specified
Physical rowid in ROWID pseudocolumn allows building secondary indexes	Logical rowid in ROWID pseudocolumn allows building secondary indexes
Access is based on rowid	Access is based on logical rowid
Sequential scan returns all rows	Full-index scan returns all rows
Can be stored in a cluster with other tables	Cannot be stored in a cluster
Can contain a column of the LONG datatype and columns of LOB datatypes	Can contain LOB columns but not LONG columns
Can contain virtual columns (only relational heap tables are supported)	Cannot contain virtual columns

This section includes the following topics:

- Benefits of Index-Organized Tables
- Index-Organized Tables with Row Overflow Area
- Secondary Indexes on Index-Organized Tables
- Bitmap Indexes on Index-Organized Tables
- Partitioned Index-Organized Tables
- B-tree Indexes on UROWID Columns for Heap- and Index-Organized Tables
- Index-Organized Table Applications

Benefits of Index-Organized Tables

Index-organized tables provide faster access to table rows by the primary key or any key that is a valid prefix of the primary key. Presence of nonkey columns of a row in the B-tree leaf block itself avoids an additional block access. Also, because rows are stored in primary key order, range access by the primary key (or a valid prefix) involves minimum block accesses.

In order to allow even faster access to frequently accessed columns, you can use a row overflow segment (as described later) to push out infrequently accessed nonkey columns from the B-tree leaf block to an optional (heap-organized) overflow segment. This allows limiting the size and content of the portion of a row that is actually stored in the B-tree leaf block, which may lead to a higher number of rows in each leaf block and a smaller B-tree.

Unlike a configuration of heap-organized table with a primary key index where primary key columns are stored both in the table and in the index, there is no such duplication here because primary key column values are stored only in the B-tree index.

Because rows are stored in primary key order, a significant amount of additional storage space savings can be obtained through the use of key compression.

Use of primary-key based logical rowids, as opposed to physical rowids, in secondary indexes on index-organized tables allows high availability. This is because, due to the logical nature of the rowids, secondary indexes do not become unusable even after a table reorganization operation that causes movement of the base table rows. At the same time, through the use of physical guess in the logical rowid, it is possible to get secondary index based index-organized table access performance that is comparable to performance for secondary index based access to an ordinary table.

> **See Also:**
>
> - "Key Compression" on page 5-30
> - "Secondary Indexes on Index-Organized Tables" on page 5-38
> - *Oracle Database Administrator's Guide* for information about creating and maintaining index-organized tables

Index-Organized Tables with Row Overflow Area

B-tree index entries are usually quite small, because they only consist of the key value and a `ROWID`. In index-organized tables, however, the B-tree index entries can be large, because they consist of the entire row. This may destroy the dense clustering property of the B-tree index.

Oracle Database provides the `OVERFLOW` clause to handle this problem. You can specify an overflow tablespace so that, if necessary, a row can be divided into the following two parts that are then stored in the index and in the overflow storage area segment, respectively:

- The index entry, containing column values for all the primary key columns, a physical rowid that points to the overflow part of the row, and optionally a few of the nonkey columns
- The overflow part, containing column values for the remaining nonkey columns

With `OVERFLOW`, you can use two clauses, `PCTTHRESHOLD` and `INCLUDING`, to control how Oracle Database determines whether a row should be stored in two parts and if so, at which nonkey column to break the row. Using `PCTTHRESHOLD`, you can specify a threshold value as a percentage of the block size. If all the nonkey column values can be accommodated within the specified size limit, the row will not be broken into two parts. Otherwise, starting with the first nonkey column that cannot be accommodated, the rest of the nonkey columns are all stored in the row overflow segment for the table.

The `INCLUDING` clause lets you specify a column name so that any nonkey column, appearing in the `CREATE TABLE` statement after that specified column, is stored in the row overflow segment. Note that additional nonkey columns may sometimes need to be stored in the overflow due to `PCTTHRESHOLD`–based limits.

> **See Also:** *Oracle Database Administrator's Guide* for examples of using the `OVERFLOW` clause

Secondary Indexes on Index-Organized Tables

Secondary index support on index-organized tables provides efficient access to index-organized table using columns that are not the primary key nor a prefix of the primary key.

Oracle Database constructs secondary indexes on index-organized tables using logical row identifiers (**logical rowids**) that are based on the table's primary key. A logical rowid includes a **physical guess**, which identifies the block location of the row. Oracle Database can use these physical guesses to probe directly into the leaf block of the index-organized table, bypassing the primary key search. Because rows in index-organized tables do not have permanent physical addresses, the physical guesses can become stale when rows are moved to new blocks.

For an ordinary table, access by a secondary index involves a scan of the secondary index and an additional I/O to fetch the data block containing the row. For index-organized tables, access by a secondary index varies, depending on the use and accuracy of physical guesses:

- Without physical guesses, access involves two index scans: a secondary index scan followed by a scan of the primary key index.

- With accurate physical guesses, access involves a secondary index scan and an additional I/O to fetch the data block containing the row.

- With inaccurate physical guesses, access involves a secondary index scan and an I/O to fetch the wrong data block (as indicated by the physical guess), followed by a scan of the primary key index.

> **See Also:** "Logical Rowids" on page 26-17

Bitmap Indexes on Index-Organized Tables

Oracle Database supports bitmap indexes on partitioned and nonpartitioned index-organized tables. A mapping table is required for creating bitmap indexes on an index-organized table.

Mapping Table

The mapping table is a heap-organized table that stores logical rowids of the index-organized table. Specifically, each mapping table row stores one logical rowid for the corresponding index-organized table row. Thus, the mapping table provides one-to-one mapping between logical rowids of the index-organized table rows and physical rowids of the mapping table rows.

A bitmap index on an index-organized table is similar to that on a heap-organized table except that the rowids used in the bitmap index on an index-organized table are those of the mapping table as opposed to the base table. There is one mapping table for each index-organized table and it is used by all the bitmap indexes created on that index-organized table.

In both heap-organized and index-organized base tables, a bitmap index is accessed using a search key. If the key is found, the bitmap entry is converted to a physical rowid. In the case of heap-organized tables, this physical rowid is then used to access the base table. However, in the case of index-organized tables, the physical rowid is then used to access the mapping table. The access to the mapping table yields a logical rowid. This logical rowid is used to access the index-organized table.

Though a bitmap index on an index-organized table does not store logical rowids, it is still logical in nature.

> **Note:** Movement of rows in an index-organized table does not leave the bitmap indexes built on that index-organized table unusable. Movement of rows in the index-organized table does invalidate the physical guess in some of the mapping table's logical rowid entries. However, the index-organized table can still be accessed using the primary key.

Partitioned Index-Organized Tables

You can partition an index-organized table by RANGE, HASH, or LIST on column values. The partitioning columns must form a subset of the primary key columns. Just like ordinary tables, local partitioned (prefixed and non-prefixed) index as well as global partitioned (prefixed) indexes are supported for partitioned index-organized tables.

B-tree Indexes on UROWID Columns for Heap- and Index-Organized Tables

UROWID datatype columns can hold logical primary key-based rowids identifying rows of index-organized tables. Oracle Database supports indexes on UROWID datatypes of a heap- or index-organized table. The index supports equality predicates on UROWID columns. For predicates other than equality or for ordering on UROWID datatype columns, the index is not used.

Index-Organized Table Applications

The superior query performance for primary key based access, high availability aspects, and reduced storage requirements make index-organized tables ideal for the following kinds of applications:

- Online transaction processing (OLTP)
- Internet (for example, search engines and portals)
- E-commerce (for example, electronic stores and catalogs)
- Data warehousing
- Analytic functions

Overview of Application Domain Indexes

Oracle Database provides **extensible indexing** to accommodate indexes on customized complex datatypes such as documents, spatial data, images, and video clips and to make use of specialized indexing techniques. With extensible indexing, you can encapsulate application-specific index management routines as an **indextype** schema object and define a **domain index** (an application-specific index) on table columns or attributes of an object type. Extensible indexing also provides efficient processing of application-specific **operators**.

The application software, called the **cartridge**, controls the structure and content of a domain index. The Oracle database server interacts with the application to build, maintain, and search the domain index. The index structure itself can be stored in the Oracle database as an index-organized table or externally as a file.

> **See Also:** *Oracle Database Data Cartridge Developer's Guide* for information about using data cartridges within the Oracle database extensibility architecture

Overview of Clusters

Clusters are an optional method of storing table data. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. For example, the `employees` and `departments` table share the `department_id` column. When you cluster the `employees` and `departments` tables, Oracle Database physically stores all rows for each department from both the `employees` and `departments` tables in the same data blocks.

Figure 5–10 shows what happens when you cluster the `employees` and `departments` tables:

Figure 5–10   Clustered Table Data
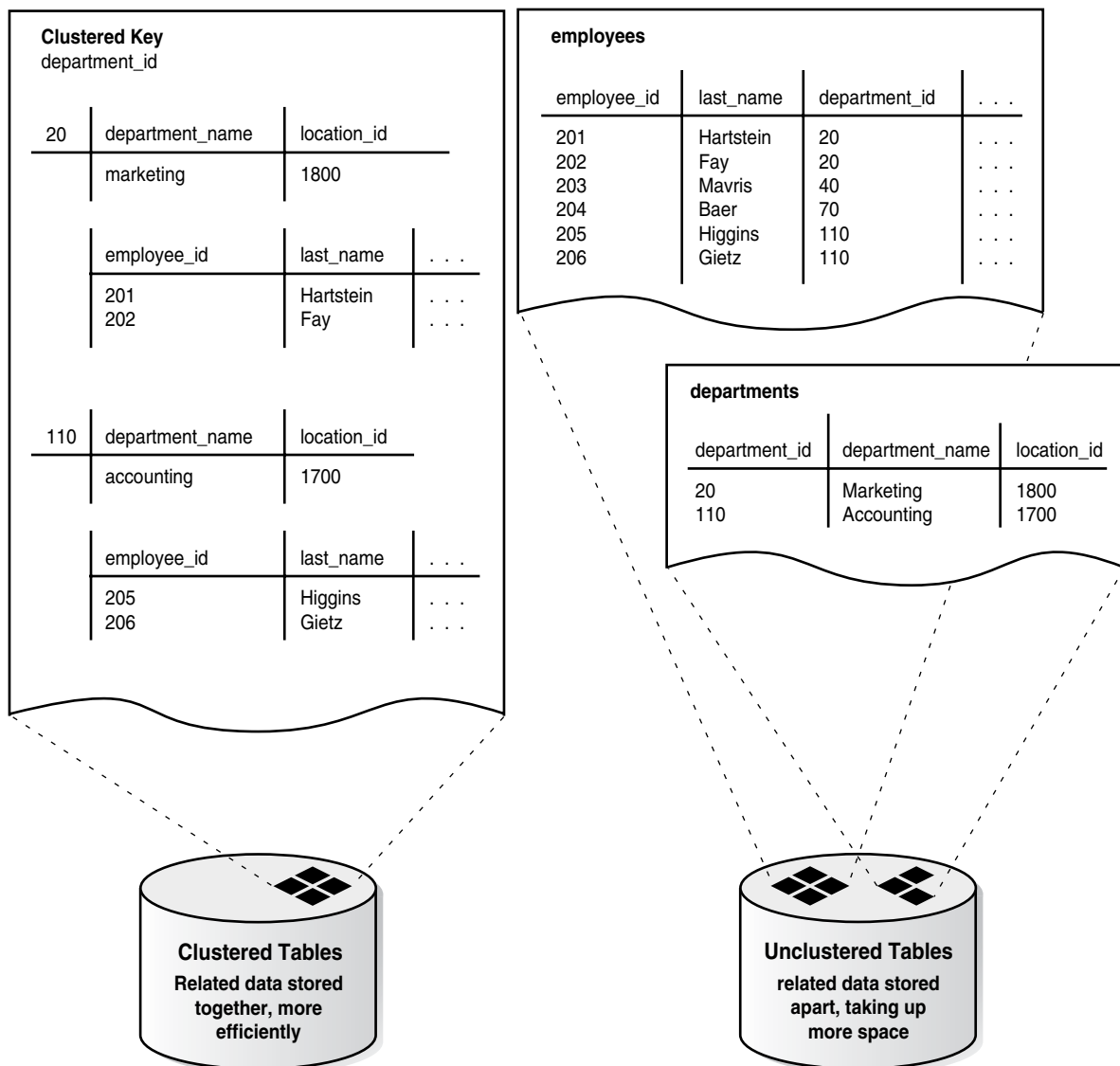
Because clusters store related rows of different tables together in the same data blocks, properly used clusters offers these benefits:

- Disk I/O is reduced for joins of clustered tables.

- Access time improves for joins of clustered tables.

■ In a cluster, a **cluster key value** is the value of the cluster key columns for a particular row. 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 is required to store related table and index data in a cluster than is necessary in nonclustered table format. For example, in Figure 5–10, notice how each cluster key (each department_id) is stored just once for many rows that contain the same value in both the employees and departments tables.

> **See Also:** *Oracle Database Administrator's Guide* for information about creating and managing clusters

Overview of Hash Clusters

Hash clusters group table data in a manner similar to regular index clusters (clusters keyed with an index rather than a hash function). However, a row is stored in a hash cluster based on the result of applying a **hash function** to the row's cluster key value. All rows with the same key value are stored together on disk.

Hash clusters are a better choice than using an indexed table or index cluster when a table is queried frequently with equality queries (for example, return all rows for department 10). For such queries, the specified cluster key value is hashed. The resulting hash key value points directly to the area on disk that stores the rows.

Hashing is an optional way of storing table data to improve the performance of data retrieval. To use hashing, create a hash cluster and load tables into the cluster. Oracle Database physically stores the rows of a table in a hash cluster and retrieves them according to the results of a hash function.

Sorted hash clusters allow faster retrieval of data for applications where data is consumed in the order in which it was inserted.

Oracle Database uses a hash function to generate a distribution of numeric values, called **hash values**, which 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, Oracle Database applies the hash function to the row's cluster key value. The resulting hash value corresponds to a data block in the cluster, which Oracle Database then reads or writes on behalf of the issued statement.

A hash cluster is an alternative to a nonclustered 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 Oracle Database stores in a separate index. To find or store a row in an indexed table or cluster, at least two 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

> **See Also:** *Oracle Database Administrator's Guide* for information about creating and managing hash clusters

6

Schema Object Dependencies

If the definition of object A references object B, then A depends on B. This chapter explains dependencies among schema objects, and how Oracle Database automatically tracks and manages these dependencies. Because of this automatic dependency management, A never uses an obsolete version of B, and you almost never have to explicitly recompile A after you change B.

Topics:

- Overview of Schema Object Dependencies
- Querying Object Dependencies
- Object Status
- Invalidation of Dependent Objects
- Guidelines for Reducing Invalidation
- Object Revalidation
- Name Resolution in Schema Scope
- Local Dependency Management
- Remote Dependency Management
- Remote Procedure Call (RPC) Dependency Management
- Shared SQL Dependency Management

Overview of Schema Object Dependencies

Some types of schema objects can reference other objects in their definitions. For example, a view is defined by a query that references tables or other views, and the body of a subprogram can include SQL statements that reference other objects. If the definition of object A references object B, then A is a **dependent object** (with respect to B) and B is a **referenced object** (with respect to A). Table 6–1 shows which object types can be dependent or referenced.

Table 6–1   Dependent and Referenced Object Types

Object Type	Can Be Dependent or Referenced
Package body	Dependent only
Package specification	Both
Sequence	Referenced only
Subprogram	Both

Table 6–1   (Cont.) Dependent and Referenced Object Types

Object Type	Can Be Dependent or Referenced
Synonym	Both
Table	Both
Trigger	Both
User-defined object	Both
User-defined collection	Both
View	Both

If you alter the definition of a referenced object, dependent objects might or might not continue to function without error, depending on the type of alteration. For example, if you drop a table, no view based on the dropped table is usable.

As an example of a schema object change that invalidates some dependents but not others, consider the two views in Example 6–1, which are based on the HR.EMPLOYEES table.

Suppose you determine that the EMAIL column in the EMPLOYEES table needs to be lengthened. You alter the table as follows:

```
ALTER TABLE employees MODIFY email VARCHAR2(100);
```

Because the COMMISSIONED view does not include EMAIL in its select list, it is not invalidated. However, because the SIXFIGURES view selects all columns in the table, it is invalidated.

```
select object_name, status from user_objects where object_type = 'VIEW';


OBJECT_NAME          STATUS
------------------- -------
COMMISSIONED         VALID
SIXFIGURES           INVALID
```

Example 6–1   Schema Object Change that Invalidates Some Dependents

```
CREATE VIEW commissioned AS
SELECT first_name, last_name, commission_pct FROM employees
WHERE commission_pct > 0.00;

CREATE VIEW sixfigures AS
SELECT * FROM employees
WHERE salary >= 100000;

select object_name, status from user_objects where object_type = 'VIEW';


OBJECT_NAME          STATUS
------------------- -------
COMMISSIONED         VALID
SIXFIGURES           VALID
```

A view depends on every object referenced in its query. The view in Example 6–2, oc_inventories, depends on the object type inventory_typ, the function warehouse_typ, and the tables inventories and warehouse.

Example 6–2   View that Depends on Multiple Objects

```
CREATE TYPE inventory_typ
```

```
  OID '82A4AF6A4CD4656DE034080020E0EE3D'
  AS OBJECT
    ( product_id        NUMBER(6)
    , warehouse         warehouse_typ
    , quantity_on_hand  NUMBER(8)
    ) ;
/
CREATE OR REPLACE VIEW oc_inventories OF inventory_typ
  WITH OBJECT OID (product_id)
  AS SELECT i.product_id,
            warehouse_typ(w.warehouse_id, w.warehouse_name, w.location_id),
            i.quantity_on_hand
     FROM inventories i, warehouses w
     WHERE i.warehouse_id=w.warehouse_id;
```

> **Notes:**
>
> - CREATE statements automatically update all dependencies.
>
> - Dynamic SQL statements do not create dependencies. For example, the following statement does not create a dependency on tab1:
>
>   EXECUTE IMMEDIATE 'SELECT * FROM tab1 ...'

Querying Object Dependencies

The static data dictionary views USER_DEPENDENCIES, ALL_DEPENDENCIES, and DBA_DEPENDENCIES describe dependencies between database objects.

The utldtree.sql SQL script creates the view DEPTREE, which contains information on the object dependency tree, and the view IDEPTREE, a presorted, pretty-print version of DEPTREE.

> **See Also:** *Oracle Database Reference* for more information about the DEPTREE, IDEPTREE, and utldtree.sql script

Object Status

Every database object has one of the status values described in Table 6–2.

Table 6–2 Database Object Status

Status	Meaning
Valid	The object was successfully compiled, using the current definition in the data dictionary.
Compiled with errors	The most recent attempt to compile the object produced errors.
Invalid	The object is marked invalid because an object that it references has changed. (Only a dependent object can be invalid.)
Unauthorized	An access privilege on a referenced object was revoked. (Only a dependent object can be unauthorized.)

> **Note:** The static data dictionary views USER_OBJECTS, ALL_OBJECTS, and DBA_OBJECTS do not distinguish between "Compiled with errors," "Invalid," and "Unauthorized"—they describe all of these as INVALID.

Invalidation of Dependent Objects

If object A depends on object B, which depends on object C, then A is a **direct dependent** of B, B is a direct dependent of C, and A is an **indirect dependent** of C.

Direct dependents are invalidated only by changes to the referenced object that affect them (changes to the signature of the referenced object).

Indirect dependents can be invalidated by changes to the reference object that do not affect them: If a change to C invalidates B, it invalidates A (and all other direct and indirect dependents of B). This is called **cascading invalidation**.

Table 6–3 shows how objects are affected by changes to other objects on which they depend.

Table 6–3   Operations that Affect Object Status

Operation	Resulting Status of Dependent Objects
ALTER TABLE *table* ADD *column*	INVALID when: ■ Dependent object is not a view and uses SELECT * on *table*. ■ Dependent object uses *table*%rowtype. ■ Dependent object performs INSERT on *table* without specifying column list. ■ Dependent object references *table* in query that contains a SQL join. ■ Dependent object references *table* in query that references a PL/SQL variable. Otherwise, no change.
ALTER TABLE *table* {MODIFY|RENAME|DROP|SET UNUSED} *column* ALTER TABLE *table* DROP CONSTRAINT *not_null_constraint*	INVALID when: ■ Dependent object directly references *column*. ■ Dependent object uses SELECT * on *table*. ■ Dependent object uses *table*%rowtype. ■ Dependent object performs INSERT on *table* without specifying column list. Otherwise, no change.

Table 6–3   (Cont.) Operations that Affect Object Status

Operation	Resulting Status of Dependent Objects
CREATE OR REPLACE VIEW *view* Online Table Redefinition (DBMS_ REDEFINITION)	INVALID when column lists of new and old definitions differ, and at least one of the following is true: ■ Dependent object references column that is modified or dropped in new view or table definition. ■ Dependent object uses *view*%rowtype or *table*%rowtype. ■ Dependent object performs INSERT on view or table without specifying column list. ■ New view definition introduces new columns, and dependent object references view or table in query that contains a SQL join. ■ New view definition introduces new columns, and dependent object references view or table in query that references a PL/SQL variable. ■ Dependent object references view or table in RELIES ON clause. Otherwise, no change.
CREATE OR REPLACE SYNONYM *synonym*	INVALID when: ■ New and old *synonym* targets differ, and one is not a table. ■ Both old and new *synonym* targets are tables, and the tables have different column lists or different privilege grants. ■ Both old and new *synonym* targets are tables, and dependent object is a view that references a column that participates in a unique index on the old target but not in a unique index on the new target. Otherwise, no change.
RENAME {TABLE│VIEW│SEQUENCE│SYNONYM}	INVALID
DROP INDEX	A dependent of the table on which the index is built becomes INVALID when: ■ The index is a function-based index and the dependent object is a trigger. ■ The index is a unique index, the dependent object is a view, and the view references a column participating in the unique index.
DROP *object*	INVALID

Table 6–3   (Cont.) Operations that Affect Object Status

Operation	Resulting Status of Dependent Objects
`CREATE OR REPLACE {PROCEDURE｜FUNCTION}`	`INVALID` if the call signature changes. The call signature is the parameter list (order, names, and types of parameters), return type, purity[1], determinism, parallelism, pipelining, and (if the procedure or function is implemented in C or Java) implementation properties. Valid for other changes, including changes to the procedure or function body.
`CREATE OR REPLACE PACKAGE`	`INVALID` when: ■ Dependent object references a dropped or renamed package item. ■ Dependent object references a package procedure or function whose call signature or entry-point number[2], changed. If referenced procedure or function has multiple overload candidates, dependent object is invalidated if any overload candidate's call signature or entry point number changed, or if a candidate was added or dropped. ■ Dependent object references a package cursor whose call signature, rowtype, or entry point number changed. ■ Dependent object references a package type or subtype whose definition changed. ■ Dependent object references a package variable or constant whose name, datatype, initial value, or offset number changed. ■ Package purity[1] changed. Otherwise, no change.
`CREATE OR REPLACE PACKAGE BODY`	No change.
`REVOKE DML-object-privilege`[3] `ON object FROM user`	All objects of *user* that depend on *object* are `INVALID`.[4]
`REVOKE DML-object-privilege`[3] `ON object FROM PUBLIC`	All objects in database that depend on *object* are `INVALID`.[4]

[1] **Purity** refers to a set of rules for preventing side effects (such as unexpected data changes) when invoking PL/SQL functions within SQL queries. **Package purity** refers to the purity of the code in the package initialization block.

[2] The **entry-point number** of a procedure or function is determined by its location in the PL/SQL package code. A procedure or function added to the end of a PL/SQL package is given a new entry-point number.

[3] DML object privileges are SELECT, INSERT, UPDATE, DELETE, and EXECUTE.

[4] Revalidation does not require recompilation. For explanation, see "Fast Revalidation of Invalid PL/SQL Objects" on page 6-8.

Topics:

■ Session State and Referenced Packages

■ Security Authorization

Session State and Referenced Packages

Each session that references a package construct has its own instantiation of that package, including a persistent state of any public and private variables, cursors, and constants. All of a session's package instantiations, including state, can be lost if any of the session's instantiated packages are subsequently invalidated and revalidated.

Security Authorization

Oracle Database notices when a DML object or system privilege is granted to or revoked from a user or PUBLIC and automatically invalidates all the owner's dependent objects. Oracle Database invalidates the dependent objects to verify that an owner of a dependent object continues to have the necessary privileges for all referenced objects.

Guidelines for Reducing Invalidation

To reduce invalidation of dependent objects, follow these guidelines:

- Add New Items to End of Package
- Reference Each Table Through a View

Add New Items to End of Package

When adding new items to a package, add them to the end of the package. This preserves the entrypoint numbers of existing top-level package items, preventing their invalidation.

For example, consider the following package:

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
END;
```

Adding an item to the end of pkg1, as follows, does not invalidate dependents that reference the get_var function:

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
```

Inserting an item between the get_var function and the set_var procedure, as follows, invalidates dependents that reference the set_var function:

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE assert_var (v VARCHAR2);
  PROCEDURE set_var (v VARCHAR2);
END;
```

Reference Each Table Through a View

Reference tables indirectly, using views. This allows you to do the following:

- Add columns to the table without invalidating dependent views or dependent PL/SQL objects
- Modify or delete columns not referenced by the view without invalidating dependent objects

The statement CREATE OR REPLACE VIEW does not invalidate an existing view or its dependents if the new ROWTYPE matches the old ROWTYPE.

Object Revalidation

An object that is not valid when it is referenced must be validated before it can be used. Validation occurs automatically when an object is referenced; it does not require explicit user action.

If an object is not valid, its status is either compiled with errors, unauthorized, or invalid. For definitions of these terms, see Table 6–2.

Topics:

- Revalidation of Objects that Compiled with Errors
- Revalidation of Unauthorized Objects
- Revalidation of Invalid SQL Objects
- Revalidation of Invalid PL/SQL Objects
- Fast Revalidation of Invalid PL/SQL Objects

Revalidation of Objects that Compiled with Errors

The compiler cannot automatically revalidate an object that compiled with errors. The compiler recompiles the object, and if it recompiles without errors, it is revalidated; otherwise, it remains invalid.

Revalidation of Unauthorized Objects

The compiler checks whether the unauthorized object has access privileges to all of its referenced objects. If so, the compiler revalidates the unauthorized object without recompiling it. If not, the compiler issues appropriate error messages.

Revalidation of Invalid SQL Objects

The SQL compiler recompiles the invalid object. If the object recompiles without errors, it is revalidated; otherwise, it remains invalid.

Revalidation of Invalid PL/SQL Objects

For an invalid PL/SQL program unit (procedure, function, or package), the PL/SQL compiler checks whether any referenced object changed in a way that affects the invalid object. If so, the compiler recompiles the invalid object. If the object recompiles without errors, it is revalidated; otherwise, it remains invalid. If not, the compiler revalidates the invalid object without recompiling it—see "Fast Revalidation of Invalid PL/SQL Objects".

Fast Revalidation of Invalid PL/SQL Objects

For an invalid PL/SQL program unit (procedure, function, or package), the PL/SQL compiler checks whether any referenced object changed in a way that affects the invalid object. If not, the compiler revalidates the invalid object without recompiling it. Fast revalidation is usually performed on objects that were invalidated due to cascading invalidation.

For example, consider the following table, package, and procedure:

```
CREATE TABLE tab1(n NUMBER);

CREATE OR REPLACE PACKAGE pkg1 IS
```

```
    TYPE rec1 IS tab1%ROWTYPE;  -- pkg1 depends on tab1
    PROCEDURE p(n NUMBER);
END pkg1;

CREATE OR REPLACE PROCEDURE proc1 IS
BEGIN
    pkg1.p(5);  -- proc1 depends on pkg1
END proc1;
```

The following statement invalidates pkg1 (which depends on tab1), and this invalidation cascades to proc1 (which depends on pkg1):

```
ALTER TABLE tab1 ADD(v VARCHAR2(20));
```

However, because the signature of pkg1.p has not changed, the PL/SQL compiler can revalidate proc1 without recompiling it.

Name Resolution in Schema Scope

Object names referenced in SQL statements have one or more pieces. Pieces are separated by periods—for example, hr.employees.department_id.

Oracle Database uses the following procedure to try to resolve an object name:

1. Try to qualify the **first piece** of the object name.

    If the object name has only one piece, then that piece is the first piece. Otherwise, the first piece is the piece to the left of the leftmost period; for example, in hr.employees.department_id, hr is the first piece.

    The procedure for trying to qualify the first piece is:

    a. If the object name is a table name that appears in the FROM clause of a SELECT statement, and the object name has more than one piece, go to step d. Otherwise, go to step b.

    b. Search the current schema for an object whose name matches the first piece.

        If found, go to step 2. Otherwise, go to step c.

    c. Search for a public synonym that matches the first piece.

        If found, go to step 2. Otherwise, go to step d.

    d. Search for a schema whose name matches the first piece.

        If found, and if the object name has a second piece, go to step e. Otherwise, return an error—the object name cannot be qualified.

    e. Search the schema found at step d for a built-in function whose name matches the second piece of the object name.

        If found, this means that the schema redefined that built-in function. The object name resolves to the original built-in function, not to the schema-defined function of the same name. Go to step 2.

        If not found, return an error—the object name cannot be qualified.

2. A schema object has been qualified. Any remaining pieces of the object name must match a valid part of this schema object.

    For example, if the object name is hr.employees.department_id, hr is qualified as a schema. If employees is qualified as a table, department_id must correspond to a column of that table. If employees is qualified as a package,

department_id must correspond to a public constant, variable, procedure, or function of that package.

Because of how Oracle Database resolves references, an object can depend on the nonexistence of other objects. This situation occurs when the dependent object uses a reference that would be interpreted differently if another object were present.

See Also: *Oracle Database Administrator's Guide* for more details

Local Dependency Management

Local dependency management occurs when Oracle Database manages dependencies among the objects in a single database. For example, a statement in a procedure can reference a table in the same database.

Remote Dependency Management

Remote dependency management occurs when Oracle Database manages dependencies in distributed environments across a network. For example, an Oracle Forms trigger can depend on a schema object in the database. In a distributed database, a local view's defining query can reference a remote table.

Oracle Database also manages distributed database dependencies. For example, an Oracle Forms application might contain a trigger that references a table. The database system must account for dependencies among such objects. Oracle Database uses different mechanisms to manage remote dependencies, depending on the objects involved.

Topics:

- Dependencies Among Local and Remote Database Procedures
- Dependencies Among Other Remote Objects
- Dependencies of Applications

Dependencies Among Local and Remote Database Procedures

Dependencies among stored procedures (including functions, packages, and triggers) in a distributed database system are managed using either time-stamp checking or signature checking (see "Time-Stamp Checking" on page 6-11 and "Signature Checking" on page 6-13).

The dynamic initialization parameter REMOTE_DEPENDENCIES_MODE determines whether time stamps or signatures govern remote dependencies.

See Also: *Oracle Database PL/SQL Language Reference*

Dependencies Among Other Remote Objects

Oracle Database does not manage dependencies among remote schema objects other than local-procedure-to-remote-procedure dependencies.

For example, assume that a local view is created and defined by a query that references a remote table. Also assume that a local procedure includes a SQL statement that references the same remote table. Later, the definition of the table is altered.

As a result, the local view and procedure are never invalidated, even if the view or procedure is used after the table is altered, and even if the view or procedure now returns errors when used. In this case, the view or procedure must be altered manually

so that errors are not returned. In such cases, lack of dependency management is preferable to unnecessary recompilations of dependent objects.

Dependencies of Applications

Code in database applications can reference objects in the connected database. For example, OCI and precompiler applications can submit anonymous PL/SQL blocks. Triggers in Oracle Forms applications can reference a schema object.

Such applications are dependent on the schema objects they reference. Dependency management techniques vary, depending on the development environment. Oracle Database does not automatically track application dependencies.

> **See Also:** Manuals for your application development tools and your operating system for more information about managing the remote dependencies within database applications

Remote Procedure Call (RPC) Dependency Management

Remote procedure call (RPC) dependency management occurs when a local stored procedure calls a remote procedure in a distributed database system.

Topics:

- Time-Stamp Checking
- Signature Checking
- Controlling Remote Dependencies

Time-Stamp Checking

In the time-stamp checking dependency model, whenever a procedure is compiled or recompiled, its **time stamp** (the time it is created, altered, or replaced) is recorded in the data dictionary. The time stamp is a record of the time the procedure is created, altered, or replaced. Additionally, the compiled version of the procedure contains information about each remote procedure that it references, including the remote procedure's schema, package name, procedure name, and time stamp.

When a dependent procedure is used, Oracle Database compares the remote time stamps recorded at compile time with the current time stamps of the remotely referenced procedures. Depending on the result of this comparison, two situations can occur:

- The local and remote procedures run without compilation if the time stamps match.

- The local procedure is invalidated if any time stamps of remotely referenced procedures do not match, and an error is returned to the calling environment. Furthermore, all other local procedures that depend on the remote procedure with the new time stamp are also invalidated. For example, assume several local procedures call a remote procedure, and the remote procedure is recompiled. When one of the local procedures is run and notices the different time stamp of the remote procedure, every local procedure that depends on the remote procedure is invalidated.

Actual time stamp comparison occurs when a statement in the body of a local procedure runs a remote procedure. Only at this moment are the time stamps compared using the distributed database's communications link. Therefore, all statements in a local procedure that precede an invalid procedure call might run

successfully. Statements subsequent to an invalid procedure call do not run at all. Compilation is required.

Depending on how the invalid procedure is called, DML statements run before the invalid procedure call are rolled back. For example, in the following, the UPDATE results are rolled back as the complete PL/SQL block changes are rolled back.

```
BEGIN
UPDATE table set ...
invalid_proc;
COMMIT;
END;
```

However, with the following, the UPDATE results are final. Only the PROC call is rolled back.

```
UPDATE table set ...
EXECUTE invalid_proc;
COMMIT;
```

If time stamps are used to handle dependencies among PL/SQL program units, then whenever you alter a program unit or a relevant schema object, all of its dependent units are marked as invalid and must be recompiled before they can be run.

Each program unit carries a time stamp that is set by the server when the unit is created or recompiled. Figure 6–1 demonstrates this graphically. Procedures P1 and P2 call stored procedure P3. Stored procedure P3 references table T1. In this example, each of the procedures is dependent on table T1. P3 depends upon T1 directly, while P1 and P2 depend upon T1 indirectly.

Figure 6–1   Dependency Relationships
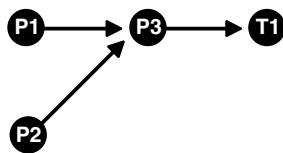
If P3 is altered, then P1 and P2 are marked as invalid immediately, if they are on the same server as P3. The compiled states of P1 and P2 contain records of the time stamp of P3. Therefore, if the procedure P3 is altered and recompiled, then the time stamp on P3 no longer matches the value that was recorded for P3 during the compilation of P1 and P2.

If P1 and P2 are on a client system, or on another Oracle Database instance in a distributed environment, then the time stamp information is used to mark them as invalid at run time.

The disadvantage of this dependency model is that it is unnecessarily restrictive. Recompilation of dependent objects across the network are often performed when not strictly necessary, leading to performance degradation.

Furthermore, on the client side, the time stamp model can lead to situations that block an application from running at all, if the client-side application is built using PL/SQL version 2. Earlier releases of tools, such as Oracle Forms, that used PL/SQL version 1 on the client side did not use this dependency model, because PL/SQL version 1 had no support for stored procedures.

For releases of Oracle Forms that are integrated with PL/SQL version 2 on the client side, the time stamp model can present problems. For example, during the installation

of the application, the application is rendered invalid unless the client-side PL/SQL procedures that it uses are recompiled at the client site. Also, if a client-side procedure depends on a server procedure, and if the server procedure is changed or automatically recompiled, then the client-side PL/SQL procedure must then be recompiled. Yet in many application environments (such as Forms run-time applications), there is no PL/SQL compiler available on the client. This blocks the application from running at all. The client application developer must then redistribute new versions of the application to all customers.

Signature Checking

Oracle Database provides the additional capability of remote dependencies using **RPC signatures**. The RPC signature capability affects only remote dependencies. Local dependencies are not affected, as recompilation is always possible in this environment.

The RPC signature of a procedure contains information about the following items:

- Name of the package, procedure, or function

- Base types of the parameters

- Modes of the parameters (IN, OUT, and IN OUT)

> **Note:** Only the types and modes of parameters are significant. The name of the parameter does not affect the RPC signature.

If the RPC signature dependency model is in effect, a dependency on a remote program unit causes an invalidation of the dependent unit if the dependent unit contains a call to a procedure in the parent unit, and the RPC signature of this procedure has been changed in an incompatible manner. A program unit can be a package, stored procedure, stored function, or trigger.

To alleviate some of the problems with the time-stamp-only dependency model, Oracle Database provides the additional capability of remote dependencies using RPC signatures. The RPC signature capability affects only remote dependencies. Local (same server) dependencies are not affected, as recompilation is always possible in this environment.

An RPC signature is associated with each compiled stored program unit. It identifies the unit using the following criteria:

- The name of the unit (the package, procedure, or function name).

- The types of each of the parameters of the subprogram.

- The modes of the parameters (IN, OUT, IN OUT).

- The number of parameters.

- The type of the return value for a function.

The user has control over whether RPC signatures or time stamps govern remote dependencies.

> **See Also:** "Controlling Remote Dependencies" on page 6-18

When the RPC signature dependency model is used, a dependency on a remote program unit causes an invalidation of the dependent unit if the dependent unit contains a call to a subprogram in the parent unit, and if the RPC signature of this subprogram has been changed in an incompatible manner.

For example, consider a procedure `get_emp_name` stored on a server in Boston (`BOSTON_SERVER`). The procedure is defined as the following:

> **Note:** You might need to set up data structures, similar to the following, for certain examples to work:
>
> ```
> CONNECT system/manager
> CREATE PUBLIC DATABASE LINK boston_server USING 'inst1_alias';
> CONNECT scott/tiger
> ```

```
CREATE OR REPLACE PROCEDURE get_emp_name (
   emp_number   IN  NUMBER,
   hire_date    OUT VARCHAR2,
   emp_name     OUT VARCHAR2) AS
BEGIN
   SELECT ename, to_char(hiredate, 'DD-MON-YY')
      INTO emp_name, hire_date
      FROM emp
      WHERE empno = emp_number;
END;
```

When `get_emp_name` is compiled on `BOSTON_SERVER`, its RPC signature, as well as its time stamp, is recorded.

Suppose that on another server in California, some PL/SQL code calls `get_emp_name` identifying it using a DBlink called `BOSTON_SERVER`, as follows:

```
CREATE OR REPLACE PROCEDURE print_ename (emp_number IN NUMBER) AS
   hire_date    VARCHAR2(12);
   ename        VARCHAR2(10);
BEGIN
   get_emp_name@BOSTON_SERVER(emp_number, hire_date, ename);
   dbms_output.put_line(ename);
   dbms_output.put_line(hire_date);
END;
```

When this California server code is compiled, the following actions take place:

- A connection is made to the Boston server.

- The RPC signature of `get_emp_name` is transferred to the California server.

- The RPC signature is recorded in the compiled state of `print_ename`.

At run time, during the remote procedure call from the California server to the Boston server, the recorded RPC signature of `get_emp_name` that was saved in the compiled state of `print_ename` gets sent to the Boston server, regardless of whether or not there were any changes.

If the timestamp dependency mode is in effect, then a mismatch in time stamps causes an error status to be returned to the calling procedure.

However, if the RPC signature mode is in effect, then any mismatch in time stamps is ignored, and the recorded RPC signature of `get_emp_name` in the compiled state of `Print_ename` on the California server is compared with the current RPC signature of `get_emp_name` on the Boston server. If they match, then the call succeeds. If they do not match, then an error status is returned to the `print_name` procedure.

The `get_emp_name` procedure on the Boston server could have changed, or its time stamp could be different from that recorded in the `print_name` procedure on the California server, possibly due to the installation of a new release of the server. As long

as the RPC signature remote dependency mode is in effect on the California server, a time stamp mismatch does not cause an error when `get_emp_name` is called.

> **Note:** `DETERMINISTIC`, `PARALLEL_ENABLE`, and purity information do not show in the RPC signature mode. Optimizations based on these settings are not automatically reconsidered if a function on a remote system is redefined with different settings. This might lead to incorrect query results when calls to the remote function occur, even indirectly, in a SQL statement, or if the remote function is used, even indirectly, in a function-based index.

Topics:

- Switching Datatype Classes
- Examples of Changing Procedure Signatures

Switching Datatype Classes

A RPC signature changes when you switch from one datatype class to another. A datatype class can include several datatypes. Changing a parameter datatype to another datatype in a class does not change the RPC signature.

Table 6–4 lists the datatype classes and the datatypes that comprise them. Datatypes that are not listed in Table 6–4, such as `NCHAR` or `TIMESTAMP`, are not part of any class; changing their type always causes a RPC signature mismatch.

Table 6–4   Datatype Classes

Datatype Class	Datatypes in Class
Character	CHAR CHARACTER
VARCHAR	VARCHAR VARCHAR2 STRING LONG ROWID
Raw	RAW LONG RAW
Integer	BINARY_INTEGER PLS_INTEGER SIMPLE_INTEGER BOOLEAN NATURAL NATURALN POSITIVE POSITIVEN

Table 6–4   (Cont.) Datatype Classes

Datatype Class	Datatypes in Class
Number	NUMBER INT INTEGER SMALLINT DEC DECIMAL REAL FLOAT NUMERIC DOUBLE PRECISION
Date	DATE TIMESTAMP TIMESTAMP WITH TIME ZONE TIMESTAMP WITH LOCAL TIME ZONE INTERVAL YEAR TO MONTH INTERVAL DAY TO SECOND

Modes Changing to or from an explicit specification of the default parameter mode IN does not change the RPC signature of a subprogram. For example, changing between:

```
PROCEDURE P1 (Param1 NUMBER);
PROCEDURE P1 (Param1 IN NUMBER);
```

does not change the RPC signature. Any other change of parameter mode *does* change the RPC signature.

Default Parameter Values Changing the specification of a default parameter value does not change the RPC signature. For example, procedure P1 has the same RPC signature in the following two examples:

```
PROCEDURE P1 (Param1 IN NUMBER := 100);
PROCEDURE P1 (Param1 IN NUMBER := 200);
```

An application developer who requires that callers get the new default value must recompile the called procedure, but no RPC signature-based invalidation occurs when a default parameter value assignment is changed.

Examples of Changing Procedure Signatures

Using the Get_emp_names procedure shown previously in this chapter, if the procedure body is changed to the following:

```
DECLARE
   Emp_number  NUMBER;
   Hire_date   DATE;
BEGIN
-- date format model changes

   SELECT Ename, To_char(Hiredate, 'DD/MON/YYYY')
      INTO Emp_name, Hire_date
      FROM Emp_tab
      WHERE Empno = Emp_number;
END;
```

The specification of the procedure has not changed, so its RPC signature has not changed.

But if the procedure specification is changed to the following:

```
CREATE OR REPLACE PROCEDURE Get_emp_name (
   Emp_number  IN  NUMBER,
   Hire_date   OUT DATE,
   Emp_name    OUT VARCHAR2) AS
```

And if the body is changed accordingly, then the RPC signature changes, because the parameter Hire_date has a different datatype.

However, if the name of that parameter changes to When_hired, and the datatype remains VARCHAR2, and the mode remains OUT, the RPC signature does not change. Changing the name of a formal parameter does not change the RPC signature of the unit.

Consider the following example:

```
CREATE OR REPLACE PACKAGE Emp_package AS
    TYPE Emp_data_type IS RECORD (
        Emp_number NUMBER,
        Hire_date  VARCHAR2(12),
        Emp_name   VARCHAR2(10));
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_type);
END;

CREATE OR REPLACE PACKAGE BODY Emp_package AS
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_type) IS
   BEGIN
       SELECT Empno, Ename, TO_CHAR(Hiredate, 'DD/MON/YY')
            INTO Emp_data
            FROM Emp_tab
            WHERE Empno = Emp_data.Emp_number;
   END;
END;
```

If the package specification is changed so that the record's field names are changed, but the types remain the same, then this does not affect the RPC signature. For example, the following package specification has the same RPC signature as the previous package specification example:

```
CREATE OR REPLACE PACKAGE Emp_package AS
    TYPE Emp_data_type IS RECORD (
        Emp_num    NUMBER,          -- was Emp_number
        Hire_dat   VARCHAR2(12),   -- was Hire_date
        Empname    VARCHAR2(10));  -- was Emp_name
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_type);
END;
```

Changing the name of the type of a parameter does not cause a change in the RPC signature if the type remains the same as before. For example, the following package specification for Emp_package is the same as the first one:

```
CREATE OR REPLACE PACKAGE Emp_package AS
    TYPE Emp_data_record_type IS RECORD (
        Emp_number NUMBER,
        Hire_date  VARCHAR2(12),
        Emp_name   VARCHAR2(10));
    PROCEDURE Get_emp_data
        (Emp_data IN OUT Emp_data_record_type);
END;
```

Controlling Remote Dependencies

The dynamic initialization parameter `REMOTE_DEPENDENCIES_MODE` controls whether the time stamp or the RPC signature dependency model is in effect.

- If the initialization parameter file contains the following specification:

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP
```

Then only time stamps are used to resolve dependencies (if this is not explicitly overridden dynamically).

- If the initialization parameter file contains the following parameter specification:

```
REMOTE_DEPENDENCIES_MODE = SIGNATURE
```

Then RPC signatures are used to resolve dependencies (if this not explicitly overridden dynamically).

- You can alter the mode dynamically by using the DDL statements. For example, this example alters the dependency model for the current session:

```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE =
    {SIGNATURE | TIMESTAMP}
Thise example alters the dependency model systemwide after startup:
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE =
    {SIGNATURE | TIMESTAMP}
```

If the `REMOTE_DEPENDENCIES_MODE` parameter is not specified, either in the `init.ora` parameter file or using the `ALTER SESSION` or `ALTER SYSTEM` DDL statements, `TIMESTAMP` is the default value. Therefore, unless you explicitly use the `REMOTE_DEPENDENCIES_MODE` parameter, or the appropriate DDL statement, your server is operating using the time-stamp dependency model.

When you use `REMOTE_DEPENDENCIES_MODE=SIGNATURE`:

- If you change the default value of a parameter of a remote procedure, then the local procedure calling the remote procedure is not invalidated. If the call to the remote procedure does not supply the parameter, then the default value is used. In this case, because invalidation/recompilation does not automatically occur, the old default value is used. If you want to see the new default values, then you must recompile the calling procedure manually.

- If you add a new overloaded procedure in a package (a new procedure with the same name as an existing one), then local procedures that call the remote procedure are not invalidated. If it turns out that this overloading results in a rebinding of existing calls from the local procedure under the time-stamp mode, then this rebinding does not happen under the RPC signature mode, because the local procedure does not get invalidated. You must recompile the local procedure manually to achieve the new rebinding.

- If the types of parameters of an existing packaged procedure are changed so that the new types have the same shape as the old ones, then the local calling procedure is not invalidated or recompiled automatically. You must recompile the calling procedure manually to get the semantics of the new type.

Topics:

- Dependency Resolution
- Suggestions for Managing Dependencies

Dependency Resolution

When REMOTE_DEPENDENCIES_MODE = TIMESTAMP (the default value), dependencies among program units are handled by comparing time stamps at run time. If the time stamp of a called remote procedure does not match the time stamp of the called procedure, then the calling (dependent) unit is invalidated and must be recompiled. In this case, if there is no local PL/SQL compiler, then the calling application cannot proceed.

In the time-stamp dependency mode, RPC signatures are not compared. If there is a local PL/SQL compiler, then recompilation happens automatically when the calling procedure is run.

When REMOTE_DEPENDENCIES_MODE = SIGNATURE, the recorded time stamp in the calling unit is first compared to the current time stamp in the called remote unit. If they match, then the call proceeds. If the time stamps do not match, then the RPC signature of the called remote subprogram, as recorded in the calling subprogram, is compared with the current RPC signature of the called subprogram. If they do not match (using the criteria described in the section "Switching Datatype Classes" on page 6-15), then an error is returned to the calling session.

Suggestions for Managing Dependencies

Follow these guidelines for setting the REMOTE_DEPENDENCIES_MODE parameter:

- Server-side PL/SQL users can set the parameter to TIMESTAMP (or let it default to that) to get the time-stamp dependency mode.

- Server-side PL/SQL users can choose to use the RPC signature dependency mode if they have a distributed system and they want to avoid possible unnecessary recompilations.

- Client-side PL/SQL users must set the parameter to SIGNATURE. This allows:

  - Installation of new applications at client sites, without the need to recompile procedures.

  - Ability to upgrade the server, without encountering time stamp mismatches.

- When using RPC signature mode on the server side, add new procedures to the end of the procedure (or function) declarations in a package specification. Adding a new procedure in the middle of the list of declarations can cause unnecessary invalidation and recompilation of dependent procedures.

Shared SQL Dependency Management

In addition to managing dependencies among schema objects, Oracle Database also manages dependencies of each shared SQL area in the shared pool. If a table, view, synonym, or sequence is created, altered, or dropped, or a procedure or package specification is recompiled, all dependent shared SQL areas are invalidated. At a subsequent execution of the cursor that corresponds to an invalidated shared SQL area, Oracle Database reparses the SQL statement to regenerate the shared SQL area.

7

The Data Dictionary

This chapter describes the central set of read-only reference tables and views of each Oracle database, known collectively as the **data dictionary**.

This chapter contains the following topics:

- Introduction to the Data Dictionary
- How the Data Dictionary Is Used
- Dynamic Performance Tables
- Database Object Metadata

Introduction to the Data Dictionary

One of the most important parts of an Oracle database is its **data dictionary**, which is a **read-only** set of tables that provides information about the database. A data dictionary contains:

- The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- How much space has been allocated for, and is currently used by, the schema objects
- Default values for columns
- Integrity constraint information
- The names of Oracle Database users
- Privileges and roles each user has been granted
- Auditing information, such as who has accessed or updated various schema objects
- Other general database information

The data dictionary is structured in tables and views, just like other database data. All the data dictionary tables and views for a given database are stored in that database's `SYSTEM` tablespace.

Not only is the data dictionary central to every Oracle database, it is an important tool for all users, from end users to application designers and database administrators. Use SQL statements to access the data dictionary. Because the data dictionary is read only, you can issue only queries (`SELECT` statements) against it's tables and views.

> **See Also:** "Bigfile Tablespaces" on page 3-5 for more information
> about SYSTEM tablespaces

This section includes the following topics:

- Structure of the Data Dictionary
- SYS, Owner of the Data Dictionary

Structure of the Data Dictionary

The data dictionary consists of the following:

Base Tables: The underlying tables that store information about the associated database. Only Oracle Database should write to and read these tables. Users rarely access them directly because they are normalized, and most of the data is stored in a cryptic format.

User-Accessible Views: The views that summarize and display the information stored in the base tables of the data dictionary. These views decode the base table data into useful information, such as user or table names, using joins and WHERE clauses to simplify the information. Most users are given access to the views rather than the base tables.

SYS, Owner of the Data Dictionary

The Oracle Database user SYS owns all base tables and user-accessible views of the data dictionary. No Oracle Database user should *ever* alter (UPDATE, DELETE, or INSERT) any rows or schema objects contained in the SYS schema, because such activity can compromise data integrity. The security administrator must keep strict control of this central account.

> **Caution:** Altering or manipulating the data in data dictionary tables can permanently and detrimentally affect the operation of a database.

How the Data Dictionary Is Used

The data dictionary has three primary uses:

- Oracle Database accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle Database modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle Database user can use the data dictionary as a read-only reference for information about the database.

This section includes the following topics:

- How Oracle Database Uses the Data Dictionary
- How to Use the Data Dictionary

How Oracle Database Uses the Data Dictionary

Data in the base tables of the data dictionary *is necessary for Oracle Database to function*. Therefore, only Oracle Database should write or change data dictionary information.

Oracle Database provides scripts to modify the data dictionary tables when a database is upgraded or downgraded.

> **Caution:** No data in any data dictionary table should be altered or deleted by any user.

During database operation, Oracle Database reads the data dictionary to ascertain that schema objects exist and that users have proper access to them. Oracle Database also updates the data dictionary continuously to reflect changes in database structures, auditing, grants, and data.

For example, if user Kathy creates a table named `parts`, then new rows are added to the data dictionary that reflect the new table, columns, segment, extents, and the privileges that Kathy has on the table. This new information is then visible the next time the dictionary views are queried.

This section includes the following topics:

- Public Synonyms for Data Dictionary Views
- Cache the Data Dictionary for Fast Access
- Other Programs and the Data Dictionary

Public Synonyms for Data Dictionary Views

Oracle Database creates public synonyms for many data dictionary views so users can access them conveniently. The security administrator can also create additional public synonyms for schema objects that are used systemwide. Users should avoid naming their own schema objects with the same names as those used for public synonyms.

Cache the Data Dictionary for Fast Access

Much of the data dictionary information is kept in the SGA in the **dictionary cache**, because Oracle Database constantly accesses the data dictionary during database operation to validate user access and to verify the state of schema objects. All information is stored in memory using the least recently used (LRU) algorithm.

Parsing information is typically kept in the caches. The `COMMENTS` columns describing the tables and their columns are not cached unless they are accessed frequently.

Other Programs and the Data Dictionary

Other Oracle Database products can reference existing views and create additional data dictionary tables or views of their own. Application developers who write programs that refer to the data dictionary should refer to the public synonyms rather than the underlying tables: the synonyms are less likely to change between software releases.

How to Use the Data Dictionary

The views of the data dictionary serve as a reference for all database users. Access the data dictionary views with SQL statements. Some views are accessible to all Oracle Database users, and others are intended for database administrators only.

The data dictionary is always available when the database is open. It resides in the `SYSTEM` tablespace, which is always online.

The data dictionary consists of sets of views. In many cases, a set consists of three views containing similar information and distinguished from each other by their prefixes, as shown in Table 7–1.

Table 7–1   Data Dictionary View Prefixes

Prefix	Scope
USER	User's view (what is in the user's schema)
ALL	Expanded user's view (what the user can access)
DBA	Database administrator's view (what is in all users' schemas)

The set of columns is identical across views, with these exceptions:

- Views with the prefix USER usually exclude the column OWNER. This column is implied in the USER views to be the user issuing the query.

- Some DBA views have additional columns containing information useful to the administrator.

> **See Also:**   *Oracle Database Reference* for a complete list of data dictionary views and their columns

This section includes the following topics:

- Views with the Prefix USER
- Views with the Prefix ALL
- Views with the Prefix DBA
- The DUAL Table

Views with the Prefix USER

The views most likely to be of interest to typical database users are those with the prefix USER. These views:

- Refer to the user's own private environment in the database, including information about schema objects created by the user, grants made by the user, and so on

- Display only rows pertinent to the user

- Have columns identical to the other views, except that the column OWNER is implied

- Return a subset of the information in the ALL views

- Can have abbreviated PUBLIC synonyms for convenience

For example, the following query returns all the objects contained in your schema:

```
SELECT object_name, object_type FROM USER_OBJECTS;
```

Views with the Prefix ALL

Views with the prefix ALL refer to the user's overall perspective of the database. These views return information about schema objects to which the user has access through public or explicit grants of privileges and roles, in addition to schema objects that the user owns. For example, the following query returns information about all the objects to which you have access:

```
SELECT owner, object_name, object_type FROM ALL_OBJECTS;
```

Views with the Prefix DBA

Views with the prefix DBA show a global view of the entire database. Synonyms are not created for these views, because DBA views should be queried only by administrators. Therefore, to query the DBA views, administrators must prefix the view name with its owner, SYS, as in the following:

```
SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;
```

Oracle recommends that you implement data dictionary protection to prevent users having the ANY system privileges from using such privileges on the data dictionary. If you enable dictionary protection (O7_DICTIONARY_ACCESSIBILITY is false), then access to objects in the SYS schema (dictionary objects) is restricted to users with the SYS schema. These users are SYS and those who connect as SYSDBA.

> **See Also:** *Oracle Database Administrator's Guide* for detailed information on system privileges restrictions

The DUAL Table

The table named DUAL is a small table in the data dictionary that Oracle Database and user-written programs can reference to guarantee a known result. This table has one column called DUMMY and one row containing the value X.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the DUAL table

Dynamic Performance Tables

Throughout its operation, Oracle Database maintains a set of virtual tables that record current database activity. These tables are called **dynamic performance tables.**

Dynamic performance tables are not true tables, and they should not be accessed by most users. However, database administrators can query and create views on the tables and grant access to those views to other users. These views are sometimes called **fixed views** because they cannot be altered or removed by the database administrator.

SYS owns the dynamic performance tables; their names all begin with V_$. Views are created on these tables, and then public synonyms are created for the views. The synonym names begin with V$. For example, the V$DATAFILE view contains information about the database's datafiles, and the V$FIXED_TABLE view contains information about all of the dynamic performance tables and views in the database.

> **See Also:** *Oracle Database Reference* for a complete list of the dynamic performance views' synonyms and their columns

Database Object Metadata

The DBMS_METADATA package provides interfaces for extracting complete definitions of database objects. The definitions can be expressed either as XML or as SQL DDL. Two styles of interface are provided:

- A flexible, sophisticated interface for programmatic control
- A simplified interface for ad hoc querying

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for
> more information about `DBMS_METADATA`

8

Memory Architecture

This chapter discusses the memory architecture of an Oracle Database instance. It contains the following topics:

- Introduction to Oracle Database Memory Structures
- Overview of the System Global Area
- Overview of the Program Global Area
- Overview of Memory Management Methods
- About Software Code Areas

> **See Also:** *Oracle Database Administrator's Guide* for instructions for configuring and managing memory

Introduction to Oracle Database Memory Structures

Oracle Database uses memory to store information such as the following:

- Program code
- Information about a connected session, even if it is not currently active
- Information needed during program execution (for example, the current state of a query from which rows are being fetched)
- Information that is shared and communicated among Oracle Database processes (for example, locking information)
- Cached data (for example, data blocks and redo log entries) that is also permanently stored on storage devices

Basic Memory Structures

The basic memory structures associated with Oracle Database include:

- Software code areas

    Software code areas are portions of memory used to store code that is being run or can be run. Oracle Database code is stored in a software area that is typically at a different location from users' programs—a more exclusive or protected location.

- 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 memory used by all individual PGAs is known as the **total instance PGA memory**, and the collection of individual PGAs is referred to as the **total instance PGA**, or just **instance PGA**. You use database initialization parameters to set the size of the instance PGA, not individual PGAs.

Figure 8–1 illustrates the relationships among these memory structures.

Figure 8–1   Oracle Database Memory Structures
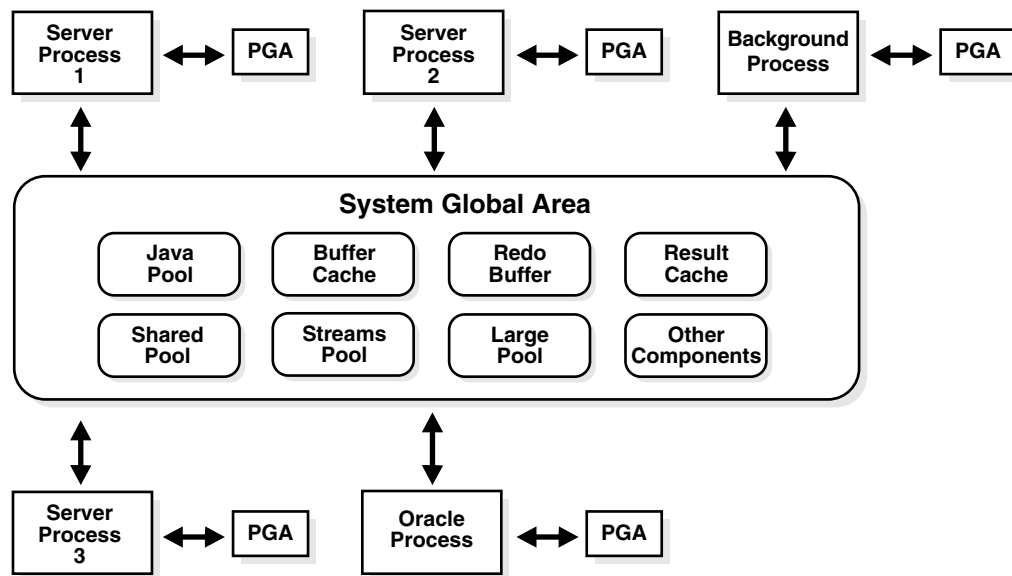
Overview of the System Global Area

The System Global Area (SGA) and the set of database processes constitute an Oracle Database instance. Oracle Database automatically allocates memory for an SGA when you start an instance, and the operating system reclaims the memory when you shut down the instance. Each instance has its own SGA.

The SGA is read/write. All database background processes and all server processes that execute on behalf of users can read information contained within the instance's SGA, and several processes write to the SGA during database operation.

Part of the SGA contains general information about the state of the database and the instance, which the background processes need to access. This is called the **fixed SGA**. No user data is stored here. The SGA also includes information communicated between processes, such as locking information.

If the system uses shared server architecture, then the request and response queues and some contents of the PGA are in the SGA.

As shown in Figure 8–1 on page 8-2, the SGA consists of a number of memory **components**, which are pools of memory used to satisfy a particular class of memory allocation requests.

The most important SGA components are the following:

- Database Buffer Cache
- Redo Log Buffer
- Shared Pool
- Large Pool
- Java Pool
- Streams Pool

> **See Also:**
>
> - "Introduction to an Oracle Instance" on page 12-1 for more information about an Oracle Database instance
> - "Overview of the Program Global Area" on page 8-9
> - "Dispatcher Request and Response Queues" on page 9-14

Database Buffer Cache

The database buffer cache is the portion of the SGA that holds copies of data blocks read from datafiles. All users concurrently connected to the instance share access to the database buffer cache.

This section includes the following topics:

- Organization of the Database Buffer Cache
- The LRU Algorithm and Full Table Scans

Organization of the Database Buffer Cache

The buffers in the cache are organized in two lists: the write list and the least recently used (LRU) list. The **write list** holds *dirty* buffers, which contain data that has been modified but has not yet been written to disk. The **LRU list** holds free buffers, *pinned* buffers, and dirty buffers that have not yet been moved to the write list. **Free buffers** do not contain any useful data and are available for use. **Pinned buffers** are currently being accessed.

When an Oracle Database process accesses a buffer, the process moves the buffer to the most recently used (MRU) end of the LRU list. As more buffers are continually moved to the MRU end of the LRU list, dirty buffers age toward the LRU end of the LRU list.

The first time an Oracle Database user process requires a particular piece of data, it searches for the data in the database buffer cache. If the process finds the data already in the cache (a **cache hit**), it can read the data directly from memory. If the process cannot find the data in the cache (a **cache miss**), it must copy the data block from a datafile on disk into a buffer in the cache before accessing the data. Accessing data through a cache hit is faster than data access through a cache miss.

Before reading a data block into the cache, the process must first find a free buffer. The process searches the LRU list, starting at the least recently used end of the list. The

process searches either until it finds a free buffer or until it has searched the threshold limit of buffers.

If the user process finds a dirty buffer as it searches the LRU list, it moves that buffer to the write list and continues to search. When the process finds a free buffer, it reads the data block from disk into the buffer and moves the buffer to the MRU end of the LRU list.

If an Oracle Database user process searches the threshold limit of buffers without finding a free buffer, the process stops searching the LRU list and signals the DBW0 background process to write some of the dirty buffers to disk.

> **See Also:** "Database Writer Process (DBWn)" on page 9-6 for more information about DBW*n* processes

The LRU Algorithm and Full Table Scans

When the user process is performing a full table scan, it reads the blocks of the table into buffers and puts them on the LRU end (instead of the MRU end) of the LRU list. This is because a fully scanned table usually is needed only briefly, so the blocks should be moved out quickly to leave more frequently used blocks in the cache.

You can control this default behavior of blocks involved in table scans on a table-by-table basis. To specify that blocks of the table are to be placed at the MRU end of the list during a full table scan, use the CACHE clause when creating or altering a table or cluster. You can specify this behavior for small lookup tables or large static historical tables to avoid I/O on subsequent accesses of the table.

> **See Also:** *Oracle Database SQL Language Reference* for information about the CACHE clause

Redo Log Buffer

The **redo log buffer** is a circular buffer in the SGA that holds information about changes made to the database. This information is stored in redo entries. **Redo entries** contain the information necessary to reconstruct, or redo, changes made to the database by INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP operations. Redo entries are used for database recovery, if necessary.

Redo entries are copied by Oracle Database processes from the user's memory space to the redo log buffer in the SGA. The redo entries take up continuous, sequential space in the buffer. The background process LGWR writes the redo log buffer to the active redo log file (or group of files) on disk.

> **See Also:**
>
> - "Log Writer Process (LGWR)" on page 9-8 for more information about how the redo log buffer is written to disk
>
> - *Oracle Database Backup and Recovery User's Guide* for information about redo log files and groups

Shared Pool

The shared pool portion of the SGA contains the library cache, the dictionary cache, the result cache, buffers for parallel execution messages, and control structures.

This section includes the following topics:

- Library Cache

- Dictionary Cache

- Result Cache

Library Cache

The library cache includes the shared SQL areas, private SQL areas (in the case of a shared server configuration), PL/SQL procedures and packages, and control structures such as locks and library cache handles.

Shared SQL areas are accessible to all users, so the library cache is contained in the shared pool within the SGA.

Shared SQL Areas and Private SQL Areas Oracle Database represents each SQL statement it runs with a shared SQL area and a private SQL area. Oracle Database recognizes when two users are executing the same SQL statement and reuses the shared SQL area for those users. However, each user must have a separate copy of the statement's private SQL area.

A shared SQL area contains the parse tree and execution plan for a given SQL statement. Oracle Database saves memory by using one shared SQL area for SQL statements run multiple times, which often happens when many users run the same application.

Oracle Database allocates memory from the shared pool when a new SQL statement is parsed, to store in the shared SQL area. The size of this memory depends on the complexity of the statement. If the entire shared pool has already been allocated, Oracle Database can deallocate items from the pool using a modified LRU (least recently used) algorithm until there is enough free space for the new statement's shared SQL area. If Oracle Database deallocates a shared SQL area, the associated SQL statement must be reparsed and reassigned to another shared SQL area at its next execution.

> **See Also:**
>
> - "Private SQL Area" on page 8-10
>
> - *Oracle Database Performance Tuning Guide*

PL/SQL Program Units and the Shared Pool Oracle Database processes PL/SQL program units (procedures, functions, packages, anonymous blocks, and database triggers) much the same way it processes individual SQL statements. Oracle Database allocates a shared area to hold the parsed, compiled form of a program unit. Oracle Database allocates a private area to hold values specific to the session that runs the program unit, including local, global, and package variables (also known as package instantiation) and buffers for executing SQL. If more than one user runs the same program unit, then a single, shared area is used by all users, while each user maintains a separate copy of his or her private SQL area, holding values specific to his or her session.

Individual SQL statements contained within a PL/SQL program unit are processed as described in the previous sections. Despite their origins within a PL/SQL program unit, these SQL statements use a shared area to hold their parsed representations and a private area for each session that runs the statement.

Allocation and Reuse of Memory in the Shared Pool In general, any item (shared SQL area or dictionary row) in the shared pool remains until it is flushed according to a modified LRU algorithm. The memory for items that are not being used regularly is freed if space is required for new items that must be allocated some space in the shared pool.

A modified LRU algorithm allows shared pool items that are used by many sessions to remain in memory as long as they are useful, even if the process that originally created the item terminates. As a result, the overhead and processing of SQL statements associated with a multiuser Oracle Database system is minimized.

When a SQL statement is submitted to Oracle Database for execution, Oracle Database automatically performs the following memory allocation steps:

1. Oracle Database checks the shared pool to see if a shared SQL area already exists for an identical statement. If so, that shared SQL area is used for the execution of the subsequent new instances of the statement. Alternatively, if there is no shared SQL area for a statement, Oracle Database allocates a new shared SQL area in the shared pool. In either case, the user's private SQL area is associated with the shared SQL area that contains the statement.

Note: A shared SQL area can be flushed from the shared pool, even if the shared SQL area corresponds to an open cursor that has not been used for some time. If the open cursor is subsequently used to run its statement, Oracle Database reparses the statement, and a new shared SQL area is allocated in the shared pool.

---

2. Oracle Database allocates a private SQL area on behalf of the session. The location of the private SQL area depends on the type of connection established for the session.

Oracle Database also flushes a shared SQL area from the shared pool in these circumstances:

- When the `ANALYZE` statement is used to update or delete the statistics of a table, cluster, or index, all shared SQL areas that contain statements referencing the analyzed schema object are flushed from the shared pool. The next time a flushed statement is run, the statement is parsed in a new shared SQL area to reflect the new statistics for the schema object.

- If a schema object is referenced in a SQL statement and that object is later modified in any way, the shared SQL area is **invalidated** (marked invalid), and the statement must be reparsed the next time it is run.

- If you change a database's global database name, all information is flushed from the shared pool.

- The administrator can manually flush all information in the shared pool to assess the performance (with respect to the shared pool, not the data buffer cache) that can be expected after instance startup without shutting down the current instance. The statement `ALTER SYSTEM FLUSH SHARED_POOL` is used to do this.

See Also:

- "Shared SQL Areas and Private SQL Areas" on page 8-5 for more information about the location of the private SQL area

- Chapter 6, "Schema Object Dependencies" for more information about the invalidation of SQL statements and dependency issues

- *Oracle Database SQL Language Reference* for information about using `ALTER SYSTEM FLUSH SHARED_POOL`

- *Oracle Database Reference* for information about `V$SQL` and `V$SQLAREA` dynamic views

Dictionary Cache

The data dictionary is a collection of database tables and views containing reference information about the database, its structures, and its users. Oracle Database accesses the data dictionary frequently during SQL statement parsing. This access is essential to the continuing operation of Oracle Database.

The data dictionary is accessed so often by Oracle Database that two special locations in memory are designated to hold dictionary data. One area is called the **data dictionary cache**, also known as the **row cache** because it holds data as rows instead of buffers (which hold entire blocks of data). The other area in memory to hold dictionary data is the library cache. All Oracle Database user processes share these two caches for access to data dictionary information.

> **See Also:** Chapter 7, "The Data Dictionary"

Result Cache

The result cache is composed of the SQL query result cache and PL/SQL function result cache, which share the same infrastructure.

The `DBMS_RESULT_CACHE` package provides administration subprograms, which, for example, flush all cached results and turn result-caching on or off system-wide. The dynamic performance views `V$RESULT_CACHE_*` allow the developer and DBA to determine, for example, the cache-hit success for a certain SQL query or PL/SQL function.

Similar to the result cache, the client result cache also caches results, except that the caching is done on the client side.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about sizing the result cache
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_RESULT_CACHE` package
>
> - *Oracle Database Reference* for information about dynamic performance ($V) views
>
> - *Oracle Call Interface Programmer's Guide* for more information about the client result cache

SQL Query Result Cache

Results of queries and query fragments can be cached in memory in the SQL query result cache. The database can then use cached results to answer future executions of these queries and query fragments. Because retrieving results from the SQL query result cache is faster than rerunning a query, frequently run queries experience a significant performance improvement when their results are cached. Users can annotate a query or query fragment with a *result cache hint* to indicate that results are to be stored in the SQL query result cache.

You can set the `RESULT_CACHE_MODE` initialization parameter to control whether the SQL query result cache is used for all queries (when possible), or only for queries that are annotated.

The database automatically invalidates a cached result whenever a transaction modifies the data or metadata of any of the database objects used to construct that cached result.

> **See Also:** *Oracle Database Performance Tuning Guide* for information about the RESULT_CACHE_MODE initialization parameter

PL/SQL Function Result Cache

A PL/SQL function is sometimes used to return the result of a computation whose inputs are one or several parameterized queries issued by the function. In some cases, these queries access data (for example, the catalog of wares in a shopping application) that changes very infrequently compared to the frequency of calling the function. You can include syntax in the source text of a PL/SQL function to request that its results be cached and, to ensure correctness, that the cache be purged when any of a list of tables experiences DML. The look-up key for the cache is the combination of actual arguments with which the function is invoked. When a particular invocation of the result-cached function is a cache hit, then the function body is not executed; instead, the cached value is returned immediately.

> **See Also:** *Oracle Database PL/SQL Language Reference* for more information about the PL/SQL function result cache

Large Pool

The database administrator can configure an optional memory area called the **large pool** to provide large memory allocations for:

- Session memory for the shared server and the Oracle XA interface (used where transactions interact with more than one database)

- I/O server processes

- Oracle Database backup and restore operations

By allocating session memory from the large pool for shared server, Oracle XA, or parallel query buffers, Oracle Database can use the shared pool primarily for caching shared SQL and avoid the performance overhead caused by shrinking the shared SQL cache.

In addition, the memory for Oracle Database backup and restore operations, for I/O server processes, and for parallel buffers is allocated in buffers of a few hundred kilobytes. The large pool is better able to satisfy such large memory requests than the shared pool.

The large pool does not have an LRU list. It is different from reserved space in the shared pool, which uses the same LRU list as other memory allocated from the shared pool.

> **See Also:**
>
> - "Shared Server Architecture" on page 9-13 for information about allocating session memory from the large pool for the shared server
>
> - *Oracle Database Advanced Application Developer's Guide* for information about Oracle XA
>
> - *Oracle Database Performance Tuning Guide* for more information about the large pool, reserve space in the shared pool, and I/O server processes
>
> - "Overview of Parallel Execution" on page 16-10 for information about allocating memory for parallel execution

Java Pool

Java pool memory is used in server memory for all session-specific Java code and data within the JVM. Java pool memory is used in different ways, depending on the mode in which Oracle Database is running.

The Java Pool Advisor statistics provide information about library cache memory used for Java and predict how changes in the size of the Java pool can affect the parse rate. The Java Pool Advisor is internally turned on when `statistics_level` is set to `TYPICAL` or higher. These statistics reset when the advisor is turned off.

> **See Also:** *Oracle Database Java Developer's Guide*

Streams Pool

The streams pool is used exclusively by Oracle Streams. The Streams pool stores buffered queue messages, and it provides memory for Oracle Streams capture processes and apply processes.

Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as needed when Oracle Streams is used.

> **See Also:** *Oracle Streams Concepts and Administration*

Overview of the Program Global Area

Oracle Database allocates a program global area (PGA) for each server process. The PGA is used to process SQL statements and to hold logon and other session information. For the purposes of memory management, the collection of all PGAs is known as the **instance PGA**. Using an initialization parameter, you set the size of the instance PGA, and the database distributes memory to individual PGAs as needed.

> **Note:** Background processes also allocate their own PGAs. This discussion focuses on server process PGAs only.

This section contains the following topics:

- Content of the PGA
- PGA Memory Use in Dedicated and Shared Server Modes

> **See Also:**

Content of the PGA

The content of the PGA memory varies, depending on whether or not the instance is running the shared server option. Generally speaking, the PGA memory is divided into the following areas:

- Session Memory
- Private SQL Area

Session Memory
Session memory is the memory allocated to hold a session's variables (logon information) and other information related to the session. For a shared server, the session memory is shared and not private.

See Also:

- for more information about sessions

- *Oracle Database Net Services Administrator's Guide*

Private SQL Area

The private SQL area contains data such as bind variable values, query execution state information, and query execution work areas. Each session that issues a SQL statement has a private SQL area. Each user that submits the same SQL statement has his or her own private SQL area that uses a single shared SQL area. Thus, many private SQL areas can be associated with the same shared SQL area.

The location of a private SQL area depends on the type of connection established for a session. If a session is connected through a dedicated server, private SQL areas are located in the server process's PGA. However, if a session is connected through a shared server, part of the private SQL area is kept in the SGA.

This section includes the following topics:

- Cursors and SQL Areas

- Private SQL Area Components

- SQL Work Areas

> **See Also:**

Cursors and SQL Areas The application developer of an Oracle Database precompiler program or OCI program can explicitly open **cursors**, or handles to specific private SQL areas, and use them as a named resource throughout the execution of the program. Recursive cursors that Oracle Database issues implicitly for some SQL statements also use shared SQL areas.

The management of private SQL areas is the responsibility of the user process. The allocation and deallocation of private SQL areas depends largely on which application tool you are using, although the number of private SQL areas that a user process can allocate is always limited by the initialization parameter OPEN_CURSORS. The default value of this parameter is 50.

A private SQL area continues to exist until the corresponding cursor is closed or the statement handle is freed. Although Oracle Database frees the runtime area after the statement completes, the persistent area remains waiting. Application developers close all open cursors that will not be used again to free the persistent area and to minimize the amount of memory required for users of the application.

> **See Also:**

Private SQL Area Components The private SQL area of a cursor is itself divided into two areas whose lifetimes are different:

- **The persistent area**—This area contains bind variable values. It is freed only when the cursor is closed.

- **The runtime area**—Oracle Database creates this area as the first step of an execute request. It contains the following structures:

  - Query execution state information

For example, for a full table scan, this area contains information on the progress of the scan

- SQL work areas

  These areas are allocated as needed for memory-intensive operations like sorting or hash-joins. More detail is provided later in this section.

For DML, the runtime area is freed when the statement finishes running. For queries, it is freed after all rows are fetched or the query is canceled.

SQL Work Areas SQL work areas are allocated to support memory-intensive operators such as the following:

- Sort-based operators (order by, group-by, rollup, window function)

- Hash-join

- Bitmap merge

- Bitmap create

For example, a sort operator uses a work area (sometimes called the sort area) to perform the in-memory sort of a set of rows. Similarly, a hash-join operator uses a work area (also called the hash area) to build a hash table from its left input. If the amount of data to be processed by these two operators does not fit into a work area, the input data is divided into smaller pieces. This enables some data pieces to be processed in memory while the rest are spilled to temporary disk storage to be processed later. Although bitmap operators do not spill to disk when their associated work area is too small, their complexity is inversely proportional to the size of their work area. Thus, these operators run faster with larger work area.

The size of a work area can be controlled and tuned. The database automatically tunes work area sizes when automatic PGA memory management is enabled. See "Overview of Memory Management Methods" on page 8-12 for more information.

Generally, bigger work areas can significantly improve the performance of a particular operator at the cost of higher memory consumption. Optimally, the size of a work area is big enough to accommodate the input data and auxiliary memory structures allocated by its associated SQL operator. If not, response time increases, because part of the input data must be spilled to temporary disk storage. In the extreme case, if the size of a work area is far too small compared to the input data size, multiple passes over the data pieces must be performed. This can dramatically increase the response time of the operator.

PGA Memory Use in Dedicated and Shared Server Modes

PGA memory allocation depends, in some specifics, on whether the system uses dedicated or shared server architecture. Table 8–1 shows the differences.

Table 8–1   Differences in Memory Allocation Between Dedicated and Shared Servers

Memory Area	Dedicated Server	Shared Server
Nature of session memory	Private	Shared
Location of the persistent area	PGA	SGA
Location of part of the runtime area for `SELECT` statements	PGA	PGA
Location of the runtime area for DML/DDL statements	PGA	PGA

Overview of Memory Management Methods

Memory management involves maintaining optimal sizes for the Oracle database instance memory structures as demands on the database change. The memory that must be managed is the system global area (SGA) memory and the instance program global area (instance PGA) memory. The instance PGA memory is the collection of memory allocations for all individual PGAs.

Oracle Database supports various memory management methods, which are chosen by initialization parameter settings. Oracle recommends that you enable the *automatic memory management* method.

Automatic Memory Management – For Both the SGA and Instance PGA

Beginning with Oracle Database 11*g*, 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.

Automatic Shared Memory Management – For the SGA

If you want to exercise more direct control over the size of the SGA, you can disable automatic memory management and enable *automatic shared memory management*. With automatic shared memory management, you set target and maximum sizes for the SGA. The database then tunes the total size of the SGA to your designated target, and dynamically tunes the sizes of all SGA components.

Manual Shared Memory Management – For the SGA

If you want complete control of individual SGA component sizes, you can disable both automatic memory management and automatic shared memory management. This effectively enables *manual shared memory management*. In this mode, 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.

Automatic PGA Memory Management – For the Instance PGA

When you disable automatic memory management and enable automatic shared memory management or manual shared memory management, you also implicitly enable *automatic PGA memory management*. With automatic PGA memory management, you set a target size for the instance PGA. The database then tunes the size of the instance PGA to your target, and dynamically tunes the sizes of individual PGAs. Because automatic PGA memory management is the default method for the instance PGA, if you do not explicitly set a target size, the database automatically computes and configures a reasonable default.

Manual PGA Memory Management – For the Instance PGA

Previous releases of Oracle Database required the DBA to manually specify the maximum work area size for each type of SQL operator (such as sort or hash-join). This proved to be very difficult, because the workload is always changing. Although the current release of Oracle Database supports this manual PGA memory management method, Oracle strongly recommends that you leave automatic PGA memory management enabled.

Summary of Memory Management Methods

Table 8–2 summarizes the various memory management methods. If you do not enable automatic memory management, you must separately configure one memory management method for the SGA and one for the PGA.

Note: When automatic memory management is not enabled, the default method for the instance PGA is automatic PGA memory management.

Table 8–2   Oracle Database Memory Management Modes

Memory Management Mode	For	You Set	Oracle Database Automatically Tunes
Automatic memory management	SGA and PGA	■ Total memory target size for the Oracle instance ■ (Optional) Maximum memory size for the Oracle instance	■ Total SGA size ■ SGA component sizes ■ Instance PGA size ■ Individual PGA sizes
Automatic shared memory management (Automatic memory management disabled)	SGA	■ SGA target size ■ (Optional) SGA maximum size	SGA component sizes
Manual shared memory management (Automatic memory management and automatic shared memory management disabled)	SGA	■ Shared pool size ■ Buffer cache size ■ Java pool size ■ Large pool size	-
Automatic PGA memory management	PGA	Instance PGA target size	Individual PGA sizes
Manual PGA memory management (not recommended)	PGA	Maximum work area size for each type of SQL operator	-

See Also: *Oracle Database Administrator's Guide* because automatic memory management is not available on all platforms

Memory Management Options and Defaults for Database Installation

If you create your database with Database Configuration Assistant (DBCA) and choose the basic installation option, automatic memory management is enabled by default. If you choose advanced installation, Database Configuration Assistant (DBCA) enables you to select from the following three memory management configurations:

■ Automatic memory management

■ Automatic shared memory management + automatic PGA memory management

■ Manual shared memory management + automatic PGA memory management

If you create the database with a CREATE DATABASE SQL statement and do not choose the memory management mode by specifying the required memory initialization parameters, manual shared memory management and automatic PGA memory management are configured by default.

See also: *Oracle Database Administrator's Guide* for more information about memory management and about memory management initialization parameters.

About Software Code Areas

Software code areas are portions of memory used to store code that is being run or can be run. Oracle Database code is stored in a software area that is typically at a different location from users' programs—a more exclusive or protected location.

Software areas are usually static in size, changing only when software is updated or reinstalled. The required size of these areas varies by operating system.

Software areas are read only and can be installed shared or nonshared. When possible, Oracle Database code is shared so that all users can access it without having multiple copies in memory. This results in a saving of real main memory and improves overall performance.

User programs can be shared or nonshared. Some Oracle tools and utilities (such as Oracle Forms and SQL*Plus) can be installed shared, but some cannot. Multiple instances of Oracle Database can use the same Oracle Database code area with different databases if running on the same computer.

> **Note:** The option of installing software shared is not available for all operating systems (for example, on PCs operating Windows).
>
> See your Oracle Database operating system-specific documentation for more information.

9

Process Architecture

This chapter discusses the processes in an Oracle database system and the different configurations available for an Oracle database system.

This chapter contains the following topics:

Introduction to Processes

All connected Oracle Database users must run two modules of code to access an Oracle Database instance.

- Application or Oracle tool: A database user runs a database application (such as a precompiler program) or an Oracle tool (such as SQL*Plus), which issues SQL statements to an Oracle database.

- Oracle database server code: Each user has some Oracle database code executing on his or her behalf, which interprets and processes the application's SQL statements.

These code modules are run by processes. A **process** is a "thread of control" or a mechanism in an operating system that can run a series of steps. (Some operating systems use the terms **job** or **task**.) A process normally has its own private memory area in which it runs.

This section includes the following topics:

Multiple-Process Oracle Systems

Multiple-process Oracle (also called **multiuser Oracle**) uses several processes to run different parts of the Oracle code and additional processes for the users—either one process for each connected user or one or more processes shared by multiple users.

Most database systems are multiuser, because one of the primary benefits of a database is managing data needed by multiple users at the same time.

Each process in an Oracle Database instance performs a specific job. By dividing the work of Oracle Database and database applications into several processes, multiple users and applications can connect to a single database instance simultaneously while the system maintains excellent performance.

Types of Processes

The processes in an Oracle Database system can be categorized into two major groups:

- User processes run the application or Oracle tool code.

- Oracle Database processes run the Oracle database server code. They include server processes and background processes.

The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

With dedicated server, for each user, the database application is run by a different process (a user process) than the one that runs the Oracle database server code (a dedicated server process).

With shared server, the database application is run by a different process (a user process) than the one that runs the Oracle database server code. Each server process that runs Oracle database server code (a **shared server process**) can serve multiple user processes.

Figure 9–1 illustrates a dedicated server configuration. Each connected user has a separate user process, and several background processes run Oracle Database.

Figure 9–1   An Oracle Database Instance
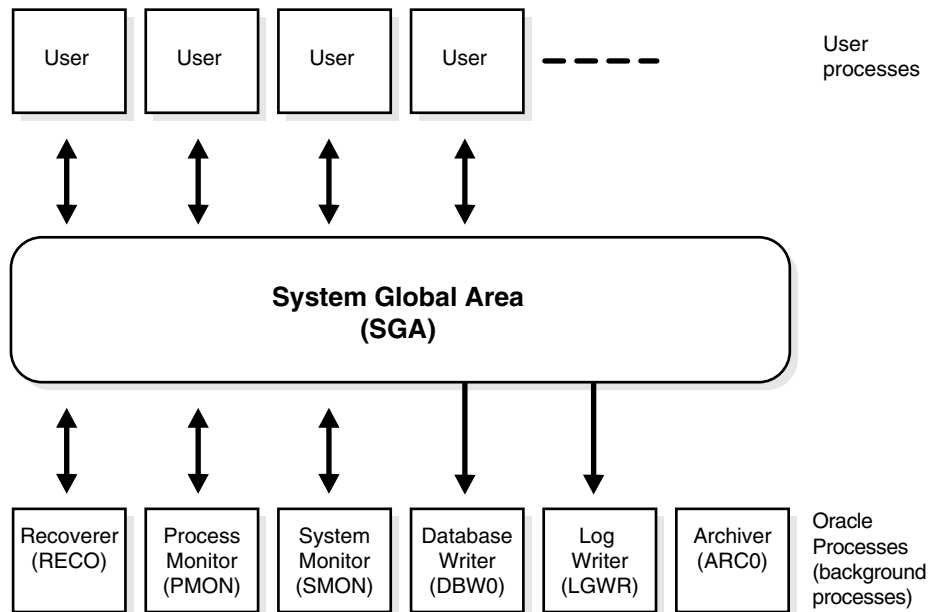
Figure 9–1 can represent multiple concurrent users running an application on the same computer as Oracle Database. This particular configuration usually runs on a mainframe or minicomputer.

See Also:

- Your Oracle Database operating system-specific documentation for more details on configuration choices

Overview of User Processes

When a user runs an application program (such as a Pro*C program) or an Oracle tool (such as Oracle Enterprise Manager or SQL*Plus), Oracle Database creates a **user process** to run the user's application.

Connections and Sessions

Connection and **session** are closely related to **user process** but are very different in meaning.

A **connection** is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle Database) or network software (when different computers run the database application and Oracle Database, and communicate through a network).

A **session** is a specific connection of a user to an Oracle Database instance through a user process. For example, when a user starts SQL*Plus, the user must provide a valid user name and password, and then a session is established for that user. A session lasts from the time the user connects until the time the user disconnects or exits the database application.

Multiple sessions can be created and exist concurrently for a single Oracle Database user using the same user name. For example, a user with the user name/password of `SCOTT/TIGER` can connect to the same Oracle Database instance several times.

In configurations without the shared server, Oracle Database creates a server process on behalf of each user session. However, with the shared server, many user sessions can share a single server process.

See Also:

Overview of Oracle Database Processes

This section describes the two types of processes that run the Oracle database server code (server processes and background processes). It also describes the trace files and alert logs, which record database events for the Oracle Database processes.

This section includes the following topics:

- Oracle Database Server Processes

- Oracle Database Background Processes

- Oracle Database Trace Files and the Alert Log

Oracle Database Server Processes

Oracle Database creates **server processes** to handle the requests of user processes connected to the instance. In some situations when the application and Oracle Database operate on the same computer, it is possible to combine the user process and corresponding server process into a single process to reduce system overhead. However, when the application and Oracle Database operate on different computers, a user process always communicates with Oracle Database through a separate server process.

Server processes (or the server portion of combined user/server processes) created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application

- Read necessary data blocks from datafiles on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA

- Return results in such a way that the application can process the information

Oracle Database Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called **background processes**.

An Oracle Database instance can have many background processes; not all are always present. There are numerous background processes. See the V$BGPROCESS view for more information on the background processes. The background processes in an Oracle Database instance can include the following:

- Archiver Processes (ARCn)

- Checkpoint Process (CKPT)

- Database Writer Process (DBWn)

- Job Queue Processes

- Log Writer Process (LGWR)

- Process Monitor Process (PMON)

- Queue Monitor Processes (QMNn)

- Recoverer Process (RECO)

- System Monitor Process (SMON)

- Other Oracle Database Background Processes

On many operating systems, background processes are created automatically when an instance is started.

Figure 9–2 illustrates how each background process interacts with the different parts of an Oracle database, and the rest of this section describes each process.

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide* and *Oracle Clusterware Administration and Deployment Guide* for more information. Oracle Real Application Clusters is not illustrated in

- Your operating system-specific documentation for details on how these processes are created

Figure 9–2   Background Processes of a Multiple-Process Oracle Database Instance
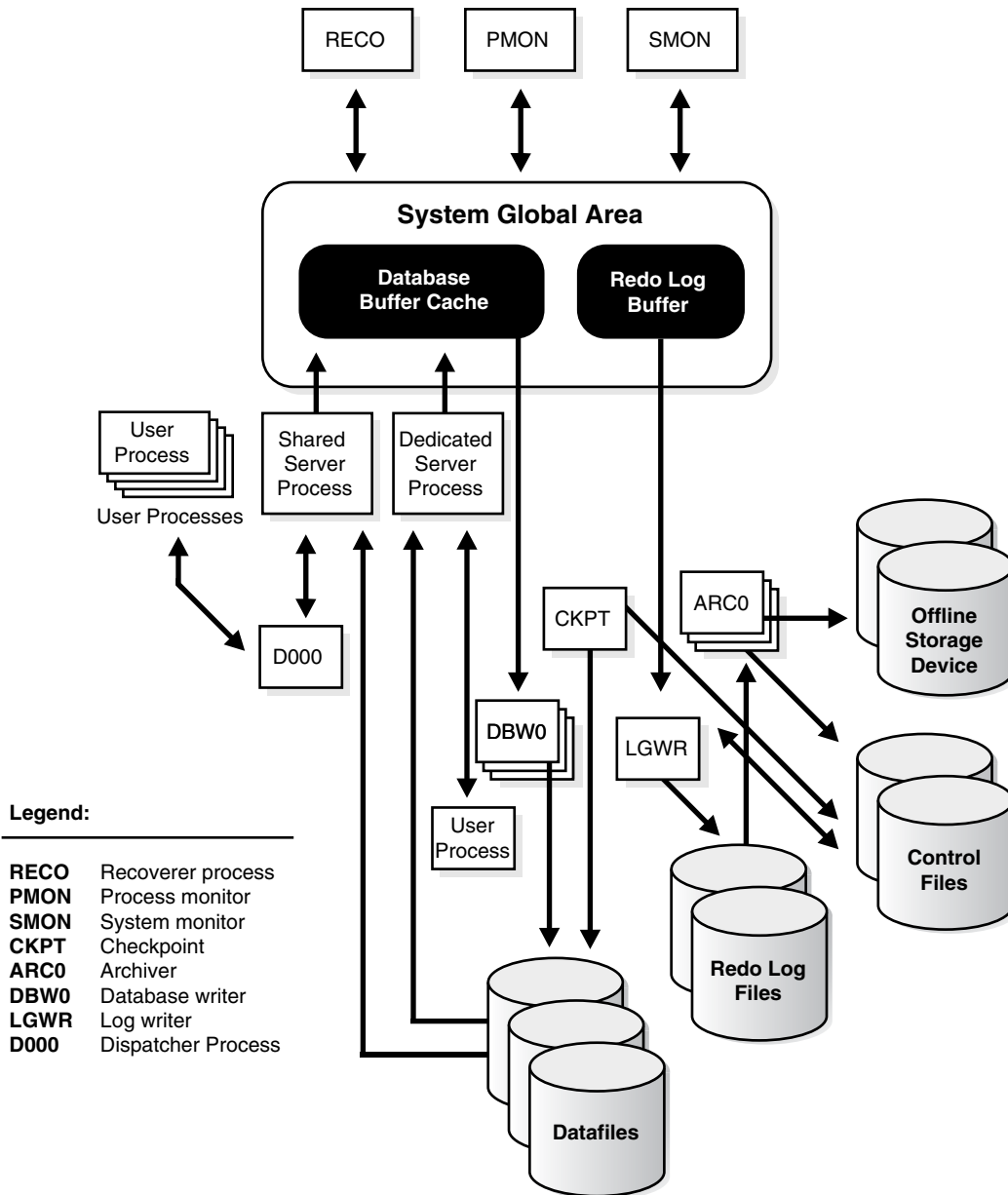
Archiver Processes (ARC*n*)

The **archiver processes (ARCn)** copy redo log files to a designated storage device after a log switch has occurred. In addition, they can collect transaction redo data and

transmit that data to standby destinations. ARC*n* processes are present only when the database is in ARCHIVELOG mode, and automatic archiving is enabled.

If you anticipate a heavy workload for archiving, such as during bulk loading of data, you can increase the maximum number of archiver processes with the LOG_ARCHIVE_ MAX_PROCESSES initialization parameter. The ALTER SYSTEM statement can change the value of this parameter dynamically to increase or decrease the number of ARC*n* processes.

> **See Also:**
>
> - "Oracle Database Trace Files and the Alert Log" on page 9-12
>
> - *Oracle Database Backup and Recovery User's Guide*
>
> - Your operating system-specific documentation for details about using the ARC*n* processes

Checkpoint Process (CKPT)

When a checkpoint occurs, Oracle Database must update the headers of all datafiles to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBW*n* always performs that work.

The statistic **DBWR checkpoints** displayed by the System_Statistics monitor in Oracle Enterprise Manager indicates the number of checkpoint requests completed.

> **See Also:** *Oracle Real Application Clusters Administration and Deployment Guide* for information about CKPT with Oracle Real Application Clustersc

Database Writer Process (DBW*n*)

The **database writer process (DBWn)** writes the contents of buffers to datafiles. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBWa through DBWj) to improve write performance if your system modifies data heavily. These additional DBW*n* processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked **dirty**. A **cold** buffer is a buffer that has not been recently used according to the least recently used (LRU) algorithm. The DBW*n* process writes cold, dirty buffers to disk so that user processes are able to find cold, clean buffers that can be used to read new blocks into the cache. As buffers are dirtied by user processes, the number of free buffers diminishes. If the number of free buffers drops too low, user processes that must read blocks from disk into the cache are not able to find free buffers. DBW*n* manages the buffer cache so that user processes can always find free buffers.

By writing cold, dirty buffers to disk, DBW*n* improves the performance of finding free buffers while keeping recently used buffers resident in memory. For example, blocks that are part of frequently accessed small tables or indexes are kept in the cache so that they do not need to be read in again from disk. The LRU algorithm keeps more frequently accessed blocks in the buffer cache so that when a buffer is written to disk, it is unlikely to contain data that will be useful soon.

The initialization parameter DB_WRITER_PROCESSES specifies the number of DBW*n* processes. The maximum number of DBW*n* processes is 20. If it is not specified by the user during startup, Oracle Database determines how to set DB_WRITER_PROCESSES based on the number of CPUs and processor groups.

The DBW*n* process writes dirty buffers to disk under the following conditions:

- When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBW*n* to write. DBW*n* writes dirty buffers to disk asynchronously while performing other processing.

- DBW*n* periodically writes buffers to advance the **checkpoint**, which is the position in the redo thread (log) from which instance recovery begins. This log position is determined by the oldest dirty buffer in the buffer cache.

In all cases, DBW*n* performs batched (multiblock) writes to improve efficiency. The number of blocks written in a multiblock write varies by operating system.

> **See Also:**
>
> - "Database Buffer Cache" on page 8-3
>
> - *Oracle Database Performance Tuning Guide* for advice on setting `DB_WRITER_PROCESSES` and for information about how to monitor and tune the performance of a single DBW0 process or multiple DBW*n* processes
>
> - *Oracle Database Backup and Recovery User's Guide*

Job Queue Processes

Job queue processes are used for batch processing. They run user jobs. They can be viewed as a scheduler service that can be used to schedule jobs as PL/SQL statements or procedures on an Oracle Database instance. Given a start date and an interval, the job queue processes try to run the job at the next occurrence of the interval.

Job queue processes are managed dynamically. This allows job queue clients to use more job queue processes when required. The resources used by the new processes are released when they are idle.

Dynamic job queue processes can run a large number of jobs concurrently at a given interval. The job queue processes run user jobs as they are assigned by the CJQ process. Here's what happens:

1. The coordinator process, named CJQ0, periodically selects jobs that need to be run from the system `JOB$` table. New jobs selected are ordered by time.

2. The CJQ0 process dynamically spawns job queue slave processes (J000…J999) to run the jobs.

3. The job queue process runs one of the jobs that was selected by the CJQ process for execution. The processes run one job at a time.

4. After the process finishes execution of a single job, it polls for more jobs. If no jobs are scheduled for execution, then it enters a sleep state, from which it wakes up at periodic intervals and polls for more jobs. If the process does not find any new jobs, then it aborts after a preset interval.

The initialization parameter `JOB_QUEUE_PROCESSES` represents the maximum number of job queue processes that can concurrently run on an instance. However, clients should not assume that all job queue processes are available for job execution.

> **Note:** The coordinator process is not started if the initialization parameter `JOB_QUEUE_PROCESSES` is set to 0.

> **See Also:** *Oracle Database Administrator's Guide* for more information about job queues

Log Writer Process (LGWR)

The **log writer process (LGWR)** is responsible for redo log buffer management—writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

- A commit record when a user process commits a transaction
- Redo log buffers
  - Every three seconds
  - When the redo log buffer is one-third full
  - When a DBW*n* process writes modified buffers to disk, if necessary

> **Note:** Before DBW*n* can write a modified buffer, all redo records associated with the changes to the buffer must be written to disk (the **write-ahead protocol**). If DBW*n* finds that some redo records have not been written, it signals LGWR to write the redo records to disk and waits for LGWR to complete writing the redo log buffer before it can write out the data buffers.

LGWR writes synchronously to the active mirrored group of redo log files. If one of the files in the group is damaged or unavailable, LGWR continues writing to other files in the group and logs an error in the LGWR trace file and in the system alert log. If all files in a group are damaged, or the group is unavailable because it has not been archived, LGWR cannot continue to function.

When a user issues a `COMMIT` statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries. The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a **fast commit** mechanism. The atomic write of the redo entry containing the transaction's commit record is the single event that determines the transaction has committed. Oracle Database returns a success code to the committing transaction, although the data buffers have not yet been written to disk.

> **Note:** Sometimes, if more buffer space is needed, LGWR writes redo log entries before a transaction is committed. These entries become permanent only if the transaction is later committed.

When a user commits a transaction, the transaction is assigned a **system change number (SCN)**, which Oracle Database records along with the transaction's redo entries in the redo log. SCNs are recorded in the redo log so that recovery operations can be synchronized in Real Application Clusters and distributed databases.

In times of high activity, LGWR can write to the redo log file using **group commits**. For example, assume that a user commits a transaction. LGWR must write the transaction's redo entries to disk, and as this happens, other users issue `COMMIT` statements. However, LGWR cannot write to the redo log file to commit these transactions until it has completed its previous write operation. After the first transaction's entries are written to the redo log file, the entire list of redo entries of waiting transactions (not yet committed) can be written to disk in one operation, requiring less I/O than do transaction entries handled individually. Therefore, Oracle Database minimizes disk I/O and maximizes performance of LGWR. If requests to commit continue at a high rate, then every write (by LGWR) from the redo log buffer can contain multiple commit records.

> **See Also:**
>
> - Redo Log Buffer on page 8-4
>
> - "Oracle Database Trace Files and the Alert Log" on page 9-12
>
> - *Oracle Real Application Clusters Administration and Deployment Guide* for more information about SCNs and how they are used
>
> - *Oracle Database Administrator's Guide* for more information about SCNs and how they are used
>
> - *Oracle Database Performance Tuning Guide* for information about how to monitor and tune the performance of LGWR

Process Monitor Process (PMON)

The **process monitor (PMON)** performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.

Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

Queue Monitor Processes (QMN*n*)

The **queue monitor process** is an optional background process for Oracle Streams Advanced Queuing, which monitors the message queues. You can configure up to 10 queue monitor processes. These processes, like the job queue processes, are different from other Oracle Database background processes in that process failure does not cause the instance to fail.

> **See Also:**
>
> - "Oracle Streams Advanced Queuing" on page 23-8
>
> - *Oracle Streams Advanced Queuing User's Guide*

Recoverer Process (RECO)

The **recoverer process (RECO)** is a background process used with the distributed database configuration that automatically resolves failures involving distributed transactions. The RECO process of a node automatically connects to other databases

involved in an in-doubt distributed transaction. When the RECO process reestablishes a connection between involved database servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved in-doubt transactions.

If the RECO process fails to connect with a remote server, RECO automatically tries to connect again after a timed interval. However, RECO waits an increasing amount of time (growing exponentially) before it attempts another connection. The RECO process is present only if the instance permits distributed transactions. The number of concurrent distributed transactions is not limited.

> **See Also:** *Oracle Database Administrator's Guide* for more information about distributed transaction recovery

System Monitor Process (SMON)

The **system monitor process (SMON)** performs recovery, if necessary, at instance startup. SMON is also responsible for cleaning up temporary segments that are no longer in use and for coalescing contiguous free extents within dictionary managed tablespaces. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

With Real Application Clusters, the SMON process of one instance can perform instance recovery for a failed CPU or instance.

> **See Also:** *Oracle Real Application Clusters Administration and Deployment Guide* for more information about SMON

Other Oracle Database Background Processes

There are several other background processes that might be running. These can include the following:

- ACMS (atomic controlfile to memory service) per-instance process is an agent that contributes to ensuring a distributed SGA memory update is either globally committed on success or globally aborted in the event of a failure in an Oracle RAC environment.

- ASMB is present in a database instance using an Automatic Storage Management disk group. It communicates with the Automatic Storage Management instance.

- DBRM (database resource manager) process is responsible for setting resource plans and other resource manager related tasks.

> **See Also:** "Overview of the Database Resource Manager" on page 14-17 for more information about the database resource manager

- DIA0 (diagnosability process 0) (only 0 is currently being used) is responsible for hang detection and deadlock resolution.

- DIAG (diagnosability) process performs diagnostic dumps and executes global `oradebug` commands.

- DSKM (slave diskmon) is used as the conduit between RDBMS and ASM instances and the Master Diskmon daemon to communicate I/O Fencing information, I/O Resource Manager Plans, and Transaction Commit Cache information to SAGE storage. It is also used to implement the `skgxp` ANT protocol between the host

nodes and SAGE storage servers. If no SAGE storage is used, the slave diskmon process will exit silently after startup of the instance.

- EMNC (event monitor coordinator) is the background server process used for database event management and notifications.

- FBDA (flashback data archiver process) archives the historical rows of tracked tables into flashback data archives. Tracked tables are tables which are enabled for flashback archive. When a transaction containing DML on a tracked table commits, this process stores the pre-image of the rows into the flashback archive. It also keeps metadata on the current rows.

  FBDA is also responsible for automatically managing the flashback data archive for space, organization, and retention and keeps track of how far the archiving of tracked transactions has occurred.

- GMON maintains disk membership in ASM disk groups.

- GTX0-j (global transaction) processes provide transparent support for XA global transactions in an Oracle RAC environment. The database autotunes the number of these processes based on the workload of XA global transactions. Global transaction processes are only seen in an Oracle RAC environment.

- KATE performs proxy I/O to an ASM metafile when a disk goes offline.

- MARK marks ASM allocation units as stale following a missed write to an offline disk.

- MMAN is used for internal database tasks.

- MMNL performs frequent and light-weight manageability-related tasks, such as session history capture and metrics computation.

- MMON performs various manageability-related background tasks, for example:

  - Issuing alerts whenever a given metrics violates its threshold value

  - Taking snapshots by spawning additional process (MMON slaves)

  - Capturing statistics value for SQL objects which have been recently modified

- ORB*n* performs the actual rebalance data extent movements in an Automatic Storage Management instance. There can be many of these at a time, called ORB0, ORB1, and so forth.

- PSP0 (process spawner) spawns Oracle processes.

- RBAL coordinates rebalance activity for disk groups in an Automatic Storage Management instance. It performs a global open on Automatic Storage Management disks.

- SMCO (space management coordinator) process coordinates the execution of various space management related tasks, such as proactive space allocation and space reclamation. It dynamically spawns slave processes (W*nnn*) to implement the task.

- VKTM (virtual keeper of time) is responsible for providing a wall-clock time (updated every second) and reference-time counter (updated every 20 ms and available only when running at elevated priority).

> **See Also:**
>
> - *Oracle Clusterware Administration and Deployment Guide* for more information about Oracle Clusterware background processes
>
> - *Oracle Real Application Clusters Administration and Deployment Guide* for more information about Oracle Real Application Clusters background processes

Oracle Database Trace Files and the Alert Log

Beginning with Oracle Database 11*g*, an advanced fault diagnosability infrastructure is included for preventing, detecting, diagnosing, and resolving problems. The problems that are targeted in particular are critical errors such as those caused by database 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.

Each server and background process can write to an associated **trace file**. When a process detects an internal error, it dumps information about the error to its trace file. If an internal error occurs and information is written to a trace file, the administrator should contact Oracle Support Services.

All filenames of trace files associated with a background process contain the name of the process that generated the trace file. The one exception to this is trace files generated by job queue processes (Jnnn).

Additional information in trace files can provide guidance for tuning applications or an instance. Background processes always write this information to a trace file when appropriate.

Each database also has an `alert.log`. The alert log of a database is a chronological log of messages and errors, including the following:

- All internal errors (ORA-600), block corruption errors (ORA-1578), and deadlock errors (ORA-60) that occur

- Administrative operations, such as the SQL statements `CREATE/ALTER/DROP DATABASE/TABLESPACE` and the Oracle Enterprise Manager or SQL*Plus statements `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`

- Several messages and errors relating to the functions of shared server and dispatcher processes

- Errors during the automatic refresh of a materialized view

Oracle Database uses the alert log to keep a record of these events as an alternative to displaying the information on an operator's console. (Many systems also display this information on the console.) If an administrative operation is successful, a message is written in the alert log as "completed" along with a time stamp.

> **See Also:**
>
> - *Oracle Database Performance Tuning Guide* for information about enabling the SQL trace facility
>
> - *Oracle Database Error Messages* for information about error messages

Shared Server Architecture

Shared server architecture eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. An idle shared server process from a shared pool of server processes picks up a request from a common queue, which means a small number of shared servers can perform the same amount of processing as many dedicated servers. Also, because the amount of memory required for each user is relatively small, less memory and process management are required, and more users can be supported.

A number of different processes are needed in a shared server system:

- A network listener process that connects the user processes to dispatchers or dedicated servers (the listener process is part of Oracle Net Services, not Oracle Database).

- One or more dispatcher processes

- One or more shared server processes

Shared server processes require Oracle Net Services or SQL*Net version 2.

Note: To use shared servers, a user process must connect through Oracle Net Services or SQL*Net version 2, even if the process runs on the same computer as the Oracle Database instance.

When an instance starts, the network listener process opens and establishes a communication pathway through which users connect to Oracle Database. Then, each dispatcher process gives the listener process an address at which the dispatcher listens for connection requests. At least one dispatcher process must be configured and started for each network protocol that the database clients will use.

When a user process makes a connection request, the listener examines the request and determines whether the user process can use a shared server process. If so, the listener returns the address of the dispatcher process that has the lightest load, and the user process connects to the dispatcher directly.

Some user processes cannot communicate with the dispatcher, so the network listener process cannot connect them to a dispatcher. In this case, or if the user process requests a dedicated server, the listener creates a dedicated server and establishes an appropriate connection.

The Oracle Database shared server architecture increases the scalability of applications and the number of clients simultaneously connected to the database. It can enable existing applications to scale up without making any changes to the application itself.

See Also:

- "Restricted Operations of the Shared Server" on page 9-16

- *Oracle Database Net Services Administrator's Guide* for more information about the network listener

This section includes the following topics:

- Dispatcher Request and Response Queues

- Restricted Operations of the Shared Server

Dispatcher Request and Response Queues

A request from a user is a single program interface call that is part of the user's SQL statement. When a user makes a call, its dispatcher places the request on the **request queue**, where it is picked up by the next available shared server process.

The request queue is in the SGA and is common to all dispatcher processes of an instance. The shared server processes check the common request queue for new requests, picking up new requests on a first-in-first-out basis. One shared server process picks up one request in the queue and makes all necessary calls to the database to complete that request.

When the server completes the request, it places the response on the calling dispatcher's **response queue**. Each dispatcher has its own response queue in the SGA. The dispatcher then returns the completed request to the appropriate user process.

For example, in an order entry system each clerk's user process connects to a dispatcher and each request made by the clerk is sent to that dispatcher, which places the request in the request queue. The next available shared server process picks up the request, services it, and puts the response in the response queue. When a clerk's request is completed, the clerk remains connected to the dispatcher, but the shared server process that processed the request is released and available for other requests. While one clerk is talking to a customer, another clerk can use the same shared server process.

Figure 9–3 illustrates how user processes communicate with the dispatcher across the program interface and how the dispatcher communicates users' requests to shared server processes.

Figure 9–3   The Shared Server Configuration and Processes
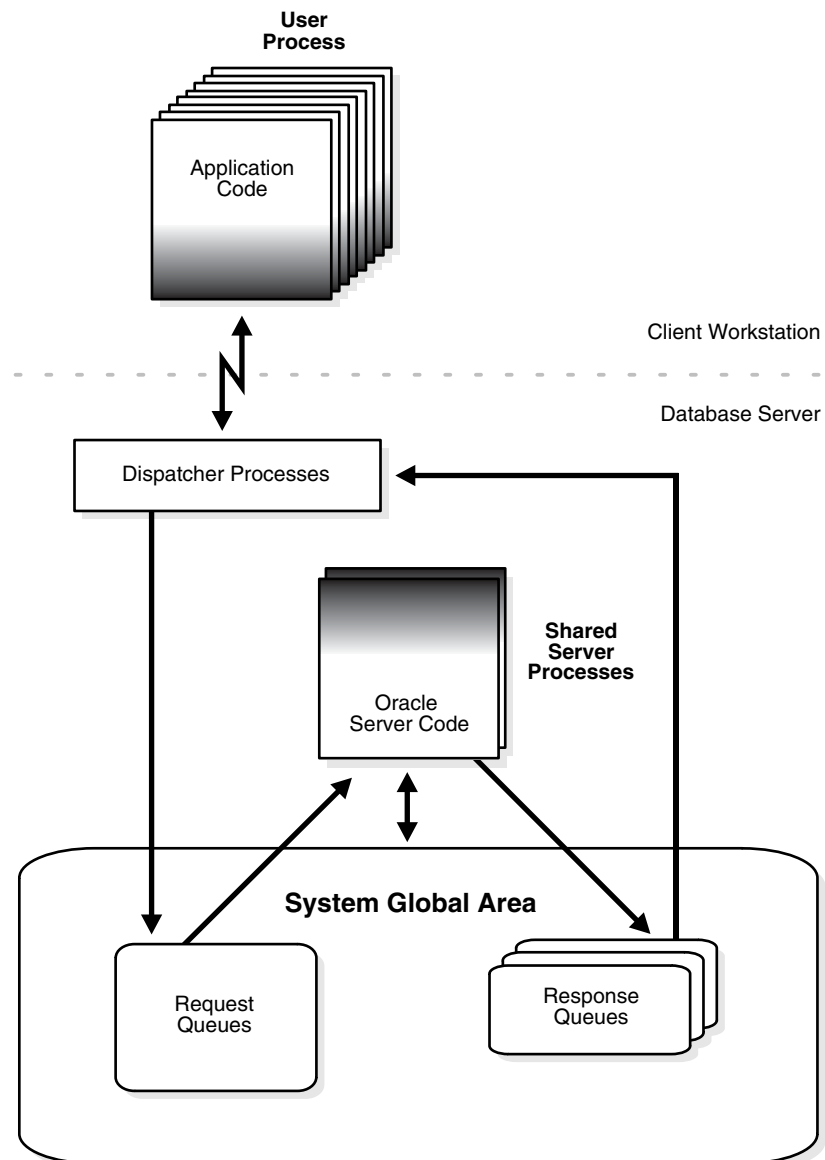
This section includes the following topics:

- Dispatcher Processes (Dnnn)
- Shared Server Processes (Snnn)

Dispatcher Processes (D*nnn*)

The **dispatcher processes** support shared server configuration by allowing user processes to share a limited number of server processes. With the shared server, fewer shared server processes are required for the same number of users, Therefore, the shared server can support a greater number of users, particularly in client/server environments where the client application and server operate on different computers.

You can create multiple dispatcher processes for a single database instance. At least one dispatcher must be created for each network protocol used with Oracle Database. The database administrator starts an optimal number of dispatcher processes

depending on the operating system limitation and the number of connections for each process, and can add and remove dispatcher processes while the instance runs.

> **Note:** Each user process that connects to a dispatcher must do so through Oracle Net Services or SQL*Net version 2, even if both processes are running on the same computer.

In a shared server configuration, a network listener process waits for connection requests from client applications and routes each to a dispatcher process. If it cannot connect a client application to a dispatcher, the listener process starts a dedicated server process, and connects the client application to the dedicated server. The listener process is not part of an Oracle Database instance; rather, it is part of the networking processes that work with Oracle Database.

> **See Also:**
>
> ■ "Shared Server Architecture" on page 9-13
>
> ■ *Oracle Database Net Services Administrator's Guide* for more information about the network listener

Shared Server Processes (S*nnn*)

Each **shared server process** serves multiple client requests in the shared server configuration. Shared server processes and dedicated server processes provide the same functionality, except shared server processes are not associated with a specific user process. Instead, a shared server process serves any client request in the shared server configuration.

The PGA of a shared server process does not contain user-related data (which needs to be accessible to all shared server processes). The PGA of a shared server process contains only stack space and process-specific variables.

All session-related information is contained in the SGA. Each shared server process needs to be able to access all sessions' data spaces so that any server can handle requests from any session. Space is allocated in the SGA for each session's data space. You can limit the amount of space that a session can allocate by setting the resource limit PRIVATE_SGA to the desired amount of space in the user's profile.

Oracle Database dynamically adjusts the number of shared server processes based on the length of the request queue. The number of shared server processes that can be created ranges between the values of the initialization parameters SHARED_SERVERS and MAX_SHARED_SERVERS .

> **See Also:**
>
> ■ "Overview of the Program Global Area" on page 8-9 for more information about the content of a PGA in different types of instance configurations
>
> ■ Chapter 20, "Database Security"for more information about resource limits and profiles

Restricted Operations of the Shared Server

Certain administrative activities cannot be performed while connected to a dispatcher process, including shutting down or starting an instance and media recovery. An error message is issued if you attempt to perform these activities while connected to a dispatcher process.

These activities are typically performed when connected with administrator privileges. When you want to connect with administrator privileges in a system configured with shared servers, you must state in your connect string that you want to use a dedicated server process (SERVER=DEDICATED) instead of a dispatcher process.

See Also:

- Your operating system-specific documentation

- *Oracle Database Net Services Administrator's Guide* for the proper connect string syntax

Dedicated Server Configuration

Figure 9–4 illustrates Oracle Database running on two computers using the dedicated server architecture. In this configuration, a user process runs the database application on one computer, and a server process runs the associated Oracle database server on another computer.

Figure 9–4 Oracle Database Using Dedicated Server Processes
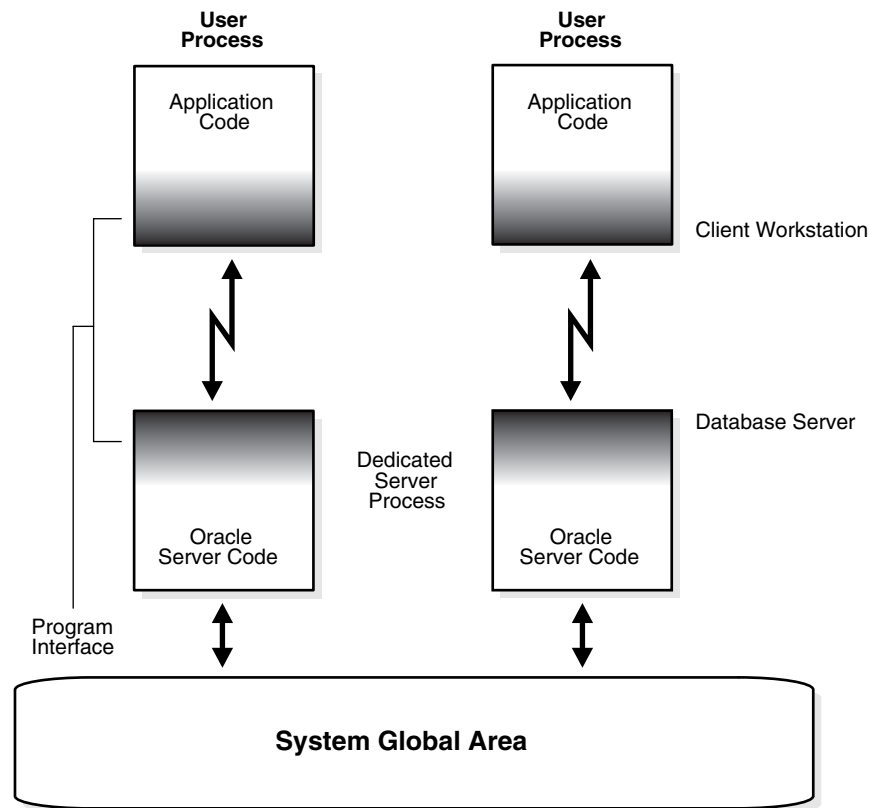
The user and server processes are separate, distinct processes. The separate server process created on behalf of each user process is called a **dedicated server process** (or **shadow** process), because this server process acts only on behalf of the associated user process.

This configuration maintains a one-to-one ratio between the number of user processes and server processes. Even when the user is not actively making a database request, the dedicated server process remains (though it is inactive and can be paged out on some operating systems).

Figure 9–4 shows user and server processes running on separate computers connected across a network. However, the dedicated server architecture is also used if the same computer runs both the client application and the Oracle database server code but the host operating system could not maintain the separation of the two programs if they were run in a single process. UNIX is a common example of such an operating system.

In the dedicated server configuration, the user and server processes communicate using different mechanisms:

- If the system is configured so that the user process and the dedicated server process run on the same computer, the program interface uses the host operating system's interprocess communication mechanism to perform its job.

- If the user process and the dedicated server process run on different computers, the program interface provides the communication mechanisms (such as the network software and Oracle Net Services) between the programs.

- Dedicated server architecture can sometimes result in inefficiency. Consider an order entry system with dedicated server processes. A customer places an order as a clerk enters the order into the database. For most of the transaction, the clerk is talking to the customer while the server process dedicated to the clerk's user process remains idle. The server process is not needed during most of the transaction, and the system is slower for other clerks entering orders. For applications such as this, the shared server architecture may be preferable.

> **See Also:**
>
> - Your operating system-specific documentation
>
> - *Oracle Database Net Services Administrator's Guide*
>
> for more information about communication links

Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) provides a connection pool in the database server for typical Web application usage scenarios. DRCP pools dedicated servers, which comprise of a server foreground combined with a database session, to create pooled servers.

A Web application typically acquires a database connection, uses the connection for a short period, and then releases the connection. DRCP enables multiple Web application threads and processes to share the pooled servers for their connection needs.

DRCP complements middle-tier connection pools that share connections between threads in a middle-tier process. DRCP also enables you to share database connections across multiple middle-tier processes. These middle-tier processes may belong to the same or different middle-tier host.

DRCP enables a significant reduction in key database resources that are required to support a large number of client connections. DRCP reduces the amount of memory required for the database server and boosts the scalability of both the database server and the middle-tier. The pool of readily available servers also reduces the cost of re-creating client connections.

DRCP is especially useful for architectures with multi-process, single-threaded application servers, such as PHP and Apache servers, that cannot do middle-tier connection pooling. The database can scale to tens of thousands of simultaneous connections with DRCP.

See Also:

- "About Database Resident Connection Pooling" in the *Oracle Database Administrator's Guide*

- *Oracle Call Interface Programmer's Guide*

The pooled server model closely follows the dedicated model, which is used to connect to Oracle by default. The pooled server model does away with the overhead of dedicating a server for every connection that requires the server for a short period. DRCP allows a connection to acquire, and then voluntarily release the pooled server for use by other similar connections. On being acquired by a connection, a pooled server essentially transforms into a dedicated server for that connection until it is released back into the pool. Clients getting connections out of the database resident connection pool connect to an Oracle background process known as the connection broker. The connection broker implements the pool functionality and multiplexes pooled servers among inbound connections from client processes.

When a client needs to perform some backend database work, the connection broker picks up a pooled server from the pool and assigns it to the client. Subsequently, the client is directly connected to the pooled server until the request is served. After the server has finished processing the client request, the server goes back into the pool and the connection from the client is restored to the connection broker process. Figure 9–5 illustrates database resident connection pooling.

Figure 9–5   Pool of Dedicated Server Processes Handling Connections Through the Connection Broker Process
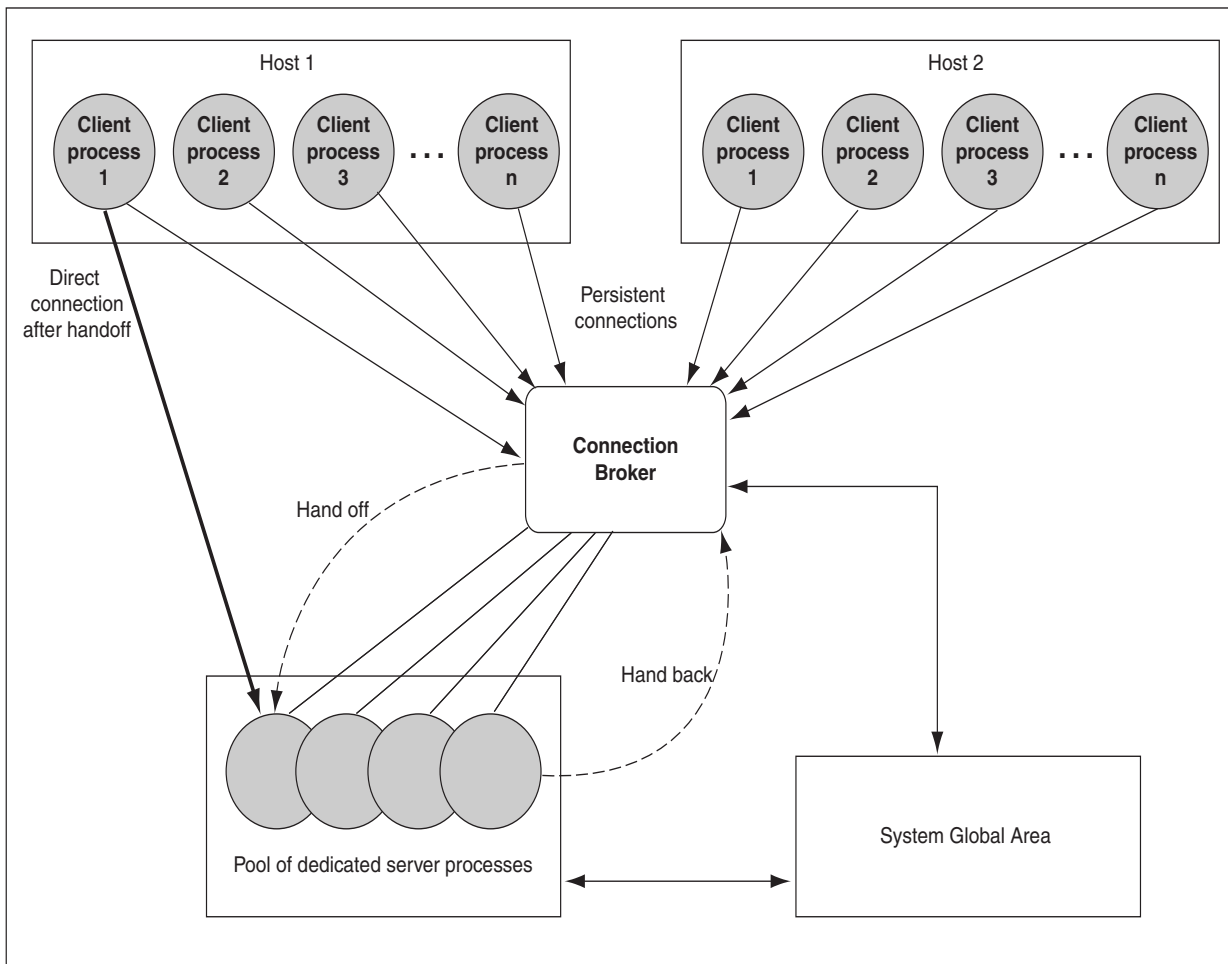
Using Database Resident Connection Pooling

Database resident connection pooling enables you to freely scale your middle-tier hardware without worrying about running out of memory on the database side. This is because a smaller pool of dedicated server processes can serve a larger number of middle-tier processes.

The default connection pool is called, SYS_DEFAULT_CONNECTION_POOL. To use database resident connection pooling, the database administrator needs to explicitly start the pool. The following example illustrates this:

Log into SQL*Plus as SYSDBA

Run the following command:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.START_POOL('SYS_DEFAULT_CONNECTION_POOL');
```

> **Note:**   Currently, only the default connection pool is supported.

> **See Also:**   "Enabling Database Resident Connection Pooling" in the *Oracle Database Administrator's Guide*

To connect to the shared pool, the server type should be set to POOLED in the database connection string. For example:

```
ServerPool = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=somehost)
        (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=testdb)(SERVER=POOLED)))
```

You can also connect to the shared pool using the easy connect naming method. You need to use the keyword POOLED with the database service name. For example:

```
CONNECT joeuser/joepasswd@myhost.mydomain.com:1521/mydb:POOLED
```

This section includes the following topics:

- Connection Classes
- Session Purity

Connection Classes

A connection class defines a logical name for the type of connection required by the application. Two different users cannot share connections, or sessions, amongst themselves. For example, a session first created for user HR is only given out to subsequent requests on behalf of the user HR. A connection class enables further separation between the sessions of a given user. The connection class lets different applications, connecting as the same database user, identify their sessions using a logical name that corresponds to the application. DRCP then ensures that sessions belonging to a particular connection class are not shared outside the connection class.

Session Purity

Session purity specifies whether the application requires a brand new session (PURITY=NEW), or whether the application logic is set up to reuse a pooled session (PURITY=SELF). If the application can reuse a pooled session, then a free session with the requested connection class is allotted to the application.

Connection classes and session purity are specified by the client as attributes of a DRCP connection. The default connection class value is *username*.SHARED. By default, sessions with the same *username* are shared when purity is SELF.

The default value for purity is NEW. The defaults can differ for different application scenarios. Please see the respective application manuals for details on using DRCP with your application.

> **See Also:** *Oracle Call Interface Programmer's Guide* for details on using connection classes and session purity

The Program Interface

The **program interface** is the software layer between a database application and Oracle Database. The program interface:

- Provides a security barrier, preventing destructive access to the SGA by client user processes
- Acts as a communication mechanism, formatting information requests, passing data, and trapping and returning errors
- Converts and translates data, particularly between different types of computers or to external user program datatypes

The **Oracle code** acts as a server, performing database tasks on behalf of an **application** (a client), such as fetching rows from data blocks. It consists of several

parts, provided by both Oracle Database software and operating system-specific software.

This section includes the following topics:

- Program Interface Structure
- Program Interface Drivers
- Communications Software for the Operating System

Program Interface Structure

The program interface consists of the following pieces:

- Oracle call interface (OCI) or the Oracle runtime library (SQLLIB)
- The client or user side of the program interface
- Various **Oracle Net Services drivers** (protocol-specific communications software)
- Operating system communications software
- The server or Oracle Database side of the program interface (also called the OPI)

Both the user and Oracle Database sides of the program interface run Oracle software, as do the drivers.

Oracle Net Services is the portion of the program interface that allows the client application program and the Oracle database server to reside on separate computers in your communication network.

Program Interface Drivers

Drivers are pieces of software that transport data, usually across a network. They perform operations such as connect, disconnect, signal errors, and test for errors. Drivers are specific to a communications protocol, and there is always a default driver.

You can install multiple drivers (such as the asynchronous or DECnet drivers) and select one as the default driver, but allow an individual user to use other drivers by specifying the desired driver at the time of connection. Different processes can use different drivers. A single process can have concurrent connections to a single database or to multiple databases (either local or remote) using different Oracle Net Services drivers.

> **See Also:**
>
> - Your system installation and configuration guide for details about choosing, installing, and adding drivers
> - Your system Oracle Net Services documentation for information about selecting a driver at runtime while accessing Oracle Database
> - *Oracle Database Net Services Administrator's Guide*

Communications Software for the Operating System

The lowest-level software connecting the user side to the Oracle Database side of the program interface is the communications software, which is provided by the host operating system. DECnet, TCP/IP, LU6.2, and ASYNC are examples. The communication software can be supplied by Oracle, but it is usually purchased separately from the hardware vendor or a third-party software supplier.

See Also: Your Oracle Database operating system-specific documentation for more information about the communication software of your system

10

Application Architecture

This chapter defines application architecture and describes how the Oracle database server and database applications work in a distributed processing environment. This material applies to almost every type of Oracle Database system environment.

This chapter contains the following topics:

- Introduction to Client/Server Architecture
- Overview of Multitier Architecture
- Overview of Oracle Net Services

Introduction to Client/Server Architecture

In the Oracle Database system environment, the database application and the database are separated into two parts: a front-end or **client** portion, and a back-end or **server** portion—hence the term **client/server architecture.** The client runs the database application that accesses database information and interacts with a user through the keyboard, screen, and pointing device, such as a mouse. The server runs the Oracle Database software and handles the functions required for concurrent, shared data access to an Oracle database.

Although the client application and Oracle Database can be run on the same computer, greater efficiency can often be achieved when the client portions and server portion are run by different computers connected through a network. The following sections discuss possible variations in the Oracle Database client/server architecture.

Distributed processing is the use of more than one processor, located in different systems, to perform the processing for an individual task. Examples of distributed processing in Oracle Database systems appear in Figure 10–1.

- In Part A of the figure, the client and server are located on different computers, and these computers are connected through a network. The server and clients of an Oracle Database system communicate through Oracle Net Services, Oracle's network interface.
- In Part B of the figure, a single computer has more than one processor, and different processors separate the execution of the client application from Oracle Database.
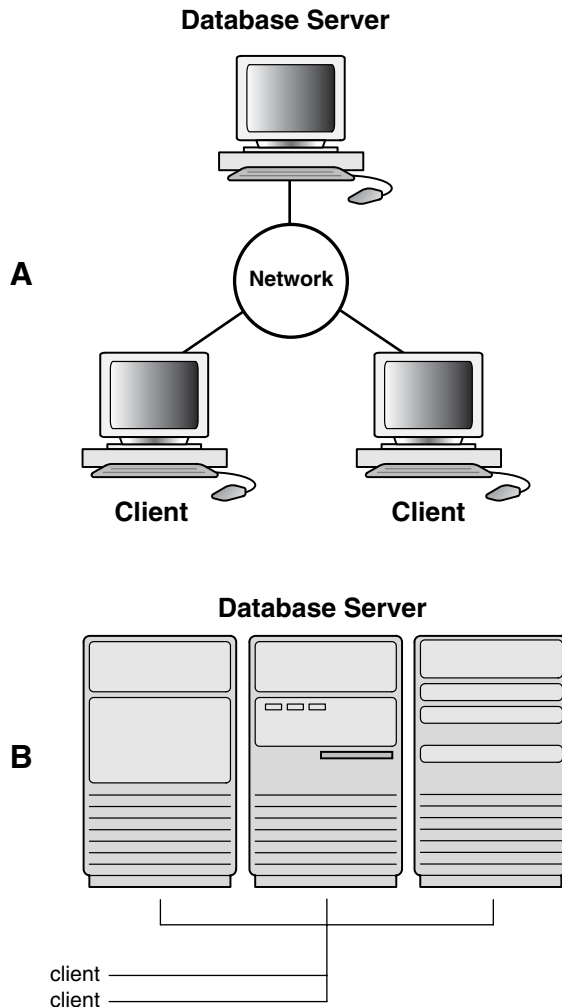
> **Note:** This chapter applies to environments with one database on one server. In a **distributed database**, one server (Oracle Database) may need to access a database on another server.

Figure 10–1   The Client/Server Architecture and Distributed Processing

Oracle Database client/server architecture in a distributed processing environment provides the following benefits:

- Client applications are not responsible for performing any data processing. Rather, they request input from users, request data from the server, and then analyze and present this data using the display capabilities of the client workstation or the terminal (for example, using graphics or spreadsheets).

- Client applications are not dependent on the physical location of the data. Even if the data is moved or distributed to other database servers, the application continues to function with little or no modification.

- Oracle Database exploits the multitasking and shared-memory facilities of its underlying operating system. As a result, it delivers the highest possible degree of concurrency, data integrity, and performance to its client applications.

- Client workstations or terminals can be optimized for the presentation of data (for example, by providing graphics and mouse support), and the server can be optimized for the processing and storage of data (for example, by having large amounts of memory and disk space).

- In networked environments, you can use inexpensive client workstations to access the remote data of the server effectively.

- If necessary, Oracle Database can be **scaled** as your system grows. You can add multiple servers to distribute the database processing load throughout the network (**horizontally scaled**), or you can move Oracle Database to a minicomputer or mainframe, to take advantage of a larger system's performance (**vertically scaled**). In either case, all data and applications are maintained with little or no modification, because Oracle Database is portable between systems.

- In networked environments, shared data is stored on the servers rather than on all computers in the system. This makes it easier and more efficient to manage concurrent access.

- In networked environments, client applications submit database requests to the server using SQL statements. After it is received, the SQL statement is processed by the server, and the results are returned to the client application. Network traffic is kept to a minimum, because only the requests and the results are shipped over the network.

> **See Also:**
>
> - "Overview of Oracle Net Services" on page 10-5 for more information about Oracle Net Services
>
> - *Oracle Database Administrator's Guide* for more information about clients and servers in distributed databases

Overview of Multitier Architecture

In a traditional multitier architecture environment, an application server provides data for clients and serves as an interface between clients and database servers. This architecture is particularly important because of the prevalence of Internet use.

This architecture enables use of an application server to:

- Validate the credentials of a client, such as a Web browser

- Connect to a database server

- Perform the requested operation

An example of a multitier architecture appears in Figure 10–2.

Figure 10–2   A Multitier Architecture Environment Example
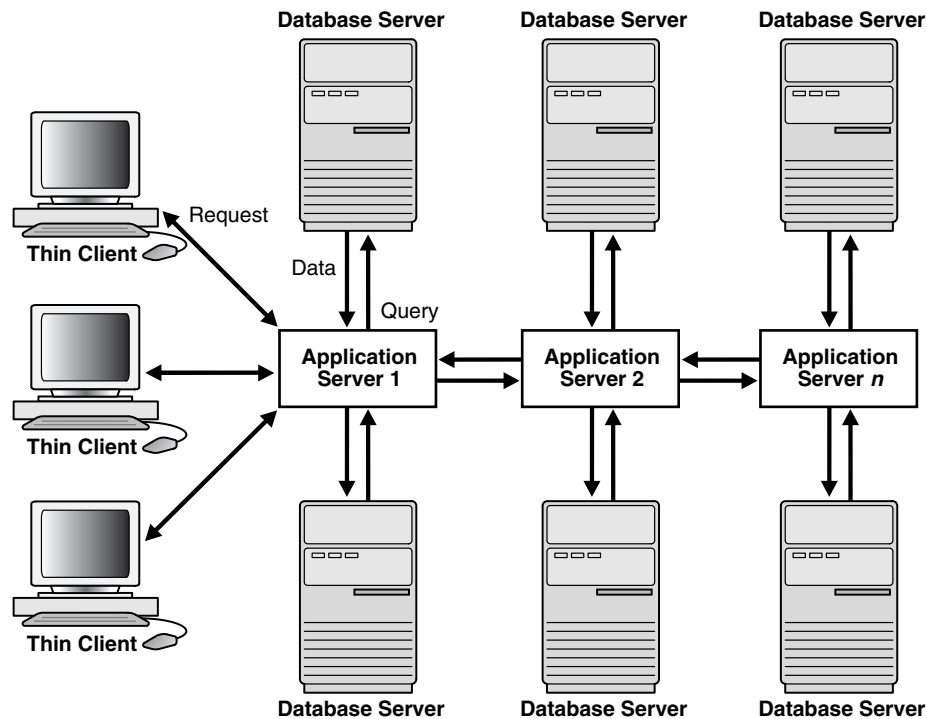
This section includes the following topics:

- Clients
- Application Servers
- Database Servers

Clients

A client initiates a request for an operation to be performed on the database server. The client can be a Web browser or other end-user process. In a multitier architecture, the client connects to the database server through one or more application servers.

Application Servers

An application server provides access to the data for the client. It serves as an interface between the client and one or more database servers, which provides an additional level of security. It can also perform some of the query processing for the client, thus removing some of the load from the database server.

The application server assumes the identity of the client when it is performing operations on the database server for that client. The application server's privileges are restricted to prevent it from performing unneeded and unwanted operations during a client operation.

Database Servers

A database server provides the data requested by an application server on behalf of a client. The database server does all of the remaining query processing.

The Oracle database server can audit operations performed by the application server on behalf of individual clients as well as operations performed by the application

server on its own behalf. For example, a client operation can be a request for information to be displayed on the client, whereas an application server operation can be a request for a connection to the database server.

> **See Also:** Chapter 20, "Database Security"

Oracle Database as a Web Service Provider

Beginning in Oracle Database 11*g*, Oracle Database can serve as a Web service provider in traditional multitier or *service oriented architecture* (SOA) environments. **SOA** is a multitier architecture in which application functionality is encapsulated in *services*. Services are designed to support interoperable machine-to-machine interaction over a network. They can be dynamically discovered, and can be queried on available functions and calling sequences.

SOA services are usually implemented as Web services. Web services can be accessed with the HTTP protocol and are based on a set of XML-based open standards, such as WSDL and SOAP.

The Oracle Database Web service capability, which is implemented as part of XML DB, must be specifically enabled by the DBA. Applications can then accomplish the following through database Web services:

- Submit SQL or XQuery queries and receive results as XML.

- Invoke standalone PL/SQL functions and receive results.

- Invoke PL/SQL package functions and receive results.

In a multitier environment, both clients and application servers can invoke database Web services.

> **Note:** Database Web services provide a simple way to add Web services to your application environment without the need for an application server. However, invoking Web services through application servers such as Oracle Fusion Middleware offers more in the way of security, scalability, UDDI registration, and reliable messaging in an SOA environment. Nevertheless, because database Web services integrate easily with Oracle Fusion Middleware, they may be an appropriate way to help optimize SOA solutions. See the Oracle Fusion Middleware documentation for more information on SOA and Web services.

> **See Also:** *Oracle XML DB Developer's Guide* for information on enabling and using database Web services.

Overview of Oracle Net Services

Oracle Net Services provides enterprise-wide connectivity solutions in distributed, heterogeneous computing environments. Oracle Net Services enables a network session from a client application to an Oracle database.

Oracle Net Services uses the communication protocols or application programmatic interfaces (APIs) supported by a wide range of networks to provide a distributed database and distributed processing for Oracle Database.

- A communication protocol is a set of rules that determine how applications access the network and how data is subdivided into packets for transmission across the network.

- An API is a set of subroutines that provide, in the case of networks, a means to establish remote process-to-process communication through a communication protocol.

After a network session is established, Oracle Net Services acts as a data courier for the client application and the database server. It is responsible for establishing and maintaining the connection between the client application and database server, as well as exchanging messages between them. Oracle Net Services is able to perform these jobs because it is located on each computer in the network.

Oracle Net Services provides location transparency, centralized configuration and management, and quick out-of-the-box installation and configuration. It also lets you maximize system resources and improve performance. The Oracle Database **shared server** architecture increases the scalability of applications and the number of clients simultaneously connected to the database.The **Virtual Interface (VI)** protocol places most of the messaging burden on high-speed network hardware, freeing the CPU for more important tasks.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for more information about these features

This section includes the following topics:

- How Oracle Net Services Works
- The Listener

How Oracle Net Services Works

Oracle's support of industry network protocols provides an interface between Oracle Database processes running on the database server and the user processes of Oracle Database applications running on other computers of the network.

The Oracle Database protocols take SQL statements from the interface of the Oracle applications and package them for transmission to Oracle Database through one of the supported industry-standard higher level protocols or programmatic interfaces. The protocols also take replies from Oracle Database and package them for transmission to the applications through the same higher level communications mechanism. This is all done independently of the network operating system.

Depending on the operation system that runs Oracle Database, the Oracle Net Services software of the database server could include the driver software and start an additional Oracle Database background process.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for more information about how Oracle Net Services works

The Listener

When an instance starts, a **listener process** establishes a communication pathway to Oracle Database. When a user process makes a connection request, the listener determines whether it should use a shared server dispatcher process or a dedicated server process and establishes an appropriate connection.

The listener also establishes a communication pathway between databases. When multiple databases or instances run on one computer, as in Oracle Real Application

Clusters, **service names** enable instances to register automatically with other listeners on the same computer. A service name can identify multiple instances, and an instance can belong to multiple services. Clients connecting to a service do not have to specify which instance they require.

Service Information Registration

Dynamic service registration reduces the administrative overhead for multiple databases or instances. Information about the services to which the listener forwards client requests is registered with the listener. Service information can be dynamically registered with the listener through a feature called **service registration** or statically configured in the listener.ora file.

Service registration relies on the **PMON process**—an instance background process—to register instance information with a listener, as well as the current state and load of the instance and **shared server** dispatchers. The registered information enables the listener to forward client connection requests to the appropriate service handler. Service registration does not require configuration in the listener.ora file.

The initialization parameter SERVICE_NAMES identifies which database services an instance belongs to. On startup, each instance registers with the listeners of other instances belonging to the same services. During database operations, the instances of each service pass information about CPU use and current connection counts to all of the listeners in the same services. This enables dynamic load balancing and connection failover.

> **See Also:**
>
> - "Shared Server Architecture" on page 9-13
>
> - "Dedicated Server Configuration" on page 9-17 for more information about server processes
>
> - *Oracle Database Net Services Administrator's Guide* for more information about the listener
>
> - Your platform-specific Oracle Real Application Clusters installation guide and *Oracle Real Application Clusters Administration and Deployment Guide* for information about instance registration and client/service connections in Oracle Real Application Clusters

11

Oracle Database Utilities

This chapter describes Oracle Database utilities for data transfer, data maintenance, and database administration.

This chapter contains the following topics:

- Introduction to Oracle Database Utilities
- Overview of Data Pump Export and Import
- Overview of the Data Pump API
- Overview of the Metadata API
- Overview of SQL*Loader
- Overview of External Tables
- Overview of LogMiner
- Overview of DBVERIFY Utility
- Overview of DBNEWID Utility
- ADRCI: ADR Command Interpreter

Introduction to Oracle Database Utilities

Oracle Database utilities let you perform the following tasks:

- High-speed movement of data and metadata from one database to another using Data Pump Export and Import
- Extract and manipulate complete representations of the metadata for database objects, using the Metadata API
- Move all or part of the data and metadata for a site from one database to another, using the Data Pump API
- Load data into Oracle Database tables from operating system files using SQL*Loader or from external sources using external tables
- Manage Oracle Database diagnostic data using the ADR Command Interpreter (ADRCI).
- Query redo log files through a SQL interface with LogMiner
- Perform physical data structure integrity checks on an offline (for example, backup) database or datafile with DBVERIFY.
- Maintain the internal database identifier (DBID) and the database name (DBNAME) for an operational database, using the DBNEWID utility

> **See Also:** *Oracle Database Utilities* for more information on all of the the utilities described in this chapter

Overview of Data Pump Export and Import

Oracle Data Pump technology enables very high-speed movement of data and metadata from one database to another. This technology is the basis for Oracle Database data movement utilities, Data Pump Export and Data Pump Import.

Data Pump enables you to specify whether a job should move a subset of the data and metadata. This is done using data filters and metadata filters, which are implemented through Export and Import parameters.

This section includes the following topics:

- Data Pump Export
- Data Pump Import

Data Pump Export

Data Pump Export (hereinafter referred to as Export for ease of reading) is a utility for unloading data and metadata into a set of operating system files called a dump file set. The dump file set can be moved to another system and loaded by the Data Pump Import utility.

The dump file set is made up of one or more disk files that contain table data, database object metadata, and control information. The files are written in a proprietary, binary format, which can be read only by Data Pump Import. During an import operation, the Data Pump Import utility uses these files to locate each database object in the dump file set.

Data Pump Import

Data Pump Import (hereinafter referred to as Import for ease of reading) is a utility for loading an export dump file set into a target system. The dump file set is made up of one or more disk files that contain table data, database object metadata, and control information. The files are written in a proprietary, binary format.

Import can also be used to load a target database directly from a source database with no intervening files, which allows export and import operations to run concurrently, minimizing total elapsed time. This is known as network import.

Import also enables you to see all of the SQL DDL that the Import job will be executing, without actually executing the SQL. This is implemented through the Import `SQLFILE` parameter.

Overview of the Data Pump API

The Data Pump API provides a high-speed mechanism to move all or part of the data and metadata for a site from one database to another. To use the Data Pump API, you use the procedures provided in the `DBMS_DATAPUMP` PL/SQL package. The Data Pump Export and Data Pump Import utilities are based on the Data Pump API.

See Also:

- *Oracle Database Utilities* for information about how the Data Pump API works

- *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_DATAPUMP package

Overview of the Metadata API

The Metadata application programming interface (API), provides a means for you to do the following:

- Retrieve an object's metadata as XML

- Transform the XML in a variety of ways, including transforming it into SQL DDL

- Submit the XML to re-create the object extracted by the retrieval

To use the Metadata API, you use the procedures provided in the DBMS_METADATA PL/SQL package. For the purposes of the Metadata API, every entity in the database is modeled as an object that belongs to an object type. For example, the table scott.emp is an object and its object type is TABLE. When you fetch an object's metadata you must specify the object type.

See Also:

- *Oracle Database Utilities* for information about how to use the Metadata API

- *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_METADATA package

Overview of SQL*Loader

SQL*Loader loads data from external files into tables of an Oracle database. It has a powerful data parsing engine that puts little limitation on the format of the data in the datafile. You can use SQL*Loader to do the following:

- Load data from multiple datafiles during the same load session.

- Load data into multiple tables during the same load session.

- Specify the character set of the data.

- Selectively load data (you can load records based on the records' values).

- Manipulate the data before loading it, using SQL functions.

- Generate unique sequential key values in specified columns.

- Use the operating system's file system to access the datafiles.

- Load data from disk, tape, or named pipe.

- Generate sophisticated error reports, which greatly aids troubleshooting.

- Load arbitrarily complex object-relational data.

- Use secondary datafiles for loading LOBs and collections.

- Use either conventional or direct path loading. While conventional path loading is very flexible, direct path loading provides superior loading performance.

A typical SQL*Loader session takes as input a control file, which controls the behavior of SQL*Loader, and one or more datafiles. The output of SQL*Loader is an Oracle database (where the data is loaded), a log file, a bad file, and potentially, a discard file.

Overview of External Tables

The external tables feature is a complement to existing SQL*Loader functionality. It lets you access data in external sources as if it were in a table in the database. External tables can be written to using the ORACLE_DATAPUMP access driver. Neither data manipulation language (DML) operations nor index creation are allowed on an external table. Therefore, SQL*Loader may be the better choice in data loading situations that require additional indexing of the staging table.

To use the external tables feature, you must have some knowledge of the file format and record format of the datafiles on your platform. You must also know enough about SQL to be able to create an external table and perform queries against it.

> **See Also:** "External Tables" on page 5-12

Overview of LogMiner

Oracle LogMiner enables you to query redo log files through a SQL interface. All changes made to user data or to the database dictionary are recorded in the Oracle Database redo log files. Therefore, redo log files contain all the necessary information to perform recovery operations.

LogMiner functionality is available through a command-line interface or through the Oracle LogMiner Viewer graphical user interface (GUI). The LogMiner Viewer is a part of Oracle Enterprise Manager.

The following are some of the potential uses for data contained in redo log files:

- Pinpointing when a logical corruption to a database, such as errors made at the application level, may have begun. This enables you to restore the database to the state it was in just before corruption.

- Detecting and whenever possible, correcting user error, which is a more likely scenario than logical corruption. User errors include deleting the wrong rows because of incorrect values in a WHERE clause, updating rows with incorrect values, dropping the wrong index, and so forth.

- Determining what actions you would have to take to perform fine-grained recovery at the transaction level. If you fully understand and take into account existing dependencies, it may be possible to perform a table-based undo operation to roll back a set of changes.

- Performance tuning and capacity planning through trend analysis. You can determine which tables get the most updates and inserts. That information provides a historical perspective on disk access statistics, which can be used for tuning purposes.

- Performing post-auditing. The redo log files contain all the information necessary to track any DML and DDL statements run on the database, the order in which they were run, and who executed them.

Overview of DBVERIFY Utility

DBVERIFY is an external command-line utility that performs a physical data structure integrity check. It can be used on offline or online databases, as well on backup files.

You use DBVERIFY primarily when you need to ensure that a backup database (or datafile) is valid before it is restored or as a diagnostic aid when you have encountered data corruption problems.

Because DBVERIFY can be run against an offline database, integrity checks are significantly faster.

DBVERIFY checks are limited to cache-managed blocks (that is, data blocks). Because DBVERIFY is only for use with datafiles, it will not work against control files or redo logs.

There are two command-line interfaces to DBVERIFY. With the first interface, you specify disk blocks of a single datafile for checking. With the second interface, you specify a segment for checking.

Overview of DBNEWID Utility

DBNEWID is a database utility that can change the internal, unique database identifier (DBID) and the database name (DBNAME) for an operational database. The DBNEWID utility lets you change any of the following:

- Only the DBID of a database

- Only the DBNAME of a database

- Both the DBNAME and DBID of a database

Therefore, you can manually create a copy of a database and give it a new DBNAME and DBID by re-creating the control file, and you can register a seed database and a manually copied database together in the same RMAN repository.

ADRCI: ADR Command Interpreter

ADRCI is a command-line tool that is part of the fault diagnosability infrastructure introduced in Oracle Database 11*g*. ADRCI enables you to:

- View diagnostic data within the **Automatic Diagnostic Repository** (ADR)

- Package incident and problem information into a zip file for transmission to Oracle Support

Diagnostic data includes incident and problem descriptions, trace files, dumps, health monitor reports, alert log entries, and more.

ADRCI has a rich command set, and can be used in interactive mode or within scripts. In addition, ADRCI can execute scripts of ADRCI commands in the same way that SQL*Plus executes scripts of SQL and PL/SQL commands.

See Also: *Oracle Database Utilities* for more information on ADRCI

12

Database and Instance Startup and Shutdown

This chapter explains the procedures involved in starting and stopping an Oracle database instance and database.

This chapter contains the following topics:

- Introduction to an Oracle Instance
- Overview of Instance and Database Startup
- Overview of Database and Instance Shutdown

Introduction to an Oracle Instance

Every running Oracle Database is associated with an Oracle database instance. When a database is started on a database server (regardless of the type of computer), Oracle Database allocates a memory area called the System Global Area (SGA) and starts one or more Oracle Database processes. This combination of the SGA and the Oracle Database processes is called an Oracle **instance**. The memory and processes of an instance manage the associated database's data efficiently and serve the one or multiple users of the database.

Figure 12–1 shows an Oracle database instance.

Figure 12–1   An Oracle Instance
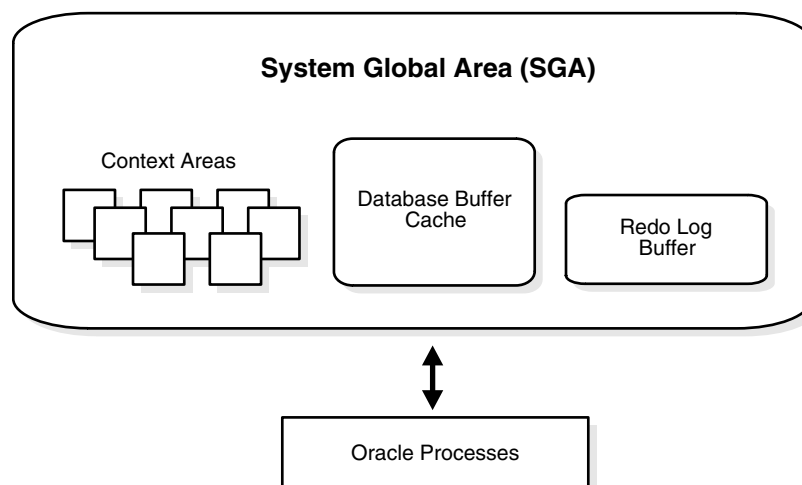
See Also:

- Chapter 8, "Memory Architecture"

- Chapter 9, "Process Architecture"

This section includes the following topics:

- The Instance and the Database
- Connection with Administrator Privileges
- Initialization Parameter Files and Server Parameter Files

The Instance and the Database

After starting an instance, Oracle Database associates the instance with the specified database. This is a **mounted database**. The database is then ready to be opened, which makes it accessible to authorized users.

Multiple instances can run concurrently on the same computer, each accessing its own physical database. In large-scale cluster systems, Oracle Real Application Clusters enables multiple instances to mount a single database.

Only the database administrator can start up an instance and open the database. If a database is open, then the database administrator can shut down the database so that it is closed. When a database is **closed**, users cannot access the data that it contains.

Security for database startup and shutdown is controlled through connections to Oracle Database with administrator privileges. Normal users do not have control over the current status of an Oracle database.

Connection with Administrator Privileges

Database startup and shutdown are powerful administrative options and are restricted to users who connect to Oracle Database with administrator privileges. Depending on the operating system, one of the following conditions establishes administrator privileges for a user:

- The user's operating system privileges allow him or her to connect using administrator privileges.

- The user is granted the SYSDBA or SYSOPER privileges and the database uses password files to authenticate database administrators.

When you connect with SYSDBA privileges, you are in the schema owned by SYS. When you connect as SYSOPER, you are in the public schema. SYSOPER privileges are a subset of SYSDBA privileges.

See Also:

- Your operating system-specific Oracle Database documentation for more information about how administrator privileges work on your operating system

- Chapter 20, "Database Security" for more information about password files and authentication schemes for database administrators

Initialization Parameter Files and Server Parameter Files

To start an instance, Oracle Database must read either an **initialization parameter file** or a **server parameter file**. These files contain a list of configuration parameters for that instance and database. Oracle Database traditionally stored initialization parameters in a text initialization parameter file. You can also choose to maintain initialization parameters in a server-side binary server parameter file (SPFILE).

Initialization parameters stored in a server parameter file are persistent, in that any changes made to the parameters while an instance is running can persist across instance shutdown and startup.

Initialization parameters are divided into two groups: basic and advanced. In the majority of cases, it is necessary to set and tune only the basic parameters to get reasonable performance. In rare situations, modification to the advanced parameters may be needed for optimal performance.

Most initialization parameters belong to one of the following groups:

- Parameters that name things, such as files

- Parameters that set limits, such as maximums

- Parameters that affect capacity, such as the size of the SGA, which are called **variable parameters**

Among other things, the initialization parameters tell Oracle Database:

- The name of the database for which to start up an instance

- How much memory to use for memory structures in the SGA

- What to do with filled redo log files

- The names and locations of the database control files

- The names of undo tablespaces in the database

> **See Also:** *Oracle Database Administrator's Guide*

This section includes the following topics:

- Server Parameter Files and Hardware Assisted Resilient Data

- Initialization Parameter Files and Server Parameter Files

Server Parameter Files and Hardware Assisted Resilient Data

The Oracle Hardware Assisted Resilient Data (HARD) initiative is a comprehensive program designed to prevent data corruptions before they happen. By implementing the Oracle data validation algorithms inside storage devices, Oracle Database can prevent corrupted data from being written to permanent storage. Starting in Oracle Database 11*g*, you can create a server parameter file in a new format that is usable on a HARD-compliant storage system. The database can read and write server parameter files in both the old and new format.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to create and manage a server parameter file

How Parameter Values Are Changed

The database administrator can adjust variable parameters to improve the performance of a database system. Exactly which parameters most affect a system depends on numerous database characteristics and variables.

Some parameters can be changed dynamically with the `ALTER SESSION` or `ALTER SYSTEM` statement while the instance is running. Unless you are using a server parameter file (`SPFILE`), changes made using the `ALTER SYSTEM` statement are only in effect for the current instance. You must manually update the text initialization parameter file for the changes to be known the next time you start up an instance.

When you use a server parameter file, you can use the `ALTER SYSTEM SET` statement to change parameter values in memory, disk, or both. The database prints the new value and the old value (if it exists) to the alert log. As a preventative measure, the database performs validation steps when you change a basic parameter to prevent illegal values from being written to the server parameter file.

Oracle Database provides values in the starter initialization parameter file provided with your database software, or as created for you by the Database Configuration Assistant. You can edit these Oracle-supplied initialization parameters and add others, depending upon your configuration and options and how you plan to tune the database. For any relevant initialization parameters not specifically included in the initialization parameter file, Oracle supplies defaults. If you are creating a database for the first time, it is suggested that you minimize the number of parameter values that you alter.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for a discussion of initialization parameters and the use of a server parameter file
> - *Oracle Database Reference* for descriptions of all initialization parameters

Overview of Instance and Database Startup

The three steps to starting an Oracle database and making it available for systemwide use are:

1. Start an instance.

2. Mount the database.

3. Open the database.

A database administrator can perform these steps using the SQL*Plus `STARTUP` statement or Enterprise Manager.

> **See Also:** *Oracle Database 2 Day DBA*

This section includes the following topics:

- How an Instance Is Started
- How a Database Is Mounted
- What Happens When You Open a Database

How an Instance Is Started

When Oracle Database starts an instance, it reads the server parameter file (SPFILE) or initialization parameter file to determine the values of initialization parameters. Then, it allocates an SGA, which is a shared area of memory used for database information, and creates background processes. At this point, no database is associated with these memory structures and processes.

When the instance starts, the database writes all explicit parameter settings to the alert log in valid parameter syntax. If necessary, you can copy and paste this text into a new parameter file and restart the instance. Note that changed

See Also:

- Chapter 8, "Memory Architecture" for information about the SGA

- Chapter 9, "Process Architecture" for information about background processes

This section includes the following topics:

- Restricted Mode of Instance Startup

- Forced Startup in Abnormal Situations

Restricted Mode of Instance Startup

You can start an instance in restricted mode (or later alter an existing instance to be in restricted mode). This restricts connections to only those users who have been granted the RESTRICTED SESSION system privilege.

Forced Startup in Abnormal Situations

In unusual circumstances, a previous instance might not have been shut down cleanly. For example, one of the instance's processes might not have terminated properly. In such situations, the database can return an error during normal instance startup. To resolve this problem, you must terminate all remnant Oracle Database processes of the previous instance before starting the new instance.

How a Database Is Mounted

The instance mounts a database to associate the database with that instance. To mount the database, the instance finds the database control files and opens them. Control files are specified in the CONTROL_FILES initialization parameter in the parameter file used to start the instance. Oracle Database then reads the control files to get the names of the database's datafiles and redo log files.

At this point, the database is still closed and is accessible only to the database administrator. The database administrator can keep the database closed while completing specific maintenance operations. However, the database is not yet available for normal operations.

This section includes the following topics:

- How a Database Is Mounted with Oracle Real Application Clusters

- How a Clone Database Is Mounted

How a Database Is Mounted with Oracle Real Application Clusters

If Oracle Database allows multiple instances to mount the same database concurrently, the database administrator can use the CLUSTER_DATABASE initialization parameter to make the database available to multiple instances. The default value of the CLUSTER_DATABASE parameter is false. Versions of Oracle Database that do not support Oracle RAC only allow CLUSTER_DATABASE to be false.

If CLUSTER_DATABASE is false for the first instance that mounts a database, then only that instance can mount the database. If CLUSTER_DATABASE is set to true on the first instance, then other instances can mount the database if their CLUSTER_

DATABASE parameters are set to `true`. The number of instances that can mount the database is subject to a predetermined maximum, which you can specify when creating the database.

> **See Also:**
>
> - *Oracle Real Application Clusters Installation and Configuration Guide*
>
> - *Oracle Real Application Clusters Administration and Deployment Guide*
>
> for more information about the use of multiple instances with a single database

How a Clone Database Is Mounted

A **clone database** is a specialized copy of a database that can be used for tablespace point-in-time recovery. When you perform tablespace point-in-time recovery, you mount the clone database and recover the tablespaces to the desired time, then export metadata from the clone to the primary database and copy the datafiles from the recovered tablespaces.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for information about clone databases and tablespace point-in-time recovery

What Happens When You Open a Database

Opening a **mounted database** makes it available for normal database operations. Any valid user can connect to an open database and access its information. Usually, a database administrator opens the database to make it available for general use.

When you open the database, Oracle Database opens the online datafiles and redo log files. If a tablespace was offline when the database was previously shut down, the tablespace and its corresponding datafiles will still be offline when you reopen the database.

If any of the datafiles or redo log files are not present when you attempt to open the database, then Oracle Database returns an error. You must perform recovery on a backup of any damaged or missing files before you can open the database.

> **See Also:** "Online and Offline Tablespaces" on page 3-11 for information about opening an offline tablespace

This section includes the following topics:

- Crash and Instance Recovery

- Undo Space Acquisition and Management

- Resolution of In-Doubt Distributed Transaction

- Open a Database in Read-Only Mode

Crash and Instance Recovery

Database buffers in the buffer cache in the SGA are written to disk only when necessary, using a least-recently-used (LRU) algorithm. Because of the way that the database writer process uses this algorithm to write database buffers to datafiles,

datafiles could contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.

Two potential problems can result if an instance failure occurs:

- Data blocks modified by a transaction might not be written to the datafiles at commit time and might only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.

- After the roll forward phase, the datafiles could contain changes that had not been committed at the time of the failure. These uncommitted changes must be rolled back to ensure transactional consistency. These changes were either saved to the datafiles before the failure, or introduced during the roll forward phase.

If the database was last closed abnormally, either because the database administrator terminated its instance or because of a power failure, Oracle Database automatically performs instance or crash recovery when the database is reopened.

Crash recovery is used to recover from a failure either when a single-instance database fails or all instances of an Oracle Real Application Clusters database fail. Instance recovery refers to the case where a surviving instance recovers a failed instance in an Oracle Real Application Clusters database.

The goal of crash and instance recovery is to restore the data block changes located in the cache of the terminated instance and to close the redo thread that was left open. Instance and crash recovery use only online redo log files and current online datafiles. Oracle Database recovers the **redo thread**s of the terminated instances together.

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 Oracle Wallet after database mount and before database open, so the recovery process can decrypt data blocks and redo.

Crash and instance recovery involve two distinct operations: rolling forward the current, online datafiles by applying both committed and uncommitted transactions contained in online redo records, and then rolling back changes made in uncommitted transactions to their original state.

Crash and instance recovery have the following shared characteristics:

- Redo the changes using the current online datafiles (as left on disk after the failure or `SHUTDOWN ABORT`)

- Use only the online redo logs and never require the use of the archived logs

- Have a recovery time governed by the number of terminated instances, amount of redo generated in each terminated redo thread since the last checkpoint, and by user-configurable factors such as the number and size of redo log files, checkpoint frequency, and the parallel recovery setting

Oracle Database performs this recovery automatically on two occasions:

- At the first database open after the failure of a single-instance database or all instances of an Oracle RAC database (crash recovery).

- When some but not all instances of an Oracle RAC configuration fail (instance recovery). The recovery is performed automatically by a surviving instance in the configuration.

The important point is that in both crash and instance recovery, Oracle Database applies the redo automatically: no user intervention is required to supply redo logs. Nevertheless, you can set parameters in the database server that can tune the duration

of instance and crash recovery performance. Also, you can tune the rolling forward and rolling back phases of instance recovery separately.

To solve this dilemma, two separate steps are generally used by Oracle Database for a successful recovery of a system failure: rolling forward with the redo log (cache recovery) and rolling back with the rollback or undo segments (transaction recovery).

This section includes the following topics:

- Cache Recovery
- Transaction Recovery

Cache Recovery To solve this dilemma, two separate steps are generally used by Oracle Database for a successful recovery of a system failure: rolling forward with the redo log (cache recovery) and rolling back with the rollback or undo segments (transaction recovery).

The **online redo log** is a set of operating system files that record all changes made to any database block, including data, index, and rollback segments, *whether the changes are committed or uncommitted.* All changes to Oracle Database blocks are recorded in the online redo log.

The first step of recovery from an instance or media failure is called **cache recovery** or **rolling forward**, and involves reapplying all of the changes recorded in the redo log to the datafiles. Because rollback data is also recorded in the redo log, rolling forward also regenerates the corresponding rollback segments.

Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time. Rolling forward usually includes online redo log files (instance recovery or media recovery) and could include archived redo log files (media recovery only).

After rolling forward, the data blocks contain all committed changes. They could also contain uncommitted changes that were either saved to the datafiles before the failure, or were recorded in the redo log and introduced during cache recovery.

Transaction Recovery After the roll forward, any changes that were not committed must be undone. Oracle Database applies undo blocks to roll back uncommitted changes in data blocks that were either written before the failure or introduced by redo application during cache recovery. This process is called **rolling back** or **transaction recovery**.

Figure 12–2 illustrates rolling forward and rolling back, the two steps necessary to recover from any type of system failure.

Figure 12–2 Basic Recovery Steps: Rolling Forward and Rolling Back
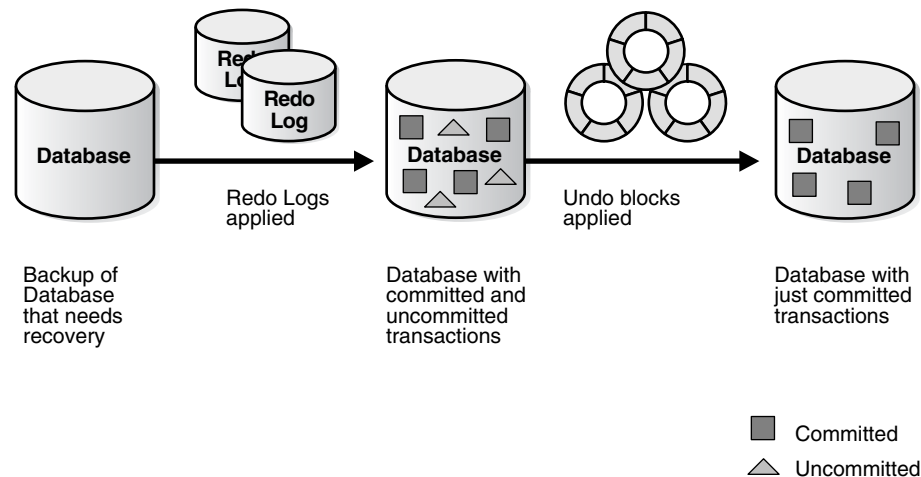
Oracle Database can roll back multiple transactions simultaneously as needed. All transactions that were active at the time of failure are marked as terminated. Instead of waiting for SMON to roll back terminated transactions, new transactions can recover blocking transactions themselves to get the row locks they need.

> **See Also:**
>
> - *Oracle Database Performance Tuning Guide* for a discussion of instance recovery mechanics and instructions for tuning instance and crash recovery
>
> - "Introduction to Undo Segments and Automatic Undo Management" on page 2-16 for more information about undo

Undo Space Acquisition and Management

When you open the database, the instance attempts to acquire an undo tablespace. If more than one undo tablespace exists, the UNDO_TABLESPACE initialization parameter designates the undo tablespace to use. If this parameter is blank, the first available undo tablespace in the database is chosen.

> **See Also:** "Introduction to Undo Segments and Automatic Undo Management" on page 2-16 for more information about undo tablespaces.

Resolution of In-Doubt Distributed Transaction

Occasionally a database closes abnormally with one or more distributed transactions **in doubt** (neither committed nor rolled back). When you reopen the database and recovery is complete, the RECO background process automatically, immediately, and consistently resolves any in-doubt distributed transactions.

> **See Also:** *Oracle Database Administrator's Guide* for information about recovery from distributed transaction failures

Open a Database in Read-Only Mode

You can open any database in read-only mode to prevent its data from being modified by user transactions. Read-only mode restricts database access to read-only transactions, which cannot write to the datafiles or to the redo log files.

Disk writes to other files, such as control files, operating system audit trails, trace files, and alert logs, can continue in read-only mode. Temporary tablespaces for sort operations are not affected by the database being open in read-only mode. However, you cannot take permanent tablespaces offline while a database is open in read-only mode. Also, job queues are not available in read-only mode.

Read-only mode does not restrict database recovery or operations that change the database's state without generating redo data. For example, in read-only mode:

- Datafiles can be taken offline and online

- Offline datafiles and tablespaces can be recovered

- The control file remains available for updates about the state of the database

One useful application of read-only mode is that **standby database**s can function as temporary reporting databases.

> **See Also:** *Oracle Database Administrator's Guide* for information about how to open a database in read-only mode

Limitations of a Read-only Database

- An application must not write database objects while executing against a read-only database. For example, an application writes database objects when it inserts, deletes, updates, or merges rows in a database table, including a global temporary table. An application writes database objects when it manipulates a database sequence. An application writes database objects when it locks rows, when it runs EXPLAIN PLAN, or when it executes DDL. Many of the functions and procedures in Oracle-supplied PL/SQL packages, such as DBMS_SCHEDULER, write database objects. If your application calls any of these functions and procedures, or if it performs any of the operations mentioned above, your application writes database objects and hence is not read-only.

- When executing on a read-only database, you must commit or roll back any in-progress transaction that involves one database link before you use another database link. This is true even if you execute a generic SELECT statement on the first database link and the *transaction* is currently read-only.

- You cannot compile or recompile PL/SQL stored procedures on a read-only database. To minimize PL/SQL invalidation because of remote procedure calls, use REMOTE_DEPENDENCIES_MODE=SIGNATURE in any session that does remote procedure calls on a read-only database.

- You cannot invoke a remote procedure (even a read-only remote procedure) from a read-only database if the remote procedure has never been called on the database. This limitation applies to remote procedure calls in anonymous PL/SQL blocks and in SQL statements. You can either put the remote procedure call in a stored procedure, or you can invoke the remote procedure in the database prior to it becoming read only.

Overview of Database and Instance Shutdown

The three steps to shutting down a database and its associated instance are:

1. Close the database.

2. Unmount the database.

3. Shut down the instance.

A database administrator can perform these steps using Enterprise Manager. Oracle Database automatically performs all three steps whenever an instance is shut down.

See Also: *Oracle Database 2 Day DBA*

This section includes the following topics:

- Close a Database
- Unmount a Database
- Shut Down an Instance

Close a Database

When you close a database, Oracle Database writes all database data and recovery data in the SGA to the datafiles and redo log files, respectively. Next, Oracle Database closes all online datafiles and redo log files. (Any offline datafiles of any offline tablespaces have been closed already. If you subsequently reopen the database, any tablespace that was offline and its datafiles remain offline and closed, respectively.) At this point, the database is closed and inaccessible for normal operations. The control files remain open after a database is closed but still mounted.

Close the Database by Terminating the Instance

In rare emergency situations, you can terminate the instance of an open database to close and completely shut down the database instantaneously. This process is fast, because the operation of writing all data in the buffers of the SGA to the datafiles and redo log files is skipped. The subsequent reopening of the database requires recovery, which Oracle Database performs automatically.

> **Note:** If a system or power failure occurs while the database is open, then the instance is, in effect, terminated, and recovery is performed when the database is reopened.

Unmount a Database

After the database is closed, Oracle Database unmounts the database to disassociate it from the instance. At this point, the instance remains in the memory of your computer.

After a database is unmounted, Oracle Database closes the control files of the database.

Shut Down an Instance

The final step in database shutdown is shutting down the instance. When you shut down an instance, the SGA is removed from memory and the background processes are terminated.

Abnormal Instance Shutdown

In unusual circumstances, shutdown of an instance might not occur cleanly; all memory structures might not be removed from memory or one of the background processes might not be terminated. When remnants of a previous instance exist, a subsequent instance startup most likely will fail. In such situations, the database administrator can force the new instance to start up by first removing the remnants of the previous instance and then starting a new instance, or by issuing a SHUTDOWN ABORT statement in SQL*Plus or using Enterprise Manager.

See Also: *Oracle Database Administrator's Guide* for more detailed
information about instance and database startup and shutdown

Part III

Oracle Database Features

Part III describes the core feature areas in the Oracle Database.

Part III contains the following chapters:

13

Data Concurrency and Consistency

This chapter explains how Oracle Database maintains consistent data in a multiuser database environment.

This chapter contains the following topics:

- Introduction to Data Concurrency and Consistency in a Multiuser Environment
- How Oracle Database Manages Data Concurrency and Consistency
- How Oracle Database Locks Data
- Overview of Oracle Flashback Query

Introduction to Data Concurrency and Consistency in a Multiuser Environment

In a single-user database, the user can modify data in the database without concern for other users modifying the same data at the same time. However, in a multiuser database, the statements within multiple simultaneous transactions can update the same data. Transactions executing at the same time need to produce meaningful and consistent results. Therefore, control of data concurrency and data consistency is vital in a multiuser database.

- **Data concurrency** means that many users can access data at the same time.

- **Data consistency** means that each user sees a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.

To describe consistent transaction behavior when transactions run at the same time, database researchers have defined a transaction isolation model called **serializability**. The serializable mode of transaction behavior tries to ensure that transactions run in such a way that they appear to be executed one at a time, or serially, rather than concurrently.

While this degree of isolation between transactions is generally desirable, running many applications in this mode can seriously compromise application throughput. Complete isolation of concurrently running transactions could mean that one transaction cannot perform an insert into a table being queried by another transaction. In short, real-world considerations usually require a compromise between perfect transaction isolation and performance.

Oracle Database offers two isolation levels, providing application developers with operational modes that preserve consistency and provide high performance.

> **See Also:** Chapter 21, "Data Integrity" for information about data integrity, which enforces business rules associated with a database

This section includes the following topics:

- Preventable Phenomena and Transaction Isolation Levels
- Overview of Locking Mechanisms

Preventable Phenomena and Transaction Isolation Levels

The ANSI/ISO SQL standard (SQL92) defines four levels of transaction isolation with differing degrees of impact on transaction processing throughput. These isolation levels are defined in terms of three phenomena that must be prevented between concurrently executing transactions.

The three preventable phenomena are:

- Dirty reads: A transaction reads data that has been written by another transaction that has not been committed yet.
- Nonrepeatable (fuzzy) reads: A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data.
- Phantom reads (or phantoms): A transaction re-runs a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition.

SQL92 defines four levels of isolation in terms of the phenomena a transaction running at a particular isolation level is permitted to experience. They are shown in Table 13–1:

Table 13–1 Preventable Read Phenomena by Isolation Level

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Oracle Database offers the read committed and serializable isolation levels, as well as a read-only mode that is not part of SQL92. Read committed is the default.

> **See Also:** "How Oracle Database Manages Data Concurrency and Consistency" on page 13-3 for a full discussion of read committed and serializable isolation levels

Overview of Locking Mechanisms

In general, multiuser databases use some form of data locking to solve the problems associated with data concurrency, consistency, and integrity. **Locks** are mechanisms that prevent destructive interaction between transactions accessing the same resource.

Resources include two general types of objects:

- User objects, such as tables and rows (structures and data)
- System objects not visible to users, such as shared data structures in the memory and data dictionary rows

> **See Also:** "How Oracle Database Locks Data" on page 13-13 for
> more information about locks

How Oracle Database Manages Data Concurrency and Consistency

Oracle Database maintains data consistency in a multiuser environment by using a
multiversion consistency model and various types of locks and transactions. The
following topics are discussed in this section:

- Multiversion Concurrency Control

- Statement-Level Read Consistency

- Transaction-Level Read Consistency

- Read Consistency with Oracle Real Application Clusters

- Oracle Database Isolation Levels

- Comparison of Read Committed and Serializable Isolation

- Choice of Isolation Level

Multiversion Concurrency Control

Oracle Database automatically provides read consistency to a query so that all the data
that the query sees comes from a single point in time **(statement-level read
consistency)**. Oracle Database can also provide read consistency to all of the queries in
a transaction **(transaction-level read consistency)**.

Oracle Database uses the information maintained in its rollback segments to provide
these consistent views. The rollback segments contain the old values of data that have
been changed by uncommitted or recently committed transactions. Figure 13–1 shows
how Oracle Database provides statement-level read consistency using data in rollback
segments.

Figure 13–1   Transactions and Read Consistency
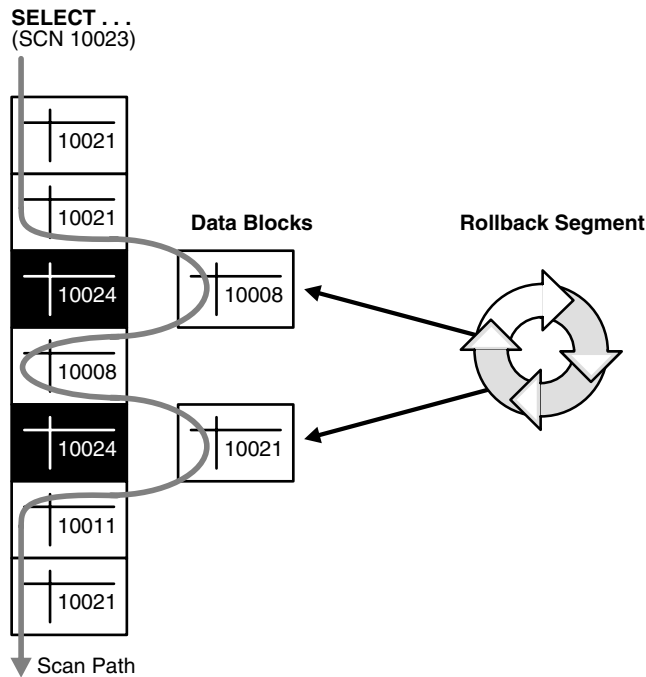
As a query enters the execution stage, the current system change number (SCN) is determined. In Figure 13–1, this system change number is 10023. As data blocks are read on behalf of the query, only blocks written with the observed SCN are used. Blocks with changed data (more recent SCNs) are reconstructed from data in the rollback segments, and the reconstructed data is returned for the query. Therefore, each query returns all committed data with respect to the SCN recorded at the time that query execution began. Changes of other transactions that occur during a query's execution are not observed, guaranteeing that consistent data is returned for each query.

Statement-Level Read Consistency

Oracle Database always enforces **statement-level** read consistency. This guarantees that all the data returned by a single query comes from a single point in time—the time that the query began. Therefore, a query never sees dirty data or any of the changes made by transactions that commit during query execution. As query execution proceeds, only data committed before the query began is visible to the query. The query does not see changes committed after statement execution begins.

A consistent result set is provided for every query, guaranteeing data consistency, with no action on the user's part. The SQL statements SELECT, INSERT with a subquery, UPDATE, and DELETE all query data, either explicitly or implicitly, and all return consistent data. Each of these statements uses a query to determine which data it will affect (SELECT, INSERT, UPDATE, or DELETE, respectively).

A SELECT statement is an explicit query and can have nested queries or a join operation. An INSERT statement can use nested queries. UPDATE and DELETE statements can use WHERE clauses or subqueries to affect only some rows in a table rather than all rows.

Queries used in INSERT, UPDATE, and DELETE statements are guaranteed a consistent set of results. However, they do not see the changes made by the DML statement itself.

In other words, the query in these operations sees data as it existed before the operation began to make changes.

> **Note:** If a `SELECT` list contains a function, then the database applies statement-level read consistency at the statement level for SQL run within the PL/SQL function code, rather than at the parent SQL level. For example, a function could access a table whose data is changed and committed by another user. For each execution of the `SELECT` in the function, a new read consistent snapshot is established.

Transaction-Level Read Consistency

Oracle Database also offers the option of enforcing **transaction-level read consistency**. When a transaction runs in serializable mode, all data accesses reflect the state of the database as of the time the transaction began. This means that the data seen by all queries within the same transaction is consistent with respect to a single point in time, except that queries made by a serializable transaction do see changes made by the transaction itself. Transaction-level read consistency produces repeatable reads and does not expose a query to phantoms.

Read Consistency with Oracle Real Application Clusters

Oracle Real Application Clusters (Oracle RAC)s uses a cache-to-cache block transfer mechanism known as Cache Fusion to transfer read-consistent images of blocks from one instance to another. Oracle RAC does this using high speed, low latency interconnects to satisfy remote requests for data blocks.

> **See Also:** *Oracle Real Application Clusters Administration and Deployment Guide*

Oracle Database Isolation Levels

Oracle Database provides these transaction isolation levels.

Isolation Level	Description
Read committed	This is the default transaction isolation level. Each query executed by a transaction sees only data that was committed before the query (not the transaction) began. An Oracle Database query never reads dirty (uncommitted) data.
	Because Oracle Database does not prevent other transactions from modifying the data read by a query, that data can be changed by other transactions between two executions of the query. Thus, a transaction that runs a given query twice can experience both nonrepeatable read and phantoms.
Serializable	Serializable transactions see only those changes that were committed at the time the transaction began, plus those changes made by the transaction itself through `INSERT`, `UPDATE`, and `DELETE` statements. Serializable transactions do not experience nonrepeatable reads or phantoms.
Read-only	Read-only transactions see only those changes that were committed at the time the transaction began and do not allow `INSERT`, `UPDATE`, and `DELETE` statements.

This section includes the following topics:

Set the Isolation Level

Application designers, application developers, and database administrators can choose appropriate isolation levels for different transactions, depending on the application and workload. You can set the isolation level of a transaction by using one of these statements at the beginning of a transaction:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION READ ONLY;
```

To save the networking and processing cost of beginning each transaction with a SET TRANSACTION statement, you can use the ALTER SESSION statement to set the transaction isolation level for all subsequent transactions:

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;

ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;
```

> **See Also:** *Oracle Database SQL Language Reference* for detailed information on any of these SQL statements

Read Committed Isolation

The default isolation level for Oracle Database is read committed. This degree of isolation is appropriate for environments where few transactions are likely to conflict. Oracle Database causes each query to run with respect to its own materialized view time, thereby permitting nonrepeatable reads and phantoms for multiple executions of a query, but providing higher potential throughput. Read committed isolation is the appropriate level of isolation for environments where few transactions are likely to conflict.

Serializable Isolation

Serializable isolation is suitable for environments:

- With large databases and short transactions that update only a few rows

- Where the chance that two concurrent transactions will modify the same rows is relatively low

- Where relatively long-running transactions are primarily read only

Serializable isolation permits concurrent transactions to make only those database changes they could have made if the transactions had been scheduled to run one after another. Specifically, Oracle Database permits a serializable transaction to modify a data row only if it can determine that prior changes to the row were made by transactions that had committed when the serializable transaction began.

To make this determination efficiently, Oracle Database uses control information stored in the data block that indicates which rows in the block contain committed and uncommitted changes. In a sense, the block contains a recent history of transactions that affected each row in the block. The amount of history that is retained is controlled by the INITRANS parameter of CREATE TABLE and ALTER TABLE.

Under some circumstances, Oracle Database can have insufficient history information to determine whether a row has been updated by a too recent transaction. This can occur when many transactions concurrently modify the same data block, or do so in a very short period. You can avoid this situation by setting higher values of `INITRANS` for tables that will experience many transactions updating the same blocks. Doing so enables Oracle Database to allocate sufficient storage in each block to record the history of recent transactions that accessed the block.

Oracle Database generates an error when a serializable transaction tries to update or delete data modified by a transaction that commits *after* the serializable transaction began:

```
ORA-08177: Cannot serialize access for this transaction
```

When a serializable transaction fails with the `Cannot serialize access` error, the application can take any of several actions:

- Commit the work executed to that point

- Execute additional (but different) statements (perhaps after rolling back to a savepoint established earlier in the transaction)

- Undo the entire transaction

Figure 13–2 shows an example of an application that rolls back and retries the transaction after it fails with the `Cannot serialize access` error:

Figure 13–2   Serializable Transaction Failure
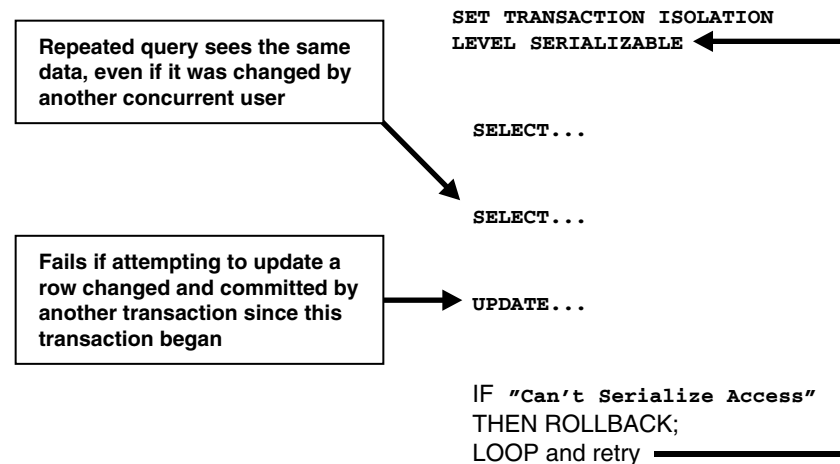
Comparison of Read Committed and Serializable Isolation

Oracle Database gives the application developer a choice of two transaction isolation levels with different characteristics. Both the read committed and serializable isolation levels provide a high degree of consistency and concurrency. Both levels provide the contention-reducing benefits of the Oracle Database read consistency multiversion concurrency control model and exclusive row-level locking implementation and are designed for real-world application deployment.

This section includes the following topics:

- Transaction Set Consistency

- Row-Level Locking

- Referential Integrity
- Distributed Transactions

Transaction Set Consistency

A useful way to view the read committed and serializable isolation levels in Oracle Database is to consider the following scenario: Assume you have a collection of database tables (or any set of data), a particular sequence of reads of rows in those tables, and the set of transactions committed at any particular time. An operation (a query or a transaction) is **transaction set consistent** if all its reads return data written by the same set of committed transactions. An operation is not transaction set consistent if some reads reflect the changes of one set of transactions and other reads reflect changes made by other transactions. An operation that is not transaction set consistent in effect sees the database in a state that reflects no single set of committed transactions.

Oracle Database provides transactions executing in read committed mode with transaction set consistency for each statement. Serializable mode provides transaction set consistency for each transaction.

Table 13–2 summarizes key differences between read committed and serializable transactions in Oracle Database.

Table 13–2   Read Committed and Serializable Transactions

	Read Committed	Serializable
Dirty write	Not possible	Not possible
Dirty read	Not possible	Not possible
Nonrepeatable read	Possible	Not possible
Phantoms	Possible	Not possible
Compliant with ANSI/ISO SQL 92	Yes	Yes
Read materialized view time	Statement	Transaction
Transaction set consistency	Statement level	Transaction level
Row-level locking	Yes	Yes
Readers block writers	No	No
Writers block readers	No	No
Different-row writers block writers	No	No
Same-row writers block writers	Yes	Yes
Waits for blocking transaction	Yes	Yes
Subject to `cannot serialize access`	No	Yes
Error after blocking transaction terminates	No	No
Error after blocking transaction commits	No	Yes

Row-Level Locking

Both read committed and serializable transactions use row-level locking, and both will wait if they try to change a row updated by an uncommitted concurrent transaction. The second transaction that tries to update a given row waits for the other transaction to commit or undo and release its lock. If that other transaction rolls back, the waiting

transaction, regardless of its isolation mode, can proceed to change the previously locked row as if the other transaction had not existed.

However, if the other blocking transaction commits and releases its locks, a read committed transaction proceeds with its intended update. A serializable transaction, however, fails with the error `Cannot serialize access` error, because the other transaction has committed a change that was made since the serializable transaction began.

Referential Integrity

Because Oracle Database does not use read locks in either read-consistent or serializable transactions, data read by one transaction can be overwritten by another. Transactions that perform database consistency checks at the application level cannot assume that the data they read will remain unchanged during the execution of the transaction even though such changes are not visible to the transaction. Database inconsistencies can result unless such application-level consistency checks are coded with this in mind, even when using serializable transactions.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for more information about referential integrity and serializable transactions

> **Note:** You can use both read committed and serializable transaction isolation levels with Oracle Real Application Clusters.

Distributed Transactions

In a distributed database environment, a given transaction updates data in multiple physical databases protected by two-phase commit to ensure all nodes or none commit. In such an environment, all servers, whether Oracle or non-Oracle, that participate in a **serializable** transaction are required to support serializable isolation mode.

If a serializable transaction tries to update data in a database managed by a server that does not support serializable transactions, the transaction receives an error. The transaction can undo and retry only when the remote server does support serializable transactions.

In contrast, **read committed** transactions can perform distributed transactions with servers that do not support serializable transactions.

> **See Also:** *Oracle Database Administrator's Guide*

Choice of Isolation Level

Application designers and developers should choose an isolation level based on application performance and consistency needs as well as application coding requirements.

For environments with many concurrent users rapidly submitting transactions, designers must assess transaction performance requirements in terms of the expected transaction arrival rate and response time demands. Frequently, for high-performance environments, the choice of isolation levels involves a trade-off between consistency and concurrency.

Application logic that checks database consistency must take into account the fact that reads do not block writes in either mode.

Oracle Database isolation modes provide high levels of consistency, concurrency, and performance through the combination of row-level locking and the Oracle Database multiversion concurrency control system. Readers and writers do not block one another in Oracle Database. Therefore, while queries still see consistent data, both read committed and serializable isolation provide a high level of concurrency for high performance, without the need for reading uncommitted data.

This section includes the following topics:

- Read Committed Isolation
- Serializable Isolation
- Quiesce Database

Read Committed Isolation

For many applications, read committed is the most appropriate isolation level. Read committed isolation can provide considerably more concurrency with a somewhat increased risk of inconsistent results due to phantoms and non-repeatable reads for some transactions.

Many high-performance environments with high transaction arrival rates require more throughput and faster response times than can be achieved with serializable isolation. Other environments that supports users with a very low transaction arrival rate also face very low risk of incorrect results due to phantoms and nonrepeatable reads. Read committed isolation is suitable for both of these environments.

Oracle Database read committed isolation provides transaction set consistency for every query. That is, every query sees data in a consistent state. Therefore, read committed isolation will suffice for many applications that might require a higher degree of isolation if run on other database management systems that do not use multiversion concurrency control.

Read committed isolation mode does not require application logic to trap the `Cannot serialize access` error and loop back to restart a transaction. In most applications, few transactions have a functional need to issue the same query twice, so for many applications protection against phantoms and non-repeatable reads is not important. Therefore many developers choose read committed to avoid the need to write such error checking and retry code in each transaction.

Serializable Isolation

The Oracle Database serializable isolation is suitable for environments where there is a relatively low chance that two concurrent transactions will modify the same rows and the long-running transactions are primarily read only. It is most suitable for environments with large databases and short transactions that update only a few rows.

Serializable isolation mode provides somewhat more consistency by protecting against phantoms and nonrepeatable reads and can be important where a read/write transaction runs a query more than once.

Unlike other implementations of serializable isolation, which lock blocks for read as well as write, Oracle Database provides nonblocking queries and the fine granularity of row-level locking, both of which reduce read/write contention. For applications that experience mostly read/write contention, Oracle Database serializable isolation can provide significantly more throughput than other systems. Therefore, some applications might be suitable for serializable isolation on Oracle Database but not on other systems.

All queries in an Oracle Database serializable transaction see the database as of a single point in time, so this isolation level is suitable where multiple consistent queries must be issued in a read/write transaction. A report-writing application that generates summary data and stores it in the database might use serializable mode because it provides the consistency that a READ ONLY transaction provides, but also allows INSERT, UPDATE, and DELETE.

> **Note:** Transactions containing DML statements with subqueries should use serializable isolation to guarantee consistent read.

Coding serializable transactions requires extra work by the application developer to check for the Cannot serialize access error and to undo and retry the transaction. Similar extra coding is needed in other database management systems to manage deadlocks. For adherence to corporate standards or for applications that are run on multiple database management systems, it may be necessary to design transactions for serializable mode. Transactions that check for serializability failures and retry can be used with Oracle Database read committed mode, which does not generate serializability errors.

Serializable mode is probably not the best choice in an environment with relatively long transactions that must update the same rows accessed by a high volume of short update transactions. Because a longer running transaction is unlikely to be the first to modify a given row, it will repeatedly need to roll back, wasting work. Note that a conventional read-locking, pessimistic implementation of serializable mode would not be suitable for this environment either, because long-running transactions—even read transactions—would block the progress of short update transactions and vice versa.

Application developers should take into account the cost of rolling back and retrying transactions when using serializable mode. As with read-locking systems, where deadlocks occur frequently, use of serializable mode requires rolling back the work done by terminated transactions and retrying them. In a high contention environment, this activity can use significant resources.

In most environments, a transaction that restarts after receiving the Cannot serialize access error is unlikely to encounter a second conflict with another transaction. For this reason, it can help to run those statements most likely to contend with other transactions as early as possible in a serializable transaction. However, there is no guarantee that the transaction will complete successfully, so the application should be coded to limit the number of retries.

Although Oracle Database serializable mode is compatible with SQL92 and offers many benefits compared with read-locking implementations, it does not provide semantics identical to such systems. Application designers must take into account the fact that reads in Oracle Database do not block writes as they do in other systems. Transactions that check for database consistency at the application level can require coding techniques such as the use of SELECT FOR UPDATE. This issue should be considered when applications using serializable mode are ported to Oracle Database from other environments.

Quiesce Database

You can put the system into **quiesced state**. The system is in quiesced state if there are no active sessions, other than SYS and SYSTEM. An active session is defined as a session that is currently inside a transaction, a query, a fetch or a PL/SQL procedure, or a session that is currently holding any shared resources (for example, enqueues--enqueues are shared memory structures that serialize access to database

resources and are associated with a session or transaction). Database administrators are the only users who can proceed when the system is in quiesced state.

Database administrators can perform certain actions in the quiesced state that cannot be safely done when the system is not quiesced. These actions include:

- Actions that might fail if there are concurrent user transactions or queries. For example, changing the schema of a database table will fail if a concurrent transaction is accessing the same table.

- Actions whose intermediate effect could be detrimental to concurrent user transactions or queries. For example, suppose there is a big table T and a PL/SQL package that operates on it. You can split table T into two tables T1 and T2, and change the PL/SQL package to make it refer to the new tables T1 and T2, instead of the old table T.

   When the database is in quiesced state, you can do the following:

   ```
   CREATE TABLE T1 AS SELECT ... FROM T;
   CREATE TABLE T2 AS SELECT ... FROM T;
   DROP TABLE T;
   ```

   You can then drop the old PL/SQL package and re-create it.

For systems that must operate continuously, the ability to perform such actions without shutting down the database is critical.

The Database Resource Manager blocks all actions that were initiated by a user other than SYS or SYSTEM while the system is quiesced. Such actions are allowed to proceed when the system goes back to normal (unquiesced) state. Users do not get any additional error messages from the quiesced state.

How a Database Is Quiesced The database administrator uses the ALTER SYSTEM QUIESCE RESTRICTED statement to quiesce the database. Only users SYS and SYSTEM can issue the ALTER SYSTEM QUIESCE RESTRICTED statement. For all instances with the database open, issuing this statement has the following effect:

- Oracle Database instructs the Database Resource Manager in all instances to prevent all inactive sessions (other than SYS and SYSTEM) from becoming active. No user other than SYS and SYSTEM can start a new transaction, a new query, a new fetch, or a new PL/SQL operation.

- Oracle Database waits for all existing transactions in all instances that were initiated by a user other than SYS or SYSTEM to finish (either commit or terminate). Oracle Database also waits for all running queries, fetches, and PL/SQL procedures in all instances that were initiated by users other than SYS or SYSTEM and that are not inside transactions to finish. If a query is carried out by multiple successive OCI fetches, Oracle Database does not wait for all fetches to finish. It waits for the current fetch to finish and then blocks the next fetch. Oracle Database also waits for all sessions (other than those of SYS or SYSTEM) that hold any shared resources (such as enqueues) to release those resources. After all these operations finish, Oracle Database places the database into quiesced state and finishes executing the QUIESCE RESTRICTED statement.

- If an instance is running in shared server mode, Oracle Database instructs the Database Resource Manager to block logins (other than SYS or SYSTEM) on that instance. If an instance is running in non-shared-server mode, Oracle Database does not impose any restrictions on user logins in that instance.

During the quiesced state, you cannot change the Resource Manager plan in any instance.

The `ALTER SYSTEM UNQUIESCE` statement puts all running instances back into normal mode, so that all blocked actions can proceed. An administrator can determine which sessions are blocking a quiesce from completing by querying the `v$blocking_quiesce` view.

See Also:

- *Oracle Database SQL Language Reference*
- *Oracle Database Administrator's Guide*

How Oracle Database Locks Data

Locks are mechanisms that prevent destructive interaction between transactions accessing the same **resource**—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.

In all cases, Oracle Database automatically obtains necessary locks when executing SQL statements, so users need not be concerned with such details. Oracle Database automatically uses the lowest applicable level of restrictiveness to provide the highest degree of data concurrency yet also provide fail-safe data integrity. Oracle Database also allows the user to lock data manually.

See Also: "Types of Locks" on page 13-16

This section includes the following topics:

- Transactions and Data Concurrency
- Deadlocks
- Types of Locks
- DML Locks
- DDL Locks
- Latches and Internal Locks
- Explicit (Manual) Data Locking
- Oracle Database Lock Management Services

Transactions and Data Concurrency

Oracle Database provides data concurrency and integrity between transactions using its locking mechanisms. Because the locking mechanisms of Oracle Database are tied closely to transaction control, application designers need only define transactions properly, and Oracle Database automatically manages locking.

Keep in mind that Oracle Database locking is fully automatic and requires no user action. Implicit locking occurs for all SQL statements so that database users never need to lock any resource explicitly. The Oracle Database default locking mechanisms lock data at the lowest level of restrictiveness to guarantee data integrity while allowing the highest degree of data concurrency.

See Also: "Explicit (Manual) Data Locking" on page 13-25

This section includes the following topics:

- Modes of Locking

- Lock Duration

- Data Lock Conversion Versus Lock Escalation

Modes of Locking

Oracle Database uses two modes of locking in a multiuser database:

- Exclusive lock mode prevents the associates resource from being shared. This lock mode is obtained to modify data. The first transaction to lock a resource exclusively is the only transaction that can alter the resource until the exclusive lock is released.

- Share lock mode allows the associated resource to be shared, depending on the operations involved. Multiple users reading data can share the data, holding share locks to prevent concurrent access by a writer (who needs an exclusive lock). Several transactions can acquire share locks on the same resource.

Lock Duration

All locks acquired by statements within a transaction are held for the duration of the transaction, preventing destructive interference including dirty reads, lost updates, and destructive DDL operations from concurrent transactions. The changes made by the SQL statements of one transaction become visible only to other transactions that start *after* the first transaction is committed.

Oracle Database releases all locks acquired by the statements within a transaction when you either commit or undo the transaction. Oracle Database also releases locks acquired after a savepoint when rolling back to the savepoint. However, only transactions not waiting for the previously locked resources can acquire locks on the now available resources. Waiting transactions will continue to wait until after the original transaction commits or rolls back completely.

Data Lock Conversion Versus Lock Escalation

A transaction holds exclusive row locks for all rows inserted, updated, or deleted within the transaction. Because row locks are acquired at the highest degree of restrictiveness, no lock conversion is required or performed.

Oracle Database automatically converts a table lock of lower restrictiveness to one of higher restrictiveness as appropriate. For example, assume that a transaction uses a SELECT statement with the FOR UPDATE clause to lock rows of a table. As a result, it acquires the exclusive row locks and a row share table lock for the table. If the transaction later updates one or more of the locked rows, the row share table lock is automatically converted to a row exclusive table lock.

Lock escalation occurs when numerous locks are held at one level of granularity (for example, rows) and a database raises the locks to a higher level of granularity (for example, table). For example, if a single user locks many rows in a table, some databases automatically escalate the user's row locks to a single table. The number of locks is reduced, but the restrictiveness of what is being locked is increased.

Oracle Database never escalates locks. Lock escalation greatly increases the likelihood of deadlocks. Imagine the situation where the system is trying to escalate locks on behalf of transaction T1 but cannot because of the locks held by transaction T2. A deadlock is created if transaction T2 also requires lock escalation of the same data before it can proceed.

> **See Also:** "Table Locks (TM)" on page 13-17

Deadlocks

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. Figure 13–3 is a hypothetical illustration of two transactions in a deadlock.

In Figure 13–3, no problem exists at time point A, as each transaction has a row lock on the row it attempts to update. Each transaction proceeds without being terminated. However, each tries next to update the row currently held by the other transaction. Therefore, a deadlock results at time point B, because neither transaction can obtain the resource it needs to proceed or terminate. It is a deadlock because no matter how long each transaction waits, the conflicting locks are held.

Figure 13–3   Two Transactions in a Deadlock
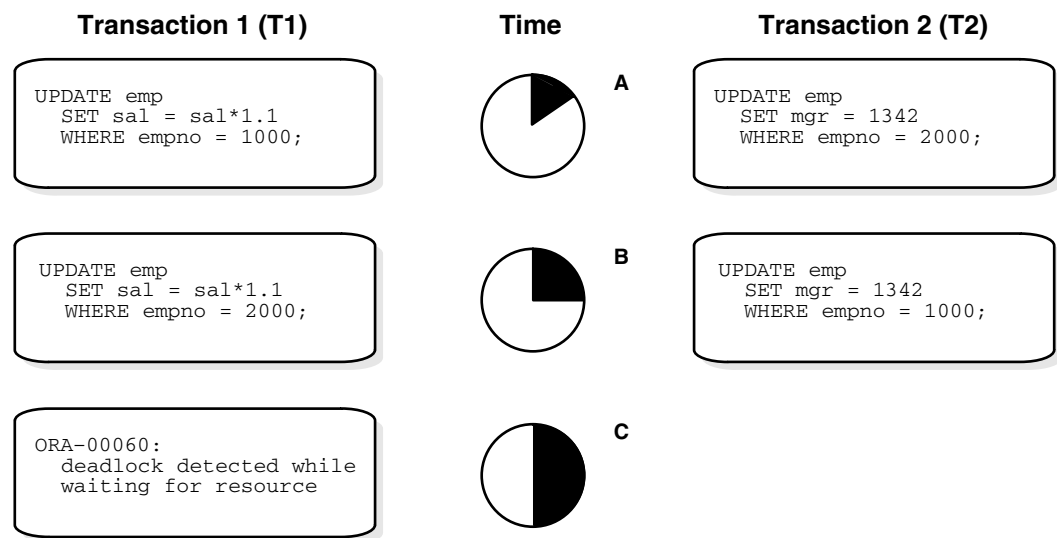
This section includes the following topics:

- Deadlock Detection
- Avoid Deadlocks

Deadlock Detection

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. A corresponding message also is returned to the transaction that undergoes statement-level rollback. The statement rolled back is the one belonging to the transaction that detects the deadlock. Usually, the signalled transaction should be rolled back explicitly, but it can retry the rolled-back statement after waiting.

> **Note:** In distributed transactions, local deadlocks are detected by analyzing wait data, and global deadlocks are detected by a time out. Once detected, nondistributed and distributed deadlocks are handled by the database and application in the same way.

Deadlocks most often occur when transactions explicitly override the default locking of Oracle Database. Because Oracle Database itself does no lock escalation and does

not use read locks for queries, but does use row-level locking (rather than page-level locking), deadlocks occur infrequently in Oracle Database.

> **See Also:** "Explicit (Manual) Data Locking" on page 13-25 for more information about manually acquiring locks

Avoid Deadlocks

Multitable deadlocks can usually be avoided if transactions accessing the same tables lock those tables in the same order, either through implicit or explicit locks. For example, all application developers might follow the rule that when both a master and detail table are updated, the master table is locked first and then the detail table. If such rules are properly designed and then followed in all applications, deadlocks are very unlikely to occur.

When you know you will require a sequence of locks for one transaction, consider acquiring the most exclusive (least compatible) lock first.

Types of Locks

Oracle Database automatically uses different types of locks to control concurrent access to data and to prevent destructive interaction between users. Oracle Database automatically locks a resource on behalf of a transaction to prevent other transactions from doing something also requiring exclusive access to the same resource. The lock is released automatically when some event occurs so that the transaction no longer requires the resource.

Throughout its operation, Oracle Database automatically acquires different types of locks at different levels of restrictiveness depending on the resource being locked and the operation being performed.

Oracle Database locks fall into one of three general categories.

Lock	Description
DML locks (data locks)	DML locks protect data. For example, table locks lock entire tables, row locks lock selected rows.
DDL locks (dictionary locks)	DDL locks protect the structure of schema objects—for example, the definitions of tables and views.
Internal locks and latches	Internal locks and latches protect internal database structures such as datafiles. Internal locks and latches are entirely automatic.

The following sections discuss DML locks, DDL locks, and internal locks.

DML Locks

The purpose of a DML lock (data lock) is to guarantee the integrity of data being accessed concurrently by multiple users. DML locks prevent destructive interference of simultaneous conflicting DML or DDL operations. DML statements automatically acquire both table-level locks and row-level locks.

> **Note:** The acronym in parentheses after each type of lock or lock mode is the abbreviation used in the Locks Monitor of Enterprise Manager. Enterprise Manager might display TM for any table lock, rather than indicate the mode of table lock (such as RS or SRX).

This section includes the following topics:

- Row Locks (TX)
- Table Locks (TM)
- DML Locks Automatically Acquired for DML Statements

Row Locks (TX)

Row-level locks are primarily used to prevent two transactions from modifying the same row. When a transaction needs to modify a row, a row lock is acquired.

There is no limit to the number of row locks held by a statement or transaction, and Oracle Database does not escalate locks from the row level to a coarser granularity. Row locking provides the finest grain locking possible and so provides the best possible concurrency and throughput.

The combination of multiversion concurrency control and row-level locking means that users contend for data only when accessing the same rows, specifically:

- Readers of data do not wait for writers of the same data rows.
- Writers of data do not wait for readers of the same data rows unless SELECT ... FOR UPDATE is used, which specifically requests a lock for the reader.
- Writers only wait for other writers if they attempt to update the same rows at the same time.

> **Note:** Readers of data may have to wait for writers of the same data blocks in some very special cases of pending distributed transactions.

A transaction acquires an exclusive row lock for each individual row modified by one of the following statements: INSERT, UPDATE, DELETE, and SELECT with the FOR UPDATE clause.

A modified row is **always** locked exclusively so that other transactions cannot modify the row until the transaction holding the lock is committed or rolled back. However, if the transaction dies due to instance failure, block-level recovery makes a row available before the entire transaction is recovered. Row locks are always acquired automatically by Oracle Database as a result of the statements listed previously.

If a transaction obtains a row lock for a row, the transaction also acquires a table lock for the corresponding table. The table lock prevents conflicting DDL operations that would override data changes in a current transaction.

> **See Also:** "DDL Locks" on page 13-22

Table Locks (TM)

Table-level locks are primarily used to do concurrency control with concurrent DDL operations, such as preventing a table from being dropped in the middle of a DML operation. When a DDL or DML statement is on a table, a table lock is acquired. Table locks do not affect concurrency of DML operations. For partitioned tables, table locks can be acquired at both the table and the subpartition level.

A transaction acquires a table lock when a table is modified in the following DML statements: INSERT, UPDATE, DELETE, SELECT with the FOR UPDATE clause, and LOCK TABLE. These DML operations require table locks for two purposes: to reserve DML access to the table on behalf of a transaction and to prevent DDL operations that would conflict with the transaction. Any table lock prevents the acquisition of an

exclusive DDL lock on the same table and thereby prevents DDL operations that require such locks. For example, a table cannot be altered or dropped if an uncommitted transaction holds a table lock for it.

A table lock can be held in any of several modes: row share (RS), row exclusive (RX), share (S), share row exclusive (SRX), and exclusive (X). The restrictiveness of a table lock's mode determines the modes in which other table locks on the same table can be obtained and held.

Table 13–3 shows the table lock modes that statements acquire and operations that those locks permit and prohibit.

Table 13–3   Summary of Table Locks

SQL Statement	Mode of Table Lock	Lock Modes Permitted?				
		RS	**RX**	**S**	**SRX**	**X**
SELECT...FROM *table*...	none	Y	Y	Y	Y	Y
INSERT INTO *table* ...	RX	Y	Y	N	N	N
UPDATE *table* ...	RX	Y*	Y*	N	N	N
DELETE FROM *table* ...	RX	Y*	Y*	N	N	N
SELECT ... FROM *table* FOR UPDATE OF ...	RS	Y*	Y*	Y*	Y*	N
LOCK TABLE *table* IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE *table* IN ROW EXCLUSIVE MODE	RX	Y	Y	N	N	N
LOCK TABLE *table* IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE *table* IN EXCLUSIVE MODE	X	N	N	N	N	N

RS: row share

RX: row exclusive

S: share

SRX: share row exclusive

X: exclusive

*Yes, if no conflicting row locks are held by another transaction. Otherwise, waits occur.

The following sections explain each mode of table lock, from least restrictive to most restrictive. They also describe the actions that cause the transaction to acquire a table lock in that mode and which actions are permitted and prohibited in other transactions by a lock in that mode.

> **See Also:**   "Explicit (Manual) Data Locking" on page 13-25

This section includes the following topics:

- Row Share Table Locks (RS)

- Row Exclusive Table Locks (RX)
- Share Table Locks (S)
- Share Row Exclusive Table Locks (SRX)
- Exclusive Table Locks (X)

Row Share Table Locks (RS)   A row share table lock (also sometimes called a **subshare table lock, SS**) indicates that the transaction holding the lock on the table has locked rows in the table and intends to update them. A row share table lock is automatically acquired for a *table* when one of the following SQL statements is run:

```
SELECT ... FROM table ... FOR UPDATE OF ... ;

LOCK TABLE table IN ROW SHARE MODE;
```

A row share table lock is the least restrictive mode of table lock, offering the highest degree of concurrency for a table.

Permitted Operations: A row share table lock held by a transaction allows other transactions to query, insert, update, delete, or lock rows concurrently in the same table. Therefore, other transactions can obtain simultaneous row share, row exclusive, share, and share row exclusive table locks for the same table.

Prohibited Operations: A row share table lock held by a transaction prevents other transactions from exclusive write access to the same table using only the following statement:

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

Row Exclusive Table Locks (RX)   A row exclusive table lock (also called a **subexclusive table lock, SX**) generally indicates that the transaction holding the lock has made one or more updates to rows in the table. A row exclusive table lock is acquired automatically for a *table* modified by the following types of statements:

```
INSERT INTO table ... ;

UPDATE table ... ;

DELETE FROM table ... ;

LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

A row exclusive table lock is slightly more restrictive than a row share table lock.

Permitted Operations: A row exclusive table lock held by a transaction allows other transactions to query, insert, update, delete, or lock rows concurrently in the same table. Therefore, row exclusive table locks allow multiple transactions to obtain simultaneous row exclusive and row share table locks for the same table.

Prohibited Operations: A row exclusive table lock held by a transaction prevents other transactions from manually locking the table for exclusive reading or writing. Therefore, other transactions cannot concurrently lock the table using the following statements:

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

Share Table Locks (S)   A share table lock is acquired automatically for the *table* specified in the following statement:

```
LOCK TABLE table IN SHARE MODE;
```

Permitted Operations: A share table lock held by a transaction allows other transactions only to query the table, to lock specific rows with SELECT ... FOR UPDATE, or to run LOCK TABLE ... IN SHARE MODE statements successfully. No updates are allowed by other transactions. Multiple transactions can hold share table locks for the same table concurrently. In this case, no transaction can update the table (even if a transaction holds row locks as the result of a SELECT statement with the FOR UPDATE clause). Therefore, a transaction that has a share table lock can update the table only if no other transactions also have a share table lock on the same table.

Prohibited Operations: A share table lock held by a transaction prevents other transactions from modifying the same table and from executing the following statements:

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

Share Row Exclusive Table Locks (SRX)   A share row exclusive table lock (also sometimes called a **share-subexclusive table lock, SSX**) is more restrictive than a share table lock. A share row exclusive table lock is acquired for a *table* as follows:

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

Permitted Operations: Only one transaction at a time can acquire a share row exclusive table lock on a given table. A share row exclusive table lock held by a transaction allows other transactions to query or lock specific rows using SELECT with the FOR UPDATE clause, but not to update the table.

Prohibited Operations: A share row exclusive table lock held by a transaction prevents other transactions from obtaining row exclusive table locks and modifying the same table. A share row exclusive table lock also prohibits other transactions from obtaining share, share row exclusive, and exclusive table locks, which prevents other transactions from executing the following statements:

```
LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

LOCK TABLE table IN ROW EXCLUSIVE MODE;

LOCK TABLE table IN EXCLUSIVE MODE;
```

Exclusive Table Locks (X)   An exclusive table lock is the most restrictive mode of table lock, allowing the transaction that holds the lock exclusive write access to the table. An exclusive table lock is acquired for a *table* as follows:

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

Permitted Operations: Only one transaction can obtain an exclusive table lock for a table. An exclusive table lock permits other transactions only to query the table.

Prohibited Operations: An exclusive table lock held by a transaction prohibits other transactions from performing any type of DML statement or placing any type of lock on the table.

DML Locks Automatically Acquired for DML Statements

The previous sections explained the different types of data locks, the modes in which they can be held, when they can be obtained, when they are obtained, and what they prohibit. The following sections summarize how Oracle Database automatically locks data on behalf of different DML operations.

Table 13–4 summarizes the information in the following sections.

Table 13–4   Locks Obtained By DML Statements

DML Statement	Row Locks?	Mode of Table Lock
SELECT ... FROM *table*		
INSERT INTO *table* ...	X	RX
UPDATE *table* ...	X	RX
DELETE FROM *table* ...	X	RX
SELECT ... FROM *table* ... FOR UPDATE OF ...	X	RS
LOCK TABLE *table* IN ...		
ROW SHARE MODE		RS
ROW EXCLUSIVE MODE		RX
SHARE MODE		S
SHARE EXCLUSIVE MODE		SRX
EXCLUSIVE MODE		X

X: exclusive

RX: row exclusive

RS: row share

S: share

SRX: share row exclusive

This section includes the following topics:

■   Default Locking for Queries

■   Default Locking for INSERT, UPDATE, DELETE, and SELECT ... FOR UPDATE

Default Locking for Queries   Queries are the SQL statements least likely to interfere with other SQL statements because they only read data. INSERT, UPDATE, and DELETE statements can have implicit queries as part of the statement. Queries include the following kinds of statements:

```
SELECT

INSERT ... SELECT ... ;

UPDATE ... ;

DELETE ... ;
```

They do **not** include the following statement:

```
SELECT ... FOR UPDATE OF ... ;
```

The following characteristics are true of all queries that do not use the FOR UPDATE clause:

- A query acquires no data locks. Therefore, other transactions can query and update a table being queried, including the specific rows being queried. Because queries lacking FOR UPDATE clauses do not acquire any data locks to block other operations, such queries are often referred to in Oracle Database as **nonblocking queries**.

- A query does not have to wait for any data locks to be released; it can always proceed. (Queries may have to wait for data locks in some very specific cases of pending distributed transactions.)

Default Locking for INSERT, UPDATE, DELETE, and SELECT ... FOR UPDATE The locking characteristics of INSERT, UPDATE, DELETE, and SELECT ... FOR UPDATE statements are as follows:

- The transaction that contains a DML statement acquires exclusive row locks on the rows modified by the statement. Other transactions cannot update or delete the locked rows until the locking transaction either commits or rolls back.

- The transaction that contains a DML statement does not need to acquire row locks on any rows selected by a subquery or an implicit query, such as a query in a WHERE clause. A subquery or implicit query in a DML statement is guaranteed to be consistent as of the start of the query and does not see the effects of the DML statement it is part of.

- A query in a transaction can see the changes made by previous DML statements in the same transaction, but cannot see the changes of other transactions begun after its own transaction.

- In addition to the necessary exclusive row locks, a transaction that contains a DML statement acquires at least a row exclusive table lock on the table that contains the affected rows. If the containing transaction already holds a share, share row exclusive, or exclusive table lock for that table, the row exclusive table lock is not acquired. If the containing transaction already holds a row share table lock, Oracle Database automatically converts this lock to a row exclusive table lock.

DDL Locks

A data dictionary lock (DDL) protects the definition of a schema object while that object is acted upon or referred to by an ongoing DDL operation. Recall that a DDL statement implicitly commits its transaction. For example, assume that a user creates a procedure. On behalf of the user's single-statement transaction, Oracle Database automatically acquires DDL locks for all schema objects referenced in the procedure definition. The DDL locks prevent objects referenced in the procedure from being altered or dropped before the procedure compilation is complete.

Oracle Database acquires a dictionary lock automatically on behalf of any DDL transaction requiring it. Users cannot explicitly request DDL locks. Only individual schema objects that are modified or referenced are locked during DDL operations. The whole data dictionary is never locked.

DDL locks fall into three categories: exclusive DDL locks, share DDL locks, and breakable parse locks.

This section includes the following topics:

- Exclusive DDL Locks
- Share DDL Locks

- Breakable Parse Locks

- Duration of DDL Locks

- DDL Locks and Clusters

Exclusive DDL Locks

Most DDL operations, except for those listed in the section, "Share DDL Locks" on page 13-23 require exclusive DDL locks for a resource to prevent destructive interference with other DDL operations that might modify or reference the same schema object. For example, a DROP TABLE operation is not allowed to drop a table while an ALTER TABLE operation is adding a column to it, and vice versa.

During the acquisition of an exclusive DDL lock, if another DDL lock is already held on the schema object by another operation, the acquisition waits until the older DDL lock is released and then proceeds.

DDL operations also acquire DML locks (data locks) on the schema object to be modified.

Share DDL Locks

Some DDL operations require share DDL locks for a resource to prevent destructive interference with conflicting DDL operations, but allow data concurrency for similar DDL operations. For example, when a CREATE PROCEDURE statement is run, the containing transaction acquires share DDL locks for all referenced tables. Other transactions can concurrently create procedures that reference the same tables and therefore acquire concurrent share DDL locks on the same tables, but no transaction can acquire an exclusive DDL lock on any referenced table. No transaction can alter or drop a referenced table. As a result, a transaction that holds a share DDL lock is guaranteed that the definition of the referenced schema object will remain constant for the duration of the transaction.

A share DDL lock is acquired on a schema object for DDL statements that include the following statements: AUDIT, NOAUDIT, COMMENT, CREATE [OR REPLACE] VIEW/ PROCEDURE/PACKAGE/PACKAGE BODY/FUNCTION/ TRIGGER, CREATE SYNONYM, and CREATE TABLE (when the CLUSTER parameter is not included).

Breakable Parse Locks

A SQL statement (or PL/SQL program unit) in the shared pool holds a parse lock for each schema object it references. Parse locks are acquired so that the associated shared SQL area can be invalidated if a referenced object is altered or dropped. A parse lock does not disallow any DDL operation and can be broken to allow conflicting DDL operations, hence the name **breakable parse lock**.

A parse lock is acquired during the parse phase of SQL statement execution and held as long as the shared SQL area for that statement remains in the shared pool.

> **See Also:** Chapter 6, "Schema Object Dependencies"

Duration of DDL Locks

The duration of a DDL lock depends on its type. Exclusive and share DDL locks last for the duration of DDL statement execution and automatic commit. A parse lock persists as long as the associated SQL statement remains in the shared pool.

DDL Locks and Clusters

A DDL operation on a cluster acquires exclusive DDL locks on the cluster and on all tables and materialized views in the cluster. A DDL operation on a table or materialized view in a cluster acquires a share lock on the cluster, in addition to a share or exclusive DDL lock on the table or materialized view. The share DDL lock on the cluster prevents another operation from dropping the cluster while the first operation proceeds.

Latches and Internal Locks

Latches and internal locks protect internal database and memory structures. Both are inaccessible to users, because users have no need to control over their occurrence or duration. The following section helps to interpret the Enterprise Manager `LOCKS` and `LATCHES` monitors.

This section includes the following topics:

- Latches
- Internal Locks

Latches

Latches are simple, low-level serialization mechanisms to protect shared data structures in the system global area (SGA). For example, latches protect the list of users currently accessing the database and protect the data structures describing the blocks in the buffer cache. A server or background process acquires a latch for a very short time while manipulating or looking at one of these structures. The implementation of latches is operating system dependent, particularly in regard to whether and how long a process will wait for a latch.

Internal Locks

Internal locks are higher-level, more complex mechanisms than latches and serve a variety of purposes.

This section includes the following topics:

- Dictionary Cache Locks
- File and Log Management Locks
- Tablespace and Rollback Segment Locks

Dictionary Cache Locks   These locks are of very short duration and are held on entries in dictionary caches while the entries are being modified or used. They guarantee that statements being parsed do not see inconsistent object definitions.

Dictionary cache locks can be shared or exclusive. Shared locks are released when the parse is complete. Exclusive locks are released when the DDL operation is complete.

File and Log Management Locks   These locks protect various files. For example, one lock protects the control file so that only one process at a time can change it. Another lock coordinates the use and archiving of the redo log files. Datafiles are locked to ensure that multiple instances mount a database in shared mode or that one instance mounts it in exclusive mode. Because file and log locks indicate the status of files, these locks are necessarily held for a long time.

Tablespace and Rollback Segment Locks   These locks protect tablespaces and rollback segments. For example, all instances accessing a database must agree on whether a

tablespace is online or offline. Rollback segments are locked so that only one instance can write to a segment.

Explicit (Manual) Data Locking

Oracle Database always performs locking automatically to ensure data concurrency, data integrity, and statement-level read consistency. However, you can override the Oracle Database default locking mechanisms. Overriding the default locking is useful in situations such as these:

- Applications require transaction-level read consistency or **repeatable reads**. In other words, queries in them must produce consistent data for the duration of the transaction, not reflecting changes by other transactions. You can achieve transaction-level read consistency by using explicit locking, read-only transactions, serializable transactions, or by overriding default locking.

- Applications require that a transaction have exclusive access to a resource so that the transaction does not have to wait for other transactions to complete.

Oracle Database automatic locking can be overridden at the transaction level or the session level.

At the transaction level, transactions that include the following SQL statements override Oracle Database default locking:

- The SET TRANSACTION ISOLATION LEVEL statement

- The LOCK TABLE statement (which locks either a table or, when used with views, the underlying base tables)

- The SELECT ... FOR UPDATE statement

Locks acquired by these statements are released after the transaction commits or rolls back.

At the session level, a session can set the required transaction isolation level with the ALTER SESSION statement.

> **Note:**   If Oracle Database default locking is overridden at any level, the database administrator or application developer should ensure that the overriding locking procedures operate correctly. The locking procedures must satisfy the following criteria: data integrity is guaranteed, data concurrency is acceptable, and deadlocks are not possible or are appropriately handled.

> **See Also:**   *Oracle Database SQL Language Reference* for detailed descriptions of the SQL statements LOCK TABLE and SELECT ... FOR UPDATE

Oracle Database Lock Management Services

With Oracle Database Lock Management services, an application developer can include statements in PL/SQL blocks that:

- Request a lock of a specific type

- Give the lock a unique name recognizable in another procedure in the same or in another instance

- Change the lock type

■ Release the lock

Because a reserved user lock is the same as an Oracle Database lock, it has all the Oracle Database lock functionality including deadlock detection. User locks never conflict with Oracle Database locks, because they are identified with the prefix UL.

The Oracle Database Lock Management services are available through procedures in the DBMS_LOCK package.

> **See Also:**
>
> ■ *Oracle Database Advanced Application Developer's Guide* for more information about Oracle Database Lock Management services
>
> ■ *Oracle Database PL/SQL Packages and Types Reference* for information about DBMS_LOCK

Overview of Oracle Flashback Query

Oracle Flashback Query lets you view and repair historical data. You can perform queries on the database as of a certain wall clock time or user-specified system change number (SCN).

Flashback Query uses the Oracle Database multiversion read-consistency capabilities to restore data by applying undo as needed. Oracle Database 11*g* automatically tunes a parameter called the undo retention period. The undo retention period indicates the amount of time that must pass before old undo information—that is, undo information for committed transactions—can be overwritten. The database collects usage statistics and tunes the undo retention period based on these statistics and on undo tablespace size.

Using Flashback Query, you can query the database as it existed this morning, yesterday, or last week. The speed of this operation depends only on the amount of data being queried and the number of changes to the data that need to be backed out.

You can query the history of a given row or a transaction. Using undo data stored in the database, you can view all versions of a row and revert to a previous version of that row. Flashback Transaction Query history lets you examine changes to the database at the transaction level.

You can audit the rows of a table and get information about the transactions that changed the rows and the times when it was changed. With the transaction ID, you can do transaction mining through LogMiner to get complete information about the transaction.

> **See Also:**
>
> ■ *Oracle Database Administrator's Guide* for more information on the automatic tuning of undo retention and on LogMiner
>
> ■

You set the date and time you want to view. Then, any SQL query you run operates on data as it existed at that time. If you are an authorized user, then you can correct errors and back out the restored data without needing the intervention of an administrator.

With the AS OF SQL clause, you can choose different snapshots for each table in the query. Associating a snapshot with a table is known as *table decoration.* If you do not decorate a table with a snapshot, then a default snapshot is used for it. All tables without a specified snapshot get the same default snapshot.

For example, suppose you want to write a query to find all the new customer accounts created in the past hour. You could do set operations on two instances of the same table decorated with different AS OF clauses.

DML and DDL operations can use table decoration to choose snapshots within subqueries. Operations such as INSERT TABLE AS SELECT and CREATE TABLE AS SELECT can be used with table decoration in the subqueries to repair tables from which rows have been mistakenly deleted. Table decoration can be any arbitrary expression: a bind variable, a constant, a string, date operations, and so on. You can open a cursor and dynamically bind a snapshot value (a timestamp or an SCN) to decorate a table with.

> **See Also:**
>
> - "Overview of High Availability Features" on page 1-22 for an overview of all Oracle Flashback features
>
> - *Oracle Database SQL Language Reference* for information on the AS OF clause

This section includes the following topics:

- Flashback Query Benefits
- Some Uses of Flashback Query

Flashback Query Benefits

This section lists some of the benefits of using Flashback Query.

- Application Transparency

  Packaged applications, like report generation tools that only do queries, can run in Flashback Query mode by using logon triggers. Applications can run transparently without requiring changes to code. All the constraints that the application needs to be satisfied are guaranteed to hold good, because there is a consistent version of the database as of the Flashback Query time.

- Application Performance

  If an application requires recovery actions, it can do so by saving SCNs and flashing back to those SCNs. This is a lot easier and faster than saving data sets and restoring them later, which would be required if the application were to do explicit versioning. Using Flashback Query, there are no costs for logging that would be incurred by explicit versioning.

- Online Operation

  Flashback Query is an online operation. Concurrent DMLs and queries from other sessions are allowed while an object is queried inside Flashback Query.The speed of these operations is unaffected. Moreover, different sessions can flash back to different Flashback times or SCNs on the same object concurrently. The speed of the Flashback Query itself depends on the amount of undo that needs to be applied, which is proportional to how far back in time the query goes.

- Easy Manageability

  There is no additional management on the part of the user, except setting the appropriate retention interval, having the right privileges, and so on. No additional logging has to be turned on, because past versions are constructed automatically, as needed.

> **Note:**
>
> - Flashback Query does *not* undo anything. It is only a query mechanism. You can take the output from a Flashback Query and perform an undo yourself in many circumstances.
>
> - Flashback Query does *not* tell you what changed. LogMiner does that.
>
> - Flashback Query can undo changes and can be very efficient if you know the rows that need to be moved back in time. You can use it to move a full table back in time, but this is very expensive if the table is large since it involves a full table copy.
>
> - Flashback Query does not work through DDL operations that modify columns, or drop or truncate tables.
>
> - LogMiner is very good for getting change history, but it gives you changes in terms of deltas (insert, update, delete), not in terms of the before and after image of a row. These can be difficult to deal with in some applications.

Some Uses of Flashback Query

This section lists some ways to use Flashback Query.

- Self-Service Repair

  Perhaps you accidentally deleted some important rows from a table and wanted to recover the deleted rows. To do the repair, you can move backward in time and see the missing rows and re-insert the deleted row into the current table.

- E-mail or Voice Mail Applications

  You might have deleted mail in the past. Using Flashback Query, you can restore the deleted mail by moving back in time and re-inserting the deleted message into the current message box.

- Account Balances

  You can view account prior account balances as of a certain day in the month.

- Packaged Applications

  Packaged applications (like report generation tools) can make use of Flashback Query without any changes to application logic. Any constraints that the application expects are guaranteed to be satisfied, because users see a consistent version of the Database as of the given time or SCN.

In addition, Flashback Query could be used after examination of audit information to see the before-image of the data. In DSS environments, it could be used for extraction of data as of a consistent point in time from OLTP systems.

> **See Also:**
>
> - *Oracle Database Advanced Application Developer's Guide* for more information about using Oracle Flashback Query
>
> - *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_FLASHBACK` package
>
> - *Oracle Database Administrator's Guide* for information about undo tablespaces and setting retention period

14

Manageability

Oracle Database 11*g* represents a major milestone in Oracle's drive toward self-managing databases. It automates many routine administrative tasks, and considerably simplifies key DBA functions, such as performance diagnostics, SQL tuning, and space and memory management. It also provides several *advisors* that guide DBAs in managing key components of the database by giving specific recommendations along with potential benefit. Furthermore, Oracle Database 11*g* proactively sends alerts when a problem is anticipated, thus facilitating proactive rather than reactive database management.

This chapter contains the following topics:

- Installing Oracle Database 11g and Getting Started
- Intelligent Infrastructure
- Performance Diagnostics and Troubleshooting
- Application and SQL Tuning
- Memory Management
- Space Management
- Storage Management
- Backup and Recovery
- Configuration Management
- Workload Management
- Automatic Storage Management
- Oracle Scheduler

Installing Oracle Database 11*g* and Getting Started

The Oracle Universal Installer is a GUI tool for installing Oracle software. It automates all installation tasks, performs comprehensive prerequisite checks (such as operating system version, software patches, and capacity), installs selected software components, and performs all postinstall configuration.

The installation process is self-contained to automatically set up the required infrastructure for routine monitoring and administration. The Oracle Enterprise Manager Database Management Console is automatically configured to let you to get started with database administrative tasks without any manual configuration. The Oracle Enterprise Manager Database Console provides all essential functionality for managing a single database, including alert notification, job scheduling, and software

management. In addition, all Oracle Database server components such as the database, listener, management framework, and so on, are configured for automated startup and shutdown.

> **See Also:** "Configuration Management" on page 14-17 for more information about Oracle Enterprise Manager

This section includes the following topics:

- Simplified Database Creation
- Instant Client
- Automated Upgrades
- Basic Initialization Parameters
- Data Loading, Transfer, and Archiving

Simplified Database Creation

The Database Creation Assistant (DBCA) is a GUI tool for database creation. It lets you create all possible configurations of the database, be it a standalone database, an Oracle Real Application Clusters database, or a standby database. During the database creation process, the DBCA guides you in setting up an automated disk-based backup and registering the database with a LDAP server, if available. A database created using the DBCA is fully setup and ready to use in all respects.

Instant Client

The Instant Client is the simplest way to deploy a full Oracle Client application built with OCI, OCCI, JDBC-OCI, or ODBC drivers. It provides the necessary Oracle Client libraries in a small set of files. Installation is as easy as copying a few shared libraries to a directory on the client computer. If this directory is accessible through the operating system library path variable (for instance, `LD_LIBRARY_PATH` or `PATH`) then the application will operate in the Instant Client mode. Instant Client deployment does not require the `ORACLE_HOME` environment, nor does it require the large number of code and data files provided in a full Oracle Client install, thereby significantly reducing the client application disk space needs. There is no loss in functionality or performance for an application deployed using Instant Client when compared to the same application running in a full `ORACLE_HOME` environment.

> **See Also:**
>
> - Chapter 24, "SQL" and Chapter 25, "Supported Application Development Languages" for more information about JDBC, OCI, and OCCI
> - *Oracle Call Interface Programmer's Guide* for more information about Instant Client

Automated Upgrades

With the Database Upgrade Assistant (DBUA), you can upgrade any database configuration, including Oracle Real Application Clusters (Oracle RAC) and standby, just by answering a few simple questions. It automatically checks that adequate resources are available, ensures adherence to the best practices – such as backing up the database before beginning the upgrade process, replacing the obsolete and

deprecate initialization parameters, and so on – and, verifies the successful completion of the operation.

The upgrade process is restartable, allowing it to automatically resume from the point of interruption. You can also get a time estimation of how long the upgrade process is likely to take.

Basic Initialization Parameters

The Oracle Database provides a number of initialization parameters to optimize its operation in diverse environments. Only a few of these parameters need to be explicitly set, because the default values are adequate in the majority of cases.

There are approximately 30 basic parameters. The remainder of the parameters are preserved to allow expert DBAs to adapt the behavior of the Oracle Database to meet unique requirements without overwhelming those who have no such requirements.

See Also: *Oracle Database Administrator's Guide*

Data Loading, Transfer, and Archiving

Data Pump enables very high-speed data and metadata loading and unloading to and from the Oracle Database. It automatically manages and schedules multiple, parallel streams of load or unload for maximum throughput.

The transportable tablespace feature lets you quickly move a tablespace across Oracle databases. This can be much faster than performing either an export/import or unload/load of the same data, because transporting a tablespace only requires the copying of datafiles and integrating the tablespace structural information. You can also use transportable tablespaces to move index data, thereby avoiding the index rebuilds you would have to perform when importing or loading table data.

Data Pump functionality together with cross-platform transportable tablespace feature provides powerful, easy to use, and high performance tools for moving data in and out of the database.

See Also:

- "Overview of Data Pump Export and Import" on page 11-2
- "Transport of Tablespaces Between Databases" on page 3-14

Intelligent Infrastructure

Oracle Database has a sophisticated self-management infrastructure that allows the database to learn about itself and use this information to adapt to workload variations or to automatically remedy any potential problem. The self-management infrastructure includes the following:

- Automatic Workload Repository
- Automatic Maintenance Tasks
- Fault Diagnosability Infrastructure
- Server-Generated Alerts
- Advisor Framework
- Hang Manager

Automatic Workload Repository

The Automatic Workload Repository (AWR) is a built-in repository that contains performance statistics used by Oracle Database for problem detection and self-tuning purposes. At regular intervals, Oracle Database makes a snapshot of vital statistics and workload information and stores them in the AWR. The data contained in the snapshots is then analyzed by the Automatic Database Diagnostic Monitor (ADDM). The difference between snapshots is compared to determine which SQL statements to capture based on the effect on the system load. This reduces the number of SQL statements that need to be captured over time. By default, the snapshots are taken once every hour and retained in the AWR for 8 days, after which they are automatically purged. You can change both the frequency and the retention period of snapshots.

Snapshots from specific time periods can be preserved in a baseline for comparison with other similar workload periods. The snapshots contained in a baseline are excluded from the automatic AWR purging process and are retained indefinitely. There are several types of available baselines in Oracle Database: fixed baselines, moving window baselines, and baseline templates. A fixed baseline corresponds to a fixed, contiguous time period in the past. Fixed baselines captured when the system is operating at an optimal level can be compared with other baselines or snapshots captured during periods of poor performance to analyze performance degradation over time. A moving window baseline corresponds to all AWR data that exists within the AWR retention period. This is useful when using adaptive thresholds because the AWR data in the entire AWR retention period can be used to compute metric threshold values. Baseline templates can be used to create baselines for contiguous time periods in the future. There are two types of baseline templates: single and repeating. A single baseline template can be used to create a baseline for a single contiguous time period in the future. This is useful if you know beforehand of a time period that you want to capture in the future. A repeating baseline template can be used to create and drop baselines based on a repeating time schedule. This is useful if you want Oracle Database to automatically capture a contiguous time period on an ongoing basis.

AWR forms the foundation for all self-management functionality of Oracle Database. It is the source of information that gives Oracle Database a historical perspective on how the database is being used, and enables ADDM to accurately diagnose and resolve potential performance problems.

> **See Also:** *Oracle Database Performance Tuning Guide* for information about the Automatic Workload Repository

Automatic Maintenance Tasks

By analyzing the information stored in AWR, the database can identify the need to perform routine maintenance tasks. The automated maintenance tasks infrastructure (known as AutoTask) enables Oracle Database to automatically schedule such operations. AutoTask schedules automatic maintenance tasks to run in a set of Oracle Scheduler windows known as *maintenance windows*. Maintenance windows are those windows that are members of the Oracle Scheduler window group `MAINTENANCE_WINDOW_GROUP`.

By default, `MAINTENANCE_WINDOW_GROUP` contains one window for each day of the week. Weekday windows (Monday through Friday) are configured to be *open* (active) for 4 hours starting at 10:00 p.m. Weekend windows (Saturday and Sunday) begin at 6:00 a.m. and remain open for 20 hours. You can customize all attributes of these maintenance windows, including start and end time, frequency, days of the week, and so on. You can also add and remove maintenance windows from the group.

The following are the tasks that AutoTask automatically schedules in these maintenance windows:

- Optimizer statistics gathering

- Automatic Segment Advisor

- SQL Tuning Advisor

Using Oracle Enterprise Manager or PL/SQL package procedures, you can adjust which of these tasks run in which maintenance windows.

Limiting Automatic Maintenance Task Resource Allocation

The impact of automated maintenance tasks on normal database operations is limited by the default Database Resource Manager resource plan. You can modify the default plan, or create your own resource plans and activate them either at the system-wide level or at the individual maintenance window level. AutoTask runs all automatic maintenance tasks as Oracle Scheduler jobs that belong to particular resource consumer groups. Resource plans then limit CPU resources that are allocated to these resource consumer groups. Because your user applications can be assigned to resource consumer groups, you can adjust the resource allocation for maintenance tasks not only relative to other maintenance tasks, but also relative to your applications.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* and *Oracle Database 2 Day DBA* for instructions for managing automatic maintenance tasks
>
> - "Oracle Scheduler" on page 14-26
>
> - "Overview of the Database Resource Manager" on page 14-17
>
> - *Oracle Database Administrator's Guide* for information about Automatic Segment Advisor
>
> - *Oracle Database Performance Tuning Guide* for information about SQL Tuning Advisor

Fault Diagnosability Infrastructure

Oracle Database includes an advanced fault diagnosability infrastructure for preventing, detecting, diagnosing, and resolving problems. The problems that are targeted are critical errors such as those caused by database code bugs, metadata corruption, and customer data corruption. The goals of the advanced fault diagnosability infrastructure are the following:

- Detecting problems proactively

- 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:

- The Health Monitor, which performs deeper analysis of a critical error upon detection, creates health check reports and adds these reports to the diagnostic data collected for the error. The DBA can also manually invoke health checks and obtain reports.

- First-failure data capture, which captures comprehensive diagnostic data upon the first occurrence of a critical error

- Standardized trace and dump formats for easier analysis

- Incident packaging service, which enables the DBA to automatically package all diagnostic information surrounding a critical error into an archive suitable for transmission to Oracle Support.

- Data Recovery Advisor, which displays data corruption problems, assesses the extent of the problems, and recommends repair options

- SQL Test Case Builder, which helps Oracle Support reproduce customer problems that are related to SQL failures

- Support Workbench, which is a guided workflow that assists you with capturing critical error diagnostic information, transmitting it to Oracle Support, and filing a service request

> **See Also:** *Oracle Database Administrator's Guide* for more information on the fault diagnosability infrastructure and on the Support Workbench

This section further discusses two components of this new infrastructure:

- Automatic Diagnostic Repository

- Incident Packaging Service

Automatic Diagnostic Repository

The Automatic Diagnostic Repository (ADR) is a file-based repository for database diagnostic data such as traces, the alert log, health monitor reports, and more. It has a unified directory structure across multiple instances and multiple products. Beginning with Oracle Database 11*g*, the database, Automatic Storage Management (ASM), and other Oracle products or components store all diagnostic data in the ADR. Each instance of each product stores diagnostic data underneath its own ADR home directory. For example, in an Oracle Real Application Clusters environment with shared storage and ASM, each database instance and each ASM instance has a home directory within the ADR. 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.

Incident Packaging Service

A DBA can automatically and easily gather all diagnostic data (traces, health check reports, SQL test cases, and more) pertaining to a critical error and package the data into a zip file suitable for transmission to Oracle Support. Because all diagnostic data relating to a critical error are tagged with that error's incident number, the DBA does not have to search through trace files and other files to determine the files that are required for analysis; the incident packaging service identifies all required files automatically and adds them to the package.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about these components
>
> - *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* for information on ADR usage

Server-Generated Alerts

For problems that cannot be resolved automatically and require administrators to be notified, such as running out of space, the Oracle Database provides server-generated alerts. Oracle Database can monitor itself and send out alerts to notify you of any problem in an efficient and timely manner.

Monitoring activities take place as the database performs its regular operation. This ensures that the database is aware of problems the moment they arise. The alerts produced by Oracle Database not only notify the problem, they also provide recommendations on how the reported problem can be resolved. This ensures quick problem resolution and helps prevent potential failures.

Advisor Framework

Oracle Database includes a number of advisors for different sub-systems in the database to automatically determine how the operation of the corresponding subcomponents could be further optimized. The SQL Tuning Advisor and the SQL Access Advisor, for example, provide recommendations for running SQL statements faster. Memory advisors help size the various memory components without resorting to trial-and-error techniques. The Segment Advisor handles space-related issues, such as recommending wasted-space reclamation and analyzing growth trends, while the Undo Advisor guides you in sizing the undo tablespace correctly. The various advisors are discussed more throughout this chapter.

To ensure the consistency and uniformity in the way advisors function and allow them to interact with each other seamlessly, Oracle Database includes an advisor framework. The advisor framework provides a consistent manner in which advisors are invoked and results are reported. Although these advisors are primarily used by the database to optimize its own performance, they can be invoked by administrators to get more insight into the functioning of a particular subcomponent.

> **See Also:** *Oracle Database 2 Day DBA* for more information about using advisors

Hang Manager

Active entities that attempt to obtain restrictive access to shared resources or request services from other Oracle Database processes, sessions, and transactions are in danger of hanging. A hang chain is a chain of processes with each one waiting on a resource held by the next, with a single process serving as the root of the hang.

Hangs in Oracle Database can cost a great deal in terms of system unavailability. Specifically, hangs lead to the following problems:

- Extended system outages. These outages may occur frequently before a fix is found, which adds to the total downtime.

- Analyzing the hang to determine where the problem lies can be lengthy, complex, and prone to error.

The Hang Manager is an Oracle Database infrastructure that can detect hangs, analyze them, and then obtain the required diagnostic data from Oracle. The Hang Manager is enabled by default in Oracle RAC databases and Automatic Storage Management (ASM) instances. Hang manager data is output to trace files.

Performance Diagnostics and Troubleshooting

Building upon the data captured in AWR, the Automatic Database Diagnostic Monitor (ADDM) lets Oracle Database diagnose its own performance and determine how identified problems could be resolved. ADDM runs automatically after each AWR statistics capture, making the performance diagnostic data readily available.

ADDM examines data captured in AWR and performs analysis to determine the major issues on the system on a proactive basis. In many cases, it recommends solutions and quantifies expected benefits. ADDM takes a holistic approach to the performance of the system, using time as a common currency between components. ADDM identifies those areas of the system that are consuming the most time. ADDM drills down to identify the root cause of problems, rather than just the symptoms, and reports the impact that the problem is having on the system overall. If a recommendation is made, it reports the benefits that can be expected in terms of time. The use of time throughout allows the impact of several problems or recommendations to be compared.

ADDM focuses on activities that the database is spending most time on and then drills down through a sophisticated problem classification tree. Some common problems detected by ADDM include the following:

- CPU bottlenecks

- Poor connection management

- Excessive parsing

- Lock contention

- I/O capacity

- Undersizing of Oracle Database memory structures; for example, PGA, buffer cache, log buffer

- High load SQL statements

- High PL/SQL and Java time

- High checkpoint load and cause; for example, small log files, aggressive MTTR setting

- Oracle RAC-specific issues

Besides reporting potential performance issues, ADDM also documents non-problem areas of the system. The subcomponents, such as I/O and memory, that are not significantly impacting system performance are pruned from the classification tree at an early stage and are listed so that you can quickly see that there is little to be gained by performing actions in those areas.

You no longer need to first collect huge volumes of diagnostic data and spend hours analyzing them in order to find out answers to performance issues. You can simply follow the recommendation made by ADDM with just a few mouse clicks.

Application and SQL Tuning

Oracle Database completely automates the SQL tuning process. ADDM identifies SQL statements consuming unusually high system resources and therefore causing performance problems. In addition, the top SQL statements in terms of CPU and shared memory consumption are automatically captured in AWR. Thus, the identification of high load SQL statements happens automatically in Oracle Database and requires no intervention.

After identifying the top resource-consuming SQL statements, Oracle Database can automatically analyze them and recommend solutions using the Automatic SQL Tuning Advisor. Automatic SQL Tuning is exposed with an advisor, called the SQL Tuning Advisor. The SQL Tuning Advisor takes one or more SQL statements as input and produces well-tuned plans along with tuning advice. You do not need to do anything other than invoke the SQL Tuning Advisor.

The solution comes right from the optimizer and not from external tools using pre-defined heuristics. This provides several advantages: a) the tuning is done by the system component that is ultimately responsible for the execution plans and SQL performance, b) the tuning process is fully cost-based, and it naturally accounts for any changes and enhancements done to the query optimizer, c) the tuning process considers the past execution statistics of a SQL statement and customizes the optimizer settings for that statement, and d) it collects auxiliary information in conjunction with the regular statistics based on what is considered useful by the query optimizer.

The recommendation of the Automatic SQL Tuning Advisor can fall into one of the following categories

- Statistics Analysis: The Automatic SQL Tuning Advisor checks each query object for missing or stale statistics and makes recommendations to gather relevant statistics. It also collects auxiliary information to supply missing statistics or correct stale statistics in case recommendations are not implemented. Because Oracle Database automatically gathers optimizer statistics, this should not be the problem unless the automatic statistics gathering functionality has been disabled.

- SQL Profiling: The Automatic SQL Tuning Advisor verifies its own estimates and collects auxiliary information to remove estimation errors. It also collects auxiliary information in the form of customized optimizer settings (for example, first rows or all rows) based on past execution history of the SQL statement. It builds a SQL profile using the auxiliary information and makes a recommendation to create it. It then enables the query optimizer (under normal mode) to generate a well-tuned plan. The most powerful aspect of SQL profiles is that they enable tuning of queries without requiring any syntactical changes and thereby proving a unique database –resident solution to tune the SQL statements embedded in packaged applications.

- Access Path Analysis: The Automatic SQL Tuning Advisor considers whether a new index can be used to significantly improve access to each table in the query and when appropriate makes recommendations to create such indexes.

- SQL Structure Analysis: The Automatic SQL Tuning Advisor tries to identify SQL statements that lend themselves to bad plans and makes relevant suggestions to restructure them. The suggested restructuring can be syntactic as well as semantic changes to the SQL code.

Both access path and SQL structure analysis can be useful in tuning the performance of an application under development or a homegrown production application where the administrators and developers have access to application code.

The SQL Access Advisor can automatically analyze the schema design for a given workload and recommend indexes, function-based indexes, partitions, and materialized views to create, retain, or drop as appropriate for the workload. For single statement scenarios, the advisor only recommends adjustments that affect the current statement. For complete business workloads, the advisor makes recommendations after considering the impact on the entire workload.

While generating recommendations, the SQL Access Advisor considers the impact of adding new indexes, partitions, and materialized views on data manipulation activities, such as insert, update, and delete, in addition to the performance

improvement they are likely to provide for queries. After the SQL Access Advisor has filtered the workload, but while it is still identifying all possible solutions, you can asynchronously interrupt the process to get the best solution up to that point in time.

The SQL Access Advisor provides an easy to use interface and requires very little system knowledge. It can be run without affecting production systems, because the data can be gathered from the production system and taken to another computer where the SQL Access Advisor can be run.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about the SQL Tuning Advisor and the SQL Access Advisor

Memory Management

Oracle Database memory management allows for dynamic resizing of system global area (SGA) and program global area (PGA) memory components, either automatically or manually.

Automatic Memory Management

By default, new database installations are configured to automatically tune the various components of the SGA and PGA. You can make simple high-level adjustments to memory allocation by changing one database parameter: MEMORY_TARGET. As you allocate more system memory to the database with this parameter, the database automatically adjusts various component sizes for optimal database performance.

The performance of each component is monitored by the Oracle database instance. The instance uses internal views and statistics to determine how to optimally distribute memory among the automatically-sized components. Thus, as the workload changes, memory is redistributed to ensure optimal performance with the new workload. The database arrives at optimal distribution by taking into consideration long term and short terms trends.

You can exercise some control over the size of the auto-tuned components by specifying minimum values for each component. This can be useful in cases where you know that an application needs a minimum amount of memory in certain components to function properly.

The sizes of the automatically-tuned components are remembered across shutdowns if a server parameter file (SPFILE) is used. This means that the system picks up where it left off from the last shutdown.

Manual Memory Management and Memory Advisors

If you want to exercise more precise control over allocation for multiple memory components, you can enable manual memory management. You can then take advantage of a set of memory *advisors*, which graphically display current component sizes and the estimated affect of changing these sizes.

The Shared Pool Advisor determines the optimal shared pool size by tracking its use by the library cache. The amount of memory available for the library cache can drastically affect the parse rate of an Oracle database instance. The shared pool advisor statistics provide information about library cache memory, letting you predict how changes in the size of the shared pool can affect aging out of objects in the shared pool.

The Buffer Cache Advisor determines the optimal size of the buffer cache. When manually configuring memory for a new instance, it is difficult to know the correct size for the buffer cache. Typically, you make a first estimate for the cache size, run a representative workload on the instance, and then examine the relevant statistics to see

whether the cache is under- or over-configured. A number of statistics can be used to examine buffer cache activity. These include the V$DB_CACHE_ADVICE view and the buffer cache hit ratio.

The Java Pool Advisor provides information about library cache memory used for Java, and predicts how changes in the size of the Java pool can affect the parse rate.

The Streams Pool Advisor determines the optimal size of the Streams pool. The view V$STREAMS_POOL_ADVICE gives estimates of the amount of bytes spilled and unspilled for the different values of the STREAMS_POOL_SIZE parameter. You can use this to tune the STREAMS_POOL_SIZE parameter for Streams and for logical standby. AWR reports on the V$STREAMS_POOL_ADVICE view and CPU usage to help you tune Streams performance.

The Program Global Area (PGA) Advisor helps you determine an appropriate setting for PGA_AGGREGATE_TARGET, which is the total amount of memory to allocate for all PGAs for server and background processes.

> **See Also:**
>
> - Chapter 8, "Memory Architecture"
>
> - *Oracle Database Performance Tuning Guide* for more information about memory advisors
>
> - *Oracle Database Administrator's Guide* for information about the various initialization parameters for manual and automatic memory management, and for information about server parameter files

Space Management

Oracle Database automatically manages its space consumption, sends alerts on potential space problems, and recommends possible solutions. Oracle Database features that help you to easily manage space include the following:

- Automatic Undo Management
- Oracle-Managed Files
- Free Space Management
- Proactive Space Management
- Intelligent Capacity Planning
- Space Reclamation

Automatic Undo Management

Earlier releases of Oracle Database used rollback segments to store undo. Space management for these rollback segments was complex. Automatic undo management eliminates the complexities of managing rollback segments by automatically managing space in an undo tablespace. Automatic undo management also optimally tunes the length of time that undo is retained before being overwritten. This automatic tuning of undo retention improves the success rate of long running queries and of certain Oracle Flashback features, which may require the presence of old undo information.

Although you can configure the database to use rollback segments, automatic undo management is the default. An autoextending undo tablespace is automatically created upon database installation.

Automatic tuning of undo retention generally achieves better results with a fixed size undo tablespace. If you want to change the undo tablespace to fixed size for this or other reasons, the Undo Advisor can help you determine the proper fixed size to allocate. You provide the desired undo retention period for your long-running queries or Oracle Flashback operations, and the Undo Advisor suggests the required undo tablespace size. The Undo Advisor makes its recommendations based on system activity statistics, including the longest running query and undo generation rate. Advisor information includes the following:

- Current undo retention

- Current undo tablespace size

- Longest query duration

- Best undo retention possible

- Undo tablespace size necessary for current undo retention

> **See Also:**
>
> - "Introduction to Undo Segments and Automatic Undo Management" on page 2-16
>
> - *Oracle Database 2 Day DBA* for information about managing undo and running the Undo Advisor
>
> - *Oracle Database Administrator's Guide* for more information about the undo tablespace and on undo retention

Oracle-Managed Files

With Oracle-managed files, you do not need to directly manage the files comprising an Oracle database. Oracle Database uses standard file system interfaces to create and delete files as needed. This automates the routine task of creation and deletion of database files.

Free Space Management

Oracle Database allows for managing free space within a table with bitmaps, as well as traditional dictionary based space management. The bitmapped implementation eliminates much space-related tuning of tables, while providing improved performance during peak loads. Additionally, Oracle Database provides automatic extension of data files, so the files can grow automatically based on the amount of data in the files. Database administrators do not need to manually track and reorganize the space usage in all the database files.

Proactive Space Management

Oracle Database introduces a non-intrusive and timely check for space utilization monitoring. It automatically monitors space utilization during normal space allocation and de-allocation operations and alerts you if the free space availability falls below the pre-defined thresholds. Space monitoring functionality is set up out of box, causes no performance impact, and is uniformly available across all tablespace types. Also, the same functionality is available both through Oracle Enterprise Manager as well as SQL. Because the monitoring is performed at the same time as space is allocated and freed up in the database, this guarantees immediate availability of space usage information whenever you need it.

Notification is performed using server-generated alerts. The alerts are triggered when certain space-related events occur in the database. For example, when the space usage threshold of a tablespace is crossed or when a resumable session encounters an out of space situation, then an alert is raised. An alert is sent instantaneously to take corrective measures. You may choose to get paged with the alert information and add space to the tablespace to allow the suspended operation to continue from where it left off.

The database comes with a default set of alert thresholds. You can override the default for a given tablespace or set a new default for the entire database through Oracle Enterprise Manager.

Intelligent Capacity Planning

Space may get overallocated because of the difficulty to predict the space requirement of an object or the inability to predict the growth trend of an object. On tables that are heavily updated, the resulting segment may have a lot of internal fragmentation and maybe even row chaining. These issues can result in a wide variety of problems from poor performance to space wastage. Oracle Database offers several features to address these challenges.

Oracle Database can predict the size of a given table based on its structure and estimated number of rows. This is a powerful "what if" tool that allows estimation of the size of an object before it is created or rebuilt. If tablespaces have different extent management policies, then the tool will help decide the tablespace that will cause least internal fragmentation.

The growth trend report takes you to the next step of capacity planning: planning for growth. Most database systems grow over time. Planning for growth is an important aspect of provisioning resources. To aid this process, Oracle Database tracks historical space utilization in the AWR and uses this information to predict the future resource requirements.

Space Reclamation

Oracle Database provides in-place reorganization of data for optimal space utilization by shrinking it. Shrinking of a segment makes unused space available to other segments in the tablespace and may improve the performance of queries and DML operations.

The segment shrink functionality both compacts the space used in a segment and then deallocates it from the segment. The deallocated space is returned to the tablespace and is available to other objects in the tablespace. Sparsely populated tables may cause a performance problem for full table scans. By performing shrink, data in the table is compacted and the high water mark of the segment is pushed down. This makes full table scans read less blocks run faster.

Segment shrink is an online operation – the table being shrunk is open to queries and DML while the segment is being shrunk. Additionally, segment shrink is performed in place. This is an advantage over online table redefinition for compaction and reclaiming space. You can schedule segment shrink for one or all the objects in the database as nightly jobs without requiring any additional space to be provided to the database.

Segment shrink works on heaps, IOTs, IOT overflow segments, LOBs, LOB segments, materialized views, and indexes with row movement enabled in tablespaces with automatic segment space management. When segment shrink is performed on tables with indexes on them, the indexes are automatically maintained when rows are moved

around for compaction. User-defined triggers are not fired, however, because compaction is a purely physical operation and does not impact the application.

> **Note:** Segment shrink can be performed only on tables with row movement enabled. Applications that explicitly track rowids of objects cannot be shrunk, because the application tracks the physical location of rows in the objects.

To easily identify candidate segments for shrinking, Oracle Database automatically runs the Segment Advisor to evaluate the entire database. The Segment Advisor performs growth trend analysis on individual objects to determine if there will be any additional space left in the object in seven days. It then uses the reclaim space target to select candidate objects to shrink.

> **Note:** The Segment Advisor does not evaluate undo and temporary tablespaces.

In addition to using the pre-computed statistics in the workload repository, the Segment Advisor performs sampling of the objects under consideration to refine the statistics for the objects. Although this operation is more resource intensive, it can be used to perform a more accurate analysis.

Although segment shrink reduces row chaining, and Oracle Database recommends online redefinition to remove chained rows, the Segment Advisor actually detects certain chained rows that are above a threshold. For example, if a row size increases during an update such that it not longer fits into the block, then the Segment Advisor recommends that the segment be reorganized to improve I/O performance.

> **Note:** The Segment Advisor does not detect chained rows created by inserts.

> **See Also:**
>
> ■ "Row Chaining and Migrating" on page 2-5 for more information about row chaining
>
> ■ *Oracle Database Administrator's Guide* and *Oracle Database 2 Day DBA* for more information about using the Segment Advisor

Storage Management

Automatic Storage Management (ASM) provides a vertical integration of the file system and volume manager specifically built for Oracle database files. ASM distributes I/O load across all available resources to optimize performance while removing the need for manual I/O tuning; spreading out the database files avoids hotspots. ASM helps you manage a dynamic database environment by enabling you to increase a database's size without having to shutdown the database to adjust the storage allocation.

ASM lets you define a pool of storage, called a disk group, and then the Oracle kernel manages the file naming and placement of the database files on that disk group. You can change the storage allocation, such as by adding or removing disks, by using SQL statements such as `CREATE DISKGROUP`, `ALTER DISKGROUP`, and `DROP`

DISKGROUP. You can also manage disk groups with Oracle Enterprise Manager and Database Configuration Assistant (DBCA).

Oracle Database provides a simplified management interface for storage resources. ASM eliminates the need for manual I/O performance tuning. It virtualizes storage to a set of disk groups and provides redundancy options to enable a high level of protection. ASM facilitates non-intrusive storage configuration changes with automatic rebalancing. It spreads database files across all available storage to optimize performance and resource utilization. ASM reduces your storage administrative overhead by automating manual storage and thereby increasing your ability to manage larger databases and more of them with increased efficiency.

Backup and Recovery

Oracle Database provides several features that help you to easily manage backup and recovery. These include the following:

- Recovery Manager
- Mean Time to Recovery
- Self Service Error Correction

Recovery Manager

Oracle Recovery Manager (RMAN) is a powerful tool that simplifies, automates, and improves the performance of backup and recovery operations. RMAN enables one time backup configuration, automatic management of backups and archived logs based on a user-specified recovery window, restartable backups and restores, and test restore/recovery.

RMAN implements a recovery window to control when backups expire. This lets you establish a period of time during which it is possible to discover logical errors and fix the affected objects by doing a database or tablespace point-in-time recovery. RMAN also automatically expires backups that are no longer required to restore the database to a point-in-time within the recovery window. Control file autobackup also allows for restoring or recovering a database, even when a RMAN repository is not available.

DBCA can automatically schedule an on disk backup procedure. All you do is specify the time window for the automatic backups to run. A unified storage location for all recovery related files and activities in an Oracle database, called the flash recovery area, can be defined with the initialization parameter DB_RECOVERY_FILE_DEST. All files needed to completely recover a database from a media failure, such as control files, archived log files, Flashback logs, RMAN backups, and so on, are part of the flash recovery area.

Allocating sufficient space to the flash recovery area ensures faster, simpler, and automatic recovery of the Oracle database. Flash recovery actually manages the files stored in this location in an intelligent manner to maximize the space utilization and avoid out of space situations to the extent possible. Based on the specified RMAN retention policy, the flash recovery area automatically deletes obsolete backups and archive logs that are no longer required based on that configuration.

Incremental backups let you back up only the changed blocks since the previous backup. When the block change tracking feature is enabled, Oracle Database tracks the physical location of all database changes. RMAN automatically uses the change tracking file to determine which blocks need to be read during an incremental backup and directly accesses that block to back it up. It reduces the amount of time needed for

daily backups, saves network bandwidth when backing up over a network, and reduces the backup file storage.

Incremental backups can be used for updating a previously made backup. With incrementally updated backups, you can merge the image copy of a datafile with a RMAN incremental backup, resulting in an updated backup that contains the changes captured by the incremental backup. This eliminates the requirement to make a whole database backup repeatedly. You can make a full database backup once for a given database and use incremental backups subsequently to keep the full back up updated. A backup strategy based on incrementally updated backups can help keep the time required for media recovery of your database to a minimum.

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
> - *Oracle Database Backup and Recovery User's Guide*

Mean Time to Recovery

Oracle Database allows for better control over database downtime by letting you specify the mean time to recover (MTTR) from system failures in number of seconds. This, coupled with dynamic initialization parameters, helps improve database availability. After you set a time limit for how long a system failure recovery can take, Oracle Database automatically and transparently makes sure that the system can restart in that time frame, regardless of the application activity running on the system at the time of the failure. This provides the fastest possible up time after a system failure.

The smaller the online logfiles are, the more aggressively DBWRs do incremental checkpoints, which means more physical writes. This may adversely affect the runtime performance of the database. Furthermore, if you set `FAST_START_MTTR_TARGET`, then the smallest logfile size may drive incremental checkpointing more aggressively than needed by the MTTR.

The Logfile Size Advisor determines the optimal smallest logfile size from the current `FAST_START_MTTR_TARGET` setting and the MTTR statistics. A smallest logfile size is considered optimal if it does not drive incremental checkpointing more aggressively than needed by `FAST_START_MTTR_TARGET`.

The MTTR Advisor helps you evaluate the effect of different MTTR settings on system performance in terms of extra physical writes. When MTTR advisor is enabled, after the system runs a typical workload, you can query `V$MTTR_TARGET_ADVICE` to see the ratio of the estimated number of cache writes under other MTTR settings to the number of cache writes under the current MTTR. For instance, a ratio of 1.2 indicates 20% more cache writes.

By looking at the different MTTR settings and their corresponding cache write ratio, you can decide which MTTR value fits your recovery and performance needs. `V$MTTR_TARGET_ADVICE` also gives the ratio on total physical writes, including direct writes, and the ratio on total I/O, including reads.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for information about using the MTTR Advisor

Self Service Error Correction

Oracle Flashback technology lets you view and rewind data back and forth in time. You can query past versions of schema objects, query historical data, perform change

analysis, or perform self-service repair to recover from logical corruptions while the database is online.

This revolutionizes recovery by just operating on the changed data. The time it takes to recover the error is equal to the amount of time it took to make the mistake.

> **See Also:**
>
> - "Overview of High Availability Features" on page 1-22
> - "Oracle Flashback Technology" on page 15-9
> - "Overview of Oracle Flashback Query" on page 13-26

Configuration Management

Oracle Enterprise Manager has several powerful configuration management facilities that help detect configuration changes and differences and enforce best practice configuration parameter settings. These capabilities also encompass the underlying hosts and operating systems.

Oracle Enterprise Manager continuously monitors the configuration of all Oracle systems for such things as best practice parameter settings, security set-up, storage and file space conditions, and recommended feature usage. Non-conforming systems are automatically flagged with a detailed explanation of the specific-system configuration issue. For example, Oracle Enterprise Manager advises you to use new functionality such as automatic undo management or locally managed tablespaces if they are not being used. This automatic monitoring of system configurations promotes best practices configuration management, reduces administrator workload and the risk of availability, performance, or security compromises.

Oracle Enterprise Manager also automatically alerts you to new critical patches – such as important security patches – and flags all systems that require that patch. In addition, you can invoke the Oracle Enterprise Manager patch wizard to find out what interim patches are available for that installation.

> **See Also:** *Oracle Enterprise Manager Concepts*

Workload Management

Oracle Database provides the following resource management features:

- Overview of the Database Resource Manager
- Overview of Services

Overview of the Database Resource Manager

The Database Resource Manager provides the ability to prioritize work within the Oracle database system. High priority users get resources, so as to minimize response time for online workers, for example, while lower priority users, such as batch jobs or reports, could take longer. This allows for more granular control over resources and provides features such as automatic consumer group switching, maximum active sessions control, query execution time estimation and undo pool quotas for consumer groups.

You can specify the maximum number of concurrently active sessions for each consumer group. When this limit is reached, the Database Resource Manager queues all subsequent requests and runs them only after existing active sessions complete.

The Database Resource Manager solves many resource allocation problems that an operating system does not manage so well:

- Excessive overhead. This 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 Oracle database servers while they hold latches, which is inefficient.

- Inappropriate allocation of resources. The operating system distributes resources equally among all active processes and is unable to prioritize one task over another.

- Inability to manage database-specific resources.

With the Database Resource Manager, you can do the following:

- Guarantee certain users a minimum amount of processing resources regardless of the load on the system and the number of users

- Distribute available processing resources by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage may be given to ROLAP (relational on-line analytical processing) applications than to batch jobs.

- Limit the degree of parallelism of any operation performed by members of a group of users

- Create an **active session pool**. This 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 terminate.

- Allow automatic switching of users from one group to another group based on administrator-defined criteria. If a member of a particular group of users creates a session that runs for longer than a specified amount of time, that session can be automatically switched to another group of users with different resource requirements.

> **Note:** Switching users or preventing operations could be based on amount of I/O, as well as amount of CPU time.

> **See Also:** *Oracle Database Administrator's Guide* for more information about automatic switching

- Create an **undo pool**. This pool consists of the amount of undo space that can be consumed in by a group of users.

- Configure an instance to use a particular method of allocating resources. You can dynamically change the method, for example, from a daytime setup to a nighttime setup, without having to shut down and restart the instance.

- Identify sessions that would block a quiesce from completing.

It is thus possible to balance one user's resource consumption against that of other users and to partition system resources among tasks of varying importance, to achieve overall enterprise goals.

Database Resource Manager Concepts

Resources are allocated to users according to a resource plan specified by the database administrator. The following terms are used in specifying a resource plan:

A **resource plan** specifies how the resources are to be distributed among various users (resource consumer groups).

Resource consumer groups let you group user sessions together by resource requirements. Resource consumer groups are different from user roles; one database user can have different sessions assigned to different resource consumer groups.

Resource allocation methods determine what policy to use when allocating for any particular resource. Resource allocation methods are used by resource plans and resource consumer groups.

Resource plan directives are a means of assigning consumer groups to particular plans and partitioning resources among consumer groups by specifying parameters for each resource allocation method.

The Database Resource Manager also allows for creation of plans within plans, called subplans. **Subplans** allow further subdivision of resources among different users of an application.

Levels provide a mechanism to specify distribution of unused resources among available users. Up to eight levels of resource allocation can be specified.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about using the Database Resource Manager
>
> - *Oracle Database Performance Tuning Guide* for information about how to tune resource plans

Overview of Services

Services represent groups of applications or a subset of a large application with common attributes, service level thresholds, and priorities. Application functions can be divided into workloads identified by services. For example, the Oracle E*Business suite can define a service for each module, such as general ledger, accounts receivable, order entry, and so on. Oracle Mail can define services for IMAP processes, postman, garbage collector, monitors, and so on. A service can span one or more instances of an Oracle database or multiple databases in a cluster, and a single instance can support multiple services.

The number of instances offering the service is transparent to the application. Services provide a single system image to manage competing applications, and they allow each workload to be managed as a single unit.

Middle tier applications and clients select a service by specifying the service name as part of the connection in the TNS connect data. For example, data sources for the Web server or the application server are set to route to a service. Using Net Easy*Connection, this connection includes the service name and network address. For example, service:IP.

Server side work, such as the Scheduler, parallel execution, and Oracle Streams Advanced Queuing set the service name as part of the workload definition. For the Scheduler, jobs are assigned to job classes, and job classes run within services. For parallel execution and parallel DML, the query coordinator connects to a service, and the parallel execution processes inherit the service for the duration of the query. For Oracle Streams Advanced Queuing, streams queues are accessed using services. Work

running under a service inherits the thresholds and attributes for the service and is measured as part of the service.

The Database Resource Manager binds services to consumer groups and priorities. This lets services be managed in the database in the order of their importance. For example, you can define separate services for high priority online users and lower priority internal reporting applications. Likewise, you can define gold, silver, and bronze services to prioritize the order in which requests are serviced for the same application.

When planning the services for a system, include the priority of each service relative to the other services. In this way, the Database Resource Manager can satisfy the highest priority services first, followed by the next priority services, and so on.

This section includes the following topics:

- Workload Management with Services
- High Availability with Services

Workload Management with Services

AWR lets you analyze the performance of workloads using the aggregation dimension for service. AWR automatically maintains response time and CPU consumption metrics, performance and resource statistics wait events, threshold-based alerts, and performance indexes for all services.

Service, module, and action tags identify operations within a service at the server. (MODULE and ACTION are set by the application) End to end monitoring enables aggregation and tracing at service, module, and action levels to identify high load operations. Oracle Enterprise Manager administers the service quality thresholds for response time and CPU consumption, monitors the top services, and provides drill down to the top modules and top actions for each service.

With AWR, performance management by the service aggregation makes sense when monitoring by sessions may not. For example, in systems using connection pools or transaction processing monitors, the sessions are shared, making accountability difficult.

The service, module, and action tags provide major and minor boundaries to discriminate the work and the processing flow. This aggregation level lets you tune groups of SQL that run together (at service, module, and action levels). These statistics can be used to manage service quality, to assess resource consumption, to adjust priorities of services relative to other services, and to point to places where tuning is required. With Oracle Real Application Clusters (Oracle RAC), services can be provisioned on different instances based on their current performance.

Connect time routing and runtime routing algorithms balance the workload across the instances offering a service. The metrics for server-side connection load balancing are extended to include service performance. Connections are shared across instances according to the current service performance. Using service performance for load balancing accommodates nodes of different sizes and workloads with competing priorities. It also prevents sending work to nodes that are hung or failed.

AWR maintains metrics for service performance continuously. These metrics are available when routing runtime requests from mid-tier servers and TP monitors to Oracle RAC. For example, Oracle JDBC connection pools use the service data when routing the runtime requests to instances offering a service.

High Availability with Services

Oracle RAC use services to enable uninterrupted database operations. Services are tightly integrated with the Oracle Clusterware high availability framework that supports Oracle RAC. When a failure occurs, the service continues uninterrupted on the nodes and instances unaffected by the failure. Those elements of the services affected by the failure are recovered fast by Oracle Clusterware, and the recovering sessions are balanced across the surviving system automatically.

For planned outages, Oracle RAC provides interfaces to relocate, disable, and enable services. Relocate migrates the service to another instance, and, as an option, the sessions are disconnected. To prevent the Oracle Clusterware system from responding to an unplanned failure that happens during maintenance or repair, the service is disabled on the node doing maintenance at the beginning of the planned outage. It is then enabled at the end of the outage.

These service-based operations, in combination with schema pre-compilation (DBMS_SCHEMA_COPY) on a service basis, minimize the downtime for many planned outages. For example, application upgrades, operating system upgrades, hardware upgrades and repairs, Oracle patches approved for rolling upgrade, and parameter changes can be implemented by isolating one or more services at a time.

The continuous service built into Oracle RAC is extended to applications and mid-tier servers. When the state of a service changes, (for example, up, down, or not restarting), the new status is notified to interested subscribers through events and callouts. Applications can use this notification to achieve very fast detection of failures, balancing of connection pools following failures, and balancing of connection pools again when the failed components are repaired. For example, when the service at an instance starts, the event and callouts are used to immediately trigger work at the service.

When the service at an instance stops, the event is used to interrupt applications using the service at that instance. Using the notification eliminates the client waiting on TCP timeouts. The events are integrated with Oracle JDBC connection pools, Oracle Data Provider for .Net Connection Pools, and Oracle Call Interface, including Transparent Application Failover (TAF).

With Oracle Data Guard, production services are offered at the production site. Other standby sites can offer reporting services when operating in read only mode. Oracle RAC and Data Guard Broker are integrated, so that when running failover, switchover, and protection mode changes, the production services are torn down at the original production site and built up at the new production site. There is a controlled change of command between Oracle Clusterware managing the services locally and Data Guard managing the transition. When the Data Guard transition is complete, Oracle Clusterware resumes management of the high availability operation automatically.

> **See Also:**
>
> - *Oracle Real Application Clusters Administration and Deployment Guide*
> - *Oracle Database Advanced Application Developer's Guide*
> - *Oracle Database PL/SQL Packages and Types Reference*
> - *Oracle Database Performance Tuning Guide*
> - "Oracle Scheduler" on page 14-26
> - "Overview of the Database Resource Manager" on page 14-17

Automatic Storage Management

Today's large databases demand minimal scheduled downtime, and DBAs are often required to manage multiple databases with an increasing number of database files. ASM lets you be more productive by making some manual storage management tasks obsolete.

ASM is a vertical integration of both the file system and the volume manager built specifically for Oracle Database files. It extends the concept of stripe and mirror everything (SAME) to optimize performance, while removing the need for manual I/O tuning (distributing the datafile layout to avoid hotspots). ASM helps manage a dynamic database environment by letting you increase the database size without shutting down the database to adjust the storage allocation. ASM also enables low cost modular storage to deliver higher performance and greater availability by supporting mirroring as well as striping.

ASM lets you create a pool of storage, and then it manages the file naming and placement of the database files on that storage. You can change the storage allocation (add to or remove disks) with the following SQL statements: `CREATE DISKGROUP`, `ALTER DISKGROUP`, and `DROP DISKGROUP`.

You can also manage the disk groups with Oracle Enterprise Manager. For creating and deleting files, the Oracle Database internally allocates storage space from these disk groups. As part of disk group setup, you can define failure groups as an additional level of redundancy to protect against loss of a subset of disks, such as a disk controller or disk array. Failure groups let ASM place a mirrored copy intelligently, by not putting copies of data on disks that would be inaccessible if a single component failed.

Oracle Database provides a simplified management interface for storage resources. ASM eliminates the need for manual I/O performance tuning. It simplifies storage to a set of disk groups and provides redundancy options to enable a high level of protection. ASM facilitates non-intrusive storage allocations and provides automatic rebalancing. It spreads database files across all available storage to optimize performance and resource utilization. It also saves time by automating manual storage tasks, which thereby increases their ability to manage more and larger databases with increased efficiency. Different versions of the database can interoperate with different versions of ASM. That is, any combination of release 10.1.x.y and 10.2.x.y for either the ASM instance or the database instance interoperate transparently.

This section includes the following topics:

- Basic Automatic Storage Management Concepts
- Benefits of Using Automatic Storage Management

Basic Automatic Storage Management Concepts

This section defines some of the basic concepts with ASM. This section includes the following topics:

- Disk Groups
- Automatic Storage Management Files
- Automatic Storage Management Templates
- Automatic Storage Management Disks
- Failure Groups
- Automatic Storage Management Instances

- SYSASM Privilege

Disk Groups

A disk group is one or more ASM disks managed as a logical unit. The data structures in a disk group are self contained and consume some of the disk space in a disk group. ASM disks can be added or dropped from a disk group while the database is running. ASM rebalances the data to ensure an even I/O load to all disks in a disk group even as the disk group configuration changes.

Any single ASM file is contained in a single disk group. However, a disk group can contain files belonging to several databases, and a single database can use storage from multiple disk groups. One or more disk groups can be specified as the default disk group for files created in a database.

Disk groups can be created when creating a database or when a new application is developed. Disk groups can also change when its database server configuration is altered.

Most installations probably have two disk groups. Reasons for having different disk groups include the following:

- To group disks of different sizes or performance characteristics together

- To group disks with different external redundancy together; disks that have the same external redundancy could be in the same disk group, but disks that have different external redundancy should be in different disk groups.

- To separate database areas and flash recovery areas for a database

Types of Disk Groups There are three types of disk groups: normal redundancy, high redundancy, and external redundancy. With normal and high redundancy, ASM provides redundancy for all files in the disk group according to the attributes specified in the disk group templates. High redundancy provides a greater degree of protection. With external redundancy, ASM does not provide any redundancy for the disk group. The underlying disks in the disk group should provide redundancy (for example, using a storage array), or the user must be willing to tolerate loss of the disk group if a disk fails (for example, in a test environment).

Automatic Storage Management Files

When the database requests it, ASM creates files. ASM assigns each file a fully qualified name ending in a dotted pair of numbers. You can create more user-friendly alias names for the ASM filenames. To see alias names for ASM files, query the V$ASM_ALIAS data dictionary view. V$ASM_ALIAS can only be queried from an ASM instance. Users can find the names for ASM files by querying the appropriate V$ view in the database instance (V$DATAFILE, V$LOGFILE, V$CONTROLFILE, and so on). In general, users need not be aware of file names.

> **Note:**   All existing situations where a filename is required are augmented with a mechanism for recognizing ASM file naming syntax.

When a file is created, certain file attributes are permanently set. Among these are its protection policy (mirroring) and its striping policy. ASM files are not visible from the operating system or its utilities, but they are visible to database instances, RMAN, and other Oracle-supplied tools.

To provide greater scalability and to improve performance, as ASM file sizes increase, the size of each new extent also increases automatically. Because ASM can dynamically resize extents, fewer extent pointers are needed to describe the file. This results in less memory being required to manage the extent maps. This feature improves ASM performance when opening and accessing large files (up to 128 TB) that are required for very large databases (VLDBs).

> **See Also:** *Oracle Database Storage Administrator's Guide* for information about ASM files and attributes

Automatic Storage Management Templates

ASM templates are collections of attributes used by ASM during file creation. Templates simplify file creation by mapping complex file attribute specifications into a single name. A default template exists for each Oracle file type. Each disk group contains its own definition of templates. Templates of the same name can exist in different disk groups with each having their own unique properties.

You can change the attributes of the default templates or add your own unique templates. This lets you specify the appropriate file creation attributes as a template. If you need to change an ASM file attribute after the file has been created, then the file must be copied using RMAN into a new file with the new attributes.

> **See Also:** *Oracle Database Storage Administrator's Guide* for a table describing the ASM default templates

Automatic Storage Management Disks

Storage is added and removed from disk groups in units of ASM disks. ASM disks can be entire physical disks, LUNs from a storage array, or pre-created files in a NAS filer. ASM disks should be independent of each other to obtain optimal I/O performance. For instance, with a storage array, you might specify a LUN that represents a hardware mirrored pair of physical disks to ASM as a single ASM disk. If you specify two separate LUNs that are striped by a storage array across the same set of physical drives, then this may cause suboptimal performance.

If using an NAS filer, the files specified as ASM disks must be a multiple of 1 megabyte. For optimal performance, if NAS files are in the same disk group, then they should have independent underlying physical drives.

You should not specify a device that contains data that should not be overwritten. For instance, on some operating systems, certain partitions contain the partition table at the beginning of the partition. Such partitions should not be specified as ASM disks.

ASM performs I/O to ASM disks through a single logical path. Therefore, if you are using a multi-pathing driver with ASM, then you should specify the logical path to the ASM disk and ensure that the ASM discovery string (ASM_DISKSTRING initialization parameter) includes only a single logical path to each ASM disk.

> **See Also:** *Oracle Database Storage Administrator's Guide* for more information about the `ASM_DISKSTRING` parameter

In a cluster, ASM disks must be visible to all ASM instances in the cluster, but the path to the ASM disk need not be identical on each node, as long as each instance's `ASM_DISKSTRING` includes the path to the disk for that instance.

A disk name is common to all nodes of the cluster. The name must be specified by the administrator, or it can be automatically generated by ASM when the disk is added to

a disk group. The ASM disk name abstraction is required, because different hosts can use different operating system names to refer to the same disk.

ASM provides mirroring to avoid data loss due to disk failure. This is necessary, because the loss of unduplicated data from a single ASM disk could damage every file in a disk group.

Failure Groups

Failure groups are administratively assigned sets of disks sharing a common resource. Failure groups are used to determine which ASM disks to use for storing redundant copies of data. The use of failure groups ensures that data and the redundant copy of the data do not both reside on disks that are likely to fail together.

The composition of failure groups is site-specific. Failure group decisions are based on what component failures the system can tolerate. For example, suppose you have five disks and one SCSI controller. The failure of the SCSI controller makes all disks unavailable. In this scenario, you should put each disk in its own failure group. However, if you have two SCSI controllers, each with two disks, and you want to tolerate a SCSI controller failure, then you should create a failure group for each controller.

By default, ASM assigns each disk to its own failure group. When creating a disk group or adding a disk to a disk group, you can specify you own grouping of disks into failure groups. ASM can then optimize file layout to reduce the unavailability of data due to failures.

A failure group is maintained in a disk group, and multiple failure groups can exist within any given disk group. However, changing a disk's failure group requires dropping the disk from the disk group and then adding the disk back to the disk group under the new failure group name.

Automatic Storage Management Instances

The ASM instance is a special Oracle database instance that manages the disks in disk groups. The ASM instance must be configured and running to enable the database instance to access ASM files. This configuration is done automatically if Database Configuration Assistant was used for database creation.

ASM instances cannot mount databases. The ASM instances simply coordinate data layout for database instances. Database instances direct the I/O to disks in disk groups without going through an ASM instance.

Multiple and separate database instances can share disk groups for their files. On a single node, there is typically a single ASM instance on the node that manages all of the disk groups. In an Oracle RAC environment, there is typically one ASM instance on each node managing all of the disk groups for its node in a coordinated manner with the rest of the cluster.

All ASM management commands must be directed to the ASM instance, and not to the Oracle database instance using the ASM files.

You can perform rolling upgrades of ASM instances clustered environments by using the rolling upgrade mode. This enables the ASM instances to operate with multiple ASM versions. When an environment is in rolling upgrade mode, you can upgrade or patch the ASM instances one at a time without affecting database availability. You can use this feature in environments that run Oracle Database 11*g* release 1 (11.1) or later. For more information about ASM rolling upgrades, refer to the *Oracle Database Storage Administrator's Guide*.

Automatic Storage Management Instance Background Processes An ASM instance contains two background processes. One process, referred to as RBAL, coordinates rebalance activity for disk groups. A second process performs the actual rebalance data extent movements. Multiple ASM rebalancing processes can exist at one time, and they are referred to as ARB0, ARB1, and so on. An ASM instance also has most of the same background processes as a database instance, such as SMON, PMON, LGWR, and so on.

Database Instance Background Processes A database instance using an ASM disk group contains a background process called ASMB that communicates with the ASM instance. Another background process called RBAL performs the global open operation on ASM disks.

SYSASM Privilege

Use the SYSASM privilege to perform ASM administration tasks instead of the SYSDBA role. Using the SYSASM privilege instead of the SYSDBA privilege provides a clearer division of responsibility between ASM administration and database administration. For more information about the SYSASM privilege, refer to the *Oracle Database Storage Administrator's Guide*.

Benefits of Using Automatic Storage Management

- ASM provides easier administration.

  There is no need to specify and manage filenames. Wherever a file is created, a disk group can be specified instead. Every new file automatically gets a new unique name. This prevents using the same filename in two different databases. Disk group naming avoids using two different names for the same file.

- For many situations, ASM provides the functions provided by external volume managers and file systems.

  ASM includes storage reliability features, such as mirroring. The storage reliability policy is applied on a file basis, rather than on a volume basis. Hence, the same disk group can contain a combination of files protected by mirroring, parity, or not protected at all.

- ASM improves performance.

  ASM maximizes performance by automatically distributing database files across all disks in a disk group. It has the performance of raw disk I/O without the inconvenience of managing raw disks.

  Unlike logical volume managers, ASM maintenance operations do not require that the database be shut down. This allows adding or dropping disks while the disks are in use.

  ASM eliminates the need for manual disk tuning. To help manage performance, file creation attributes are controlled by disk group-specific templates.

  > **See Also:** *Oracle Database Storage Administrator's Guide* for detailed information about using ASM

Oracle Scheduler

Oracle Database includes a feature rich job scheduler. You can schedule jobs to run at a designated date and time (such as every weeknight at 11:00pm), or upon the

occurrence of a designated event (such as when inventory drops below a certain level). You can define custom calendars such as the last workday of every fiscal quarter.

You create and manipulate Scheduler objects such as jobs, programs, and schedules with the DBMS_SCHEDULER package or with Oracle Enterprise Manager. Because Scheduler objects are standard database objects, you can control access to them with system and object privileges.

Program objects (or *programs*) contain metadata about the command that the Scheduler will run, including default values for any arguments. Schedule objects (*schedules*) contain information about run date and time and recurrence patterns. Job objects (*jobs*) associate a program with a schedule, and are the principal object that you work with in the Scheduler. You can create multiple jobs that refer to the same program but that run at different schedules. A job can override the default values of program arguments, so multiple jobs can refer to the same program but provide different argument values.

The Scheduler provides comprehensive job logging in Oracle Enterprise Manager and in a variety of views available from SQL*Plus. You can configure a job to raise an event when a specified job state change occurs. Your application can process the event and take appropriate action. For example, the Scheduler can page or send an e-mail to the DBA if a job terminates abnormally.

The Scheduler also includes **chains**, which are named groups of steps that work together to accomplish a task. Steps in the chain can be a program, subchain or an event, and you specify rules that determine when each step runs and what the dependencies between steps are. An example of a chain is to run programs A and B, and only run program C if programs A and B complete successfully, otherwise run program D.

The Scheduler is integrated with the Database Resource Manager. You can associate Scheduler jobs with resource consumer groups, and you can create Scheduler objects called windows that 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 Scheduler job can name a window as its schedule instead of a schedule object. Such a job runs when the named window opens. Additionally, windows can be grouped into *window groups*, and a job can name a window group as its schedule. Such a job runs whenever any of the windows in the named window group opens.

> **See Also:** *Oracle Database Administrator's Guide* for a detailed overview of the Scheduler and for information about how to use and administer the Scheduler

What Can the Scheduler Do?

The Scheduler provides complex enterprise scheduling functionality. You can use this functionality to do the following:

- Schedule Job Execution

- Time-Based Scheduling

- Event-Based Scheduling

- Define Multi-Step Jobs

- Schedule Job Processes that Model Business Requirements

- Manage and Monitor Jobs

- Execute and Manage Jobs in a Clustered Environment

Schedule Job Execution

The most basic capability of a job scheduler is to schedule the execution of a job. The Scheduler supports both time-based and event-based scheduling.

Time-Based Scheduling

Time-based scheduling enables users to specify a fixed date and time (for example, Jan. 23rd 2006 at 1:00 AM), a repeating schedule (for example, every Monday), or a defined rule (for example the last Sunday of every other month or the fourth Thursday in November which defines Thanksgiving).

Users can create new composite schedules with minimum effort by combining existing schedules. For example if a HOLIDAY and WEEKDAY schedule were already defined, a WORKDAY schedule can be easily created by excluding the HOLIDAY schedule from the WEEKDAY schedule.

Companies often use a fiscal calendar as opposed to a regular calendar and thus have the requirement to schedule jobs on the last workday of their fiscal quarter. The Scheduler supports user-defined frequencies which enables users to define not only the last workday of every month but also the last workday of every fiscal quarter.

Event-Based Scheduling

Event-based scheduling as the name implies triggers jobs based on real-time events. Events are defined as any state changes or occurrences in the system such as the arrival of a file. Scheduling based on events enables you to handle situations where a precise time is not known in advance for when you would want a job to execute.

Define Multi-Step Jobs

The Scheduler has support for single or multi-step jobs. Multi-step jobs are defined using a Chain. A Chain consists of multiple steps combined using dependency rules. Since each step represents a task, Chains enable users to specify dependencies between tasks, for example execute task C one hour after the successful completion of task A and task B.

Schedule Job Processes that Model Business Requirements

The Scheduler enables job processing in a way that models your business requirements. It enables limited computing resources to be allocated appropriately among competing jobs, thus aligning job processing with your business needs. Jobs that share common characteristic and behavior can be grouped into larger entities called job classes. You can prioritize among the classes by controlling the resources allocated to each class. This lets you ensure that critical jobs have priority and enough resources to complete. Jobs can also be prioritized within a job class.

The Scheduler also provides the ability to change the prioritization based on a schedule. Because the definition of a critical job can change across time, the Scheduler lets you define different class priorities at different times.

Manage and Monitor Jobs

There are multiple states that a job undergoes from its creation to its completion. All Scheduler activity is logged, and information, such as the status of the job and the time to completion, can be easily tracked. This information is stored in views. It can be queried with Oracle Enterprise Manager or a SQL query. The views provide information about jobs and their execution that can help you schedule and manage your jobs better. For example, you can easily track all jobs that failed for user scott.

In order to facilitate the monitoring of jobs, users can also flag the Scheduler to raise an event if unexpected behavior occurs and indicate the actions that should be taken if the specified event occurs. For example if a job failed an administrator should be notified.

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 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 service where you want a job to run.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about transferring files with the DBMS_SCHEDULER package and also the DBMS_FILE_TRANSFER package
>
> - *Oracle Database SQL Language Reference* for more information about fixed user database links

15

Backup and Recovery

Backup and recovery procedures protect your database against data loss and reconstruct the data, should loss occur. This chapter introduces concepts fundamental to designing a backup and recovery strategy.

This chapter contains the following topics:

- Introduction to Backup and Recovery
- Database Backups
- Problems Requiring Data Repair
- Data Repair

> **See Also:**
>
> - "Overview of Database Backup and Recovery Features" on page 1-20
> - *Oracle Database Backup and Recovery User's Guide* for backup and recovery concepts and tasks

Introduction to Backup and Recovery

A backup is a copy of data. This copy can include important parts of the database such as datafiles, which contain user data, and the server parameter file and control file, which contain configuration information.

The main purpose of a backup is as a safeguard against unexpected data loss and application errors. For example, a disk may fail, causing the loss of datafiles. You can restore a backup of the data and reconstruct the lost data through media recovery. Media recovery refers to the various operations involved in restoring, rolling forward, and rolling back a backup of database files.

You have two ways to perform backup and recovery of an Oracle database: **Recovery Manager (RMAN)** and user-managed techniques. RMAN is an Oracle Database utility that can back up, restore, and recover database files. It is a feature of Oracle Database and does not require separate installation. You can also use operating system commands for backups and SQL*Plus for media recovery. This technique, also called user-managed backup and recovery, is fully supported by Oracle, although use of RMAN is recommended because it is more robust and simplifies administration.

Oracle Flashback Technology is an alternative to traditional backup and recovery. You can use flashback features to view past states of data, and move data back and forth in time, without restoring data from backups. Instead, you can issue a single command to rewind your entire database, or a single table, to a time in the past. The flashback

features of Oracle Database are more efficient and less disruptive than media recovery in most circumstances in which they are applicable.

No matter which backup and recovery tool you use, it is recommended that you configure a flash recovery area to manage your recovery-related files.

Flash Recovery Area

The **flash recovery area** is an optional Oracle Database-managed directory, file system, or Automatic Storage Management disk group that provides a centralized disk location for backup and recovery files. You can configure the flash recovery area when creating a database with the Database Configuration Assistant or add it later.

Oracle Database can write archived logs to the flash recovery area. RMAN can store backups in the flash recovery are and restore them from the flash recovery area during media recovery. The flash recovery area also acts as a disk cache for tape.

Oracle Database recovery components interact with the flash recovery area to ensure that the database is completely recoverable by using files stored in the recovery area. All files necessary to recover the database following a media failure are part of the flash recovery area.

The following recovery-related files are stored in the flash recovery area:

- Current control file

- Online redo logs

- Archived redo logs

- Flashback logs

- Control file autobackups

- Datafile and control file copies

- Backup pieces

Oracle Database enables you to define a disk limit, which is the amount of space that the database can use in the flash recovery area. A disk limit enables you to use the remaining disk space for other purposes and not to dedicate a complete disk for the flash recovery area. It does not include any overhead that is not known to Oracle Database. For example, the disk limit does not include the extra size of a file system that is compressed, mirrored, or uses some other redundancy mechanism.

Oracle Database and RMAN create files in the flash recovery area until the space used reaches the recovery area disk limit. When it needs to make room for new files, Oracle Database deletes files from the flash recovery area that are obsolete, redundant, or backed up to tertiary storage. Oracle Database prints a warning when available disk space is less than 15%, but it continues to fill the disk to 100% of the disk limit.

The bigger the flash recovery area, the more useful it becomes. The recommended disk limit is the sum of the database size, the size of incremental backups, and the size of all archive logs that have not been copied to tape.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for the rules that define the priority of file deletion, as well as other information about the flash recovery area
>
> - *Oracle Database Administrator's Guide* for information about how to set up and administer the flash recovery area

Database Backups

This section describes physical backups. This section includes the following topics:

- What Are Database Backups?
- Whole Database and Partial Database Backups
- Consistent and Inconsistent Backups
- RMAN and User-Managed Backups

What Are Database Backups?

Database backups can be either physical or logical. Physical backups, which are the primary concern in a backup and recovery strategy, are copies of physical database files. You can make physical backups with either RMAN or operating system utilities.

In contrast, logical backups contain logical data such as tables and stored procedures. You can extract the logical data with an Oracle Database utility such as Data Pump Export and store it in a binary file. Logical backups can supplement physical backups.

The primary purpose of a database backup is for data protection, but you can also create archival database backups for data preservation. For example, suppose you have a business requirement to preserve customer transaction records for a specified period of time. You can use RMAN to create an archival backup of the database, along with the redo necessary to make it consistent, for offsite storage. You can control how long this database backup is exempt from the RMAN retention policies that govern the deletion of obsolete backups.

Whole Database and Partial Database Backups

A **whole database backup** is a backup of every datafile in the database, plus the control file. Whole database backups are the most common type of backup.

As shown in Figure 15–1, a whole database backups can be taken in either ARCHIVELOG or NOARCHIVELOG mode and is either a **consistent backup** or an **inconsistent backup**. Whether a backup is consistent determines whether you need to apply redo logs after restoring the backup.

Figure 15–1   Whole Database Backup Options
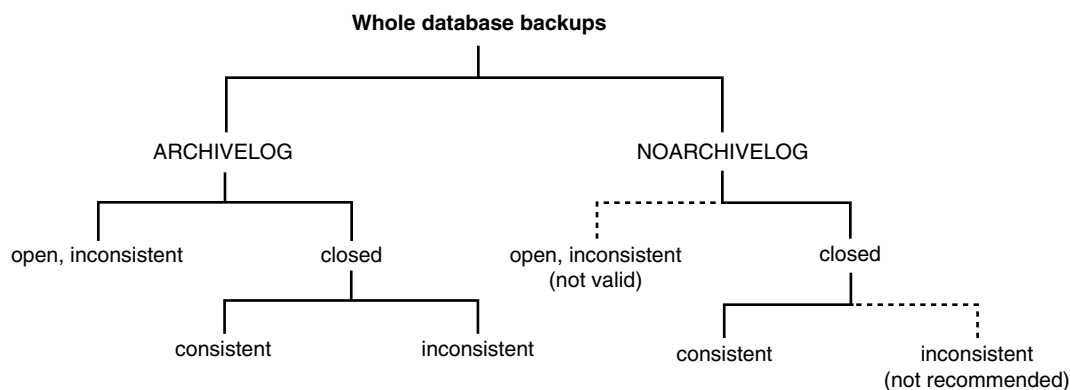
A partial backup includes a subset of the database, that is, individual tablespaces or datafiles. A tablespace backup is a backup of the datafiles that make up the tablespace. Tablespace backups, whether online or offline, are valid only if the database is

operating in `ARCHIVELOG` mode. The reason is that redo is required to make the restored tablespace consistent with the other tablespaces in the database.

A datafile backup is a backup of a single datafile. Datafile backups, which are not as common as tablespace backups, are valid in `ARCHIVELOG` databases.

> **See Also:** *Oracle Database Backup and Recovery Reference* and *Oracle Database Utilities* for information about logical backups

Consistent and Inconsistent Backups

Database backups are either consistent or inconsistent. This section explains the difference between them.

This section includes the following topics:

- Overview of Consistent Backups
- Overview of Inconsistent Backups

Overview of Consistent Backups

In a consistent database backup, all read/write datafiles and control files are checkpointed with the same **system change number (SCN)**. The files in the backup are guaranteed to contain all changes up to the same SCN. Unlike an inconsistent backup, a consistent whole database backup does not require recovery after it is restored.

The only way to make a consistent whole database backup is to shut down the database with the `NORMAL`, `IMMEDIATE`, or `TRANSACTIONAL` options and make the backup while the database is closed. If a database is not shut down consistently, for example, an instance fails or you issue a `SHUTDOWN ABORT` statement, then the datafiles are always inconsistent—unless the database is a **read-only database**.

The important point is that you can open the database after restoring a consistent whole database backup *without needing recovery* because the data is already consistent: no action is required to make the data in the restored datafiles correct. Thus, you can restore a year-old consistent backup of your database without performing media recovery and without the database performing instance recovery.

> **Note:** When you restore a consistent whole database backup without applying redo, you lose all transactions that were made after the backup was taken.

A consistent whole database backup is the only valid backup option for databases operating in `NOARCHIVELOG` mode. Other backup options require recovery for consistency, which is not possible without archived redo logs.

A consistent whole database backup is also a valid backup option for databases operating in `ARCHIVELOG` mode. When this type of backup is restored and archived logs are available, you have the option of either opening the database immediately and losing transactions that were made after the backup was made, or applying the archived logs to recover those transactions.

Overview of Inconsistent Backups

In an inconsistent database backup, read/write datafiles and control files are not guaranteed to be checkpointed to the same SCN. The files in the backup can contain data taken from different points in time, which means that changes can be missing. This situation can occur when datafiles are modified while backups are being taken.

If you back up the database when it is open or mounted after an inconsistent shutdown, then the backup is inconsistent. A backup of online datafiles is called an **online backup**. You must run the database in `ARCHIVELOG` mode for online backups.

As long as the database runs in `ARCHIVELOG` mode, and you back up the archived redo logs and datafiles, inconsistent backups can be the foundation for a sound backup and recovery strategy. Inconsistent backups offer superior availability because you do not have to shut down the database to make backups that fully protect the database.

Oracle Database recovery makes inconsistent backups consistent by reading all archived and online redo logs, starting with the earliest SCN in any of the datafile headers, and applying the changes from the logs back into the datafiles. After making an inconsistent backup, always ensure that you have the redo necessary to recover the backup by archiving the unarchived redo logs. If you do not have all archived redo logs produced during the backup, then you cannot recover it because you do not have all the redo necessary to make it consistent.

> **See Also:** *Oracle Database Backup and Recovery User's Guide*

RMAN and User-Managed Backups

The RMAN `BACKUP` command generates either image copies or backup sets. An image copy is an exact duplicate of a datafile, control file, or archived log. You can create image copies of physical files with operating system utilities or RMAN, and you can restore them as-is without performing additional processing by using either operating system utilities or RMAN.

> **Note:** Unlike operating system copies, RMAN validates the blocks in the file and records the image copy in the repository.

A backup set is a backup in a proprietary format that consists of one or more physical files called backup pieces. A backup set can contain multiple datafiles. The smallest unit of a backup set is a binary file called a backup piece. Backup sets, which are only created and accessed through RMAN, are the only form in which RMAN can write backups to sequential devices such as tape drives.

This section includes the following topics:

- Online Backups
- Control File Backups
- Archived Redo Log Backups

Online Backups

Because the database continues writing to the file during an online backup, it is possible to back up inconsistent data within a block. For example, assume that either RMAN or an operating system utility reads the block while database writer is in the middle of updating the block. In this case, RMAN or the copy utility could read the new data in the first half of the block and the old data in the second half of the block. The block is fractured, meaning that the data in this block is not consistent.

During an RMAN backup, the Oracle database reads the datafiles, not an operating system utility. The server reads each block and determines whether the block is fractured. If the block is fractured, then the database re-reads the block until it gets a valid block.

When you back up an online datafile with an operating system utility rather than with RMAN, you must use a different method to handle fractured blocks. You must first place the files in backup mode with the `ALTER TABLESPACE BEGIN BACKUP` statement (to back up an individual tablespace), or the `ALTER DATABASE BEGIN BACKUP` statement (to back up the entire database). After an online backup is completed, you must run the `ALTER TABLESPACE...END BACKUP` or `ALTER DATABASE END BACKUP` statement to take the files out of backup mode.

When updates are made to files in backup mode, additional redo data is logged. This additional data is needed to repair fractured blocks that might be backed up by the operating system utility.

Control File Backups

Backing up the control file is a crucial aspect of backup and recovery. Without a control file, you cannot mount or open the database. You can instruct RMAN to automatically backup the control file whenever you run backup jobs by executing `CONFIGURE CONTROLFILE AUTOBACKUP ON`. Because the autobackup uses a default filename, RMAN can restore this backup even if the RMAN repository is unavailable. Hence, this feature is extremely useful in a disaster recovery scenario.

You can make manual backups of the control file by using the following methods:

- The RMAN `BACKUP CURRENT CONTROLFILE` command makes a binary backup of the control file, as either a backup set or an image copy.

- The SQL statement `ALTER DATABASE BACKUP CONTROLFILE` makes a binary backup of the control file.

- The SQL statement `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` exports the control file contents to a SQL script file. You can use the script to create a new control file. Trace file backups have one major disadvantage: they contain no records of archived redo logs, and RMAN backups and copies. For this reason, binary backups are preferable.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide*
> - *Oracle Database Backup and Recovery Reference*

Archived Redo Log Backups

You can use archived redo logs to roll a backup forward in time. To recover a backup through the most recent archived redo log, every log generated after the backup was made must be available. In other words, you cannot recover from archived redo log 100 to log 200 if log 173 is missing. If log 173 is missing, then you must halt recovery after applying log 172 and open the database with the `RESETLOGS` option.

Because archived redo logs are essential to recovery, you should back them up regularly. If you use a media manager, then back up the logs regularly to tape. You can make backups of archived logs by using the following methods:

- The RMAN `BACKUP ARCHIVELOG` command

- The RMAN `BACKUP...PLUS ARCHIVELOG` command

- An operating system utility

See Also:

- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Database Backup and Recovery Reference*

Problems Requiring Data Repair

The following failures may require DBA intervention, and may even crash a database instance, but will not generally cause data loss or the need to recover from backup.

- Instance failures
- Network failures
- Failure of Oracle Database background processes
- Failure of a statement to execute due to, for example, exhaustion of some resource such as space in a datafile

Typically, data recovery is a response to media failures or user errors.

This section includes the following topics:

- Media Failures
- User Errors

Media Failures

A media failure occurs when a problem external to the database prevents Oracle Database from reading from or writing to a file during database operations. Typical media failures include physical failures, such as head crashes, and the overwriting, deletion or corruption of a database file. Media failures are less common than user or application errors, but your backup and recovery strategy should prepare for them.

Database operation after a media failure of online redo log files or control files depends on whether the files are protected by **multiplexing**. When an online redo log or control file is multiplexed, the database maintains multiple copies of the file.

If a media failure damages a disk containing one copy of a multiplexed online redo log, then the database can usually continue to operate without significant interruption. Damage to a nonmultiplexed online redo log causes database operation to halt and may cause permanent loss of data.

Damage to any control file, whether it is multiplexed or not, halts the database when it attempts to read or write to the damaged control file. The database accesses the control file frequently, for example, at every checkpoint and online redo log switch.

Media failures are either **read errors** or **write errors**. In a read error, the instance cannot read a datafile and an operating system error is returned to the application, along with an error indicating that the file cannot be found, cannot be opened, or cannot be read. The database continues to run, but the error is returned each time an unsuccessful read occurs. At the next checkpoint, a write error will occur when the database attempts to write to the datafile header as part of the checkpoint process.

The effect of a datafile write error depends upon which tablespace the datafile is in. If the instance cannot write to a datafile in the **SYSTEM tablespace**, an undo tablespace, or a datafile with active rollback segments, then the database issues an error and shuts down. All files in the SYSTEM tablespace and all datafiles containing undo or rollback segments must be online in order for the database to operate properly.

If the instance cannot write to a datafile other than those in the preceding list, then the result depends on whether the database is running in ARCHIVELOG mode. In ARCHIVELOG mode, the database records an error in the database writer trace file and takes the affected datafile offline. All other datafiles in the tablespace containing this datafile remain online. You can then rectify the underlying problem and restore and recover the affected tablespace.

In NOARCHIVELOG mode, the database writer background process fails and the instance fails. The cause of the problem determines the required response. If the problem is temporary, then crash recovery usually can be performed using the online redo log files. In such situations, the instance can be restarted without resorting to media recovery. If a datafile is damaged, however, then you must restore a **consistent backup** of the entire database.

User Errors

A user or application may make unwanted changes to your database, such as erroneous updates, deleting the contents of a table, or dropping database objects. An adequate backup and recovery strategy uses the many features of Oracle Database to let you return your database to the desired state, with the minimum possible impact upon database availability, and minimal DBA effort.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* to learn how to perform point-in-time recovery for an entire database
>
> - *Oracle Database Backup and Recovery User's Guide* to learn how to perform tablespace point-in-time recovery
>
> - *Oracle Database Backup and Recovery User's Guide* to learn how to use the flashback features of Oracle Database

Data Repair

Typically, you have more than one way to solve the problems described in "Problems Requiring Data Repair" on page 15-7.

Data Repair Advisor is an integrated solution that performs much of the diagnosis and repair work for you. Data Repair Advisor can diagnose failures, suggest both manual and automated repair options, and in some cases automatically repair failures.

To correct problems caused by logical data corruptions or user errors, you can use Oracle Flashback as an alternative to media recovery. Oracle Flashback features enable you to rewind the whole database or a subset of the database to a previous time.

To correct media failures, you can use media recovery. Media recovery is the application of redo or incremental backups to a backup to update it with lost changes. Block media recovery is a more specialized operation that you use when just a few blocks in one or more files have been corrupted.

This section includes the following topics:

- Data Recovery Advisor

- Oracle Flashback Technology

- Media Recovery

Data Recovery Advisor

Oracle Database includes the **Data Recovery Advisor** tool, which automatically diagnoses persistent data failures, presents appropriate repair options, and executes them at your request. You can use Data Repair Advisor either through the Enterprise Manager interface or through the RMAN client.

A checker is a diagnostic operation or procedure registered with the Health Monitor to assess the health of the database or its components. The health assessment is known as a data integrity check and can be invoked reactively or proactively.

Failures are normally detected reactively. A database operation involving corrupted data results in an error, which automatically invokes a data integrity check that searches the database for failures related to the error. If failures are diagnosed, then they are recorded in the Automatic Diagnostic Repository (ADR). You can also invoke a data integrity check proactively through the Health Monitor or by checking for block corruption with the VALIDATE and BACKUP commands in RMAN.

You can use Data Recovery Advisor to generate repair advice and repair failures only after failures have been detected by the database and stored in ADR. Each failure has a status: open or closed. Each failure also has a priority: critical, high, or low. Failures with critical priority require immediate attention because they make the whole database unavailable. Failures with high priority make a database partly unavailable or unrecoverable, and usually have to be repaired in a reasonably short time. Examples of high-priority failures include data block corruptions and non-fatal I/O errors. Low priority failures can wait until more important failures are fixed.

Data Repair Advisor automatically determines the best repair options and their impact on the database. Typically, Data Repair Advisor generates both manual and automated repair options for each failure or group of failures. The manual options are categorized as either mandatory or optional.

Before presenting an automated repair option, Data Repair Advisor validates it with respect to the specific environment, as well as availability of media components required to complete the proposed repair. If you choose an automatic repair, then Oracle Database executes it for you. The Data Repair Advisor tool verifies the repair success and closes the appropriate failures.

Oracle Flashback Technology

Oracle Database provides a group of features known as Oracle Flashback Technology that support viewing past states of data, and winding data back and forth in time, without requiring the restore of the database from backup. Depending on the changes to your database, Flashback features can often reverse the unwanted changes more quickly and with less impact on database availability than media recovery.

> **See Also:** "Overview of High Availability Features" on page 1-22 for an overview of all Oracle Flashback features, including those not directly related to backup and recovery

This section includes the following topics:

- Oracle Flashback Database
- Oracle Flashback Table
- Oracle Flashback Drop

Oracle Flashback Database

Oracle Flashback Database enables you to rewind an Oracle database to a previous time to correct problems caused by logical data corruptions or user errors.

If a flash recovery area is configured, and if you have enabled the Flashback database functionality, then you can use the RMAN or SQL FLASHBACK DATABASE command to return the database to a prior time. Flashback Database is not true media recovery because it does not involve restoring physical files. Flashback Database is preferable to using the RESTORE and RECOVER commands in some cases because it is faster and easier and does not require restoring the whole database.

When you use Flashback Database, Oracle Database uses past block images to back out changes to the database. During normal database operation, Oracle Database occasionally logs these block images in flashback logs. Flashback logs are written sequentially and are not archived. Oracle Database automatically creates, deletes, and resizes flashback logs in the flash recovery area. You only need to be aware of flashback logs for monitoring performance and deciding how much disk space to allocate to the flash recovery area for flashback logs.

The time it takes to rewind a database with FLASHBACK DATABASE is proportional to how far back in time you need to go and the amount of database activity after the target time. The time it would take to restore and recover the whole database could be much longer. The before images in the flashback logs are only used to restore the database to a point in the past, and forward recovery is used to bring the database to a consistent state at some time in the past. Oracle Database returns datafiles to the previous point-in-time, but not auxiliary files, such as initialization parameter files.

Flashback database can also be used to compliment Data Guard, Recovery Advisor, and for synchronizing clone databases.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for details about using Oracle Flashback Database
> - *Oracle Database SQL Language Reference* for information about the FLASHBACK DATABASE statement
> - *Oracle Data Guard Concepts and Administration* for information on how flashback database compliments Oracle Data Guard
> - *Oracle Database High Availability Overview* for information on further uses of flashback database and restore points

Oracle Flashback Table

Oracle Flashback Table enables you to rewind tables to a specified point in time with a single statement. You can restore table data along with associated indexes, triggers, and constraints, while the database is online, undoing changes to only the specified

tables. Oracle Flashback Table does not address physical corruption such as bad disks or data segment and index inconsistencies.

Oracle Flashback Table works like a self-service repair tool. Suppose a user accidentally deletes some important rows from a table and wants to recover the deleted rows. You can restore the table to the time before the deletion and see the missing rows in the table with the `FLASHBACK TABLE` statement.

You can restore the table and its contents to a certain wall clock time or user-specified system change number (SCN). Use Oracle Flashback Table with Oracle Flashback Version Query and Oracle Flashback Transaction Query to find a time to which to restore the table.

For Oracle Flashback Table to succeed, the system must retain enough undo information to satisfy the specified SCN or timestamp, and the integrity constraints specified on the tables cannot be violated. Also, row movement must be enabled on the table.

The availability of retained undo information for Oracle Flashback Table is controlled by the automatically tuned undo retention period of the system. The undo retention period indicates the amount of time that must pass before old undo information—that is, undo information for committed transactions—can be overwritten. The database collects usage statistics and tunes the undo retention period based on these statistics and on undo tablespace size. You can request a minimum undo retention period by setting the `UNDO_RETENTION` initialization parameter.

> **Note:** Automatic tuning of undo retention occurs only when the database is in automatic undo management mode (the default). The database may or may not be able to honor your request for a minimum undo retention period. This depends on a number of factors, including the current transaction activity on the system, whether the undo tablespace is autoextending or fixed size, and whether you specified `RETENTION GUARANTEE` for the undo tablespace.
>
> See *Oracle Database Administrator's Guide* for more information about the automatic tuning of undo retention.

See Also:

-

- *Oracle Database Backup and Recovery User's Guide* for details about using Oracle Flashback Table

- *Oracle Database SQL Language Reference* for information on the `UNDO_RETENTION` initialization parameter and information about the `FLASHBACK TABLE` statement

Oracle Flashback Drop

Oracle Flashback Drop reverses the effects of a `DROP TABLE` operation. Flashback Drop is substantially faster than other recovery mechanisms that can be used in this situation, such as point-in-time recovery, and does not lead to any loss of recent transactions or downtime.

When you drop a table, the database does not immediately remove the space associated with the table. Instead, the table is renamed and, along with any associated objects, is placed in the recycle bin of the database. Oracle Database uses the recycle

bin to manage dropped database objects until the space they occupied is needed to store new data. The recycle bin is actually a data dictionary table that contains information about the dropped objects.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for details about using Oracle Flashback Drop

Media Recovery

To restore a physical backup of a datafile or control file is to reconstruct it and make it available to the Oracle database. To recover a restored datafile is to update it by applying archived redo logs and online redo logs, that is, records of changes made to the database after the backup was taken. If you use RMAN, then you can also recover datafiles with incremental backups, which are backups of a datafile that contain only blocks that changed after a previous incremental backup.

After the necessary files are restored, media recovery must be initiated by the user. Media recovery involves various operations to restore, roll forward, and roll back a backup of database files.

Media recovery applies archived redo logs and online redo logs to recover the datafiles. Whenever a change is made to a datafile, the change is first recorded in the online redo logs. Media recovery selectively applies the changes recorded in the online and archived redo logs to the restored datafile to roll it forward.

Figure 15–2 illustrates the basic principle of backing up, restoring, and performing media recovery on a database.

Figure 15–2   Media Recovery
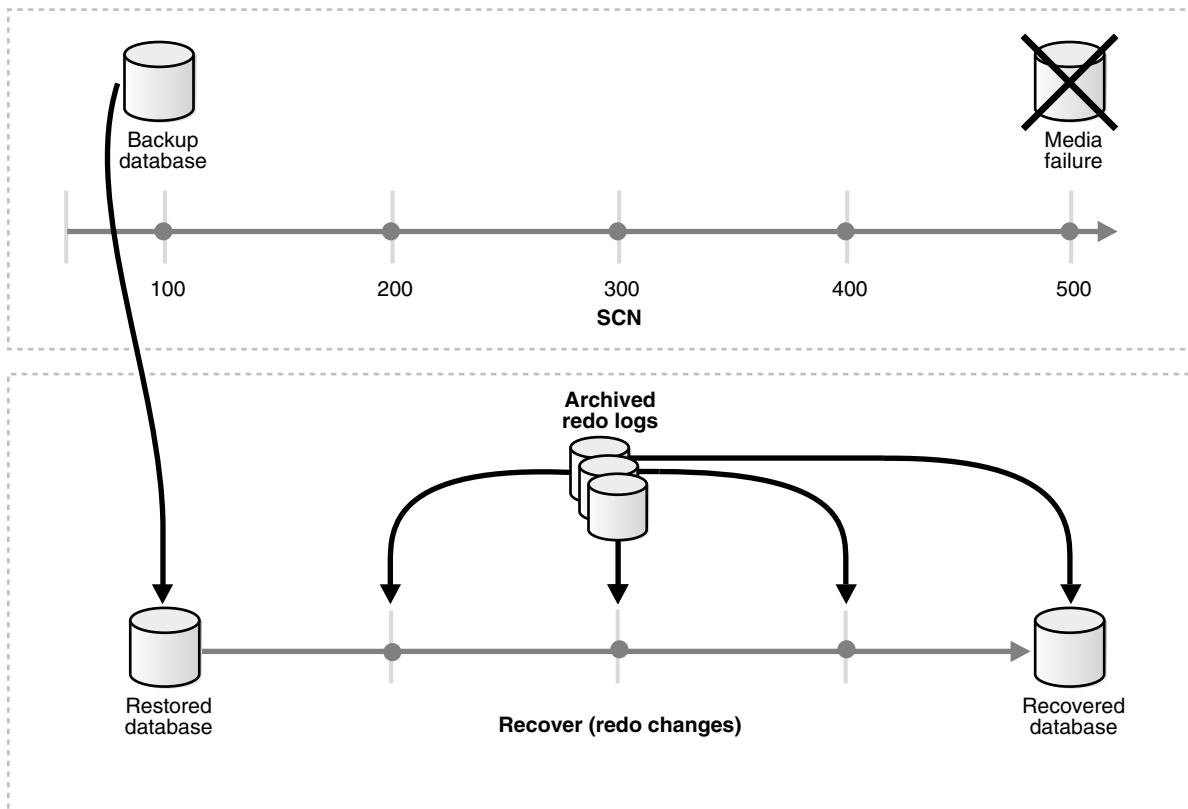
Unlike media recovery, Oracle Database performs crash recovery and instance recovery automatically after an instance failure. Crash and instance recovery recover a database to its transaction-consistent state just before instance failure. Crash recovery is the recovery of a database in a single-instance configuration or an Oracle Real Application Clusters configuration after all instances have crashed. In contrast, instance recovery is the recovery of one or more failed instances by a live instance in an Oracle Real Application Clusters configuration.

This section includes the following topics:

- Datafile Media Recovery
- Block Media Recovery
- Complete Recovery
- Database Point-in-Time Recovery
- RMAN and User-Managed Recovery

Datafile Media Recovery

Datafile media recovery is used to recover from a lost or damaged current datafile or control file. It is also used to recover changes that were lost when a tablespace went offline without the OFFLINE NORMAL option. Both datafile media recovery and instance recovery must repair database integrity. However, these types of recovery differ with respect to their additional features. Media recovery has the following characteristics:

- Applies changes to restored backups of damaged datafiles.
- Can use archived logs as well as online logs.
- Requires explicit invocation by a user.
- Does not detect media failure (that is, the need to restore a backup) automatically. After a backup has been restored, however, detection of the need to recover it through media recovery *is* automatic.
- Has a recovery time governed solely by user policy (for example, frequency of backups, parallel recovery parameters, number of database transactions since the last backup) rather than by Oracle Database internal mechanisms.

The database cannot be opened if any of the online datafiles needs media recovery, nor can a datafile that needs media recovery be brought online until media recovery is complete. The following scenarios necessitate media recovery:

- You restore a backup of a datafile.
- You restore a backup control file (even if all datafiles are current).
- A datafile is taken offline (either by you or automatically by Oracle Database) without the OFFLINE NORMAL option.

Unless the database is not open by any instance, datafile media recovery can only operate on offline datafiles.

Block Media Recovery

Block media recovery is a technique for restoring and recovering individual data blocks while all database files remain online and available. If only a few blocks are corrupt, then block media recovery may be preferable to datafile recovery.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* to learn how to perform block media recovery

Complete Recovery

Complete recovery applies *all* of the redo changes contained in the archived and online logs to a backup. Typically, you perform complete media recovery after a media failure damages datafiles or the control file. You can perform complete recovery on a database, tablespace, or datafile.

If you are performing complete recovery on the whole database, then you must:

- Mount the database
- Ensure that all datafiles you want to recover are online
- Restore a backup of the whole database
- Run the RMAN `RECOVER DATABASE` command, which will apply the correct redo logs and incremental backups.

If you are performing complete recovery on a tablespace or datafile, then you must:

- Take the tablespace or datafile to be recovered offline if the database is open
- Restore a backup of the datafiles you want to recover
- Apply online or archived redo logs, or a combination of the two

Database Point-in-Time Recovery

Database point-in-time recovery, which is also called incomplete recovery, results in a noncurrent version of the database. In other words, you do not apply all of the redo records generated after the restored backup. Typically, you perform point-in-time recovery of the whole database in the following situations:

- Media failure destroys some or all of the online redo logs.
- A user error causes data loss, for example, a user inadvertently drops a table.
- You cannot perform complete recovery because an archived redo log is missing.
- Complete recovery is possible with a backup control file. If using RMAN it is seamless and automatic.

To perform database point-in-time recovery, you must restore all datafiles from backups created prior to the time to which you want to recover and then open the database with the `RESETLOGS` option when recovery completes. The `RESETLOGS` operation creates a new incarnation of the database—in other words, a database with a new stream of log sequence numbers starting with log sequence 1.

Before using the `OPEN RESETLOGS` command to open the database in read/write mode after an incomplete recovery, it is a good idea to first open the database in read-only mode, and inspect the data to make sure that the database was recovered to the correct point. If the recovery was done to the wrong point, then it is easier to re-run the recovery if no `OPEN RESETLOGS` has been done. If you open the database read-only and discover that not enough recovery was done, then just run the recovery again to the desired time. If you discover that too much recovery was done, then you must restore the database again and re-run the recovery.

> **Note:** Flashback Database is an alternative to database point-in-time recovery.

See Also: "Oracle Flashback Database" on page 15-10

Tablespace Point-in-Time Recovery The tablespace point-in-time recovery (TSPITR) feature lets you recover one or more tablespaces to a point in time older than the rest of the database. TSPITR is most useful when you want to:

- Recover from an erroneous drop or truncate table operation

- Recover a table that has become logically corrupted

- Recover from an incorrect batch job or other DML statement that has affected only a subset of the database

- Recover one independent schema to a point different from the rest of a physical database (in cases where there are multiple independent schemas in separate tablespaces of one physical database)

- Recover a tablespace on a very large database (VLDB) rather than restore the whole database from a backup and perform a complete database roll-forward

TSPITR has the following limitations:

- You cannot use it on the SYSTEM tablespace, an UNDO tablespace, or any tablespace that contains rollback segments.

- Tablespaces that contain interdependent data must be recovered together. For example, if two tables are in separate tablespaces and have a foreign key relationship, then both tablespaces must be recovered at the same time; you cannot recover just one of them. Oracle Database can enforce this limitation when it detects data relationships that have been explicitly declared with database constraints. There could be other data relationships that are not declared with database constraints. Oracle Database cannot detect these relationships, so the DBA must be careful to always restore a consistent set of tablespaces.

See Also: *Oracle Database Backup and Recovery User's Guide* and *Oracle Database Backup and Recovery Reference* for more information on TSPITR

RMAN and User-Managed Recovery

You have a choice between two basic techniques for recovering physical files. You can:

- Use the RMAN utility to restore and recover the database

- Restore backups by means of operating system utilities, and then recover them by running the SQL*Plus RECOVER command

Whichever method you choose, you can recover a database, tablespace, or datafile. Before performing media recovery, you need to determine which datafiles to recover. Often you can use the fixed view V$RECOVER_FILE. This view lists all files that require recovery and explains the error that necessitates recovery.

See Also: *Oracle Database Backup and Recovery Reference* for more about using V$ views in a recovery scenario

RMAN Restore and Recovery The basic RMAN recovery commands are RESTORE and RECOVER. Use RESTORE to restore datafiles from backup sets or from image copies on disk, either to their current location or to a new location. You can also restore backup sets containing archived redo logs, but this is usually unnecessary, because RMAN automatically restores the archived logs that are needed for recovery and deletes them

after the recovery is finished. Use the RMAN RECOVER command to perform media recovery and apply archived logs or incremental backups.

> **See Also:** *Oracle Database Backup and Recovery Reference* for details about how to restore and recover using RMAN

User-Managed Restore and Recovery If you do not use RMAN, then you can restore backups with operating system utilities and then run the SQL*Plus RECOVER command to recover the database.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for details about how to restore and recover with operating system utilities and SQL*Plus

16

Business Intelligence

This chapter describes some of the basic ideas in business intelligence.

This chapter contains the following topics:

- Introduction to Data Warehousing and Business Intelligence
- Overview of Extraction, Transformation, and Loading (ETL)
- Overview of Materialized Views for Data Warehouses
- Overview of Bitmap Indexes in Data Warehousing
- Overview of Parallel Execution
- Overview of Analytic SQL
- Overview of OLAP Capabilities
- Overview of Data Mining

Introduction to Data Warehousing and Business Intelligence

A data warehouse is a relational database that is designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources.

In addition to a relational database, a data warehouse environment includes an extraction, transportation, transformation, and loading (ETL) solution, an online analytical processing (OLAP) engine, Oracle Warehouse Builder, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users.

This section includes the following topics:

- Characteristics of Data Warehousing
- Differences Between Data Warehouse and OLTP Systems
- Data Warehouse Architecture

Characteristics of Data Warehousing

Data warehouses all share the following basic characteristics:

- Subject Oriented
- Integrated

- Nonvolatile

- Time Variant

Subject Oriented

Data warehouses are designed to help you analyze data. For example, to learn more about your company's sales data, you can build a warehouse that concentrates on sales. Using this warehouse, you can answer questions like "Who was our best customer for this item last year?" This ability to define a data warehouse by subject matter, sales in this case, makes the data warehouse subject oriented.

Integrated

Integration is closely related to subject orientation. Data warehouses must put data from disparate sources into a consistent format. They must resolve such problems as naming conflicts and inconsistencies among units of measure. When they achieve this, they are said to be integrated.

Nonvolatile

Nonvolatile means that, once entered into the warehouse, data should not change. This is logical because the purpose of a warehouse is to enable you to analyze what has occurred.

Time Variant

In order to discover trends in business, analysts need large amounts of data. This is very much in contrast to online transaction processing (OLTP) systems, where performance requirements demand that historical data be moved to an archive. A data warehouse's focus on change over time is what is meant by the term time variant.

Typically, data flows from one or more online transaction processing (OLTP) databases into a data warehouse on a monthly, weekly, or daily basis. The data is normally processed in a **staging file** before being added to the data warehouse. Data warehouses commonly range in size from tens of gigabytes to a few terabytes. Usually, the vast majority of the data is stored in a few very large fact tables.

Differences Between Data Warehouse and OLTP Systems

Data warehouses and OLTP systems have very different requirements. Here are some examples of differences between typical data warehouses and OLTP systems:

- Workload

- Data Modifications

- Schema Design

- Typical Operations

- Historical Data

Workload

Data warehouses are designed to accommodate ad hoc queries. You might not know the workload of your data warehouse in advance, so a data warehouse should be optimized to perform well for a wide variety of possible query operations.

OLTP systems support only predefined operations. Your applications might be specifically tuned or designed to support only these operations.

Data Modifications

A data warehouse is updated on a regular basis by the ETL process (run nightly or weekly) using bulk data modification techniques. The end users of a data warehouse do not directly update the data warehouse.

In OLTP systems, end users routinely issue individual data modification statements to the database. The OLTP database is always up to date, and reflects the current state of each business transaction.

Schema Design

Data warehouses often use denormalized or partially denormalized schemas (such as a star schema) to optimize query performance.

OLTP systems often use fully normalized schemas to optimize update/insert/delete performance, and to guarantee data consistency.

Typical Operations

A typical data warehouse query scans thousands or millions of rows.For example, "Find the total sales for all customers last month."

A typical OLTP operation accesses only a handful of records. For example, "Retrieve the current order for this customer."

Historical Data

Data warehouses usually store many months or years of data. This is to support historical analysis.

OLTP systems usually store data from only a few weeks or months. The OLTP system stores only historical data as needed to successfully meet the requirements of the current transaction.

Data Warehouse Architecture

Data warehouses and their architectures vary depending upon the specifics of an organization's situation. Three common architectures are:

- Data Warehouse Architecture (Basic)

- Data Warehouse Architecture (with a Staging Area)

- Data Warehouse Architecture (with a Staging Area and Data Marts)

Data Warehouse Architecture (Basic)

Figure 16–1 shows a simple architecture for a data warehouse. End users directly access data derived from several source systems through the data warehouse.

Figure 16–1   Architecture of a Data Warehouse

Data Sources   **Warehouse**   **Users**
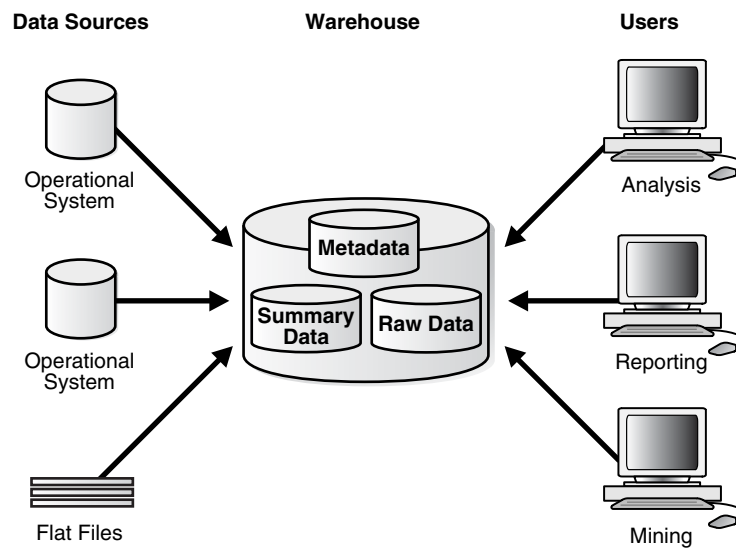
In Figure 16–1, the metadata and raw data of a traditional OLTP system is present, as is an additional type of data, summary data. Summaries are very valuable in data warehouses because they pre-compute long operations in advance. For example, a typical data warehouse query is to retrieve something like August sales.

Summaries in Oracle Database are called materialized views.

Data Warehouse Architecture (with a Staging Area)

Figure 16–1, you need to clean and process your operational data before putting it into the warehouse. You can do this programmatically, although most data warehouses use a staging area instead. A staging area simplifies building summaries and general warehouse management. Figure 16–2 illustrates this typical architecture.

Figure 16–2   Architecture of a Data Warehouse with a Staging Area

Data Sources   **Staging Area**   **Warehouse**   **Users**
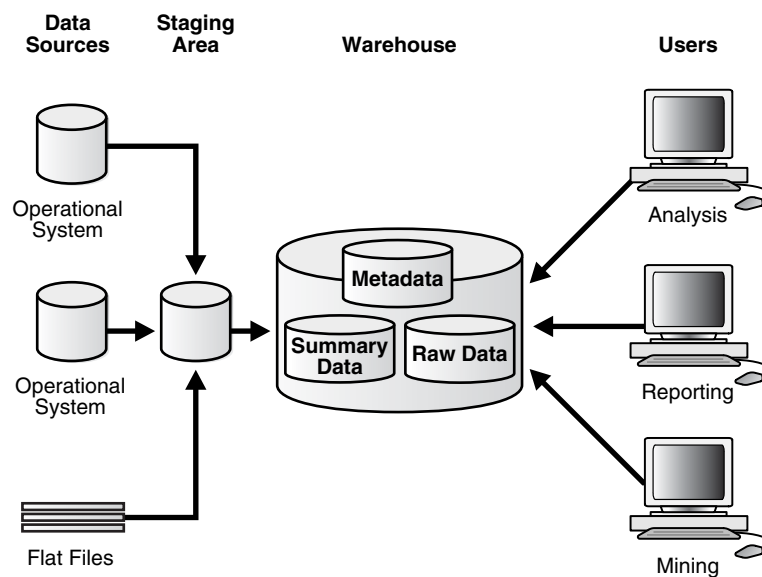
Data Warehouse Architecture (with a Staging Area and Data Marts)

Although the architecture in Figure 16–2 is quite common, you might want to customize your warehouse's architecture for different groups within your organization.

Do this by adding data marts, which are systems designed for a particular line of business. Figure 16–3 illustrates an example where purchasing, sales, and inventories are separated. In this example, a financial analyst might want to analyze historical data for purchases and sales.

Figure 16–3   Architecture of a Data Warehouse with a Staging Area and Data Marts
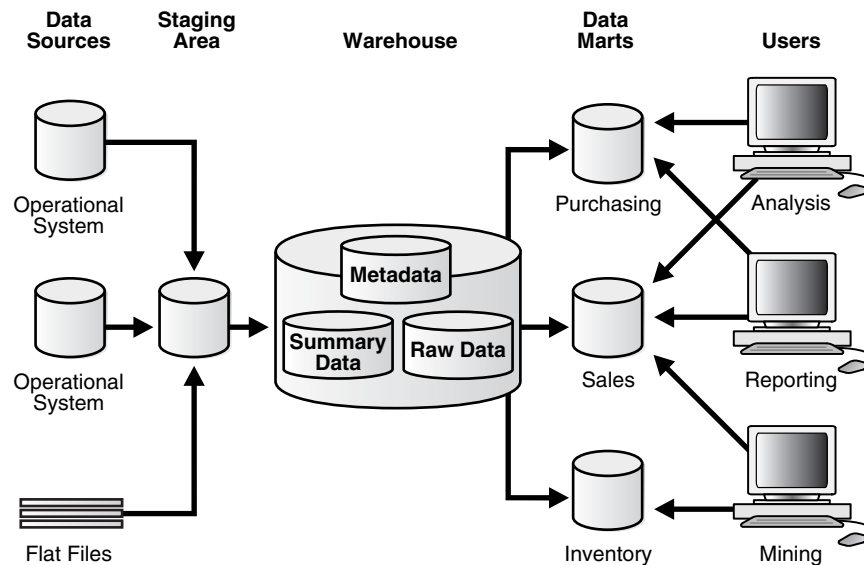
See Also: *Oracle Database Data Warehousing Guide*

Overview of Extraction, Transformation, and Loading (ETL)

You need to load your data warehouse regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational systems needs to be extracted and copied into the warehouse. The process of extracting data from source systems and bringing it into the data warehouse is commonly called **ETL**, which stands for extraction, transformation, and loading. The acronym ETL is perhaps too simplistic, because it omits the transportation phase and implies that each of the other phases of the process is distinct. The entire process, including data loading, is referred to as ETL. You should understand that ETL refers to a broad process, and not three well-defined steps.

The methodology and tasks of ETL have been well known for many years, and are not necessarily unique to data warehouse environments: a wide variety of proprietary applications and database systems are the IT backbone of any enterprise. Data has to be shared between applications or systems, trying to integrate them, giving at least two applications the same picture of the world. This data sharing was mostly addressed by mechanisms similar to what is now called ETL.

Data warehouse environments face the same challenge with the additional burden that they not only have to exchange but to integrate, rearrange and consolidate data over many systems, thereby providing a new unified information base for business intelligence. Additionally, the data volume in data warehouse environments tends to be very large.

What happens during the ETL process? During extraction, the desired data is identified and extracted from many different sources, including database systems and applications. Very often, it is not possible to identify the specific subset of interest, therefore more data than necessary has to be extracted, so the identification of the relevant data will be done at a later point in time. Depending on the source system's capabilities (for example, operating system resources), some transformations may take place during this extraction process. The size of the extracted data varies from hundreds of kilobytes up to gigabytes, depending on the source system and the business situation. The same is true for the time delta between two (logically) identical extractions: the time span may vary between days/hours and minutes to near real-time. Web server log files for example can easily become hundreds of megabytes in a very short period of time.

After extracting data, it has to be physically transported to the target system or an intermediate system for further processing. Depending on the chosen way of transportation, some transformations can be done during this process, too. For example, a SQL statement which directly accesses a remote target through a gateway can concatenate two columns as part of the SELECT statement.

If any errors occur during loading, an error is logged and the operation can continue.

This section includes the following topics:

- Transportable Tablespaces
- Table Functions
- External Tables
- Table Compression
- Change Data Capture

Transportable Tablespaces

Transportable tablespaces are the fastest way for moving large volumes of data between two Oracle databases. You can transport tablespaces between different computer architectures and operating systems.

Previously, the most scalable data transportation mechanisms relied on moving flat files containing raw data. These mechanisms required that data be unloaded or exported into files from the source database. Then, after transportation, these files were loaded or imported into the target database. Transportable tablespaces entirely bypass the unload and reload steps.

Using transportable tablespaces, Oracle Database data files (containing table data, indexes, and almost every other Oracle database object) can be directly transported from one database to another. Furthermore, like import and export, transportable tablespaces provide a mechanism for transporting metadata in addition to transporting data.

The most common applications of transportable tablespaces in data warehouses are in moving data from a staging database to a data warehouse, or in moving data from a data warehouse to a data mart.

Table Functions

Table functions provide the support for pipelined and parallel execution of transformations implemented in PL/SQL, C, or Java. Scenarios as mentioned earlier can be done without requiring the use of intermediate staging tables, which interrupt the data flow through various transformations steps.

A table function is defined as a function that can produce a set of rows as output. Additionally, table functions can take a set of rows as input. Table functions extend database functionality by allowing:

- Multiple rows to be returned from a function

- Results of SQL subqueries (that select multiple rows) to be passed directly to functions

- Functions take cursors as input

- Functions can be parallelized

- Returning result sets incrementally for further processing as soon as they are created. This is called incremental pipelining

Table functions can be defined in PL/SQL using a native PL/SQL interface, or in Java or C using the Oracle Data Cartridge Interface (ODCI).

External Tables

External tables let you use external data as a virtual table that can be queried and joined directly and in parallel without requiring the external data to be first loaded in the database. You can then use SQL, PL/SQL, and Java to access the external data.

External tables enable the pipelining of the loading phase with the transformation phase. The transformation process can be merged with the loading process without any interruption of the data streaming. It is no longer necessary to stage the data inside the database for further processing inside the database, such as comparison or transformation. For example, the conversion functionality of a conventional load can be used for a direct-path `INSERT AS SELECT` statement in conjunction with the `SELECT` from an external table. Figure 16–4 illustrates a typical example of pipelining.

Figure 16–4   Pipelined Data Transformation
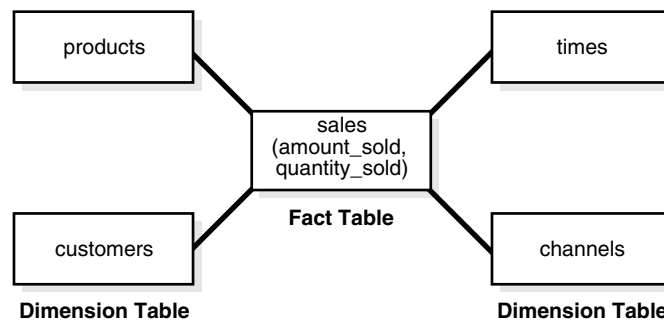
The main difference between external tables and regular tables is that externally organized tables are read-only. No DML operations (`UPDATE`/`INSERT`/`DELETE`) are possible and no indexes can be created on them.

External tables are a complement to SQL*Loader and are especially useful for environments where the complete external source has to be joined with existing database objects and transformed in a complex manner, or where the external data volume is large and used only once. SQL*Loader, on the other hand, might still be the better choice for loading of data where additional indexing of the staging table is necessary. This is true for operations where the data is used in independent complex transformations or the data is only partially used in further processing.

Table Compression

You can save disk space by compressing heap-organized tables. A typical type of heap-organized table you should consider for table compression is partitioned tables.

To reduce disk use and memory use (specifically, the buffer cache), you can store tables and partitioned tables in a compressed format inside the database. This often leads to a better scaleup for read-only operations. Table compression can also speed up query execution. There is, however, a slight cost in CPU overhead.

Table compression should be used with highly redundant data, such as tables with many foreign keys. You should avoid compressing tables with much update or other DML activity. Although compressed tables or partitions are updatable, there is some overhead in updating these tables, and high update activity may work against compression by causing some space to be wasted.

> **See Also:** "Table Compression" on page 5-7

Change Data Capture

Change Data Capture efficiently identifies and captures data that has been added to, updated, or removed from Oracle Database relational tables, and makes the change data available for use by applications.

Oftentimes, data warehousing involves the extraction and transportation of relational data from one or more source databases into the data warehouse for analysis. Change Data Capture quickly identifies and processes only the data that has changed, not entire tables, and makes the change data available for further use.

Change Data Capture does not depend on intermediate flat files to stage the data outside of the relational database. It captures the change data resulting from INSERT, UPDATE, and DELETE operations made to user tables. The change data is then stored in a database object called a change table, and the change data is made available to applications in a controlled way.

> **See Also:** *Oracle Database Data Warehousing Guide*

Overview of Materialized Views for Data Warehouses

One technique employed in data warehouses to improve performance is the creation of summaries. Summaries are special kinds of aggregate views that improve query execution times by precalculating expensive joins and aggregation operations prior to execution and storing the results in a table in the database. For example, you can create a table to contain the sums of sales by region and by product.

The summaries or aggregates that are referred to in this book and in literature on data warehousing are created in Oracle Database using a schema object called a **materialized view**. Materialized views can perform a number of roles, such as improving query performance or providing replicated data.

Previously, organizations using summaries spent a significant amount of time and effort creating summaries manually, identifying which summaries to create, indexing the summaries, updating them, and advising their users on which ones to use. Summary management eased the workload of the database administrator and meant that the user no longer needed to be aware of the summaries that had been defined. The database administrator creates one or more materialized views, which are the equivalent of a summary. The end user queries the tables and views at the detail data level.

The query rewrite mechanism in Oracle Database automatically rewrites the SQL query to use the summary tables. This mechanism reduces response time for returning results from the query. Materialized views within the data warehouse are transparent to the end user or to the database application.

Although materialized views are usually accessed through the query rewrite mechanism, an end user or database application can construct queries that directly access the summaries. However, serious consideration should be given to whether users should be allowed to do this because any change to the summaries will affect the queries that reference them.

To help you select from among the many possible materialized views in your schema, Oracle Database provides a collection of materialized view analysis and advisor functions and procedures in the DBMS_ADVISOR package. Collectively, these functions are called the SQL Access Advisor, and they are callable from any PL/SQL program. The SQL Access Advisor recommends materialized views from a hypothetical or user-defined workload or one obtained from the SQL cache. You can run the SQL Access Advisor from Oracle Enterprise Manager or by invoking the DBMS_ADVISOR package.

> **See Also:** *Oracle Database Performance Tuning Guide* for information about materialized views and the SQL Access Advisor

Overview of Bitmap Indexes in Data Warehousing

Bitmap indexes are widely used in data warehousing environments. The environments typically have large amounts of data and ad hoc queries, but a low level of concurrent DML transactions. For such applications, bitmap indexing provides:

- Reduced response time for large classes of ad hoc queries

- Reduced storage requirements compared to other indexing techniques

- Dramatic performance gains even on hardware with a relatively small number of CPUs or a small amount of memory

- Efficient maintenance during parallel DML and loads

Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of space because the indexes can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

An index provides pointers to the rows in a table that contain a given key value. A regular index stores a list of rowids for each key corresponding to the rows with that key value. In a bitmap index, a bitmap for each key value replaces a list of rowids.

Each bit in the bitmap corresponds to a possible rowid, and if the bit is set, it means that the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid, so that the bitmap index provides the same functionality as a regular index. If the number of different key values is small, bitmap indexes save space.

Bitmap indexes are most effective for queries that contain multiple conditions in the WHERE clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically. A good candidate for a bitmap index would be a gender column due to the low number of possible values.

Parallel query and parallel DML work with bitmap indexes as they do with traditional indexes. Bitmap indexing also supports parallel create indexes and concatenated indexes.

> **See Also:** *Oracle Database Data Warehousing Guide*

Overview of Parallel Execution

When Oracle Database runs SQL statements in parallel, multiple processes work together simultaneously to run a single SQL statement. By dividing the work necessary to run a statement among multiple processes, Oracle Database can run the statement more quickly than if only a single process ran it. This is called **parallel execution** or **parallel processing**.

Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with decision support systems (DSS) and data warehouses. Symmetric multiprocessing (SMP), clustered systems, and large-scale cluster systems gain the largest performance benefits from parallel execution because statement processing can be split up among many CPUs on a single Oracle Database system. You can also implement parallel execution on certain types of online transaction processing (OLTP) and hybrid systems.

Parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time. An example of this is when 12 processes handle 12 different months in a year instead of one process handling all 12 months by itself. The improvement in performance can be quite high.

Parallel execution helps systems scale in performance by making optimal use of hardware resources. If your system's CPUs and disk controllers are already heavily loaded, you need to alleviate the system's load or increase these hardware resources before using parallel execution to improve performance.

In Oracle RAC environments, parallel execution is controlled by the service placement of a particular service. Specifically, parallel processes run on the nodes on which you have configured the service. The default behavior is for Oracle Database to run the parallel process only on the instance that offers the service that you used to connect to the database. This does not affect other parallel operations such as parallel recovery or the processing of GV$queries.

Some tasks are not well-suited for parallel execution. For example, many OLTP operations are relatively fast, completing in mere seconds or fractions of seconds, and the overhead of utilizing parallel execution would be large, relative to the overall execution time.

> **See Also:** *Oracle Database Data Warehousing Guide* for specific information on tuning your parameter files and database to take full advantage of parallel execution and the *Oracle Real Application Clusters Administration and Deployment Guide* for considerations regarding parallel execution in Oracle RAC environments

How Parallel Execution Works

When parallel execution is not used, a single server process performs all necessary processing for the sequential execution of a SQL statement. For example, to perform a full table scan (such as SELECT * FROM emp), one process performs the entire operation, as illustrated in Figure 16–5.

Figure 16–5   Serial Full Table Scan

Serial Process

```
SELECT *
   FROM EMP;
```
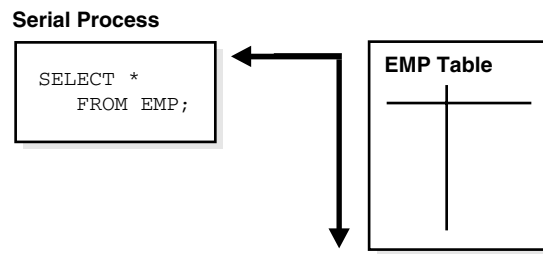
EMP Table

Figure 16–6 illustrates several parallel execution servers performing a scan of the table `emp`. The table is divided dynamically (**dynamic partitioning**) into load units called granules and each granule is read by a single parallel execution server. The granules are generated by the coordinator. Each granule is a range of physical blocks of the table `emp`. The mapping of granules to execution servers is not static, but is determined at execution time. When an execution server finishes reading the rows of the table `emp` corresponding to a granule, it gets another granule from the coordinator if there are any granules remaining. This continues until all granules are exhausted, in other words, until the entire table `emp` has been read. The parallel execution servers send results back to the parallel execution coordinator, which assembles the pieces into the desired full table scan.

Figure 16–6   Parallel Full Table Scan

**Parallel Execution
Coordinator**

**Parallel Execution
Server**

```
SELECT *
   FROM EMP;
```

EMP Table
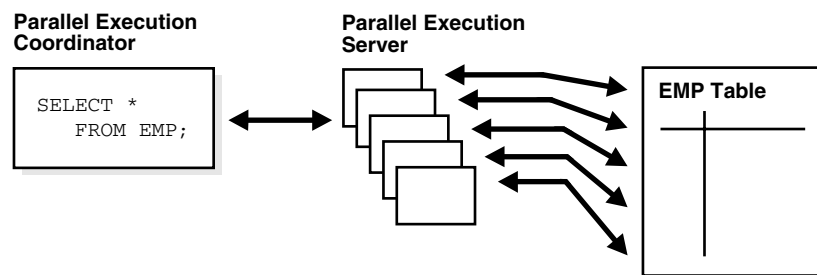
Given a query plan for a SQL query, the parallel execution coordinator breaks down each operator in a SQL query into parallel pieces, runs them in the right order as specified in the query, and then integrates the partial results produced by the parallel execution servers executing the operators. The number of parallel execution servers assigned to a single operation is the degree of parallelism (DOP) for an operation. Multiple operations within the same SQL statement all have the same degree of parallelism.

> **See Also:**   *Oracle Database Data Warehousing Guide* for information on granules as well as how Oracle Database divides work and handles DOP in multiuser environments

Overview of Analytic SQL

Oracle has introduced many SQL operations for performing analytic operations in the database. These operations include ranking, moving averages, cumulative sums, ratio-to-reports, and period-over-period comparisons. Although some of these calculations were previously possible using SQL, this syntax offers much better performance.

This section discusses:

- [SQL for Aggregation](#)
- [SQL for Analysis](#)
- [SQL for Modeling](#)

SQL for Aggregation

Aggregation is a fundamental part of data warehousing. To improve aggregation performance in your warehouse, Oracle Database provides extensions to the GROUP BY clause to make querying and reporting easier and faster. Some of these extensions enable you to:

- Aggregate at increasing levels of aggregation, from the most detailed up to a grand total

- Calculate all possible combinations of aggregations with a single statement

- Generate the information needed in cross-tabulation reports with a single query

These extension let you specify exactly the groupings of interest in the GROUP BY clause. This allows efficient analysis across multiple dimensions without performing a CUBE operation. Computing a full cube creates a heavy processing load, so replacing cubes with grouping sets can significantly increase performance. CUBE, ROLLUP, and grouping sets produce a single result set that is equivalent to a UNION ALL of differently grouped rows.

To enhance performance, these extensions can be parallelized: multiple processes can simultaneously run all of these statements. These capabilities make aggregate calculations more efficient, thereby enhancing database performance, and scalability.

One of the key concepts in decision support systems is multidimensional analysis: examining the enterprise from all necessary combinations of dimensions. The term **dimension** is used to mean any category used in specifying questions. Among the most commonly specified dimensions are time, geography, product, department, and distribution channel, but the potential dimensions are as endless as the varieties of enterprise activity. The events or entities associated with a particular set of dimension values are usually referred to as **facts**. The facts might be sales in units or local currency, profits, customer counts, production volumes, or anything else worth tracking.

Here are some examples of multidimensional requests:

- Show total sales across all products at increasing aggregation levels for a geography dimension, from state to country to region, for 1999 and 2000.

- Create a cross-tabular analysis of our operations showing expenses by territory in South America for 1999 and 2000. Include all possible subtotals.

- List the top 10 sales representatives in Asia according to 2000 sales revenue for automotive products, and rank their commissions.

All these requests involve multiple dimensions. Many multidimensional questions require aggregated data and comparisons of data sets, often across time, geography or budgets.

> **See Also:** *Oracle Database Data Warehousing Guide*

SQL for Analysis

Oracle has advanced SQL analytical processing capabilities using a family of analytic SQL functions. These analytic functions enable you to calculate:

- Rankings and percentiles

- Moving window calculations

- Lag/lead analysis

- First/last analysis

- Linear regression statistics

Ranking functions include cumulative distributions, percent rank, and N-tiles. Moving window calculations allow you to find moving and cumulative aggregations, such as sums and averages. Lag/lead analysis enables direct inter-row references so you can calculate period-to-period changes. First/last analysis enables you to find the first or last value in an ordered group.

Other features include the `CASE` expression. `CASE` expressions provide if-then logic useful in many situations.

To enhance performance, analytic functions can be parallelized: multiple processes can simultaneously run all of these statements. These capabilities make calculations easier and more efficient, thereby enhancing database performance, scalability, and simplicity.

See Also: *Oracle Database Data Warehousing Guide*

SQL for Modeling

The Oracle `MODEL` clause brings a new level of power and flexibility to SQL calculations. With the `MODEL` clause, you can create a multidimensional array from query results and then apply formulas to this array to calculate new values. The formulas can range from basic arithmetic to simultaneous equations using recursion. For some applications, the `MODEL` clause can replace PC-based spreadsheets. Models in SQL leverage the Oracle Database strengths in scalability, manageability, collaboration, and security. The core query engine can work with unlimited quantities of data. By defining and executing models within the database, users avoid transferring large datasets to and from separate modeling environments. Models can be shared easily across workgroups, ensuring that calculations are consistent for all applications. Just as models can be shared, access can also be controlled precisely with the Oracle Database security features. With its rich functionality, the `MODEL` clause can enhance all types of applications.

See Also: *Oracle Database Data Warehousing Guide*

Overview of OLAP Capabilities

Oracle online analytical processing (OLAP) adds power to your SQL applications by providing extensive analytic content and fast query response times. A SQL query interface enables any application to query cubes and dimensions without any knowledge of OLAP.

The OLAP option automatically generates a set of relational views on cubes, dimensions, and hierarchies. SQL applications query these views to display the information-rich contents of these objects to analysts and decision makers. You can also create custom views that comply with the structure expected by your applications, using the system-generated views like base tables.

Analysts can choose any SQL query and analysis tool for selecting, viewing, and analyzing the data. You can use your favorite tool or application, or use one of the

tools supplied with Oracle Database, such as Oracle Application Express and Business Intelligence Publisher.

> **See Also:** *Oracle OLAP User's Guide*

This section includes the following topics:

- Full Integration of Multidimensional Technology
- Ease of Application Development
- Ease of Administration
- Security
- Unmatched Performance and Scalability
- Reduced Costs

Full Integration of Multidimensional Technology

By integrating multidimensional objects and analytics into the database, Oracle provides the best of both worlds: the power of multidimensional analysis along with the reliability, availability, security, and scalability of Oracle Database.

Oracle OLAP is fully integrated into Oracle Database. At a technical level, this means:

- The OLAP engine runs within the kernel of Oracle Database
- Dimensional objects are stored in Oracle Database in their native multidimensional format
- Cubes and other dimensional objects are first class data objects represented in the Oracle data dictionary
- Data security is administered in the standard way, by granting and revoking privileges to Oracle Database users and roles
- Applications can query dimensional objects using SQL

The benefits to your organization are significant. Oracle OLAP offers the power of simplicity. One database, standard administration and security, standard interfaces and development tools.

Ease of Application Development

Oracle OLAP makes it easy to enrich your database and your applications with interesting analytic content. Native SQL access to Oracle multidimensional objects and calculations greatly eases the task of developing dashboards, reports, business intelligence, and analytical applications of any type compared to systems that offer proprietary interfaces. Moreover, SQL access means that the power of Oracle OLAP analytics can be used by any database application, not just by the traditional limited collection of OLAP applications.

Ease of Administration

Because Oracle OLAP is completely embedded in Oracle Database, there is no administration learning curve as is typically associated with stand-alone OLAP servers. You can leverage your existing DBA staff, rather than invest in specialized administration skills.

One major administrative advantage of Oracle's embedded OLAP technology is automated cube maintenance. With stand-alone OLAP servers, the burden of refreshing the cube is left entirely to the administrator. This can be a complex and potentially error-prone job. The administrator must create procedures to extract the changed data from the relational source, move the data from the source system to the system running the stand-alone OLAP server, load and rebuild the cube. The administrator must take responsibility for the security of the changed values during this process, as well.

With Oracle OLAP, in contrast, cube refresh is handled entirely by Oracle Database. The database tracks the staleness of the dimensional objects, automatically keeps track of the deltas in the source tables, and automatically applies only the changed values during the refresh process. The administrator simply schedules the refresh at appropriate intervals, and Oracle Database takes care of everything else.

Security

With Oracle OLAP, standard Oracle Database security features are used to secure your multidimensional data.

In contrast, with a stand-alone OLAP server, administrators must manage security twice: once on the relational source system and again on the OLAP server system. Additionally, they must manage the security of data in transit from the relational system to the stand-alone OLAP system.

Unmatched Performance and Scalability

Business intelligence and analytical applications are dominated by actions such as drilling up and down hierarchies and comparing aggregate values such as period-over-period, share of parent, projections onto future time periods, and a myriad of similar calculations. Often these actions are essentially random across the entire space of potential hierarchical aggregations. Because Oracle OLAP pre-computes or efficiently computes on the fly all aggregates in the defined multidimensional space, it delivers unmatched performance for typical business intelligence applications.

Oracle OLAP queries take advantage of Oracle shared cursors, dramatically reducing memory requirements and increasing performance.

When Oracle Database is installed with Oracle Real Application Clusters (Oracle RAC), OLAP applications receive the same benefits in performance, scalability, failover, and load balancing as any other application.

Reduced Costs

All these features add up to reduced costs. Administrative costs are reduced because existing personnel skills can be leveraged. Moreover, Oracle Database can manage the refresh of dimensional objects, a complex task left to administrators in other systems. Standard security reduces administration costs as well. Application development costs are reduced because the availability of a large pool of application developers who are SQL knowledgeable, and a large collection of SQL-based development tools means applications can be developed and deployed more quickly. Any SQL-based development tool can take advantage of Oracle OLAP. Hardware costs are reduced by Oracle OLAP's efficient management of aggregations, use of shared cursors, and Oracle RAC, which enables highly scalable systems to be built from low-cost commodity components.

Overview of Data Mining

Oracle Data Mining embeds data mining in the Oracle Database. The data never leaves the database — data preparation, model building, and model scoring are all performed within the database. Since the data never leaves the database, there are significant advantages in scalability, manageability, and user access. Thus, the Oracle Database provides an infrastructure for application developers to integrate data mining seamlessly with database applications. Data mining is often used in applications such as call centers, ATMs, ERM, and business planning.

As of Oracle Database 11*g*, Oracle Data Mining models are implemented as data dictionary objects in the SYS schema. A set of new data dictionary views present mining models and their properties. New system and object privileges control access to mining model objects.

Support of Generalized Linear Models (GLM) is new for Oracle Data Mining 11*g*. Oracle Data Mining supports two forms of GLM, one for classification and one for regression:

- Binary Logistic Regression, used for classification, predicts the probability for each row of scoring data. The dependent variable (target) is binary and categorical. For example, demographic attributes might be used to predict whether customer response to a promotion is low or high.

- Multivariate Linear Regression, used for regression, predicts the best estimate within a continuum for each row of scoring data. For example, demographic attributes such as age bracket, income level, gender, and town of residence might be used to predict sales per customer.

Oracle Data Mining GLM can handle many hundreds or thousands of input attributes, unlike traditional implementations that typically support 30 or fewer input attributes.

Data mining activities such as model building, testing, and scoring are accomplished through a PL/SQL API, a Java API, and SQL Data Mining functions. The Java API is compliant with the data mining standard JSR 73. The Java API and the PL/SQL API are fully interoperable.

Optionally, Oracle Data Mining can automatically perform all algorithm-required data preparation, such as binning, normalization, and outlier treatment. Additionally, user-specified data transformations can be integrated with the algorithm-specific data preparation to simplify testing and scoring; models like this are supermodel.

The SQL Data Mining functions are SQL language operators for the deployment of data mining models. The Data Mining functions support the scoring of classification, regression, clustering, and feature extraction models. Within the context of standard SQL statements, pre-created models can be applied to new data and the results returned for further processing.

Predictive Analytics is a technology that captures data mining processes in simple routines. Sometimes called "one-click data mining," predictive analytics simplify and automate the data mining process. The procedure returns the results of analytic processing. The models and other intermediate objects are not preserved. The DBMS_ PREDICTIVE_ANALYTICS PL/SQL package implements Predictive Analytics with the following procedures:

- EXPLAIN - Ranks attributes in order of strongest relationships with a target attribute.

- PREDICT - Predicts the value of a target attribute.

- PROFILE - Creates rules that identify the records that have the same target value.

Oracle Data Mining supports the following algorithms (Generalized Linear Models are new for Oracle Database 11*g*):

- For classification, Naive Bayes, Decision Tree, Generalized Linear Models (Binary Logistic Regression), and Support Vector Machine

- For regression, Support Vector Machine and Generalized Linear Models (Multivariate Linear Regression)

- For associations (market basket analysis), Apriori

- For clustering, *k*-Means and O-Cluster

- For attribute importance, Minimum Description Length

- For anomaly detection, One Class Support Vector Machine

- For feature extraction, Non-Negative Matrix Factorization

> **See Also:**
>
> - *Oracle Data Mining Concepts*
>
> - *Oracle Data Mining Administrator's Guide*
>
> - *Oracle Data Mining Application Developer's Guide*
>
> - *Oracle Data Mining Java API Reference* contains Javadoc descriptions of the classes that constitute the Oracle Data Mining Java API
>
> - The PL/SQL API is described in the `DBMS_DATA_MINING`, `DBMS_DATA_MINING_TRANSFORM`, and `DBMS_PREDICTIVE_ANALYTICS` chapters of *Oracle Database PL/SQL Packages and Types Reference*
>
> - The SQL Data Mining functions are described in *Oracle Database SQL Language Reference*

17

High Availability

This chapter discusses the concept of database availability and introduces you to Oracle Database high availability products and features.

> **Note:** Availability is influenced by many choices you make other than your database software: hardware, application and operating system software, storage media, network reliability, and operational processes are all important.

This chapter includes these topics:

- Introduction to High Availability
- Causes Of Downtime
- Protection Against Computer Failures
- Protection Against Data Failures
- Avoiding Downtime During Planned Maintenance
- Maximum Availability Architecture (MAA) Best Practices

> **See Also:**
>
> - *Oracle Database High Availability Overview*
> - *Oracle Database High Availability Best Practices*
>
> These books provide complete information about best practices for deploying a highly available environment and describes the Oracle products and features that support high availability.

Introduction to High Availability

Enterprises have used their information technology (IT) infrastructure to provide competitive advantage, increase productivity, and empower users to make faster and more informed decisions. However, with these benefits has come an increasing dependence on that infrastructure. Revenue and customers can be lost, penalties can be owed, and bad press can have a lasting effect on customers and a company's reputation. Building a high availability IT infrastructure is critical to the success and well being of all enterprises in today's fast moving economy.

Trends in computing technology are also enabling a new IT architecture, referred to as Grid computing, to be deployed. The Grid computing architecture effectively pools large numbers of servers and storage into a flexible, on-demand computing resource

for all enterprise computing needs. Technology innovations like low-cost blade servers, small and inexpensive multiprocessor servers, modular storage technologies, and open source operating systems (such as Linux) provide the raw materials for the Grid. By harnessing these technologies and leveraging the Grid technology available in the Oracle Database, enterprises can deliver an extremely high quality of service to users while vastly reducing expenditures on IT. The Oracle Database enables you to capture the cost advantages of Grid enterprise computing without sacrificing performance, scalability, security, manageability, functionality, or system availability.

This chapter examines the causes of downtime and looks at the technology available in the Oracle Database that avoids costly downtime and enables rapid recovery from failures.

Causes Of Downtime

One of the challenges when designing a highly available IT Grid infrastructure is examining and addressing all the possible causes of downtime. Figure 17–1 shows a diagram that classifies system downtime into two primary categories: unplanned and planned downtime. It is important to consider the causes of both unplanned and planned downtime when designing a fault tolerant and resilient IT infrastructure.

Figure 17–1   Causes of Downtime
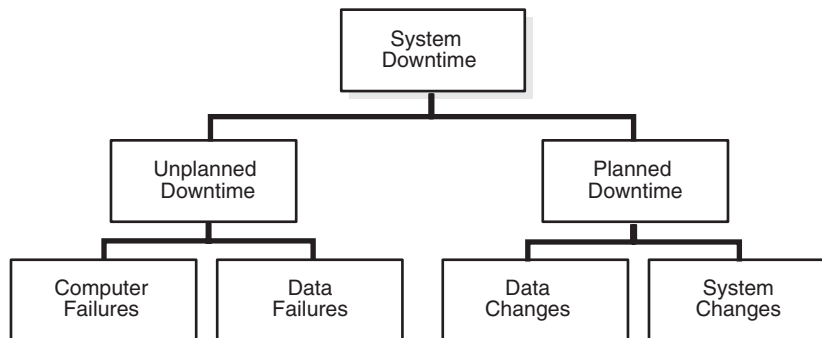
Unplanned downtime results from computer failures or data failures. Planned downtime is primarily due to data changes or system changes that must be applied to the production system. The following sections examine each of these causes of downtime and describes the Oracle technology you can apply to avoid downtime.

Protection Against Computer Failures

A computer failure occurs when the computer system or database server unexpectedly fails and causes a service interruption. In most cases, computer failures are due to hardware breakdown. These types of failures are best remedied by taking advantage of cluster technology and fast database crash recovery. The recommended solutions include Enterprise grids with Oracle Real Application Clusters (Oracle RAC), fast start fault recovery, Oracle Data Guard, and Oracle Streams.

This section includes the following topics:

- Overview of Enterprise Grids with Oracle Real Application Clusters and Oracle Clusterware

- Fast Start Fault Recovery

- Oracle Data Guard

- Oracle Streams

Overview of Enterprise Grids with Oracle Real Application Clusters and Oracle Clusterware

With Oracle RAC, the enterprise can build database servers across multiple systems that are highly available and highly scalable. In an Oracle RAC environment, Oracle Database runs on two or more systems in a cluster while concurrently accessing a single shared database. This provides a single database system that spans multiple hardware systems yet appears to the application as a single unified database system. This extends the following availability and scalability benefits for all of your applications:

- Flexibility and cost effectiveness in capacity planning, so that a system can scale to any desired capacity on demand and as business needs change.

- Fault tolerance within the cluster, especially computer failures.

The following list describes the features of an Oracle RAC environment:

Enterprise Grids—Oracle RAC enables enterprise Grids. Enterprise Grids are built out of large configurations of standardized, commodity-priced components: processors, servers, network, and storage. Oracle RAC is the only technology that can harness these components into a useful processing system for the enterprise. Oracle RAC and the Grid dramatically reduce operational costs and provide flexibility so that systems become more adaptive, proactive, and agile. Dynamic provisioning of nodes, storage, CPUs, and memory allow service levels to be easily and efficiently maintained while lowering cost still further through improved use. In addition, Oracle RAC is completely transparent to the application accessing the Oracle RAC database, thereby allowing existing applications to be deployed on Oracle RAC without requiring any modifications.

Scalability—Oracle RAC gives you the flexibility to add nodes to the cluster as the demand for capacity increases, scaling the system up incrementally to save costs and eliminating the need to replace smaller single node systems with larger ones. Grid pools of standard low-cost computers and modular disk arrays make this solution even more powerful with Oracle Database. It makes the capacity upgrade process much easier and faster because you can incrementally add one or more nodes to the cluster, compared to replacing existing systems with new and larger nodes to upgrade systems. The Cache Fusion technology implemented in Oracle RAC and the InfiniBand support provided in Oracle Database enable you to scale the capacity almost linearly, without making any changes to your application.

Fault Tolerance—Another key advantage of the Oracle RAC cluster architecture is the inherent fault tolerance provided by multiple nodes. Because the physical nodes run independently, the failure of one or more nodes will not affect other nodes in the cluster. Failover can happen to any node on the Grid. In the extreme case, an Oracle RAC system will still provide database service even when all but one node is down. This architecture allows a group of nodes to be transparently put online or taken offline, for maintenance, while the rest of the cluster continues to provide database service. Oracle RAC provides built in integration with the Oracle Application Server for failing over connection pools. With this capability, an application is immediately notified of any failure rather than having to wait tens of minutes for a TCP timeout to occur. The application can immediately take the appropriate recovery action. And Grid load balancing redistributes load over time.

Oracle Clusterware—Oracle RAC also provides a complete set of clusterware to manage the cluster. Oracle Clusterware provides all of the features required to run the

cluster, including node membership, messaging services, and locking. Because Oracle Clusterware is a fully integrated stack with common event and management APIs, it can be centrally managed from Oracle Enterprise Manager. There is no need to purchase additional software to support your cluster, which helps avoid the additional efforts required to integrate and test third-party clusterware. Oracle Clusterware also provides the same interface and operates the same way across all of the platforms on which Oracle Database is available. While Oracle continues to support third-party clusterware for use with Oracle RAC, there is no need or advantage to using third-party clusterware.

You can extend the high availability capabilities of the Oracle Clusterware framework to your applications. That is, you can use the same high availability mechanisms of Oracle Database and Oracle RAC to make your custom applications highly available. You can use Oracle Clusterware to monitor, relocate, and restart your applications, thus allowing you to integrate and coordinate failover of your applications with database failover.

Services—Oracle RAC supports an entity referred to as a service that you can define to group database workloads and route work to the optimal instances that are assigned to offer the service. Services represent classes of database users or applications. You define and apply business policies to these services to perform tasks such as to allocate nodes for times of peak processing or to automatically handle a server failure. Using services ensures the application of system resources where and when they are needed to achieve business goals.

> **See Also:** *Oracle Database 2 Day + Real Application Clusters Guide*

Fast Start Fault Recovery

One of the most common causes of unplanned downtime is a system fault or crash. System faults are the result of hardware failures, power failures, and operating system or server crashes. The amount of disruption these failures cause depends on the number of affected users and how quickly service is restored. High availability systems are designed to quickly and automatically recover from failures, should they occur. Users of critical systems look to the IT organization for a commitment that recovery from a failure will be fast and take a predictable amount of time. Periods of downtime longer than this commitment can have a direct effect on operations and lead to lost revenue and productivity.

Oracle Database provides very fast recovery from system faults and crashes. However, equally important to being fast is being predictable. The fast start fault recovery technology included in Oracle Database automatically bounds database crash recovery time and is unique to Oracle Database. The database self tunes checkpoint processing to safeguard the desired recovery time objective. This makes recovery time fast and predictable and improves the ability to meet service-level objectives. The Oracle fast start fault recovery feature can reduce recovery time on a heavily loaded database from tens of minutes to a few seconds.

> **See Also:** *Oracle Database Performance Tuning Guide* for information on fast start fault recovery

Oracle Data Guard

Oracle Data Guard ensures high availability, data protection, and disaster recovery for enterprise data. Data Guard maintains standby databases as transactionally consistent copies of the primary (production) database. Then, if the primary database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any

standby database to the primary role, minimizing the downtime associated with the outage. Automated failover using Data Guard fast-start failover and fast application notification with integrated Oracle clients provides a high level of data protection and data availability.

> **See Also:** *Oracle Data Guard Concepts and Administration*

Oracle Streams

You can use Streams to configure flexible high availability environments. With Oracle Streams, you can create a local or remote copy of a production database. In the event of human error or a catastrophe, the copy can be used to resume processing.

> **See Also:** *Oracle Streams Concepts and Administration*

Protection Against Data Failures

A data failure is the loss, damage, or corruption of critical enterprise data. The causes of data failure are more complex and subtle than computer failure and can be caused by a failure of the storage hardware, human error, corruption, or site failure.

Figure 17–2 focuses on the four types of data failures: storage failure, human error, corruption, and site failure.

Figure 17–2   Downtime Due to Data Failures
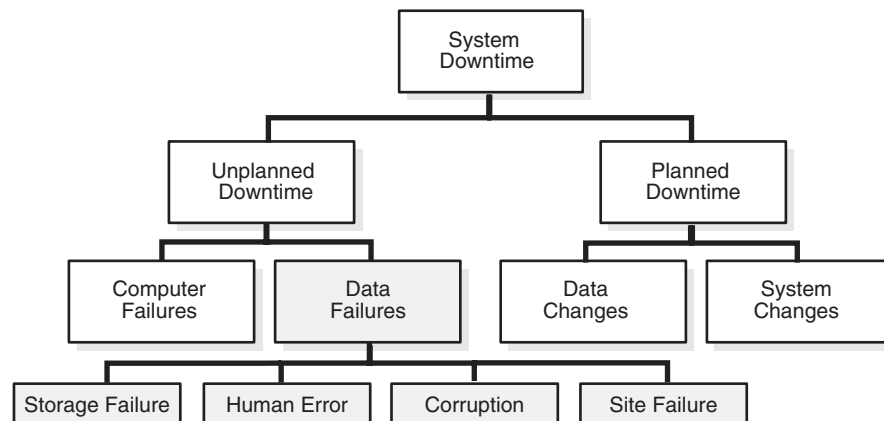
It is extremely important to design a solution to protect against and recover from data failures. A system or network fault may prevent users from accessing data, but data failures without proper backups or recovery technology can result in either the recovery operation taking many hours to perform, or in lost data.

Oracle Database provides many data protection capabilities. The motivation for many of these enhancements is the new economics around data protection and recovery. Over the last twenty years, disk capacity has grown by three orders of magnitude while the cost per megabyte has fallen dramatically. This is a trend that shows no sign of abating. This has made the cost of disk storage competitive with tape as a backup media. Plus, disk storage has additional benefits of being online and able to provide random access to the data.

These trends allowed Oracle to rethink and make its recovery strategy hierarchical to take advantage of these economic dynamics. By making additional disk storage available to Oracle Database, you can reduce backup and recovery time from hours to minutes. In essence, you can trade inexpensive disk storage for expensive downtime.

This section includes the following topics:

- Protecting Against Storage Failures
- Protecting Against Human Errors

Protecting Against Storage Failures

Provisioning storage for a single-database instance, much less for an entire enterprise, can be complex. Typically, the process includes the following steps:

1. Estimate the amount of space needed

2. Map out what you hope will be an optimal layout (where to put data files, archive files, and so on to avoid hot spots)

3. Create logical volumes

4. Create file systems

5. Define and set up how you will protect and mirror your data

6. Define and implement your backup and recovery plan for the data

7. Install the Oracle software

8. Create the database

Then, the hard work begins—looking for *hot spots* that negatively affect performance; moving datafiles around to reduce contention, and dreading the day when a disk crash occurs or when you run out of space and must add more disks and shift all the files around to rebalance across your updated storage configuration.

Fortunately, that scenario is changing dramatically with the Automatic Storage Management (ASM) feature of Oracle Database. ASM provides a vertically integrated file system and volume manager directly in the Oracle kernel, resulting in much less work to provision database storage.

ASM provides a higher level of availability, without the expense, installation, and maintenance of specialized storage products, and provides unique capabilities for database applications. ASM spreads its files across all available storage for optimal performance, and it can mirror as well, providing protection against data loss. ASM extends the concept of SAME (stripe and mirror everything) and adds more flexibility in that it can do mirroring at the database file level instead of having to mirror at the entire disk level.

Most importantly, ASM eliminates the complexity associated with managing data and disks, and it simplifies the processes of setting up mirroring, adding disks, and removing disks. Rather than managing hundreds, possibly thousands of files (as in a large data warehouse) database administrators using ASM create and administer a larger-grained object, the disk group, which identifies the set of disks to be managed as a logical unit. The automation of the file naming and placement of the underlying database files save the DBAs time and ensures that best practice standards are followed.

Optionally, you can use the ASM native mirroring mechanism to protect against storage failures. Mirroring is enabled by default and triple mirroring is also available. With ASM mirroring, you can provide an additional level of data protection with the use of failure groups. A failure group is a set of disks sharing a common resource (disk controller or an entire disk array) whose failure can be tolerated.

Once defined, an ASM failure group intelligently places redundant copies of the data in separate failure groups to ensure that the data will be available and transparently

protected against the failure of any component in the storage subsystem. In addition, ASM supports the Hardware Assisted Resilient Data capability (discussed below in the Protecting Against Data Corruptions section) to further protect your data.

See Also: *Oracle Database 2 Day DBA*

Protecting Against Human Errors

Most research performed on the causes of downtime identifies human error as the single largest cause of downtime. Human errors—such as the inadvertent deletion of important data or when an incorrect `WHERE` clause in an `UPDATE` statement updates many more rows than were intended—must be prevented wherever possible and must be undone when the precautions against them fail. Oracle Database provides easy to use yet powerful tools that help administrators quickly diagnose and recover from these errors. It also includes features that allow end users to recover from problems without administrator involvement, reducing the support burden on the administrators and speeding recovery of the lost and damaged data.

The following sections describe the Oracle features that protect against human errors:

- Guarding Against Human Errors
- Oracle Flashback Technology
- LogMiner SQL-Based Log Analyzer
- Protecting Against Data Corruptions
- Protecting Against Site Failures

Guarding Against Human Errors

The best way to prevent errors is to restrict a user's access to data and services they truly need to conduct their business. Oracle Database provides a wide range of security tools to control user access to application data by authenticating users and then allowing administrators to grant users only those privileges required to perform their duties. In addition, the security model of Oracle Database helps you restrict data access at a row level, using the Virtual Private Database (VPD) feature. This further isolates users from data they do not need access to.

Oracle Flashback Technology

When authorized people make mistakes, you need the tools to correct these errors. Oracle Database provides a family of human error correction technology called Flashback. Flashback revolutionizes data recovery. In the past, it might take minutes to damage a database but hours to recover it. With Flashback technology, the time to correct errors equals the time it took to make the error. It is also easy to use a single, short command to recover the entire database instead of following some complex procedure. Flashback is unique to Oracle Database and provides:

- A SQL interface to quickly analyze and repair human errors.
- Fine-grained surgical analysis and repair for localized damage, such as when the wrong customer order is deleted.
- Correction of more widespread damage yet does it quickly to avoid long downtime, such as when all of this month's customer orders have been deleted.
- Recovery at all levels including the row, transaction, table, tablespace, and database wide.

Table 17–1 describes how Flashback technology corrects human errors.

Table 17–1   Protecting Against Human Errors with Oracle Flashback Technology

Feature	Description
Flashback Query	Oracle Flashback Query allows you to query any data at some point in time in the past. You can use Flashback Query to view and reconstruct lost data that may have been deleted or changed by accident. For example: `SELECT * FROM employee AS OF TIMESTAMP TO_TIMESTAMP('19-APR-05 02:00:00 PM') WHERE …` This statement displays rows from the employee table as of 2:00pm on the specified date. Developers can use this feature to build self-service error correction into their applications, empowering end users to undo and correct their errors without delay rather than burdening administrators to perform this task. Flashback Query is simple to manage, because the database automatically keeps the necessary information to reconstruct data for a configurable time into the past.

Flashback Versions Query	The Flashback Versions Query provides a way to view changes made to the database at the row level. It is an extension to SQL and allows you to retrieve all of the different versions of a row across a specified time interval. For example:
	
	`SELECT * FROM employee VERSIONS BETWEEN TIMESTAMP TO_TIMESTAMP('19-APR-05`
	`02:00:00 PM') AND TIMESTAMP TO_TIMESTAMP('19-APR-05 03:00:00 PM') WHERE …`
	
	This statement displays each version of the row, with each row changed by a different transaction between 2:00 and 3:00 p.m. on 19 April. This helps you to pinpoint when and how data is changed and trace it back to the user, application, or transaction, and tracks down the source of a logical corruption in the database and correct it. Flashback Versions Query also helps application developers debug code.

Flashback Transaction	Oracle Flashback Transaction backs out a transaction and its dependent transactions. The `DBMS_FLASHBACK.TRANSACTION_BACKOUT()` procedure rolls back a transaction and its dependent transactions while the database remains online. This recovery operation uses undo data to create and execute the compensating transactions that return the affected data to its original state. You can query the `DBA_FLASHBACK_TRANSACTION_STATE` view to see the current state of a transaction with respect to whether the transaction has been backed out using dependency rules or forced out by either: ■ Backing out nonconflicting rows ■ Applying undo SQL Oracle Flashback Transaction increases availability during logical recovery by easily and quickly backing out a specific transaction or set of transactions and their dependent transactions, with one command while the database remains online.
Flashback Transaction Query	Flashback Transaction Query provides a way to view changes made to the database at the transaction level. It is an extension to SQL that allows you to see all changes made by a transaction. For example: `SELECT * FROM FLASHBACK_TRANSACTION_QUERY WHERE XID = '000200030000002D';` This query shows all of the resultant changes made by this transaction. In addition, compensating SQL statements are returned and can be used to undo changes made to all rows by this transaction. Using a precision tool like this, the database administrator and application developer can precisely diagnose and correct logical problems in the database or application.
Flashback Database	To bring an Oracle database to a previous point in time, the traditional method is to do point-in-time recovery. However, point-in-time recovery can take hours, or even days, because it requires the whole database to be restored from backup and recovered to the point in time just before the error was introduced into the database. With the size of databases constantly growing, it will take hours or even days just to restore the whole database. Flashback Database is a strategy for doing point in time recovery. It quickly rewinds an Oracle database to a previous time to correct any problems caused by logical data corruption or user error. It uses flashback logs to capture old versions of changed blocks. This is similar to a continuous backup or storage snapshot. When you need to perform recovery, the Flashback logs are quickly replayed to restore the database to a point in time before the error and just the changed blocks are restored. Flashback Database is fast and reduces recovery time from hours to minutes. For example, issue the following command to recover a database to 2:05 p.m.: `FLASHBACK DATABASE TO TIMESTAMP TO_TIMESTAMP('19-APR-05 02:05:00 PM');`

Table 17–1   (Cont.) Protecting Against Human Errors with Oracle Flashback Technology

Feature	Description
Flashback Table	Flashback Table recovers a table or a set of tables to a specified point in time in the past. In many cases, Flashback Table alleviates the need to perform more complicated point-in-time recovery operations. For example:
	`FLASHBACK TABLE orders, order_items TIMESTAMP TO_TIMESTAMP('07-APR-2005 02:33:00 PM');`
	This command rewinds any updates to the `orders` and `order_items` tables that were made between the current time and the specified timestamp in the past. Flashback Table performs this operation online and in place and it maintains any referential integrity constraints between the tables. Flashback Table is very similar to having a rewind or undo button for a table, or a set of related tables.
Flashback Drop	Dropping, or deleting, database objects by accident is a mistake people make, and probably always will make, thinking they were connected to the test database when they actually deleted that table. Users soon realize their mistake but by then it is too late and there is no way to easily recover the dropped tables and its indexes, constraints, and triggers. Objects once dropped are lost forever. For important tables or other objects (like indexes, partitions or clusters), you need to perform a point-of-time recovery, which may be very time consuming and lead to loss of recent transactions.
	Flashback Drop provides a safety net when dropping objects in Oracle Database. When a user drops a table, Oracle Database places it in a recycle bin. The recycle bin is a virtual container where all dropped objects reside. Objects remain in the recycle bin until you decide to permanently remove the objects or until the space pressure is placed on the tablespace containing the table. You can *undrop* the dropped table and its dependent objects from the Recycle Bin. For example, the table employee and all of its dependent objects would be undropped using the following command:
	`FLASHBACK TABLE employee TO BEFORE DROP;`
Flashback Restore Points	When an Oracle database point-in-time recovery operation is required, you must determine a time or System Change Number (SCN or transaction time) to which the data must be rolled back. Oracle Database uses restore points, which is a user-defined label that can be substituted for an SCN or clock time when used in conjunction with Flashback Database, Flashback Table, and Recovery Manager (RMAN). Restore points provides the ability to bookmark a known time when the database was in a good state, allowing quick and easy rewind of an inappropriate action done to the database. It also provides the ability to flashback through a previous database recovery and open resetlogs. Guaranteed restore points allow major database changes to be quickly undone (for example, database batch job, upgrade, or patch) by ensuring that the undo required to rewind the database is retained. Restore points provide an easy way to rewind back to a known time.
	In an Oracle Data Guard environment, this capability also allows a physical standby database that has been opened read-write to later flash back the changes and later convert the database back to a physical standby database that is synchronized with the production database. If a logical error is discovered after a switchover operation, the primary and standby databases can be flashed back to an SCN or a point in time prior to the switchover operation. In addition, to quickly synchronize the standby database with the production database, you can apply an incremental backup of the production database to the standby database instead of applying all the redo data generated since the two databases diverged. This can significantly reduce the time to resynchronization the two databases.
Block recovery using flashback logs	Block recovery can optionally retrieve a more recent copy of a data block from the flashback logs to reduce recovery time. Furthermore, a corrupted block encountered during instance recovery does not result in instance recovery failing. The block is automatically marked as corrupt and added to the RMAN corruption list in the `V$DATABASE_BLOCK_CORRUPTION` table. You can subsequently issue the RMAN `RECOVER BLOCK` command to fix the associated block.
Flashback Data Archive	A Flashback Data Archive provides the ability to track and store all transactional changes to a table over its lifetime. It is no longer necessary to build this intelligence into your application. A Flashback Data Archive is useful for compliance with record stage policies and audit reports. See or complete information.

See Also:

- Chapter 13, "Data Concurrency and Consistency" for information on Oracle Flashback Query

- Chapter 15, "Backup and Recovery" for information on Oracle Flashback Database and Oracle Flashback Table

- *Oracle Database Advanced Application Developer's Guide* for more information on Flashback Transaction

LogMiner SQL-Based Log Analyzer

Oracle log files contain a wealth of useful information about the activities and history of an Oracle database. Log files contain all of the data needed to perform database recovery. They also record every change made to data and metadata in the database. LogMiner is a relational tool that allows redo log files to be read, analyzed, and interpreted using SQL. You can use analysis of the log files with LogMiner to track or audit changes to data, provide supplemental information for tuning and capacity planning, retrieve critical information for debugging complex applications, or recover deleted data.

> **See Also:**   "Overview of LogMiner" on page 11-4

Protecting Against Data Corruptions

A corruption is created by a faulty component in the I/O stack. For example, the database issues I/O operations as the result of an update transaction. The database I/O is passed to:

1. The I/O code in the operating system

2. The file system

3. The volume manager

4. The device driver

5. The host bus adapter

6. The storage controller

7. The disk drive to which the I/O is finally written

Bugs or a hardware failure in any component in the I/O sequence could *flip some bits* in the data resulting in corrupt data being written to the database. This corruption could be due to database control information or user data either of which could be catastrophic to the functioning and availability of the database. Similarly, a disk failure could damage database files requiring backups be used to recover the database.

Protection against data corruptions include the following topics:

- Oracle Hardware Assisted Resilient Data (HARD)

- Lost Writes

- Data Recovery Advisor

- Flash Backup And Recovery

Oracle Hardware Assisted Resilient Data (HARD) The Oracle Hardware Assisted Resilient Data (HARD) prevents data corruptions before they happen. Data corruptions while rare, can have a catastrophic effect on a database, and therefore a business. By implementing the Oracle data validation algorithms inside storage devices, Oracle prevents corrupted data from being written to the database files on persistent storage. This type of end-to-end high-level software to low-level hardware validation is a unique capability provided by Oracle and its storage partners.

Oracle validates and adds protection information to the database blocks and this protection information is further validated by the storage device. HARD detects corruptions from being introduced into the I/O path between the database and storage, and eliminates a large class of failures that the database industry has previously been powerless to prevent. RAID has gained a wide following in the storage industry by ensuring the physical protection of data, HARD takes data

protection to the next level by going beyond protecting physical bits, to protecting business data.

> **See Also:**
>
> http://www.oracle.com/technology/deploy/availability /htdocs/HARD.html for more information about HARD

Lost Writes A lost write is another form of data corruption, but it is much more evasive to detect and repair quickly. A data block lost write occurs when:

- An I/O subsystem acknowledges the completion of the block write, while in fact the write did not occur in the persistent storage. On a subsequent block read, the I/O subsystem returns the stale version of the data block, which might be used to update other blocks of the database, thereby corrupting it.

- The write I/O completed but it was written somewhere else, and a subsequent read operation returns the stale value.

- A read I/O from one cluster node returns stale data after a write I/O on another node. For example, this could occur if an NFS caching policy is incompatible with Oracle RAC.

Data block corruption prevention and detection Prior to Oracle Database 11*g*, block corruptions detected by RMAN were recorded in V$DATABASE_BLOCK_CORRUPTION. In Oracle Database 11*g*, several database components and utilities, including RMAN, can now detect a corrupt block and record it in that view. Oracle Database automatically updates this view when block corruptions are detected or repaired (for example, using block media recovery or data file recovery). The benefit is that the time it takes to discover block corruptions is shortened.

Data Recovery Advisor Data Recovery Advisor automatically diagnoses persistent (on disk) data failures, presents appropriate repair options, and runs repair operations at your request. It includes the following functionality:

- Failure diagnosis

- Failure impact assessment

- Repair generation

- Repair feasibility checks

- Repair automation

- Validation of data consistency and database recoverability

- Early detection of corruption

- Integration of data validation and repair

Note that the initial release of Data Recovery Advisor does not support Oracle RAC. In addition, while you can use Data Recovery Advisor to manage a primary database in a Data Guard configuration, you cannot use Data Recovery Advisor to troubleshoot a physical standby database. Data Recovery Advisor only takes the presence of a standby database into account when recommending repair strategies if you are using Oracle Enterprise Manager 11*g* Grid Control.

Flash Backup And Recovery There is no substitute for backups of enterprise data. Although rare, multiple failures can render even data mirrored in the storage subsystem unusable. Fortunately, Oracle provides online tools to properly backup all

your data, to restore data from a previous backup, and to recover changes to that data up to the time just before the failure occurred.

Backing up a large database system is no simple task. A large database can be composed of hundreds of files spread over many different disks. Neglecting to backup a critical file can render the entire database backup unusable. Often these damaged files are not discovered until they are needed. Recovery Manager (RMAN) is a tool that manages the backup, restore, and recovery process for Oracle Database. It creates and maintains backup policies, and catalogs all backup and recovery activities. All data blocks can be analyzed for corruption during backup and restore to prevent propagation of corrupt data through to the backups. Most importantly, Recovery Manager ensures all necessary data files are backed up, and the database is recoverable.

Recovery Manager automatically keeps track of the files needed to restore the database within a user-specified window. It can automatically restart interrupted operations, handle corrupted log files, and restore an individual data block while the remainder of the database remains online.

RMAN radically enhances database backup and recovery. RMAN can automatically manage backing up and recover all of your data to the Flash Recovery Area. The Flash Recovery Area is a unified disk-based storage location for all recovery related files and activities in an Oracle database. Making backups to disk, instead of tape, enables faster backups. But more importantly, if database media recovery is required, then datafile backups are readily available, radically speeding database recovery time.

Recovery Manager manages the recovery files in the Flash Recovery Area. RMAN automatically creates all backups in the Flash Recovery Area and manages the space. The archiver writes redo data to the Flash Recovery Area and RMAN automatically deletes, or moves to tape, obsolete backups and archived redo logs that are no longer required. If you set the RETENTION POLICY to a recovery window of 7 days, then RMAN retains all backups required to recover the database 7 days back. If you require recovery to a time further than 7 days in the past, RMAN restores the data from tape. Oracle Enterprise Manager provides a complete interface to drive Flash Backup and Recovery, including implementing best practices.

Incremental backups have been part of RMAN since it was first released in the Oracle8 Database. Incremental backups record only the changed blocks since the previous backup. Oracle Database delivers the ability for faster incremental backups with the implementation of block change tracking. Oracle Database tracks the physical location of all database changes. RMAN automatically uses this change tracking information to determine which blocks need to be read during an incremental backup and directly accesses the block to back it up. The incremental backups can then be merged into a previously created image backup to minimize the time for recovery. A backup strategy based on incrementally updated backups keeps the time required for media recovery to a minimum. By making incremental backups with change tracking part of your backup strategy, you can: reduce the amount of time needed for daily backups, save network bandwidth when backing up over a network, recover unlogged changes to database, reduce the backup file storage, and reduce the time for database recovery.

Oracle Database backup and recovery also includes many other innovative capabilities

- Compressed backups

- Automated failover to a previous backup when restore discovers a missing or corrupt backup

- Automated recovery through a previous point in time recovery, or recovery through resetlogs

- Automated creation of new database and temporary files during recovery

- Automated channel failover on backup or restore

- Automated tablespace point-in-time recovery

- Full database BEGIN BACKUP command for faster mirror split

- Faster recovery due to improved recovery parallelism (2 to 4 times)

- Tablespace Rename

- Proxy (third party) backup for archived redo logs

- Time window based throttling of backups

- Cross Platform Transportable Tablespaces

> **See Also:**
>
> - Chapter 15, "Backup and Recovery" for information on backup and recovery solutions
>
> - *Oracle Database Backup and Recovery User's Guide* for information on RMAN and backup and recovery solutions

Protecting Against Site Failures

Data protection features provide protection from catastrophic events that cripple processing at a site for an extended period of time. Examples include file corruptions, natural disasters, power and communication outages, and even terrorism. Oracle Database offers a variety of data protection solutions that provide the ability to create and maintain a local or remote copy of a production database. In the event of a corruption or disaster, users of the data can continue to function by accessing the remote database.

The simplest form of data protection is off-site storage of database backups. In the event a data center is unable to resume services in a reasonable amount of time, the backups can be restored on a system at another site, and users can connect to the backup system. Unfortunately, restoring backups on another system will be time consuming, and the backup may not be completely up to date. To more quickly recover and maintain continuous database service even in the event of a disaster, Oracle provides Data Guard, which is described in the following topics:

- Oracle Data Guard and standby databases

- Zero Data Loss Redo Transport

- Real-Time Apply and Flashback Database

- Data Guard Broker

- Fast-Start Failover

Oracle Data Guard and standby databases Data Guard should be the foundation of any Oracle Database disaster recovery plan. Data Guard provides the ability to set up and maintain a standby copy of your production database. You can locate the standby database a half a world away from the production database or in the same data center. Data Guard includes enhancements to automate complex tasks and provide significant monitoring, alerting and control mechanisms. It enables your database to survive a data center disaster. Data Guard also works transparently across Grid clusters and you can add the servers dynamically to the standby database in the event a failover is required.

Oracle Data Guard supports physical standby databases that use Redo Apply technology, snapshot standby databases, and logical standby databases that use SQL Apply technology:

■ Physical standby databases and Redo Apply

Data Guard in Redo Apply mode maintains a copy of a production database, called a physical standby database, and keeps it synchronized with the production database. The redo data from the primary database are shipped to the standby and physically applied via media recovery. The standby database is physically identical to the primary (although it may lag the primary). Additionally, you can open the standby database in read-only mode while redo is being applied so the database can also be used to off load reporting work from the production database. Backup processing may also be offloaded from the production database as backups created at the standby database can be used to perform recovery of the production database.

Physical standby databases are good for providing protection from disasters and data errors. In the event of an error or disaster, the physical standby can be opened, and be used to provide data services to applications and end users. Because the efficient media recovery mechanism is used to apply changes to the standby database, it is supported with every application, and can easily and efficiently keep up with even the largest transaction workloads.

■ Snapshot standby databases

A snapshot standby database is an updatable standby database that you create from a physical standby database. A snapshot standby database receives and archives redo data received from the primary database, but the snapshot standby database does not apply redo data from the primary database while the standby is open read-write. Because of this, the snapshot standby typically diverges from the primary database over time. Moreover, local updates to the snapshot standby database cause additional divergence.

Redo data is not applied until you convert the snapshot standby database back into a physical standby database, and after all local updates to the snapshot standby database are discarded. With a single command, you can revert a snapshot standby back to a physical standby database, at which time the changes made to the snapshot standby state are discarded, and Redo Apply automatically resynchronizes the physical standby database with the primary database using the redo that was archived.

■ Logical standby databases and SQL Apply

Data Guard in SQL Apply mode takes redo data received from the primary database, transforms the redo into SQL transactions, and applies them to an open standby database. Although a logical standby database can be physically different from the primary database, it is logically the same as the primary and it can take over processing if the primary database is destroyed. Because transactions are applied using SQL to an open database, the standby can be used concurrently for other tasks, and can have a different physical structure than the production database. For example, the logical standby can be used for decision support, and be optimized for reporting by using additional indexes and materialized views that do not exist on the primary database.

SQL Apply is most importantly a data protection feature because it compares before-change values in the log files to the before-change values in the logical standby database providing a check against logical corruption. A logical standby database can therefore offer protection from the widest possible range of corruptions.

Because logical standby databases are open for read/write I/O during recovery, you can query the standby database while SQL Apply applies changes in the redo logs.

Zero Data Loss Redo Transport   Both physical and logical standby databases use the same transport services. Data Guard offers customers the choice of synchronous and asynchronous transport methods. Data Guard synchronous transport services provide *zero data loss* protection insuring that if a disaster should strike the primary database, the redo data necessary to preserve all previously committed transactions is available at the standby site.

You can also choose to transmit redo data asynchronously to the standby site. This minimizes any potential data loss while providing optimal performance over large distances, and provides protection from network failures.

Real-Time Apply and Flashback Database   With real-time apply, Data Guard apply services can apply redo data on the standby database as soon as it is received from the primary database, without waiting for the current log file to be archived at the standby database. This enables standby databases to be closely synchronized with the primary database, enabling up to date and real-time reporting. This also enables faster switchover and failover times, which in turn reduces planned and unplanned downtime for the business. You may also choose to use Flashback Database on both the primary and standby database to quickly revert the databases to an earlier point in time to back out user errors. Alternatively, if you decide to failover to a standby database, but those user errors were already applied to the standby database (because real-time apply was enabled), you can flash back the standby database to a safe point in time. The use of these two features eliminate the tradeoff sometimes necessary to ensure the standby database stays current and delays applying redo data as way of preventing human errors on the production database from propagating to the standby database.

Data Guard Broker   The primary and standby databases, as well as their various interactions, may be managed by using SQL*Plus. For easier manageability, Data Guard also offers the Data Guard Broker distributed management framework, which automates and centralizes the creation, maintenance, and monitoring of a Data Guard configuration. You can use either Oracle Enterprise Manager or the Broker's own specialized command-line interface (DGMGRL) to take advantage of the Broker's management capabilities. From the easy-to-use Oracle Enterprise Manager GUI, you can initiate failover processing with a single mouse click from the primary to either type of standby database. The Broker and Oracle Enterprise Manager make it easy to manage and operate the standby database. By facilitating activities such as failover and switchover, the possibility of errors is greatly reduced.

Fast-Start Failover   Fast-start failover enables the creation of a fault-tolerant standby database environment by providing the ability to totally automate the failover of database processing from the production to standby database, without any human intervention. Fast-start failover automatically, quickly, and reliably fails over to a designated, synchronized standby database in the event of loss of the primary database, without requiring administrators to perform complex manual steps to invoke and implement the failover operation. This greatly reduces the length of an outage. You set up fast-start failover using either Oracle Enterprise Manager or the Data Guard Broker. The Observer, which monitors the Data Guard environment, automatically triggers and completes the failover when required. After a fast-start failover occurs, the old primary database, upon reconnection to the configuration, is automatically reinstated as a new standby database by the Broker. This enables the Data Guard configuration to restore disaster protection in the configuration easily and

quickly, improving the robustness of the Data Guard configuration. With these capabilities, Data Guard not only helps maintain transparent business continuity, but also reduces the management costs for the disaster-recovery configuration.

> **See Also:**
>
> - *Oracle Data Guard Concepts and Administration*
> - *Oracle Data Guard Broker*

Avoiding Downtime During Planned Maintenance

Planned downtime can be just as disruptive to operations, especially in global enterprises that support users in multiple time zones. In this case it is important to design a system to minimize planned interruptions. Planned downtime includes routine operations, periodic maintenance, and new deployments.

Routine operations are frequent maintenance tasks that include backups, performance management, user and security management, and batch operations. Periodic maintenance, such as installing a patch or reconfiguring the system, is occasionally necessary to update the database, application, operating system, middleware, or network. New deployments may be necessitated by major upgrades to the hardware, operating system, database, application, middleware, or network. It is important to consider not only the time to perform the upgrade, but also the effect the changes may have on the overall application.

The Internet has made it easy to share data globally, but brings new challenges and requirements for data availability. As global users access data 24 hours per day, maintenance windows have all but evaporated. Planned downtime is becoming as disruptive as unplanned downtime. There are no longer any windows of time during which users are not affected. When the volume of data stored in a database becomes very large, maintenance operations can be quite time consuming. It is important that these operations be performed without affecting the users of the data.

This section includes the following topics:

- Avoiding Downtime for Data Changes
- Avoiding Downtime for System Changes

Avoiding Downtime for Data Changes

Figure 17–3 focuses on the three types of downtime due to data changes: online schema and data reorganization, partitioned tables and indexes, and dynamic resource provisioning.

Figure 17–3   Downtime Due to Data Changes
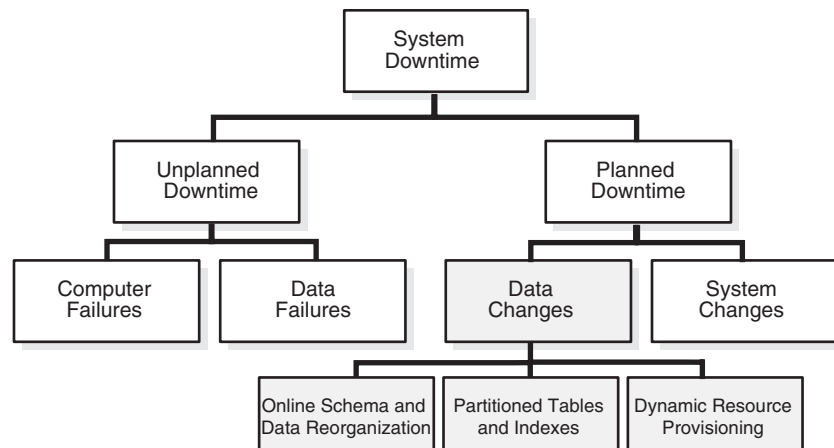
This section includes the following topics:

- Online Schema and Data Reorganization
- Partitioned Tables and Indexes
- Dynamic Resource Provisioning

Online Schema and Data Reorganization

Oracle Database allows you to perform most maintenance operations without disrupting database operations or data availability to users. Indexes can be added, rebuilt, or defragmented while the database is online and while end users are reading or updating data. Similarly, you can relocate or defragment tables online. Tables can be redefined, changing table types, adding, dropping or renaming columns, and changing storage parameters without interrupting end users who are viewing or updating the underlying data. Oracle Database capabilities include:

- Support for easy cloning of indexes, grants, constraints, and other characteristics of the table
- Conversion from the long to LOB datatype online
- Allowing unique indexes instead of requiring a primary key
- Ability to the modify the bodies of PL/SQL stored procedures and views that are referenced through synonyms, without recompiling other PL/SQL packages that reference them
- Ability to perform an online segment shrink to compact the space used in a segment online and in place, allowing space management to be performed without impacting system or data availability.

Partitioned Tables and Indexes

As databases grow larger, they may become extremely cumbersome to manage. The ability to partition database tables and indexes allows administrators to divide large tables up into smaller, more manageable pieces. While most operations and schema changes can be made online, partitioning allows maintenance tasks to be performed one partition at a time. This allows the bulk of the data to be unaffected during maintenance. In addition, partitions enable the use of parallel execution to perform most operations much faster.

Another benefit of partitions is fault containment. A failure, such as a media failure or corruption, is contained to partitions resident on the failed disk. Only that partition is affected and needs to be recovered. This not only reduces the time to recover, but allows the other unaffected partitions to remain online while the failed partition is recovered.

Often, not all data in a large table has the same access characteristics. Pending orders may be accessed more frequently than closed orders, or analysis of last quarter's sales may be more common than analysis of sales from a quarter three years ago. Partitioning allows for intelligent storage management of data. Frequently accessed data can be stored on the fastest disks, and heavily accessed data can be striped across many drives.

Dynamic Resource Provisioning

Oracle Database continues to broaden support for dynamic reconfiguration enabling it to adapt to changes in demand and hardware with no disruption of service. Oracle Database dynamically accommodates changes to hardware configurations such as:

- Add and remove of processors from an SMP server

- Add and remove nodes in an Oracle RAC cluster

- Dynamically grow and shrink its shared memory allocation and automatically tune memory online

- Add and remove database disks online without disturbing database activities

- Automatically rebalance IO load across the database storage

- Move datafiles online

These capabilities eliminate the impact of system changes and provide true capacity on demand provisioning that is a fundamental requirement of enterprise Grid computing.

Avoiding Downtime for System Changes

Planned system changes occur when you perform routine and periodic maintenance operations and new deployments. Planned system changes include any scheduled changes to the operating environment that occur outside of the organizational data structure in the database.

The service level impact of planned system changes varies significantly depending on:

- The nature and scope of the planned outage

- The testing and validation efforts made prior to implementing the change

- The technologies and features in place to minimize the impact

This section includes the following topics:

- Rolling Patch Updates

- Rolling Release Upgrade

Rolling Patch Updates

Oracle Database supports the application of patches to the nodes of an Oracle RAC system in a rolling fashion. To perform a rolling upgrade involves performing the following high-level steps:

1. Copy the patch software to the primary upgrade node

2. Shut down Oracle RAC instances on the upgrade nodes

3. Stop all Oracle processes on the upgrade nodes

4. Start OUI and complete the upgrade process on the upgrade nodes

An Oracle RAC system runs with all nodes actively processing transactions on the behalf of database clients. Step 1 of the patch application procedure is to quiesce the first instance to which the patch is to be applied. In step 2, you use an Oracle patch tool (Opatch) to apply the patch to the quiesced instance (for example, the Oracle Home for instance 1 is updated). In step 3, you reactivate the patched instance which then rejoins the cluster. The Oracle RAC system is now running with one instance at a higher maintenance level than the other nodes in the cluster.

An Oracle RAC system can run in this mixed mode for an arbitrary period to ensure the patch corrects the original problem, and has not introduced some other problem. This procedure is then repeated for the remaining nodes in the cluster. When all nodes in the cluster have been patched, the rolling patch update is complete and all nodes are running the same version of the Oracle Database software. In addition, Opatch has the ability to *rollback* the application of a patch. If you observe some aberrant behavior on the updated instance, you can either uninstall the offending patch or roll it back, without forcing a clusterwide outage. The rollback procedure is the same as the patch apply procedure, but in this case the Opatch utility removes a previously applied patch.

Rolling Release Upgrade

Oracle Database supports the installation of database software upgrades, and the application of patchsets, in a rolling fashion—with near zero database downtime—by using Data Guard SQL Apply and logical standby databases.

Oracle Clusterware also supports upgrades in a rolling fashion for software upgrades and application of patchsets. Additionally, once on Oracle Database 11*g*, ASM supports rolling release upgrades.

Physical standby database users can elect to add a logical standby database to the configuration for the duration of a rolling upgrade, or they can use a transient logical standby database to avoid creating a second standby database. A transient logical standby is a new feature for Data Guard in Oracle Database 11*g* that allows you to temporarily convert a physical standby database to a logical standby database for the purpose of executing a rolling database upgrade. Then, once the upgrade is complete, you can convert it back into a physical standby database. Using a transient logical standby database avoids the need to create a separate logical standby database to perform upgrades.

> **See Also:** *Oracle Data Guard Concepts and Administration* for step-by-step instructions about performing a rolling upgrade with a transient logical standby database

By supporting rolling upgrades and rolling patch updates, Oracle has eliminated a large portion of the maintenance windows database administrators reserve for administrative tasks, and enables 24x7 operation of their enterprise.

Maximum Availability Architecture (MAA) Best Practices

Operational best practices are the key to a successful implementation of IT infrastructure. Technology alone is not enough. Oracle's Maximum Availability Architecture (MAA) is a fully integrated and proven blueprint for building highly available systems. The MAA blueprint details the combined use of key Oracle Database features for high availability including Oracle RAC and Oracle Clusterware,

Data Guard, Recovery Manager, Flashback Technologies, Streams, and Enterprise Manager.

Conceptualized with the philosophy that designing an HA system involving Oracle technologies should not be complex, and should not involve guesswork, the MAA design and configuration recommendations have been extensively reviewed and tested to ensure optimum system availability and reliability. MAA also addresses the configuration and integration of other critical components of highly available systems including servers, storage, networking, and the application server. Enterprises that base their system architecture on MAA can more quickly and efficiently achieve business requirements for high availability and data protection.

You can find more information on MAA at
http://otn.oracle.com/deploy/availability/htdocs/maa.htm

18

Very Large Databases (VLDB)

This chapter contains an overview of VLDB topics, with emphasis on partitioning as a key component of the VLDB strategy. It covers the following topics:

- Introduction to Partitioning
- Overview of Partitioned Indexes
- Partitioning to Improve Performance

> **See Also:** *Oracle Database VLDB and Partitioning Guide* for more information about VLDB topics such as partitioning

> **Note:** This functionality is available only if you purchase the Partitioning option.

Introduction to Partitioning

Partitioning addresses key issues in supporting very large tables and indexes by letting you decompose them into smaller and more manageable pieces called **partitions**. SQL queries and DML statements do not need to be modified in order to access partitioned tables. However, after partitions are defined, DDL statements can access and manipulate individuals partitions rather than entire tables or indexes. This is how partitioning can simplify the manageability of large database objects. Also, partitioning is entirely transparent to applications.

Each partition of a table or index must have the same logical attributes, such as column names, datatypes, and constraints, but each partition can have separate physical attributes such as pctfree, pctused, and tablespaces.

Partitioning is useful for many different types of applications, particularly applications that manage large volumes of data. OLTP systems often benefit from improvements in manageability and availability, while data warehousing systems benefit from performance and manageability.

Partitioning offers these advantages:

- Partitioning enables data management operations such data loads, index creation and rebuilding, and backup/recovery at the partition level, rather than on the entire table. This results in significantly reduced times for these operations.

- Partitioning improves query performance. In many cases, the results of a query can be achieved by accessing a subset of partitions, rather than the entire table. For some queries, this technique (called **partition pruning**) can provide order-of-magnitude gains in performance.

■ Partitioning can significantly reduce the impact of scheduled downtime for maintenance operations.

Partition independence for partition maintenance operations lets you perform concurrent maintenance operations on different partitions of the same table or index. You can also run concurrent SELECT and DML operations against partitions that are unaffected by maintenance operations.

■ Partitioning increases the availability of mission-critical databases if critical tables and indexes are divided into partitions to reduce the maintenance windows, recovery times, and impact of failures.

■ Partitioning can be implemented without requiring any modifications to your applications. For example, you could convert a nonpartitioned table to a partitioned table without needing to modify any of the SELECT statements or DML statements which access that table. You do not need to rewrite your application code to take advantage of partitioning.

This section includes the following topics:

■ Partition Key

■ Partitioned Tables

■ Partitioned Index-Organized Tables

■ Partitioning Methods

Partition Key

Each row in a partitioned table is unambiguously assigned to a single partition. The partition key is a set of one or more columns that determines the partition for each row. Oracle Database automatically directs insert, update, and delete operations to the appropriate partition through the use of the partition key. A partition key:

■ Consists of an ordered list of 1 to 16 columns

■ Cannot contain a LEVEL, ROWID, or MLSLABEL pseudocolumn or a column of type ROWID

■ Can contain columns that are NULLable

Partitioned Tables

Tables can be partitioned into up to 1024K-1 separate partitions. Any table can be partitioned except those tables containing columns with LONG or LONG RAW datatypes. You can, however, use tables containing columns with CLOB or BLOB datatypes.

> **Note:** To reduce disk use and memory use (specifically, the buffer cache), you can store tables and partitioned tables in a compressed format inside the database. This often leads to a better scaleup for read-only operations. Table compression can also speed up query execution. There is, however, a slight cost in CPU overhead.

> **See Also:** "Table Compression" on page 16-8

Partitioned Index-Organized Tables

You can partition index-organized tables by range, list, or hash. Partitioned index-organized tables are very useful for providing improved manageability, availability, and performance for index-organized tables. In addition, data cartridges that use index-organized tables can take advantage of the ability to partition their stored data.

For partitioning an index-organized table:

- Partition columns must be a subset of primary key columns

- Secondary indexes can be partitioned—locally and globally

- OVERFLOW data segments are always equipartitioned with the table partitions

Partitioning Methods

Oracle provides the following partitioning methods:

- Range partitioning maps data to partitions based on ranges of partition key values that you establish for each partition.

- Interval partitioning is an extension of range partitioning which instructs the database to automatically create partitions of a specified interval when data inserted into the table exceeds all of the range partitions.

- Hash partitioning maps data to partitions based on a hashing algorithm that evenly distributes rows among partitions, giving partitions approximately the same size.

- List partitioning enables you to explicitly control how rows map to partitions by specifying a list of discrete values in the description for each partition.

- Reference partitioning enables you to partition a table based on the partitioning scheme of the table referenced in its referential constraint.

- Composite partitioning is a combination of two partitioning methods to further divide the data into subpartitions:

  – Composite range-range partitioning partitions data using the range method, and within each partition, subpartitions it using the range method.

  – Composite range-hash partitioning partitions data using the range method, and within each partition, subpartitions it using the hash method.

  – Composite range-list partitioning partitions data using the range method, and within each partition, subpartitions it using the list method.

  – Composite list-range partitioning partitions data using the list method, and within each partition, subpartitions it using the range method.

  – Composite list-hash partitioning partitions data using the list method, and within each partition, subpartitions it using the hash method.

  – Composite list-list partitioning partitions data using the list method, and within each partition, subpartitions it using the list method.

- System partitioning enables application-controlled partitioning for arbitrary tables.

Overview of Partitioned Indexes

Just like partitioned tables, partitioned indexes improve manageability, availability, performance, and scalability. They can either be partitioned independently (global indexes) or automatically linked to a table's partitioning method (local indexes). In general, you should use global indexes for OLTP applications and local indexes for data warehousing or DSS applications. Also, whenever possible, you should try to use local indexes because they are easier to manage. When deciding what kind of partitioned index to use, you should consider the following guidelines in order:

1. If the table partitioning column is a subset of the index keys, use a local index. If this is the case, you are finished. If this is not the case, continue to guideline 2.

2. If the index is unique, use a global index. If this is the case, you are finished. If this is not the case, continue to guideline 3.

3. If your priority is manageability, use a local index. If this is the case, you are finished. If this is not the case, continue to guideline 4.

4. If the application is an OLTP one and users need quick response times, use a global index. If the application is a DSS one and users are more interested in throughput, use a local index.

> **See Also:** *Oracle Database VLDB and Partitioning Guide* and *Oracle Database Administrator's Guide* for more information about partitioned indexes and how to decide which type to use

This section includes the following topics:

- Local Partitioned Indexes
- Global Partitioned Indexes
- Global Nonpartitioned Indexes
- Miscellaneous Information about Creating Indexes on Partitioned Tables
- Using Partitioned Indexes in OLTP Applications
- Using Partitioned Indexes in Data Warehousing and DSS Applications
- Partitioned Indexes on Composite Partitions

Local Partitioned Indexes

Local partitioned indexes are easier to manage than other types of partitioned indexes. They also offer greater availability and are common in DSS environments. The reason for this is equipartitioning: each partition of a local index is associated with exactly one partition of the table. This enables Oracle Database to automatically keep the index partitions in sync with the table partitions, and makes each table-index pair independent. Any actions that make one partition's data invalid or unavailable only affect a single partition.

Local partitioned indexes support more availability when there are partition or subpartition maintenance operations on the table. A type of index called a local nonprefixed index is very useful for historical databases. In this type of index, the partitioning is not on the left prefix of the index columns.

> **See Also:** *Oracle Database VLDB and Partitioning Guide* more information about prefixed indexes

You cannot explicitly add a partition to a local index. Instead, new partitions are added to local indexes only when you add a partition to the underlying table. Likewise, you cannot explicitly drop a partition from a local index. Instead, local index partitions are dropped only when you drop a partition from the underlying table.

A local index can be unique. However, in order for a local index to be unique, the partitioning key of the table must be part of the index's key columns. Unique local indexes are useful for OLTP environments.

Global Partitioned Indexes

Oracle Database offers two types of global partitioned index: range partitioned and hash partitioned.

This section includes the following topics:

- Global Range Partitioned Indexes
- Global Hash Partitioned Indexes
- Maintenance of Global Partitioned Indexes

Global Range Partitioned Indexes

Global range partitioned indexes are flexible in that the degree of partitioning and the partitioning key are independent from the table's partitioning method. They are commonly used for OLTP environments and offer efficient access to any individual record.

The highest partition of a global index must have a partition bound, all of whose values are MAXVALUE. This ensures that all rows in the underlying table can be represented in the index. Global prefixed indexes can be unique or nonunique.

You cannot add a partition to a global index because the highest partition always has a partition bound of MAXVALUE. If you wish to add a new highest partition, use the ALTER INDEX SPLIT PARTITION statement. If a global index partition is empty, you can explicitly drop it by issuing the ALTER INDEX DROP PARTITION statement. If a global index partition contains data, dropping the partition causes the next highest partition to be marked unusable. You cannot drop the highest partition in a global index.

Global Hash Partitioned Indexes

Global hash partitioned indexes improve performance by spreading out contention when the index is monotonically growing. In other words, most of the index insertions occur only on the right edge of an index.

Maintenance of Global Partitioned Indexes

By default, the following operations on partitions on a heap-organized table mark all global indexes as unusable:

```
ADD (HASH)
COALESCE (HASH)
DROP
EXCHANGE
MERGE
MOVE
SPLIT
TRUNCATE
```

These indexes can be maintained by appending the clause `UPDATE INDEXES` to the SQL statements for the operation. The two advantages to maintaining global indexes:

- The index remains available and online throughout the operation. Hence no other applications are affected by this operation.

- The index doesn't have to be rebuilt after the operation.

Example:

```
ALTER TABLE DROP PARTITION P1 UPDATE INDEXES;
```

> **Note:** This feature is supported only for heap-organized tables.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the `UPDATE INDEXES` clause

Global Nonpartitioned Indexes

Global nonpartitioned indexes behave just like a nonpartitioned index. They are commonly used in OLTP environments and offer efficient access to any individual record.

Miscellaneous Information about Creating Indexes on Partitioned Tables

You can create bitmap indexes on partitioned tables, with the restriction that the bitmap indexes must be local to the partitioned table. They cannot be global indexes.

Global indexes can be unique. Local indexes can only be unique if the partitioning key is a part of the index key.

Using Partitioned Indexes in OLTP Applications

Here are a few guidelines for OLTP applications:

- Global indexes and unique, local indexes provide better performance than nonunique local indexes because they minimize the number of index partition probes.

- Local indexes offer better availability when there are partition or subpartition maintenance operations on the table.

- Hash-partitioned global indexes offer better performance by spreading out contention when the index is monotonically growing. In other words, most of the index insertions occur only on the right edge of an index.

Using Partitioned Indexes in Data Warehousing and DSS Applications

Here are a few guidelines for data warehousing and DSS applications:

- Local indexes are preferable because they are easier to manage during data loads and during partition-maintenance operations.

- Local indexes can improve performance because many index partitions can be scanned in parallel by range queries on the index key.

Partitioned Indexes on Composite Partitions

Here are a few points to remember when using partitioned indexes on composite partitions:

- Subpartitioned indexes are always local and stored with the table subpartition by default.

- Tablespaces can be specified at either index or index subpartition levels.

Partitioning to Improve Performance

Partitioning can help you improve performance and manageability. Some topics to keep in mind when using partitioning for these reasons are:

- Partition Pruning
- Partition-wise Joins

Partition Pruning

Oracle Database explicitly recognizes partitions and subpartitions. It then optimizes SQL statements to mark the partitions or subpartitions that need to be accessed and eliminates (prunes) unnecessary partitions or subpartitions from access by those SQL statements. In other words, partition pruning is the skipping of unnecessary index and data partitions or subpartitions in a query.

For each SQL statement, depending on the selection criteria specified, unneeded partitions or subpartitions can be eliminated. For example, if a query only involves March sales data, then there is no need to retrieve data for the remaining eleven months. Such intelligent pruning can dramatically reduce the data volume, resulting in substantial improvements in query performance.

The SQL Access Advisor can automatically analyze the schema design for a given workload and recommend indexes, function-based indexes, partitions, and materialized views to create, retain, or drop as appropriate for the workload.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about the SQL Access Advisor

If the optimizer determines that the selection criteria used for pruning are satisfied by all the rows in the accessed partition or subpartition, it removes those criteria from the predicate list (WHERE clause) during evaluation in order to improve performance. However, the optimizer cannot prune partitions if the SQL statement applies a function to the partitioning column (with the exception of the TO_DATE function). Similarly, the optimizer cannot use an index if the SQL statement applies a function to the indexed column, unless it is a function-based index.

Pruning can eliminate index partitions even when the underlying table's partitions cannot be eliminated, but only when the index and table are partitioned on different columns. You can often improve the performance of operations on large tables by creating partitioned indexes that reduce the amount of data that your SQL statements need to access or modify.

Equality, range, LIKE, and IN-list predicates are considered for partition pruning with range or list partitioning, and equality and IN-list predicates are considered for partition pruning with hash partitioning.

Partition Pruning Example

There is a partitioned table called `cust_orders`. The partition key for `cust_orders` is `order_date`. Assume that `cust_orders` has six months of data, January to June, with a partition for each month of data. If the following query is run:

```
SELECT SUM(value)
FROM cust_orders
WHERE order_date BETWEEN '28-MAR-98' AND '23-APR-98';
```

Partition pruning is achieved by:

- First, partition elimination of January, February, May, and June data partitions. Then either:

  - An index scan of the March and April data partition due to high index selectivity

    or

  - A full scan of the March and April data partition due to low index selectivity

Partition-wise Joins

A partition-wise join is a join optimization for joining two tables that are both partitioned along the join column(s). With partition-wise joins, the join operation is broken into smaller joins that are performed sequentially or in parallel. Another way of looking at partition-wise joins is that they minimize the amount of data exchanged among parallel slaves during the execution of parallel joins by taking into account data distribution.

> **See Also:** *Oracle Database VLDB and Partitioning Guide* for more information about partitioning methods and partition-wise joins

19

Content Management

This chapter provides an overview of Oracle's content management features.

This chapter contains the following topics:

- Introduction to Content Management
- Overview of XML in Oracle Database
- Overview of LOBs
- Overview of Oracle Text
- Overview of Oracle Ultra Search
- Overview of Oracle Multimedia
- Overview of Oracle Spatial

Introduction to Content Management

Oracle Database includes datatypes to handle all the types of rich Internet content such as relational data, object-relational data, XML, text, audio, video, image, and spatial. These datatypes appear as native types in the database. They can all be queried using SQL. A single SQL statement can include data belonging to any or all of these datatypes.

As applications evolve to encompass increasingly richer semantics, they encounter the need to deal with the following kinds of data:

- Simple structured data
- Complex structured data
- Semi-structured data
- Unstructured data

Traditionally, the relational model has been very successful at dealing with simple structured data—the kind which can fit into simple tables. Oracle added object-relational features so that applications can deal with complex structured data—collections, references, user-defined types, and so on. Queuing technologies, such as Oracle Streams Advanced Queuing, deal with messages and other semi-structured data. This chapter discusses Oracle's technologies to support unstructured data.

Unstructured data cannot be decomposed into standard components. Data about an employee can be "structured" into a name (probably a character string), an identification (likely a number), a salary, and so on. But if you are given a photo, you find that the data really consists of a long stream of 0s and 1s. These 0s and 1s are used

to switch pixels on or off, so that you see the photo on a display, but it cannot be broken down into any finer structure in terms of database storage.

Unstructured data such as text, graphic images, still video clips, full motion video, and sound waveforms tend to be large -- a typical employee record may be a few hundred bytes, but even small amounts of multimedia data can be thousands of times larger. Some multimedia data may reside on operating system files, and it is desirable to access them from the database.

Overview of XML in Oracle Database

Extensible Markup Language (XML) is a tag-based markup language that lets developers create their own tags to describe data that's exchanged between applications and systems. XML is widely adopted as the common language of information exchange between companies. It is human-readable; that is, it is plain text. Because it is plain text, XML documents and XML-based messages can be sent easily using common protocols, such as HTTP or FTP.

Oracle XML DB treats XML as a native datatype in the database. Oracle XML DB is not a separate server. The XML data model encompasses both unstructured content and structured data. Applications can use standard SQL and XML operators to generate complex XML documents from SQL queries and to store XML documents.

Oracle XML DB provides capabilities for both content-oriented and data-oriented access. For developers who see XML as documents (news stories, articles, and so on), Oracle XML DB provides an XML repository accessible from standard protocols and SQL.

For others, the structured-data aspect of XML (invoices, addresses, and so on) is more important. For these users, Oracle XML DB provides a native XMLType, support for XML Schema, XPath, XSLT, DOM, and so on. The data-oriented access is typically more query-intensive.

The Oracle XML developer's kits (XDK) contain the basic building blocks for reading, manipulating, transforming, and viewing XML documents, whether on a file system or stored in a database. They are available for Java, C, and C++. Unlike many shareware and trial XML components, the production Oracle XDKs are fully supported and come with a commercial redistribution license. Oracle XDKs consist of the following components:

- XML Parsers: supporting Java, C, and C++, the components create and parse XML using industry standard DOM and SAX interfaces.

- XSLT Processor: transforms or renders XML into other text-based formats, such as HTML.

- XML Schema Processor: supporting Java, C, and C++, allows use of XML simple and complex datatypes.

- XML Class Generator: automatically generates Java and C++ classes from XSL schemas to send XML data from Web forms or applications.

- XML Java Beans: visually view and transform XML documents and data with Java components.

- XML SQL Utility: supporting Java, generates XML documents, DTDs, and schemas from SQL queries.

- XSQL Servlet: combines XML, SQL, and XSLT in the server to deliver dynamic Web content.

See Also:

- *Oracle XML DB Developer's Guide*

- *Oracle XML Developer's Kit Programmer's Guide*

- *Oracle XML Developer's Kit API Reference*

- *Oracle Database XML Java API Reference*

Overview of LOBs

The large object (LOB) datatypes `BLOB`, `CLOB`, `NCLOB`, and `BFILE` enable you to store and manipulate large blocks of unstructured data (such as text, graphic images, video clips, and sound waveforms) in binary or character format. They provide efficient, random, piece-wise access to the data.

With the growth of the internet and content-rich applications, it has become imperative that databases support a datatype that fulfills the following:

- Can store unstructured data with compression, encryption, or deduplication.

- Is optimized for large amounts of such data: up to 128 terabytes.

- Provides a uniform way of accessing large unstructured data within the database or outside in operating system files which are read-only.

See Also: "Overview of LOB Datatypes" on page 26-11

For the LOBs with `STORE AS SECUREFILE` option (introduced in release 11.1) you can specify the SQL parameter `DEDUPLICATE` in `CREATE TABLE` and `ALTER TABLE` statements. This enables you to specify that LOB data that are identical in two or more rows in a LOB column will all share the same data blocks, thus saving disk space. `KEEP_DUPLICATES` turns off this capability. The following options are also used with `SECUREFILE`:

The parameter `COMPRESS` turns on LOB compression. `NOCOMPRESS` turns LOB compression off.

The parameter `ENCRYPT` turns on LOB encryption and optionally selects an encryption algorithm. `NOENCRYPT` turns off LOB encryption.

The pre-release 11.1 LOBs paradigm is the default. It is also now explicitly set by the option `STORE AS BASICFILE`.

The following SQL and PL/SQL statements, and OCI functions are used with the `SECUREFILE` features:

See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide*, for complete details of relevant SQL functions and cross-references to PL/SQL packages.

- *Oracle Call Interface Programmer's Guide*, functions `OCILobGetDuplicateRegions()`, `OCILobSetOptions()`, and `OCILobGetOptions()`

Overview of Oracle Text

Oracle Text indexes any document or textual content to add fast, accurate retrieval of information to internet content management applications, e-Business catalogs, news

services, job postings, and so on. It can index content stored in file systems, databases, or on the Web.

Oracle Text allows text searches to be combined with regular database searches in a single SQL statement. It can find documents based on their textual content, metadata, or attributes. The Oracle Text SQL API makes it simple and intuitive to create and maintain Text indexes and run Text searches.

Oracle Text is completely integrated with Oracle Database, making it inherently fast and scalable. The Text index is in the database, and Text queries are run in the Oracle Database process. The Oracle Database optimizer can choose the best execution plan for any query, giving the best performance for ad hoc queries involving Text and structured criteria. Additional advantages include the following:

- Oracle Text supports multilingual querying and indexing.

- You can index and define sections for searching in XML documents. Section searching lets you narrow down queries to blocks of text within documents. Oracle Text can automatically create XML sections for you.

- A Text index can span many Text columns, giving the best performance for Text queries across more than one column.

- Oracle Text has enhanced performance for operations that are common in Text searching, like count hits.

- Oracle Text leverages scalability features, such as replication.

- Oracle Text supports local partitioned index.

This section includes the following topics:

- Oracle Text Index Types

- Oracle Text Document Services

- Oracle Text Query Package

- Oracle Text Advanced Features

Oracle Text Index Types

There are three Oracle Text index types to cover all text search needs.

- Standard index type for traditional full-text retrieval over documents and Web pages. The context index type provides a rich set of text search capabilities for finding the content you need, without returning pages of spurious results.

- Catalog index type, designed specifically for e-Business catalogs. This catalog index provides flexible searching and sorting at Web-speed.

- Classification index type for building classification or routing applications. This index is created on a table of queries, where the queries define the classification or routing criteria.

Oracle Text also provides substring and prefix indexes. Substring indexing improves performance for left-truncated or double-truncated wildcard queries. Prefix indexing improves performance for right truncated wildcard queries.

Oracle Text Document Services

Oracle Text provides a number of utilities to view text, no matter how that text is stored.

- Oracle Text supports over 150 document formats through its Inso filtering technology, including all common document formats like XML, PDF, and MS Office. You can also create your own custom filter.

- You can view the HTML version of any text, including formatted documents such as PDF, MS Office, and so on.

- You can view the HTML version of any text, with search terms highlighted and with navigation to next/previous term in the text.

- Oracle Text provides markup information; for example, the offset and length of each search term in the text, to be used for example by a third party viewer.

Oracle Text Query Package

The CTX_QUERY PL/SQL package can be used to generate query feedback, count hits, and create stored query expressions.

> **See Also:** *Oracle Text Reference* for information about this package

Oracle Text Advanced Features

With Oracle Text, you can find, classify, and cluster documents based on their text, metadata, or attributes.

Document classification performs an action based on document content. Actions can be assigned category IDs to a document for future lookup or for sending a document to a user. The result is a set, or stream, of categorized documents. For example, assume that there is an incoming stream of news articles. You can define a rule to represent the category of Finance. The rule is essentially one or more queries that select documents about the subject of finance. The rule might have the form "stocks or bonds or earnings." When a document arrives that satisfies the rules for this category, the application takes an action, such as tagging the document as Finance or e-mailing one or more users.

Clustering is the unsupervised division of patterns into groups. The interface lets users select the appropriate clustering algorithm. Each cluster contains a subset of documents of the collection. A document within a cluster is believed to be more similar with documents inside the cluster than with outside documents. Clusters can be used to build features like presenting similar documents in the collection.

> **See Also:** *Oracle Text Application Developer's Guide*

Overview of Oracle Ultra Search

Oracle Ultra Search is built on Oracle Database and Oracle Text technology that provides uniform search-and-locate capabilities over multiple repositories: Oracle Databases, other ODBC compliant databases, IMAP mail servers, HTML documents served up by a Web server, files on disk, and more.

Oracle Ultra Search uses a *crawler* to index documents; the documents stay in their own repositories, and the crawled information is used to build an index that stays within your firewall in a designated Oracle database. Oracle Ultra Search also provides APIs for building content management solutions.

Oracle Ultra Search offers the following:

- A complete text query language for text search inside the database

- Full integration with Oracle Database and the SQL query language

- Advanced features like concept searching and theme analysis

- Indexing of all common file formats (150+)

- Full globalization, including support for Chinese, Japanese and Korean (CJK), and Unicode

> **See Also:** *Oracle Ultra Search Administrator's Guide*

Overview of Oracle Multimedia

Oracle Multimedia (formerly known as Oracle *inter*Media) is a feature that enables Oracle Database to store, manage, and retrieve images, Digital Imaging and Communications in Medicine (DICOM), audio, and video data in an integrated fashion with other enterprise information. Oracle Multimedia extends Oracle Database reliability, availability, and data management to media content and medical image content in traditional, medical, Internet, electronic commerce, and media-rich applications.

Oracle Multimedia manages media content by providing the following:

- Storage and retrieval of media data in the database to synchronize the media data with the associated business data

- Support for popular image, audio, and video formats

- Extraction of format and application metadata into XML documents

- Full object and relational interfaces to Oracle Multimedia services

- Access through traditional and Web interfaces

- Querying using associated relational data and extracted metadata

- Image processing, such as thumbnail generation

- Delivery through RealNetworks and Windows Media Streaming Servers

Oracle Multimedia manages DICOM content by providing the following:

- Storage and retrieval of medical imaging data in the database to synchronize the DICOM data with the associated business data

- Full object and relational interfaces to Oracle Multimedia DICOM services

- Extraction of DICOM metadata into user-specifiable XML documents

- Querying using associated relational data and extracted metadata

- Image processing, such as thumbnail generation

- Creation of new DICOM objects

- Conformance validation based on a set of user-specified conformance rules

- Making DICOM objects anonymous based on user-defined rules that specify the set of attributes to be made anonymous and how to make those attributes anonymous

- The ability to update run-time behaviors, such as the version of the DICOM standard supported, without installing a new release of Oracle Database

See Also:

- *Oracle Multimedia User's Guide*

- *Oracle Multimedia Reference*

- *Oracle Multimedia DICOM Developer's Guide*

- *Oracle Multimedia Java API Reference*

- *Oracle Multimedia Servlets and JSP Java API Reference*

- *Oracle Multimedia DICOM Java API Reference*

Overview of Oracle Spatial

Oracle Spatial is designed to make spatial data management easier and more natural to users of location-enabled applications and geographic information system (GIS) applications. When spatial data is stored in Oracle Database, it can be easily manipulated, retrieved, and related to all other data stored in the database.

A common example of spatial data can be seen in a road map. A road map is a two-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information. The location of cities, roads, and political boundaries that exist on the surface of the Earth are projected onto a two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects.

The data that indicates the Earth location (such as longitude and latitude) of these rendered objects is the spatial data. When the map is rendered, this spatial data is used to project the locations of the objects on a two-dimensional piece of paper. A GIS is often used to store, retrieve, and render this Earth-relative spatial data.

Types of spatial data (other than GIS data) that can be stored using Spatial include data from computer-aided design (CAD) and computer-aided manufacturing (CAM) systems. Instead of operating on objects on a geographic scale, CAD/CAM systems work on a smaller scale, such as for an automobile engine or printed circuit boards.

The differences among these systems are in the size and precision of the data, not the data's complexity. The systems might all involve the same number of data points. On a geographic scale, the location of a bridge can vary by a few tenths of an inch without causing any noticeable problems to the road builders, whereas if the diameter of an engine's pistons is off by a few tenths of an inch, the engine will not run.

In addition, the complexity of data is independent of the absolute scale of the area being represented. For example, a printed circuit board is likely to have many thousands of objects etched on its surface, containing in its small area information that may be more complex than the details shown on a road builder's blueprints.

These applications all store, retrieve, update, or query some collection of features that have both nonspatial and spatial attributes. Examples of nonspatial attributes are name, soil_type, landuse_classification, and part_number. The spatial attribute is a coordinate geometry, or vector-based representation of the shape of the feature.

Oracle Spatial provides a SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial features in Oracle Database. Spatial consists of the following:

- A schema (MDSYS) that prescribes the storage, syntax, and semantics of supported geometric datatypes

- A spatial indexing mechanism
- Operators, functions, and procedures for performing area-of-interest queries, spatial join queries, and other spatial analysis operations
- Functions and procedures for utility and tuning operations
- Topology data model for working with data about nodes, edges, and faces in a topology.
- Network data model for representing capabilities or objects that are modeled as nodes and links in a network.
- GeoRaster, a feature that lets you store, index, query, analyze, and deliver GeoRaster data, that is, raster image and gridded data and its associated metadata.

> **See Also:** *Oracle Spatial GeoRaster Developer's Guide* and *Oracle Spatial Topology and Network Data Models Developer's Guide*

20

Database Security

This chapter provides an overview of Oracle Database database security.

This chapter contains the following topics:

- Introduction to Database Security
- Overview of Transparent Data Encryption
- Overview of Authentication Methods
- Overview of Authorization
- Overview of Access Restrictions on Tables, Views, Synonyms, or Rows
- Overview of Security Policies
- Overview of Database Auditing

> **See Also:** *Oracle Database Security Guide* for more detailed information on everything in this chapter

Introduction to Database Security

Database security entails allowing or disallowing user actions on the database and the objects within it. Oracle Database uses schemas and security domains to control access to data and to restrict the use of various database resources.

Oracle Database provides comprehensive discretionary access control. **Discretionary access control** regulates all user access to named objects through privileges. A privilege is permission to access a named object in a prescribed manner; for example, permission to query a table. Privileges are granted to users at the discretion of other users.

This section includes the following topics:

- Database Users and Schemas
- Privileges
- Roles
- Storage Settings and Quotas

Database Users and Schemas

Each Oracle database has a list of user names. To access a database, a user must use a database application and attempt a connection with a valid user name of the database. Each user name has an associated password to prevent unauthorized use.

Security Domain

Each user has a **security domain**—a set of properties that determine such things as:

- The actions (privileges and roles) available to the user

- The tablespace quotas (available disk space) for the user

- The system resource limits (for example, CPU processing time) for the user

Each property that contributes to a user's security domain is discussed in the following sections.

Privileges

A **privilege** is a right to run a particular type of SQL statement. Some examples of privileges include the right to:

- Connect to the database (create a session)

- Create a table in your schema

- Select rows from someone else's table

- Run someone else's stored procedure

> **See Also:** "Introduction to Privileges" on page 20-12

Roles

Oracle Database provides for easy and controlled privilege management through roles. **Roles** are named groups of related privileges that you grant to users or other roles.

> **See Also:** "Introduction to Roles" on page 20-13 information about role properties

Storage Settings and Quotas

You can direct and limit the use of disk space allocated to the database for each user, including default and temporary tablespaces and tablespace quotas.

This section includes the following topics:

- Default Tablespace

- Temporary Tablespace

- Tablespace Quotas

- Profiles and Resource Limits

Default Tablespace

Each user is associated with a **default tablespace**. When a user creates a table, index, or cluster and no tablespace is specified to physically contain the schema object, the user's default tablespace is used if the user has the privilege to create the schema object and a quota in the specified default tablespace. The default tablespace provides Oracle Database with information to direct space use in situations where schema object's location is not specified.

Temporary Tablespace

Each user has a **temporary tablespace**. When a user runs a SQL statement that requires the creation of temporary segments (such as the creation of an index), the user's

temporary tablespace is used. By directing all users' temporary segments to a separate tablespace, the temporary tablespace can reduce I/O contention among temporary segments and other types of segments.

Tablespace Quotas

Oracle Database can limit the collective amount of disk space available to the objects in a schema. **Quotas** (space limits) can be set for each tablespace available to a user. This permits selective control over the amount of disk space that can be consumed by the objects of specific schemas.

Profiles and Resource Limits

Each user is assigned a **profile** that specifies limitations on several system resources available to the user, including the following:

- Number of concurrent sessions the user can establish

- CPU processing time available for the user's session and a single call to Oracle Database made by a SQL statement

- Amount of logical I/O available for the user's session and a single call to Oracle Database made by a SQL statement

- Amount of idle time available for the user's session

- Amount of connect time available for the user's session

- Password restrictions:

  - Account locking after multiple unsuccessful login attempts

  - Password expiration and grace period

  - Password reuse and complexity restrictions

    See Also:

Overview of Transparent Data Encryption

Oracle Database provides security in the form of authentication, authorization, and auditing. Authentication ensures that only legitimate users gain access to the system. Authorization ensures that those users only have access to resources they are permitted to access. Auditing ensures accountability when users access protected resources. Although these security mechanisms effectively protect data in the database, they do not prevent access to the operating system files where the data is stored.

Transparent data encryption enables encryption of sensitive data in database columns as it is stored in the operating system files. In addition, it provides for secure storage and management of encryption keys in a security module external to the database.

Using an external security module separates ordinary program functions from those that pertain to security, such as encryption. Consequently, it is possible to divide administration duties between DBAs and security administrators, a strategy that enhances security because no administrator is granted comprehensive access to data. External security modules generate encryption keys, perform encryption and decryption, and securely store keys outside of the database.

Transparent data encryption is a key-based access control system that enforces authorization by encrypting data with a key that is kept secret. There can be only one key for each database table that contains encrypted columns regardless of the number

of encrypted columns in a given table. Each table's column encryption key is, in turn, encrypted with the database server's master key. No keys are stored in the database. Instead, they are stored in an Oracle wallet, which is part of the external security module.

Before you can encrypt any database columns, you must generate or set a master key. This master key is used to encrypt the column encryption key which is generated automatically when you issue a SQL command with the ENCRYPT clause on a database column.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for details about using transparent data encryption

Tablespace Encryption

Tablespace encryption is a new feature introduced in this release. Tablespace encryption enables you to encrypt an entire tablespace. This secures all data stored in the tablespace. When an authorized user accesses data in the tablespace, the data is transparently decrypted for him.

Tablespace encryption eliminates the need for granular analysis of applications to determine which columns to encrypt. You can use tablespace encryption to encrypt entire tables that might contain sensitive data.

Transparent encryption/decryption takes place during disk input/output (I/O) and not for every logical access to the data. This leads to improved performance.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for details about using tablespace encryption

Overview of Authentication Methods

Authentication means verifying the identity of someone (a user, device, or other entity) who wants to use data, resources, or applications. Validating that identity establishes a trust relationship for further interactions. Authentication also enables accountability by making it possible to link access and actions to specific identities. After authentication, authorization processes can allow or limit the levels of access and action permitted to that entity.

For simplicity, the same authentication method is generally used for all database users, but Oracle Database allows a single database instance to use any or all methods. Oracle Database requires special authentication procedures for database administrators, because they perform special database operations. Oracle Database also encrypts passwords during transmission to ensure the security of network authentication.

To validate the identity of database users and prevent unauthorized use of a database user name, you can authenticate using any combination of the methods described in the following sections:

- Authentication by the Operating System

- Authentication by the Network

- Authentication by Oracle Database

- Multitier Authentication and Authorization

- Authentication by the Secure Socket Layer Protocol

- Authentication of Database Administrators

Authentication by the Operating System

Some operating systems let Oracle Database use information they maintain to authenticate users, with the following benefits:

- Once authenticated by the operating system, users can connect to Oracle Database more conveniently, without specifying a user name or password. For example, an operating-system-authenticated user can invoke SQL*Plus and skip the user name and password prompts by entering the following:

  ```
  SQLPLUS /
  ```

- With control over user authentication centralized in the operating system, Oracle Database need not store or manage user passwords, though it still maintains user names in the database.

- Audit trails in the database and operating system use the same user names.

When an operating system is used to authenticate database users, managing distributed database environments and database links requires special care.

Authentication by the Network

Oracle Database supports the following methods of authentication by the network:

- Third Party-Based Authentication Technologies
- Public-Key-Infrastructure-Based Authentication
- Remote Authentication

> **Note:** These methods require Oracle Database Enterprise Edition with the Oracle Advanced Security option.

Third Party-Based Authentication Technologies

If network authentication services are available to you (such as DCE, Kerberos, or SESAME), Oracle Database can accept authentication from the network service. If you use a network authentication service, then some special considerations arise for network roles and database links.

Public-Key-Infrastructure-Based Authentication

Authentication systems based on public key cryptography issue digital certificates to user clients, which use them to authenticate directly to servers in the enterprise without directly involving an authentication server. Oracle Database provides a public key infrastructure (PKI) for using public keys and certificates, consisting of the following components:

- Authentication and secure session key management using Secure Sockets Layer (SSL).
- Oracle Call Interface (OCI) and PL/SQL functions to sign user-specified data using a private key and certificate, and verify the signature on data using a trusted certificate.
- Trusted certificates, identifying third-party entities that are trusted as signers of user certificates when an identity is being validated as the entity it claims to be.
- Oracle wallets, which are data structures that contain a user private key, a user certificate, and the user's set of trust points (trusted certificate authorities).

- Oracle Wallet Manager, a standalone Java application used to manage and edit the security credentials in Oracle wallets.

- X.509v3 certificates obtained from (and signed by) a trusted entity, a certificate authority outside of Oracle Database.

- Oracle Internet Directory to manage security attributes and privileges for users, including users authenticated by X.509 certificates. It enforces attribute-level access control and enables read, write, or update privileges on specific attributes to be restricted to specific named users, such as administrators.

- Oracle Enterprise Security Manager, provides centralized privilege management to make administration easier and increase your level of security. This lets you store and retrieve roles from Oracle Internet Directory.

- Oracle Enterprise Login Assistant, a Java-based tool to open and close a user wallet to enable or disable secure SSL-based communications for an application.

Remote Authentication

Oracle Database supports remote authentication of users through Remote Dial-In User Service (RADIUS), a standard lightweight protocol used for user authentication, authorization, and accounting.

Authentication by Oracle Database

Oracle Database can authenticate users attempting to connect to a database by using information stored in that database.

To set up Oracle Database to use database authentication, create each user with an associated password that must be supplied when the user attempts to establish a connection. This prevents unauthorized use of the database, since the connection will be denied if the user provides an incorrect password. Oracle Database stores a user's password in the data dictionary in an encrypted format to prevent unauthorized alteration, but a user can change the password at any time.

Database authentication includes the following facilities:

- Password Encryption

- Account Locking

- Password Lifetime and Expiration

- Password Complexity Verification

Password Encryption

To protect password confidentiality, Oracle Database always encrypts passwords before sending them over the network. Oracle Database encrypts the passwords using a modified AES (Advanced Encryption Standard) algorithm.

Account Locking

Oracle Database can lock a user's account after a specified number of consecutive failed log-in attempts. You can configure the account to unlock automatically after a specified time interval or to require database administrator intervention to be unlocked. The database administrator can also lock accounts manually, so that they must be unlocked explicitly by the database administrator.

Password Lifetime and Expiration

The database administrator can specify a lifetime for passwords, after which they expire and must be changed before account login is again permitted. A grace period can be established, during which each attempt to login to the database account receives a warning message to change the password. If it is not changed by the end of that period, then the account is locked. No further logins to that account are allowed without assistance by the database administrator.

The database administrator can also set the password state to expired, causing the user's account status to change to expired. The user or the database administrator must then change the password before the user can log in to the database.

The password history option checks each newly specified password to ensure that a password is not reused for a specified amount of time or for a specified number of password changes.

Password Complexity Verification

Complexity verification checks that each password is complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.

The Oracle Database default password complexity verification routine checks that each password meet the following requirements:

- Be a minimum of four characters in length

- Not equal the userid

- Include at least one alphabet character, one numeric character, and one punctuation mark

- Not match any word on an internal list of simple words like welcome, account, database, user, and so on

- Differ from the previous password by at least three characters

Multitier Authentication and Authorization

In a multitier environment, Oracle Database controls the security of middle-tier applications by limiting their privileges, preserving client identities through all tiers, and auditing actions taken on behalf of clients. In applications that use a heavy middle tier, such as a transaction processing monitor, the identity of the client connecting to the middle tier must be preserved. Yet one advantage of a middle tier is **connection pooling**, which allows multiple users to access a data server without each of them needing a separate connection. In such environments, you must be able to set up and break down connections very quickly.

For these environments, Oracle database administrators can use the Oracle Call Interface (OCI) to create **lightweight sessions**, allowing database password authentication for each user. This preserves the identity of the real user through the middle tier without the overhead of a separate database connection for each user.

You can create lightweight sessions with or without passwords. However, if a middle tier is outside or on a firewall, then security is better when each lightweight session has its own password. For an internal application server, lightweight sessions without passwords might be appropriate.

Oracle Database 11*g* enables you to implement server-side connection pooling. This allows different applications and application processes to share database connections.

Server-side connection pooling supports only password based authentication. Advanced Security Option (ASO) and enterprise users are currently not supported.

> **See Also:** *Oracle Database Administrator's Guide* and *Oracle Call Interface Programmer's Guide* for more details on server-side connection pooling

Authentication by the Secure Socket Layer Protocol

The Secure Socket Layer (SSL) protocol is an application layer protocol. Users identified either externally or globally (external or global users) can authenticate to a database through SSL.

Authentication of Database Administrators

Database administrators perform special operations (such as shutting down or starting up a database) that should not be performed by normal database users. Oracle Database provides a more secure authentication scheme for database administrator user names.

You can choose between operating system authentication or password files to authenticate database administrators. Figure 20–1 illustrates the choices you have for database administrator authentication schemes. Different choices apply to administering your database locally (on the computer where the database resides) and to administering many different database computers from a single remote client.

Figure 20–1   Database Administrator Authentication Methods
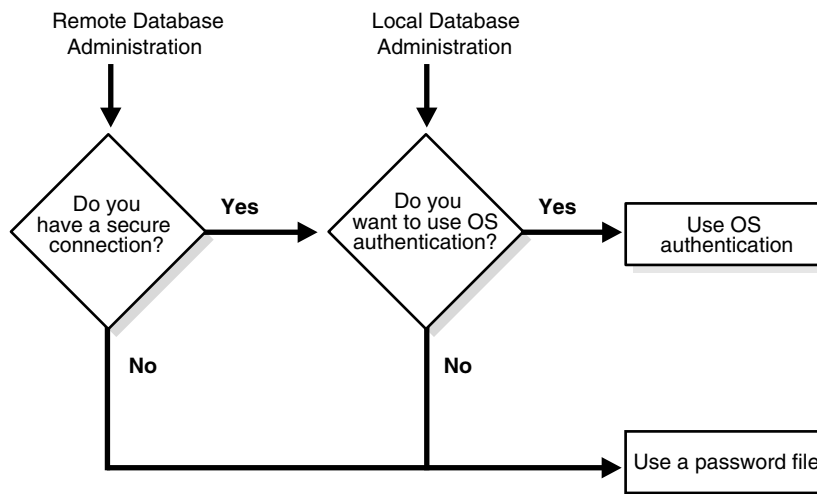
Operating system authentication for a database administrator typically involves placing his operating system user name in a special group or giving it a special process right. (On UNIX systems, the group is the **dba** group.)

The database uses password files to keep track of database user names that have been granted the SYSDBA and SYSOPER privileges, enabling the following operations:

- SYSOPER lets database administrators perform STARTUP, SHUTDOWN, ALTER DATABASE OPEN/MOUNT, ALTER DATABASE BACKUP, ARCHIVE LOG, and RECOVER, and includes the RESTRICTED SESSION privilege.

- SYSDBA contains all system privileges with ADMIN OPTION, and the SYSOPER system privilege. Permits CREATE DATABASE and time-based recovery.

   See Also:

   - *Oracle Database Administrator's Guide* for information on authentication and distributed database concepts

   - *Oracle Database Advanced Security Administrator's Guide* for information about the Oracle Advanced Security option

   - Your Oracle Database operating system-specific documentation for information about authenticating

Overview of Authorization

Authorization primarily includes two processes:

- Permitting only certain users to access, process, or alter data

- Applying varying limitations on users' access or actions. The limitations placed on (or removed from) users can apply to objects, such as schemas, tables, or rows; or to resources, such as time (CPU, connect, or idle times).

This section introduces the basic concepts and mechanisms for placing or removing such limitations on users, individually or in groups.

This section includes the following topics:

- User Resource Limits and Profiles

- Introduction to Privileges

- Introduction to Roles

- Secure Application Roles

User Resource Limits and Profiles

You can set limits on the amount of various system resources available to each user as part of a user's security domain. By doing so, you can prevent the uncontrolled consumption of valuable system resources such as CPU time.

This is very useful in large, multiuser systems, where system resources are expensive. Excessive consumption of resources by one or more users can detrimentally affect the other users of the database.

Manage a user's resource limits and password management preferences with his or her profile—a named set of resource limits that you can assign to that user. Each database can have an unlimited number of profiles. The security administrator can enable or disable the enforcement of profile resource limits universally.

If you set resource limits, then a slight degradation in performance occurs when users create sessions. This is because Oracle Database loads all resource limit data for the user when a user connects to a database.

   See Also: *Oracle Database Administrator's Guide* for information about security administrators

Resource limits and profiles are discussed in the following sections:

- Types of System Resources and Limits

- Profiles

Types of System Resources and Limits

Oracle Database can limit the use of several types of system resources, including CPU time and logical reads. In general, you can control each of these resources at the session level, the call level, or both.

This section includes the following topics:

- Session Level
- Call Level
- CPU Time
- Logical Reads
- Other Resources

Session Level Each time a user connects to a database, a session is created. Each session consumes CPU time and memory on the computer that runs Oracle Database. You can set several resource limits at the session level.

If a user exceeds a session-level resource limit, Oracle Database terminates (rolls back) the current statement and returns a message indicating that the session limit has been reached. At this point, all previous statements in the current transaction are intact, and the only operations the user can perform are COMMIT, ROLLBACK, or disconnect (in this case, the current transaction is committed). All other operations produce an error. Even after the transaction is committed or rolled back, the user can accomplish no more work during the current session.

Call Level Each time a SQL statement is run, several steps are taken to process the statement. During this processing, several calls are made to the database as part of the different execution phases. To prevent any one call from using the system excessively, Oracle Database lets you set several resource limits at the call level.

If a user exceeds a call-level resource limit, Oracle Database halts the processing of the statement, rolls back the statement, and returns an error. However, all previous statements of the current transaction remain intact, and the user's session remains connected.

CPU Time When SQL statements and other types of calls are made to Oracle Database, an amount of CPU time is necessary to process the call. Average calls require a small amount of CPU time. However, a SQL statement involving a large amount of data or a runaway query can potentially consume a large amount of CPU time, reducing CPU time available for other processing.

To prevent uncontrolled use of CPU time, limit the CPU time for each call and the total amount of CPU time used for Oracle Database calls during a session. Limits are set and measured in CPU one-hundredth seconds (0.01 seconds) used by a call or a session.

Logical Reads Input/output (I/O) is one of the most expensive operations in a database system. SQL statements that are I/O intensive can monopolize memory and disk use and cause other database operations to compete for these resources.

To prevent single sources of excessive I/O, Oracle Database lets you limit the logical data block reads for each call and for each session. Logical data block reads include data block reads from both memory and disk. The limits are set and measured in number of block reads performed by a call or during a session.

Other Resources Oracle Database also provides for the limitation of several other resources at the session level:

- You can limit the number of **concurrent sessions for each user**. Each user can create only up to a predefined number of concurrent sessions.

- You can limit the **idle time** for a session. If the time between Oracle Database calls for a session reaches the idle time limit, then the current transaction is rolled back, the session is aborted, and the resources of the session are returned to the system. The next call receives an error that indicates the user is no longer connected to the instance. This limit is set as a number of elapsed minutes.

  Shortly after a session is aborted because it has exceeded an idle time limit, the process monitor (PMON) background process cleans up after the aborted session. Until PMON completes this process, the aborted session is still counted in any session/user resource limit.

- You can limit the elapsed connect time for each session. If a session's duration exceeds the elapsed time limit, then the current transaction is rolled back, the session is dropped, and the resources of the session are returned to the system. This limit is set as a number of elapsed minutes.

  Oracle Database does not constantly monitor the elapsed idle time or elapsed connection time. Doing so would reduce system performance. Instead, it checks every few minutes. Therefore, a session can exceed this limit slightly (for example, by five minutes) before Oracle Database enforces the limit and aborts the session.

- You can limit the amount of private SGA space (used for private SQL areas) for a session. This limit is only important in systems that use the shared server configuration. Otherwise, private SQL areas are located in the PGA. This limit is set as a number of bytes of memory in an instance's SGA. Use the characters **K** or **M** to specify kilobytes or megabytes.

  > **See Also:** *Oracle Database Administrator's Guide* for instructions about enabling and disabling resource limits

Profiles

In the context of system resources, a profile is a named set of specified resource limits that can be assigned to a valid user name in Oracle Database. Profiles provide for easy management of resource limits. Profiles are also the way in which you administer password policy.

Different profiles can be created and assigned individually to each user of the database. A default profile is present for all users not explicitly assigned a profile. The resource limit feature prevents excessive consumption of global database system resources.

This section includes the following topics:

- When to Use Profiles

- Determine Values for Resource Limits of a Profile

When to Use Profiles You need to create and manage user profiles only if resource limits are a requirement of your database security policy. To use profiles, first categorize the related types of users in a database. Just as roles are used to manage the privileges of related users, profiles are used to manage the resource limits of related users. Determine how many profiles are needed to encompass all types of users in a database and then determine appropriate resource limits for each profile.

Determine Values for Resource Limits of a Profile   Before creating profiles and setting the resource limits associated with them, determine appropriate values for each resource limit. You can base these values on the type of operations a typical user performs. Usually, the best way to determine the appropriate resource limit values for a given user profile is to gather historical information about each type of resource usage.

You can gather statistics for other limits using the Monitor feature of Oracle Enterprise Manager (or SQL*Plus), specifically the Statistics monitor.

Introduction to Privileges

A **privilege** is a right to run a particular type of SQL statement or to access another user's object.

Grant privileges to users so that they can accomplish tasks required for their job. Grant privileges only to users who absolutely require them. Excessive granting of unnecessary privileges can compromise security. A user can receive a privilege in two different ways:

- You can grant privileges to users explicitly. For example, you can explicitly grant the privilege to insert records into the employees table to the user SCOTT.

- You can grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, you can grant the privileges to select, insert, update, and delete records from the employees table to the role named clerk, which in turn you can grant to the users scott and brian.

Because roles allow for easier and better management of privileges, you should generally grant privileges to roles and not to specific users.

There are two distinct categories of privileges:

- System Privileges
- Schema Object Privileges

> **See Also:**   *Oracle Database Administrator's Guide* for a list of all system and schema object privileges, as well as instructions for privilege management

System Privileges

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges. There are over 100 distinct system privileges.

Schema Object Privileges

A **schema object privilege** is a privilege or right to perform a particular action on a specific schema object:

Different object privileges are available for different types of schema objects. For example, the privilege to delete rows from the departments table is an object privilege.

Some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges. Their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege.

A schema object and its synonym are equivalent with respect to privileges. That is, the object privileges granted for a table, view, sequence, procedure, function, or package apply whether referencing the base object by name or using a synonym.

Granting object privileges on a table, view, sequence, procedure, function, or package to a **synonym** for the object has the same effect as if no synonym were used. When a synonym is dropped, all grants for the underlying schema object remain in effect, even if the privileges were granted by specifying the dropped synonym.

> **See Also:** *Oracle Database Security Guide* for more information about schema object privileges

Introduction to Roles

Managing and controlling privileges is made easier by using **roles**, which are named groups of related privileges that you grant, as a group, to users or other roles. Within a database, each role name must be unique, different from all user names and all other role names. Unlike schema objects, roles are not contained in any schema. Therefore, a user who creates a role can be dropped with no effect on the role.

Roles ease the administration of end-user system and schema object privileges. However, roles are not meant to be used by application developers, because the privileges to access schema objects within stored programmatic constructs must be granted directly.

Table 20–1 lists properties of roles that enable easier privilege management within a database.

Table 20–1   Properties of Roles

Property	Description
Reduced privilege administration	Rather than granting the same set of privileges explicitly to several users, you can grant the privileges for a group of related users to a role, and then only the role needs to be granted to each member of the group.
Dynamic privilege management	If the privileges of a group must change, then only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role.
Selective availability of privileges	You can selectively enable or disable the roles granted to a user. This allows specific control of a user's privileges in any given situation.
Application awareness	The data dictionary records which roles exist, so you can design applications to query the dictionary and automatically enable (or disable) selective roles when a user attempts to run the application by way of a given user name.
Application-specific security	You can protect role use with a password. Applications can be created specifically to enable a role when supplied the correct password. Users cannot enable the role if they do not know the password.

Database administrators often create roles for a database application. The DBA grants a secure application role all privileges necessary to run the application. The DBA then grants the secure application role to other roles or users. An application can have several different roles, each granted a different set of privileges that allow for more or less data access while using the application.

The DBA can create a role with a password to prevent unauthorized use of the privileges granted to the role. Typically, an application is designed so that when it starts, it enables the proper role. As a result, an application user does not need to know the password for an application's role.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for instructions for enabling roles from an application

This section includes the following topics:

- Common Uses for Roles
- Role Mechanisms
- The Operating System and Roles

Common Uses for Roles

In general, you create a role to serve one of two purposes:

- To manage the privileges for a database application
- To manage the privileges for a user group

Figure 20–2 and the sections that follow describe the two uses of roles.

Figure 20–2   Common Uses for Roles
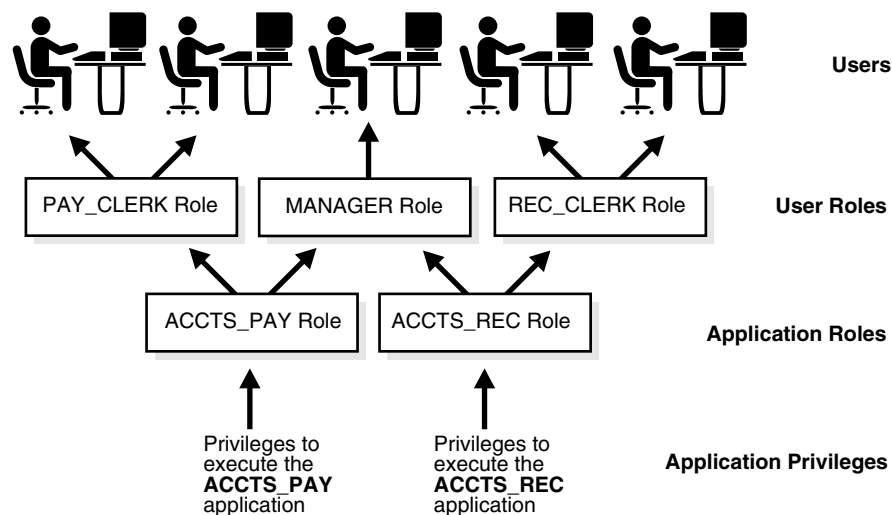
This section includes the following topics:

- Application Roles
- User Roles

Application Roles   You grant an application role all privileges necessary to run a given database application. Then, you grant the secure application role to other roles or to specific users. An application can have several different roles, with each role assigned a different set of privileges that allow for more or less data access while using the application.

User Roles   You create a user role for a group of database users with common privilege requirements. You manage user privileges by granting secure application roles and privileges to the user role and then granting the user role to appropriate users.

Role Mechanisms

Database roles have the following functionality:

- A role can be granted system or schema object privileges.

- A role can be granted to other roles. However, a role cannot be granted to itself and cannot be granted circularly. For example, role A cannot be granted to role B if role B has previously been granted to role A.

- Any role can be granted to any database user.

- Each role granted to a user is, at a given time, either enabled or disabled. A user's security domain includes the privileges of all roles currently enabled for the user and excludes the privileges of any roles currently disabled for the user. Oracle Database allows database applications and users to enable and disable roles to provide selective availability of privileges.

- An indirectly granted role is a role granted to a role. It can be explicitly enabled or disabled for a user. However, by enabling a role that contains other roles, you implicitly enable all indirectly granted roles of the directly granted role.

The Operating System and Roles

In some environments, you can administer database security using the operating system. The operating system can be used to manage the granting (and revoking) of database roles and to manage their password authentication. This capability is not available on all operating systems.

Secure Application Roles

Oracle Database provides secure application roles, which are roles that can only be enabled by authorized PL/SQL packages. This mechanism restricts the enabling of such roles to the invoking application.

Security is strengthened when passwords are not embedded in application source code or stored in a table. Instead, a secure application role can be created, specifying which PL/SQL package is authorized to enable the role. Package identity is used to determine whether privileges are sufficient to enable the roles. Before enabling the role, the application can perform authentication and customized authorization, such as checking whether the user has connected through a proxy.

Because of the restriction that users cannot change security domain inside definer's right procedures, secure application roles can only be enabled inside invoker's right procedures.

See Also:

- *Oracle Database PL/SQL Language Reference*
- *Oracle Database Advanced Application Developer's Guide*

Overview of Access Restrictions on Tables, Views, Synonyms, or Rows

This section describes restrictions associated not with users, but with objects. The restrictions provide protection regardless of the entity who seeks to access or alter them.

You provide this protection by designing and using policies to restrict access to specific tables, views, synonyms, or rows. These policies invoke functions that you design to

specify dynamic predicates establishing the restrictions. You can also group established policies, applying a policy group to a particular application.

Having established such protections, you need to be notified when they are threatened or breached. Given notification, you can strengthen your defenses or deal with the consequences of inappropriate actions and the entities who caused them.

This section includes the following topics:

- Fine-Grained Access Control
- Application Context
- Fine-Grained Auditing

Fine-Grained Access Control

Fine-grained access control lets you use functions to implement security policies and to associate those security policies with tables, views, or synonyms. The database server automatically enforces your security policies, no matter how the data is accessed (for example, by ad hoc queries).

You can:

- Use different policies for SELECT, INSERT, UPDATE, and DELETE (and INDEX, for row level security policies).

- Use security policies only where you need them (for example, on salary information).

- Use more than one policy for each table, including building on top of base policies in packaged applications.

- Distinguish policies between different applications, by using policy groups. Each policy group is a set of policies that belong to an application. The database administrator designates an application context, called a driving context, to indicate the policy group in effect. When tables, views, or synonyms are accessed, the fine-grained access control engine looks up the driving context to determine the policy group in effect and enforces all the associated policies that belong to that policy group.

The PL/SQL package DBMS_RLS let you administer your security policies. Using this package, you can add, drop, enable, disable, and refresh the policies (or policy groups) you create.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about package implementation
>
> - *Oracle Database Advanced Application Developer's Guide* for information and examples on establishing security policies

Dynamic Predicates

Dynamic predicates are acquired at statement parse time, when the base table or view is referenced in a DML statement, rather than having the security rules embedded in views.

The function or package that implements the security policy you create returns a predicate (a WHERE condition). This predicate controls access according to the policy specifications. Rewritten queries are fully optimized and shareable.

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface.

Application Context

Application context helps you apply fine-grained access control because you can associate your function-based security policies with applications.

Each application has its own application-specific context, which users cannot arbitrarily change (for example, through SQL*Plus). Context attributes are accessible to the functions implementing your security policies. For example, context attributes for a human resources application could include "position," "organizational unit," and "country," whereas attributes for an order-entry control might be "customer number" and "sales region".

Application contexts thus permit flexible, parameter-based access control using attributes of interest to an application.

You can:

- Base predicates on context values

- Use context values within predicates, as bind variables

- Set user attributes

- Access user attributes

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference*
>
> - *Oracle Database Advanced Application Developer's Guide*

Dynamic Contexts

Your policies can identify run-time efficiencies by specifying whether a policy is static, shared, context-sensitive, or dynamic.

If it is static, producing the same predicate string for anyone accessing the object, then it is run once and cached in SGA. Policies for statements accessing the same object do not re-run the policy function, but use the cached predicate instead.

This is also true for shared-static policies, for which the server first looks for a cached predicate generated by the same policy function of the same policy type. Shared-static policies are ideal for data partitions on hosting because almost all objects share the same function and the policy is static.

If you label your policy context-sensitive, then the server always runs the policy function on statement parsing; it does not cache the value returned. The policy function is not re-evaluated at statement execution time unless the server detects context changes since the last use of the cursor. (For session pooling where multiple clients share a database session, the middle tier must reset context during client switches.)

When a context-sensitive policy is shared, the server first looks for a cached predicate generated by the same policy function of the same policy type within the same database session. If the predicate is found in the session memory, then the policy function is not re-run and the cached value is valid until session private application context changes occur.

For dynamic policies, the server assumes the predicate may be affected by any system or session environment at any time, and so always re-runs the policy function on each statement parsing or execution.

Fine-Grained Auditing

Fine-grained auditing allows the monitoring of data access based on content. It provides granular auditing of queries, as well as INSERT, UPDATE, and DELETE operations. For example, a central tax authority needs to track access to tax returns to guard against employee snooping, with enough detail to determine what data was accessed. It is not enough to know that SELECT privilege was used by a specific user on a particular table. Fine-grained auditing provides this deeper functionality.

In general, fine-grained auditing policy is based on simple user-defined SQL predicates on table objects as conditions for selective auditing. During fetching, whenever policy conditions are met for a returning row, the query is audited. Later, Oracle Database runs user-defined audit event handlers using autonomous transactions to process the event.

Fine-grained auditing can be implemented in user applications using the DBMS_FGA package or by using database triggers.

> **See Also :** *Oracle Database Advanced Application Developer's Guide*

Overview of Security Policies

This section contains the following topics:

- System Security Policy
- Data Security Policy
- User Security Policy
- Password Management Policy
- Auditing Policy

System Security Policy

Each database has one or more administrators responsible for maintaining all aspects of the security policy: the security administrators. If the database system is small, then the database administrator might have the responsibilities of the security administrator. However, if the database system is large, then a special person or group of people might have responsibilities limited to those of a security administrator.

A security policy must be developed for every database. A security policy should include several sub-policies, as explained in the following sections.

This section includes the following topics:

- Database User Management
- User Authentication
- Operating System Security

Database User Management

Depending on the size of a database system and the amount of work required to manage database users, the security administrator might be the only user with the privileges required to create, alter, or drop database users. Or, there may be several

administrators with privileges to manage database users. Regardless, only trusted individuals should have the powerful privileges to administer database users.

User Authentication

Database users can be **authenticated** (verified as the correct person) by Oracle Database using database passwords, the host operating system, network services, or by Secure Sockets Layer (SSL).

> **See Also:**

Operating System Security

If applicable, the following security issues must also be considered for the operating system environment executing Oracle Database and any database applications:

- Database administrators must have the operating system privileges to create and delete files.

- Typical database users should not have the operating system privileges to create or delete files related to the database.

- If the operating system identifies database roles for users, then the security administrators must have the operating system privileges to modify the security domain of operating system accounts.

Data Security Policy

Data security includes mechanisms that control access to and use of the database at the object level. Your data security policy determines which users have access to a specific schema object, and the specific types of actions allowed for each user on the object. For example, user scott can issue SELECT and INSERT statements but not DELETE statements using the employees table. Your data security policy should also define the actions, if any, that are audited for each schema object.

Your data security policy is determined primarily by the level of security you want for the data in your database. For example, it might be acceptable to have little data security in a database when you want to allow any user to create any schema object, or grant access privileges for their objects to any other user of the system. Alternatively, it might be necessary for data security to be very controlled when you want to make a database or security administrator the only person with the privileges to create objects and grant access privileges for objects to roles and users.

Overall data security should be based on the sensitivity of data. If information is not sensitive, then the data security policy can be more lax. However, if data is sensitive, then a security policy should be developed to maintain tight control over access to objects.

Some means of implementing data security include system and object privileges, and through roles. A role is a set of privileges grouped together that can be granted to users. Views can also implement data security because their definition can restrict access to table data. They can exclude columns containing sensitive data.

Another means of implementing data security is through fine-grained access control and use of an associated application context. Fine-grained access control lets you implement security policies with functions and associate those security policies with tables or views. In effect, the security policy function generates a WHERE condition that is appended to a SQL statement, thereby restricting the users access to rows of data in the table or view. An application context is a secure data cache for storing information used to make access control decisions.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more about views
> - *Oracle Database Advanced Application Developer's Guide* for more about fine-grained access control and application context
> - *Oracle Database PL/SQL Packages and Types Reference*

User Security Policy

This section describes aspects of user security policy, and contains the following topics:

- General User Security
- End-User Security
- Administrator Security
- Application Developer Security
- Application Administrator Security

General User Security

For all types of database users, consider password security and privilege management.

If user authentication is managed by the database, then security administrators should develop a password security policy to maintain database access security. For example, database users must change their passwords at regular intervals. By forcing a user to modify passwords, unauthorized database access can be reduced.

Also consider issues related to privilege management for all types of users. For example, a database with many users, applications, or objects, would benefit from using roles to manage the privileges available to users. Alternatively, in a database with a handful of user names, it might be easier to grant privileges explicitly to users and avoid the use of roles.

End-User Security

Security administrators must define a policy for end-user security. If a database has many users, then the security administrator can decide which groups of users can be categorized into user groups, and then create user roles for these groups. The security administrator can grant the necessary privileges or application roles to each user role, and assign the user roles to the users. To account for exceptions, the security administrator must also decide what privileges must be explicitly granted to individual users.

Roles are the easiest way to grant and manage the common privileges needed by different groups of database users. You can also manage users and their authorizations centrally, in a directory service, through the enterprise user and enterprise role features of Oracle Advanced Security.

Administrator Security

Security administrators should have a policy addressing database administrator security. For example, when the database is large and there are several types of database administrators, the security administrator might decide to group related administrative privileges into several administrative roles. The administrative roles can then be granted to appropriate administrator users. Alternatively, when the database is small and has only a few administrators, it might be more convenient to create one administrative role and grant it to all administrators.

Protection for Connections as SYS and SYSTEM After database creation, and if you used the default passwords for `SYS` and `SYSTEM`, *immediately* change the passwords for the `SYS` and `SYSTEM` administrative user names. Connecting as `SYS` or `SYSTEM` gives a user powerful privileges to modify a database.

If you have installed options that have caused other administrative user names to be created, then such user name accounts are initially created locked.

Protection for Administrator Connections Only database administrators should have the capability to connect to a database with administrative privileges. For example:

```
CONNECT username/password AS SYSDBA/SYSOPER
```

Connecting as `SYSOPER` gives a user the ability to perform basic operational tasks (such as `STARTUP`, `SHUTDOWN`, and recovery operations). Connecting as `SYSDBA` gives the user these abilities plus unrestricted privileges to do anything to a database or the objects within a database (including, `CREATE`, `DROP`, and `DELETE`). `SYSDBA` puts a user in the `SYS` schema, where they can alter data dictionary tables.

Application Developer Security

Security administrators must define a special security policy for the application developers using a database. A security administrator could grant the privileges to create necessary objects to application developers. Or, alternatively, the privileges to create objects could be granted only to a database administrator, who then receives requests for object creation from developers.

Application Developers and Their Privileges Database application developers are unique database users who require special groups of privileges to accomplish their jobs. Unlike end users, developers need system privileges, such as `CREATE TABLE`, `CREATE PROCEDURE`, and so on. However, only specific system privileges should be granted to developers to restrict their overall capabilities in the database.

In many cases, application development is restricted to test databases and is not allowed on production databases. This restriction ensures that application developers do not compete with end users for database resources, and that they cannot detrimentally affect a production database. After an application has been thoroughly developed and tested, it is permitted access to the production database and made available to the appropriate end users of the production database.

Security administrators can create roles to manage the privileges required by the typical application developer.

While application developers are typically given the privileges to create objects as part of the development process, security administrators must maintain limits on what and how much database space can be used by each application developer. For example, the security administrator should specifically set or restrict the following limits for each application developer:

- The tablespaces in which the developer can create tables or indexes
- The quota for each tablespace accessible to the developer

Both limitations can be set by altering a developer's security domain.

Application Administrator Security

In large database systems with many database applications, consider assigning application administrators responsible for the following types of tasks:

- Creating roles for an application and managing the privileges of each application role

- Creating and managing the objects used by a database application

- Maintaining and updating the application code and Oracle Database procedures and packages, as necessary

Often, an application administrator is also the application developer who designed an application. However, an application administrator could be any individual familiar with the database application.

Password Management Policy

Database security systems dependent on passwords require that passwords be kept secret at all times. But, passwords are vulnerable to theft, forgery, and misuse. To allow for greater control over database security, the Oracle Database password management policy is controlled by DBAs and security officers through user profiles.

> **See Also:** "Authentication by Oracle Database" on page 20-6

Auditing Policy

Security administrators should define a policy for the auditing procedures of each database. You may decide to have database auditing disabled unless questionable activities are suspected. When auditing is required, decide what level of detail to audit the database; usually, general system auditing is followed by more specific types of auditing after the origins of suspicious activity are determined. Auditing is discussed in the following section.

Overview of Database Auditing

Auditing is the monitoring and recording of selected user database actions. It can be based on individual actions, such as the type of SQL statement run, or on combinations of factors that can include name, application, time, and so on. Security policies can cause auditing when specified elements in Oracle Database are accessed or altered, including content.

Auditing is generally used to:

- Enable future accountability for current actions taken in a particular schema, table, or row, or affecting specific content

- Investigate suspicious activity. For example, if an unauthorized user is deleting data from tables, then the security administrator could audit all connections to the database and all successful and unsuccessful deletions of rows from all tables in the database.

- Monitor and gather data about specific database activities. For example, the database administrator can gather statistics about which tables are being updated, how many logical I/Os are performed, or how many concurrent users connect at peak times.

You can use Enterprise Manager to view and configure audit-related initialization parameters and administer audited objects for statement auditing and schema object auditing. For example, Enterprise Manager shows the properties for current audited statements, privileges, and objects. You can view the properties of each object, and you can search audited objects by their properties. You can also turn on and turn off auditing on objects, statements, and privileges.

Types and Records of Auditing

Oracle Database allows audit options to be focused or broad. You can audit:

- Successful statement executions, unsuccessful statement executions, or both

- Statement executions once in each user session or once every time the statement is run

- Activities of all users or of a specific user

Oracle Database auditing enables the use of several different mechanisms, with the features listed in Table 20–2.

Table 20–2   Types of Auditing

Type of Auditing	Meaning/Description
Statement auditing	Audits SQL statements by type of statement, not by the specific schema objects on which they operate. Typically broad, statement auditing audits the use of several types of related actions for each option. For example, `AUDIT TABLE` tracks several DDL statements regardless of the table on which they are issued. You can also set statement auditing to audit selected users or every user in the database.
Privilege auditing	Audits the use of powerful system privileges enabling corresponding actions, such as `AUDIT CREATE TABLE`. Privilege auditing is more focused than statement auditing because it audits only the use of the target privilege. You can set privilege auditing to audit a selected user or every user in the database.
Schema object auditing	Audits specific statements on a particular schema object, such as `AUDIT SELECT ON employees`. Schema object auditing is very focused, auditing only a specific statement on a specific schema object. Schema object auditing always applies to all users of the database.
Fine-grained auditing	Audits data access and actions based on content. Using `DBMS_FGA`, the security administrator creates an audit policy on the target table. If any rows returned from a DML statement block match the audit condition, then an audit event entry is inserted into the audit trail.

Audit Records and the Audit Trails

Audit records include information such as the operation that was audited, the user performing the operation, and the date and time of the operation. Audit records can be stored in either a data dictionary table, called the **database audit trail**, or in operating system files, called an operating system audit trail.

This section includes the following topics:

- Database Audit Trail

- Auditing in a Distributed Database

- Operating System Audit Trail

- Operating System Audit Records

- Records Always in the Operating System Audit Trail

- When Are Audit Records Created?

Database Audit Trail The database audit trail is a single table named `SYS.AUD$` in the `SYS` schema of each Oracle database's data dictionary. Several predefined views are provided to help you use the information in this table.

Audit trail records can contain different types of information, depending on the events audited and the auditing options set. The following information is always included in each audit trail record, if the information is meaningful to the particular audit action:

- User name

- Instance number

- Process identifier

- Session identifier

- Terminal identifier

- Name of the schema object accessed

- Operation performed or attempted

- Completion code of the operation

- Date and time stamp

- System privileges used

Auditing in a Distributed Database Auditing is site autonomous. An instance audits only the statements issued by directly connected users. A local Oracle Database node cannot audit actions that take place in a remote database. Because remote connections are established through the user account of a database link, statements issued through the database link's connection are audited by the remote Oracle Database node.

Operating System Audit Trail Oracle Database allows audit trail records to be directed to an operating system audit trail if the operating system makes such an audit trail available to Oracle Database. If not, then audit records are written to a file outside the database, with a format similar to other Oracle Database trace files.

Oracle Database allows certain actions that are *always* audited to continue, even when the operating system audit trail (or the operating system file containing audit records) is unable to record the audit record. The usual cause of this is that the operating system audit trail or the file system is full and unable to accept new records.

System administrators configuring operating system auditing should ensure that the audit trail or the file system does not fill completely. Most operating systems provide administrators with sufficient information and warning to ensure this does not occur. Note, however, that configuring auditing to use the database audit trail removes this vulnerability, because Oracle Database prevents audited events from occurring if the audit trail is unable to accept the database audit record for the statement.

Operating System Audit Records The operating system audit trail is encoded, but it is decoded in data dictionary files and error messages.

- **Action code** describes the operation performed or attempted. The `AUDIT_ACTIONS` data dictionary table describes these codes.

- **Privileges used** describes any system privileges used to perform the operation. The `SYSTEM_PRIVILEGE_MAP` table describes all of these codes.

- **Completion code** describes the result of the attempted operation. Successful operations return a value of zero, and unsuccessful operations return the Oracle Database error code describing why the operation was unsuccessful.

Records Always in the Operating System Audit Trail Some database-related actions are always recorded into the operating system audit trail regardless of whether database auditing is enabled:

- At instance startup, an audit record is generated that details the operating system user starting the instance, the user's terminal identifier, the date and time stamp, and whether database auditing was enabled or disabled. This information is recorded into the operating system audit trail, because the database audit trail is not available until after startup has successfully completed. Recording the state of database auditing at startup also acts as an auditing flag, inhibiting an administrator from performing unaudited actions by restarting a database with database auditing disabled.

- At instance shutdown, an audit record is generated that details the operating system user shutting down the instance, the user's terminal identifier, the date and time stamp.

- During connections with administrator privileges, an audit record is generated that details the operating system user connecting to Oracle Database with administrator privileges. This record provides accountability regarding users connected with administrator privileges.

On operating systems that do not make an audit trail accessible to Oracle Database, these audit trail records are placed in an Oracle Database audit trail file in the same directory as background process trace files.

When Are Audit Records Created? Any authorized database user can set his own audit options at any time, but the recording of audit information is enabled or disabled by the security administrator.

When auditing is enabled in the database, an audit record is generated during the execute phase of statement execution.

SQL statements inside PL/SQL program units are individually audited, as necessary, when the program unit is run.

The generation and insertion of an audit trail record is independent of a user's transaction being committed. That is, even if a user's transaction is rolled back, the audit trail record remains committed.

Statement and privilege audit options in effect at the time a database user connects to the database remain in effect for the duration of the session. Setting or changing statement or privilege audit options in a session does not cause effects in that session. The modified statement or privilege audit options take effect only when the current session is ended and a new session is created. In contrast, changes to schema object audit options become effective for current sessions immediately.

Operations by the `SYS` user and by users connected through `SYSDBA` or `SYSOPER` can be fully audited with the `AUDIT_SYS_OPERATIONS` initialization parameter. Successful SQL statements from `SYS` are audited indiscriminately. The audit records for sessions established by the user `SYS` or connections with administrative privileges are sent to an operating system location. Sending them to a location separate from the usual database audit trail in the `SYS` schema provides for greater auditing security.

See Also:

- *Oracle Database Administrator's Guide* for instructions on enabling and disabling auditing

- Chapter 24, "SQL" for information about the different phases of SQL statement processing and shared SQL

21

Data Integrity

This chapter explains how to use integrity constraints to enforce the business rules associated with your database and prevent the entry of invalid information into tables.

This chapter contains the following topics:

- Introduction to Data Integrity
- Overview of Integrity Constraints
- Types of Integrity Constraints
- The Mechanisms of Constraint Checking
- Deferred Constraint Checking

Introduction to Data Integrity

It is important that column data adhere to a predefined set of rules, as determined by the database administrator or application developer.

For example, some columns in a database table can have specific rules that constrain the data contained within them. These constraints can affect how data columns in one table relate to those in another table.

This section includes the following topics:

- Data Integrity Rules
- How Oracle Database Enforces Data Integrity
- Constraint States

Data Integrity Rules

This section describes the rules that can be applied to table columns to enforce different types of data integrity.

Null rule: A null rule is a rule defined on a single column that allows or disallows inserts or updates of rows containing a null (the absence of a value) in that column.

Unique column values: A unique value rule defined on a column (or set of columns) allows the insert or update of a row only if it contains a unique value in that column (or set of columns).

Primary key values: A primary key value rule defined on a key (a column or set of columns) specifies that each row in the table can be uniquely identified by the values in the key.

Referential integrity rules: A referential integrity rule is a rule defined on a key (a column or set of columns) in one table that guarantees that the values in that key match the values in a key in a related table (the referenced value).

Referential integrity also includes the rules that dictate what types of data manipulation are allowed on referenced values and how these actions affect dependent values. The rules associated with referential integrity are:

- **Restrict**: Disallows the update or deletion of referenced data.

- **Set to null**: When referenced data is updated or deleted, all associated dependent data is set to `NULL`.

- **Set to default**: When referenced data is updated or deleted, all associated dependent data is set to a default value.

- **Cascade**: When referenced data is updated, all associated dependent data is correspondingly updated. When a referenced row is deleted, all associated dependent rows are deleted.

- **No action**: Disallows the update or deletion of referenced data. This differs from `RESTRICT` in that it is checked at the end of the statement, or at the end of the transaction if the constraint is deferred. (Oracle Database uses No Action as its default action.)

Complex integrity checking: A user-defined rule for a column (or set of columns) that allows or disallows inserts, updates, or deletes of a row based on the value it contains for the column (or set of columns).

How Oracle Database Enforces Data Integrity

Oracle Database enables you to define and enforce each type of data integrity rule defined in the previous section. Most of these rules are easily defined using either integrity constraints or database triggers (stored database procedures automatically invoked on insert, update, or delete operations).

You cannot enforce referential integrity using declarative integrity constraints if child and parent tables are on different nodes of a distributed database. However, you can enforce referential integrity in a distributed database using database triggers.

> **See Also:** Chapter 22, "Triggers" for examples of triggers used to enforce data integrity

Constraint States

- `ENABLE` ensures that all incoming data conforms to the constraint

- `DISABLE` allows incoming data, regardless of whether it conforms to the constraint

- `VALIDATE` ensures that existing data conforms to the constraint

- `NOVALIDATE` means that some existing data may not conform to the constraint

In addition:

- `ENABLE VALIDATE` is the same as `ENABLE`. The constraint is checked and is guaranteed to hold for all rows.

- `ENABLE NOVALIDATE` means that the constraint is checked, but it does not have to be true for all rows. This allows existing rows to violate the constraint, while ensuring that all new or modified rows are valid.

In an `ALTER TABLE` statement, `ENABLE NOVALIDATE` resumes constraint checking on disabled constraints without first validating all data in the table.

- `DISABLE NOVALIDATE` is the same as `DISABLE`. The constraint is not checked and is not necessarily true.

- `DISABLE VALIDATE` disables the constraint, drops the index on the constraint, and disallows any modification of the constrained columns.

  For a `UNIQUE` constraint, the `DISABLE VALIDATE` state enables you to load data efficiently from a nonpartitioned table into a partitioned table using the `EXCHANGE PARTITION` clause of the `ALTER TABLE` statement.

Transitions between these states are governed by the following rules:

- `ENABLE` implies `VALIDATE`, unless `NOVALIDATE` is specified.

- `DISABLE` implies `NOVALIDATE`, unless `VALIDATE` is specified.

- `VALIDATE` and `NOVALIDATE` do not have any default implications for the `ENABLE` and `DISABLE` states.

- When a unique or primary key moves from the `DISABLE` state to the `ENABLE` state, if there is no existing index, a unique index is automatically created. Similarly, when a unique or primary key moves from `ENABLE` to `DISABLE` and it is enabled with a unique index, the unique index is dropped.

- When any constraint is moved from the `NOVALIDATE` state to the `VALIDATE` state, all data must be checked (this can be very slow). However, moving from `VALIDATE` to `NOVALIDATE` simply forgets that the data was ever checked.

- Moving a single constraint from the `ENABLE NOVALIDATE` state to the `ENABLE VALIDATE` state does not block reads, writes, or other DDL statements. It can be done in parallel.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for general information about constraints
>
> - *Oracle Database Administrator's Guide* for information about managing constraints

Overview of Integrity Constraints

Oracle Database uses integrity constraints to prevent invalid data entry into the base tables of the database. You can define integrity constraints to enforce the business rules you want to associate with the information in a database. If any of the results of a DML statement execution violate an integrity constraint, Oracle Database rolls back the statement and returns an error.

> **Note:** Operations on views (and synonyms for tables) are subject to the integrity constraints defined on the underlying base tables.

For example, assume that you define an integrity constraint for the `salary` column of the `employees` table that enforces a rule that no row in this table can contain a numeric value greater than 10,000 in this column. If an `INSERT` or `UPDATE` statement attempts to violate this integrity constraint, Oracle Database rolls back the statement and returns an information error message.

This section includes the following topics:

- [Advantages of Integrity Constraints]
- [The Performance Cost of Integrity Constraints]

Advantages of Integrity Constraints

This section describes some of the advantages that integrity constraints associated with database tables have over other alternatives. These advantages are:

- Enforcing business rules in the code of a database application
- Using stored procedures to completely control access to data
- Enforcing business rules with triggered stored database procedures

> **See Also:** Chapter 22, "Triggers"

This section includes the following topics:

- [Declarative Ease]
- [Centralized Rules]
- [Maximum Application Development Productivity]
- [Immediate User Feedback]
- [Flexibility for Data Loads and Identification of Integrity Violations]

Declarative Ease

Declarative integrity constraints are preferable to application code and database triggers. Because you define integrity constraints using SQL statements, no additional programming is required when you define or alter a table. The SQL statements are easy to write and eliminate programming errors. Oracle Database controls their functionality.

The declarative approach is also better than using stored procedures, because the stored procedure solution to data integrity controls data access, but integrity constraints do not eliminate the flexibility of random data access.

The semantics of integrity constraint declarations are clearly defined, and performance optimizations are implemented for each specific declarative rule. The Oracle Database optimizer can use declarations to learn more about data to improve overall query performance. (Also, taking integrity rules out of application code and database triggers guarantees that checks are only made when necessary.)

Centralized Rules

Integrity constraints are defined for tables (not applications) and are stored in the data dictionary. Therefore, any data entered by any application must adhere to the same integrity constraints associated with the table. By keeping business rules in application code centralized integrity constraints rather than in application code, the tables of a database are guaranteed to contain valid data, no matter which database application manipulates the information.

Maximum Application Development Productivity

If a business rule enforced by an integrity constraint changes, then the administrator need only change that integrity constraint and all applications automatically adhere to the modified constraint. In contrast, if the business rule were enforced by the code of

each database application, developers would have to modify all application source code and recompile, debug, and test the modified applications.

Immediate User Feedback

Oracle Database stores specific information about each integrity constraint in the data dictionary. You can design database applications to use this information to provide immediate user feedback about integrity constraint violations, even before Oracle Database runs and checks the SQL statement. For example, an Oracle Forms application can use integrity constraint definitions stored in the data dictionary to check for violations as values are entered into the fields of a form, even before the application issues a statement.

Flexibility for Data Loads and Identification of Integrity Violations

You can disable integrity constraints temporarily so that large amounts of data can be loaded without the overhead of constraint checking. When the data load is complete, you can easily enable the integrity constraints, and you can automatically report any new rows that violate integrity constraints to a separate exceptions table.

The Performance Cost of Integrity Constraints

The advantages of enforcing data integrity rules come with some loss in performance. In general, the cost of including an integrity constraint is, at most, the same as executing a SQL statement that evaluates the constraint.

Types of Integrity Constraints

You can use integrity constraints to impose restrictions on the input of values in both normal and virtual columns. You can use the following constraints:

- NOT NULL Integrity Constraints

- UNIQUE Key Integrity Constraints

- PRIMARY KEY Integrity Constraints

- Referential Integrity Constraints

- CHECK Integrity Constraints

> **See Also:** "Overview of Tables" on page 5-3 for a conceptual description of virtual columns, and *Oracle Database SQL Language Reference* for reference information about virtual columns

NOT NULL Integrity Constraints

By default, all columns in a table allow nulls. **Null** means the absence of a value. A `NOT NULL` constraint requires a column of a table contain no null values.

For certain types of tables and column datatyptes, when adding a column that has both a `NOT NULL` constraint and a default value, the database can optimize the operation and reduce the amount of time that the table is locked for DML. The database stores metadata in the table that describes the default value in the added column. As a result, the database does not need to populate every row with the default value when you add the column, thereby minimizing the time that the table is locked.

You can only add a column with a `NOT NULL` constraint if the table does not contain any rows or if you specify a default value.

See Also:

- *Oracle Database Administrator's Guide* to learn how to add columns to a table

- *Oracle Database SQL Language Reference* to learn about the ALTER TABLE statement

UNIQUE Key Integrity Constraints

A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table have duplicate values in a specified column or set of columns.

This section includes the following topics:

- Unique Keys

- Combining UNIQUE Key and NOT NULL Integrity Constraints

Unique Keys

The columns included in the definition of the UNIQUE key constraint are called the **unique key**. If the unique key consists of more than one column, then that group of columns is called a **composite unique key**.

Unique key is often incorrectly used as a synonym for the term *UNIQUE key constraint* or *UNIQUE index*. However, *key* refers only to the column or set of columns used in the definition of the integrity constraint.

For example, the UNIQUE key constraint might let you enter an area code and telephone number any number of times, but the combination of a given area code and given telephone number cannot be duplicated in the table. This eliminates unintentional duplication of a telephone number.

Combining UNIQUE Key and NOT NULL Integrity Constraints

Columns with both unique keys and NOT NULL integrity constraints are common. This combination forces the user to enter values in the unique key and also eliminates the possibility that any new row's data will ever conflict with an existing row's data.

> **Note:** Because of the search mechanism for UNIQUE constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite UNIQUE key constraint.

PRIMARY KEY Integrity Constraints

Each table in the database can have at most one PRIMARY KEY constraint. The values in the group of one or more columns subject to this constraint constitute the unique identifier of the row. In effect, each row is named by its primary key values.

The Oracle Database implementation of the PRIMARY KEY integrity constraint guarantees that both of the following are true:

- No two rows of a table have duplicate values in the specified column or set of columns.

- The primary key columns do not allow nulls. That is, a value must exist for the primary key columns in each row.

This section includes the following topics:

- Primary Keys
- PRIMARY KEY Constraints and Indexes

Primary Keys

The columns included in the definition of a table's `PRIMARY KEY` integrity constraint are called the *primary key*. Although it is not required, every table should have a primary key so that:

- Each row in the table can be uniquely identified
- No duplicate rows exist in the table

PRIMARY KEY Constraints and Indexes

Oracle Database enforces all `PRIMARY KEY` constraints using indexes. The primary key constraint created for a column is enforced by the implicit creation of:

- A unique index on that column
- A `NOT NULL` constraint for that column

Composite primary key constraints are limited to 32 columns, which is the same limitation imposed on composite indexes. The name of the index is the same as the name of the constraint. Also, you can specify the storage options for the index by including the `ENABLE` clause in the `CREATE TABLE` or `ALTER TABLE` statement used to create the constraint. If a usable index exists when a primary key constraint is created, then the primary key constraint uses that index rather than implicitly creating a new one.

Referential Integrity Constraints

Different tables in a relational database can be related by common columns, and the rules that govern the relationship of the columns must be maintained. Referential integrity rules guarantee that these relationships are preserved.

Table 21–1 lists terms associated with referential integrity constraints.

Table 21–1   Referential Integrity Constraint Terms

Term	Definition
Foreign key	The column or set of columns included in the definition of the referential integrity constraint that reference a referenced key.
Referenced key	The unique key or primary key of the same or different table that is referenced by a foreign key.
Dependent or child table	The table that includes the foreign key. Therefore, it is the table that is dependent on the values present in the referenced unique or primary key.
Referenced or parent table	The table that is referenced by the child table's foreign key. It is this table's referenced key that determines whether specific inserts or updates are allowed in the child table.

A referential integrity constraint requires that for each row of a table, the value in the foreign key matches a value in a parent key.

Figure 21–1 shows a foreign key defined on the `deptno` column of the `emp` table. It guarantees that every value in this column must match a value in the primary key of the `dept` table (also the `deptno` column). Therefore, no erroneous department numbers can exist in the `deptno` column of the `emp` table.

Foreign keys can be defined as multiple columns. However, a composite foreign key must reference a composite primary or unique key with the same number of columns and the same datatypes. Because composite primary and unique keys are limited to 32 columns, a composite foreign key is also limited to 32 columns.

Figure 21–1   Referential Integrity Constraints

Parent Key
Primary key of referenced table

Table DEPT

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Referenced or Parent Table

Foreign Key
(values in dependent table must match a value in unique key or primary key of referenced table)

Table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO		17–DEC–85	9,000.00		20
7499	ALLEN	VP–SALES	7329	20–FEB–90	300.00	100.00	30
7521	WARD	MANAGER	7499	22–FEB–90	500.00	200.00	30
7566	JONES	SALESMAN	7521	02–APR–90		400.00	20

Dependent or Child Table

INSERT INTO

This row violates the referential constraint because "40" is not present in the referenced table's primary key; therefore, the row is not allowed in the table.

| 7571 | FORD | MANAGER | 7499 | 23–FEB–90 | 5,000.00 | 200.00 | 40 |
| 7571 | FORD | MANAGER | 7499 | 23–FEB–90 | 5,000.00 | 200.00 | |

This row is allowed in the table because a null value is entered in the DEPTNO column; however, if a not null constraint is also defined for this column, this row is not allowed.

This section includes the following topics:

- Self-Referential Integrity Constraints
- Nulls and Foreign Keys
- Actions Defined by Referential Integrity Constraints
- Concurrency Control, Indexes, and Foreign Keys

Self-Referential Integrity Constraints

Another type of referential integrity constraint, shown in Figure 21–2, is called a self-referential integrity constraint. This type of foreign key references a parent key in the same table.

In Figure 21–2, the referential integrity constraint ensures that every value in the `mgr` column of the `emp` table corresponds to a value that currently exists in the `empno` column of the same table, but not necessarily in the same row, because every manager must also be an employee. This integrity constraint eliminates the possibility of erroneous employee numbers in the `mgr` column.

Figure 21–2   Single Table Referential Constraints
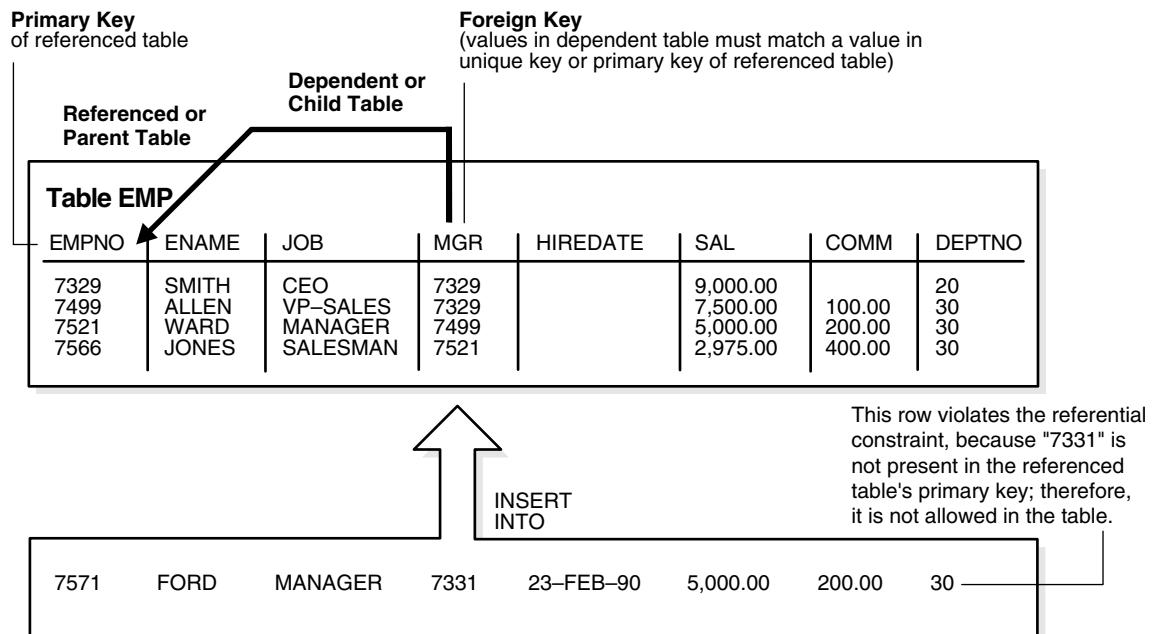
Nulls and Foreign Keys

The relational model permits the value of foreign keys to match either the referenced primary or unique key value, or be null. If any column of a composite foreign key is null, then the non-null portions of the key do not have to match any corresponding portion of a parent key.

Actions Defined by Referential Integrity Constraints

Referential integrity constraints can specify particular actions to be performed on the dependent rows in a child table if a referenced parent key value is modified. The referential actions supported by the `FOREIGN KEY` integrity constraints of Oracle Database are `UPDATE` and `DELETE NO ACTION`, and `DELETE CASCADE`.

> **Note:**   Other referential actions not supported by `FOREIGN KEY` integrity constraints of Oracle Database can be enforced using database triggers.
>
> See Chapter 22, "Triggers" for more information.

This section includes the following topics:

■ DELETE NO ACTION

DELETE CASCADE

DELETE SET NULL

■ DML Restrictions with Respect to Referential Actions

DELETE NO ACTION The No Action (default) option specifies that referenced key values cannot be updated or deleted if the resulting data would violate a referential integrity constraint. For example, if a primary key value is referenced by a value in the foreign key, then the referenced primary key value cannot be deleted because of the dependent data.

DELETE CASCADE A **delete cascades** when rows containing referenced key values are deleted, causing all rows in child tables with dependent foreign key values to also be deleted. For example, if a row in a parent table is deleted, and this row's primary key value is referenced by one or more foreign key values in a child table, then the rows in the child table that reference the primary key value are also deleted from the child table.

DELETE SET NULL A delete **sets null** when rows containing referenced key values are deleted, causing all rows in child tables with dependent foreign key values to set those values to null. For example, if `employee_id` references `manager_id` in the `TMP` table, then deleting a manager causes the rows for all employees working for that manager to have their `manager_id` value set to null.

DML Restrictions with Respect to Referential Actions Table 21–2 outlines the DML statements allowed by the different referential actions on the primary/unique key values in the parent table, and the foreign key values in the child table.

Table 21–2   DML Statements Allowed by Update and Delete No Action

DML Statement	Issued Against Parent Table	Issued Against Child Table
INSERT	Always OK if the parent key value is unique.	OK only if the foreign key value exists in the parent key or is partially or all null.
UPDATE NO ACTION	Allowed if the statement does not leave any rows in the child table without a referenced parent key value.	Allowed if the new foreign key value still references a referenced key value.
DELETE NO ACTION	Allowed if no rows in the child table reference the parent key value.	Always OK.
DELETE CASCADE	Always OK.	Always OK.
DELETE SET NULL	Always OK.	Always OK.

Concurrency Control, Indexes, and Foreign Keys

You almost always index foreign keys. The only exception is when the matching unique or primary key is never updated or deleted.

Oracle Database maximizes the concurrency control of parent keys in relation to dependent foreign key values. You can control what concurrency mechanisms are used to maintain these relationships, and, depending on the situation, this can be highly beneficial. The following sections explain the possible situations and give recommendations for each.

This section includes the following topics:

- No Index on the Foreign Key

- Index on the Foreign Key

No Index on the Foreign Key   Figure 21–3 illustrates the locking mechanisms used by Oracle Database when no index is defined on the foreign key and when rows are being updated or deleted in the parent table. Inserts into the parent table do not require any locks on the child table.

Unindexed foreign keys cause DML on the primary key to get a share row exclusive table lock (also sometimes called a **share-subexclusive table lock, SSX**) on the foreign key table. This prevents DML on the table by other transactions. The SSX lock is released immediately after it is obtained. If multiple primary keys are updated or deleted, the lock is obtained and released once for each row.

> **See Also:**   "DML Locks" on page 13-16

Figure 21–3   Locking Mechanisms When No Index Is Defined on the Foreign Key

Table Parent

Row 1	Key 1
Row 2	Key 2
Row 3	**Key 3**
Row 4	Key 4

Index

Key 1
Key 2
Key 3
Key 4

Table Child

Row 1	Key 1
Row 2	Key 1
Row 3	Key 3
Row 4	Key 2
Row 5	Key 2

Share lock acquired

Exclusive row lock acquired

Newly updated row
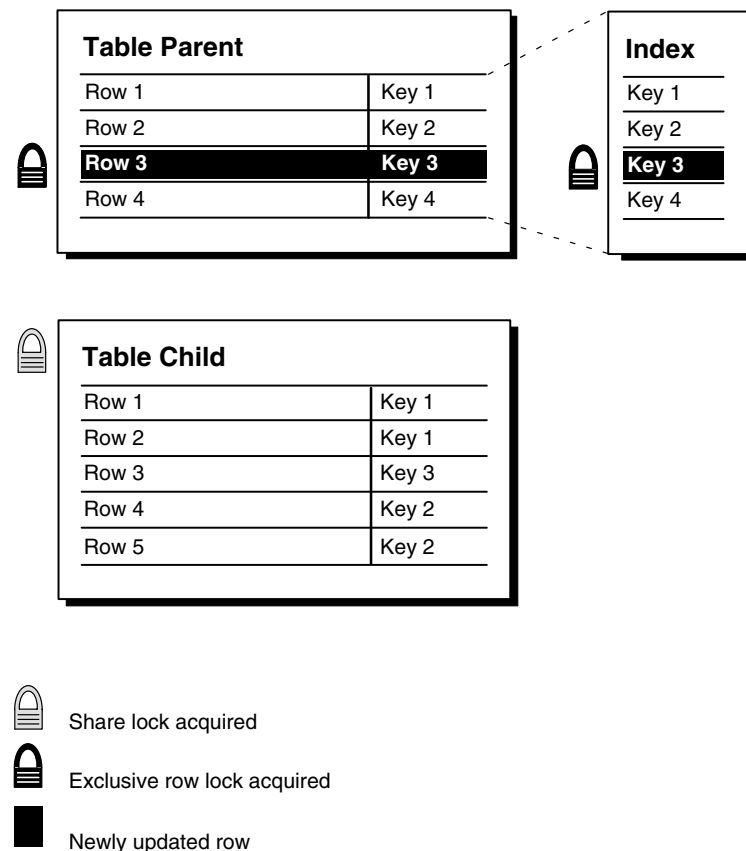
Index on the Foreign Key   Figure 21–4 illustrates the locking mechanisms used when an index is defined on the foreign key, and new rows are inserted, updated, or deleted in the child table.

Indexed foreign keys cause a row share table lock (also sometimes called a **subshare table lock, SS**). This prevents other transactions from exclusive locking the whole table, but it does not block DML on the parent or the child table.

This situation is preferable if there is any update or delete activity on the parent table while update activity is taking place on the child table. Inserts, updates, and deletes on

the parent table do not require any locks on the child table, although updates and deletes will wait for row-level locks on the indexes of the child table to clear.

Figure 21–4   Locking Mechanisms When Index Is Defined on the Foreign Key

Exclusive row lock acquired

Newly updated row
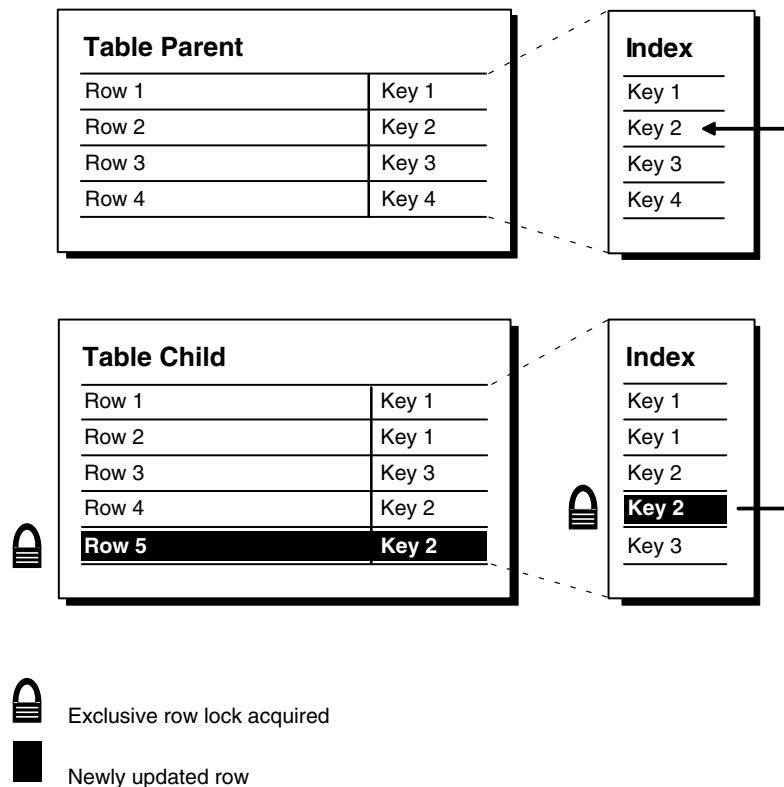
If the child table specifies ON DELETE CASCADE, then deletes from the parent table can result in deletes from the child table. In this case, waiting and locking rules are the same as if you deleted yourself from the child table after performing the delete from the parent table.

CHECK Integrity Constraints

A CHECK integrity constraint on a column or set of columns requires that a specified condition be true or unknown for every row of the table. If a DML statement results in the condition of the CHECK constraint evaluating to false, then the statement is rolled back.

This section includes the following topics:

- The Check Condition
- Multiple CHECK Constraints

The Check Condition

CHECK constraints let you enforce very specific integrity rules by specifying a check condition. The condition of a CHECK constraint has some limitations:

- It must be a Boolean expression evaluated using the values in the row being inserted or updated, and
- It cannot contain subqueries; sequences; the SQL functions SYSDATE, UID, USER, or USERENV; or the pseudocolumns LEVEL or ROWNUM.

In evaluating CHECK constraints that contain string literals or SQL functions with globalization support parameters as arguments (such as TO_CHAR, TO_DATE, and TO_NUMBER), Oracle Database uses the database globalization support settings by default. You can override the defaults by specifying globalization support parameters explicitly in such functions within the CHECK constraint definition.

> **See Also:** *Oracle Database Globalization Support Guide* for more information on globalization support features

Multiple CHECK Constraints

A single column can have multiple CHECK constraints that reference the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

If you create multiple CHECK constraints for a column, design them carefully so their purposes do not conflict. Do not assume any particular order of evaluation of the conditions. Oracle Database does not verify that CHECK conditions are not mutually exclusive.

The Mechanisms of Constraint Checking

To know what types of actions are permitted when constraints are present, it is useful to understand when Oracle Database actually performs the checking of constraints. Assume the following:

- The emp table has been defined as in Figure 21–2 on page 21-9.

- The self-referential constraint makes the entries in the mgr column dependent on the values of the empno column. For simplicity, the rest of this discussion addresses only the empno and mgr columns of the emp table.

Consider the insertion of the first row into the emp table. No rows currently exist, so how can a row be entered if the value in the mgr column cannot reference any existing value in the empno column? Three possibilities for doing this are:

- A null can be entered for the mgr column of the first row, assuming that the mgr column does not have a NOT NULL constraint defined on it. Because nulls are allowed in foreign keys, this row is inserted successfully into the table.

- The same value can be entered in both the empno and mgr columns. This case reveals that Oracle Database performs its constraint checking *after* the statement has been completely run. To allow a row to be entered with the same values in the parent key and the foreign key, Oracle Database must first run the statement (that is, insert the new row) and then check to see if any row in the table has an empno that corresponds to the new row's mgr.

- A multiple row INSERT statement, such as an INSERT statement with nested SELECT statement, can insert rows that reference one another. For example, the first row might have empno as 200 and mgr as 300, while the second row might have empno as 300 and mgr as 200.

  This case also shows that constraint checking is deferred until the complete execution of the statement. All rows are inserted first, then all rows are checked for constraint violations. You can also defer the checking of constraints until the end of the **transaction**.

Consider the same self-referential integrity constraint in this scenario. The company has been sold. Because of this sale, all employee numbers must be updated to be the current value plus 5000 to coordinate with the new company's employee numbers.

Because manager numbers are really employee numbers, these values must also increase by 5000 (see Figure 21–5).

Figure 21–5   The EMP Table Before Updates


```
UPDATE employees
  SET employee_id = employee_id + 5000,
      manager_id = manager_id + 5000;
```

Even though a constraint is defined to verify that each `mgr` value matches an `empno` value, this statement is legal because Oracle Database effectively performs its constraint checking after the statement completes. Figure 21–6 shows that Oracle Database performs the actions of the entire SQL statement before any constraints are checked.

Figure 21–6   Constraint Checking
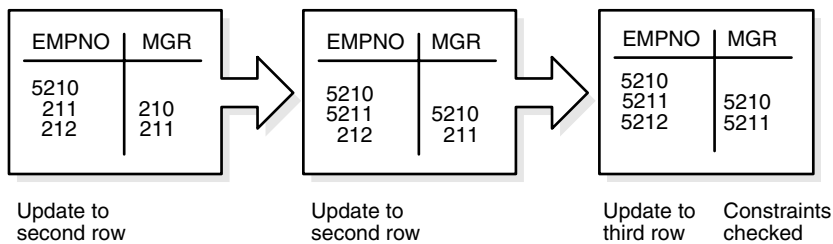
The examples in this section illustrate the constraint checking mechanism during `INSERT` and `UPDATE` statements. The same mechanism is used for all types of DML statements, including `UPDATE`, `INSERT`, and `DELETE` statements.

The examples also used self-referential integrity constraints to illustrate the checking mechanism. The same mechanism is used for all types of constraints, including the following:

- `NOT NULL`
- `UNIQUE` key
- `PRIMARY KEY`
- All types of `FOREIGN KEY` constraints
- `CHECK` constraints

> **See Also:** "Deferred Constraint Checking" on page 21-15

Default Column Values and Integrity Constraint Checking

Default values are included as part of an `INSERT` statement before the statement is parsed. Therefore, default column values are subject to all integrity constraint checking.

Deferred Constraint Checking

You can **defer** checking constraints for validity until the end of the transaction.

- A constraint is **deferred** if the system checks that it is satisfied only on commit. If a deferred constraint is violated, then commit causes the transaction to undo.

- If a constraint is **immediate** (not deferred), then it is checked at the end of each statement. If it is violated, the statement is rolled back immediately.

If a constraint causes an **action** (for example, delete cascade), that action is always taken as part of the statement that caused it, whether the constraint is deferred or immediate.

This section includes the following topics:

- Constraint Attributes

- SET CONSTRAINTS Mode

- Unique Constraints and Indexes

Constraint Attributes

You can define constraints as either **deferrable** or **not deferrable**, and either **initially deferred** or **initially immediate**. These attributes can be different for each constraint. You specify them with keywords in the `CONSTRAINT` clause:

- `DEFERRABLE` or `NOT DEFERRABLE`

- `INITIALLY DEFERRED` or `INITIALLY IMMEDIATE`

Constraints can be added, dropped, enabled, disabled, or validated. You can also modify a constraint's attributes.

> **See Also:** *Oracle Database SQL Language Reference* for information about constraint attributes and their default values

SET CONSTRAINTS Mode

The `SET CONSTRAINTS` statement makes constraints either `DEFERRED` or `IMMEDIATE` for a particular transaction (following the ANSI SQL92 standards in both syntax and semantics). You can use this statement to set the mode for a list of constraint names or for `ALL` constraints.

The `SET CONSTRAINTS` mode lasts for the duration of the transaction or until another `SET CONSTRAINTS\` statement resets the mode.

`SET CONSTRAINTS ... IMMEDIATE` causes the specified constraints to be checked immediately on execution of each constrained statement. Oracle Database first checks any constraints that were deferred earlier in the transaction and then continues immediately checking constraints of any further statements in that transaction, as long as all the checked constraints are consistent and no other `SET CONSTRAINTS` statement is issued. If any constraint fails the check, an error is signaled. At that point, a `COMMIT` causes the whole transaction to undo.

The `ALTER SESSION` statement also has clauses to `SET CONSTRAINTS IMMEDIATE` or `DEFERRED`. These clauses imply setting `ALL` deferrable constraints (that is, you cannot specify a list of constraint names). They are equivalent to making a `SET CONSTRAINTS` statement at the start of each transaction in the current session.

Making constraints **immediate** at the end of a transaction is a way of checking whether `COMMIT` can succeed. You can avoid unexpected rollbacks by setting

constraints to `IMMEDIATE` as the last statement in a transaction. If any constraint fails the check, you can then correct the error before committing the transaction.

The `SET CONSTRAINTS` statement is disallowed inside of triggers.

`SET CONSTRAINTS` can be a distributed statement. Existing database links that have transactions in process are told when a `SET CONSTRAINTS ALL` statement occurs, and new links learn that it occurred as soon as they start a transaction.

Unique Constraints and Indexes

A user sees inconsistent constraints, including duplicates in unique indexes, when that user's transaction produces these inconsistencies. You can place deferred unique and foreign key constraints on materialized views, allowing fast and complete refresh to complete successfully.

Deferrable unique constraints always use nonunique indexes. When you remove a deferrable constraint, its index remains. This is convenient because the storage information remains available after you disable a constraint. Not-deferrable unique constraints and primary keys also use a nonunique index if the nonunique index is placed on the key columns before the constraint is enforced.

22

Triggers

This chapter discusses triggers, which are procedures stored in PL/SQL or Java that run (fire) implicitly whenever a table or view is modified or when some user actions or database system actions occur.

This chapter contains the following topics:

- Introduction to Triggers
- Components of a Trigger
- Types of Triggers
- Trigger Execution

Introduction to Triggers

You can write triggers that fire whenever one of the following operations occurs:

1. DML statements (`INSERT`, `UPDATE`, `DELETE`) on a particular table or view, issued by any user

2. DDL statements (`CREATE` or `ALTER` primarily) issued either by a particular schema/user or by any schema/user in the database

3. Database events, such as logon/logoff, errors, or startup/shutdown, also issued either by a particular schema/user or by any schema/user in the database

Triggers are similar to stored procedures. A trigger stored in the database can include SQL and PL/SQL or Java statements to run as a unit and can invoke stored procedures. However, procedures and triggers differ in the way that they are invoked. A procedure is explicitly run by a user, application, or trigger. Triggers are implicitly fired by Oracle Database when a triggering event occurs, no matter which user is connected or which application is being used.

Figure 22–1 shows a database application with some SQL statements that implicitly fire several triggers stored in the database. Notice that the database stores triggers separately from their associated tables.

Figure 22–1   Triggers
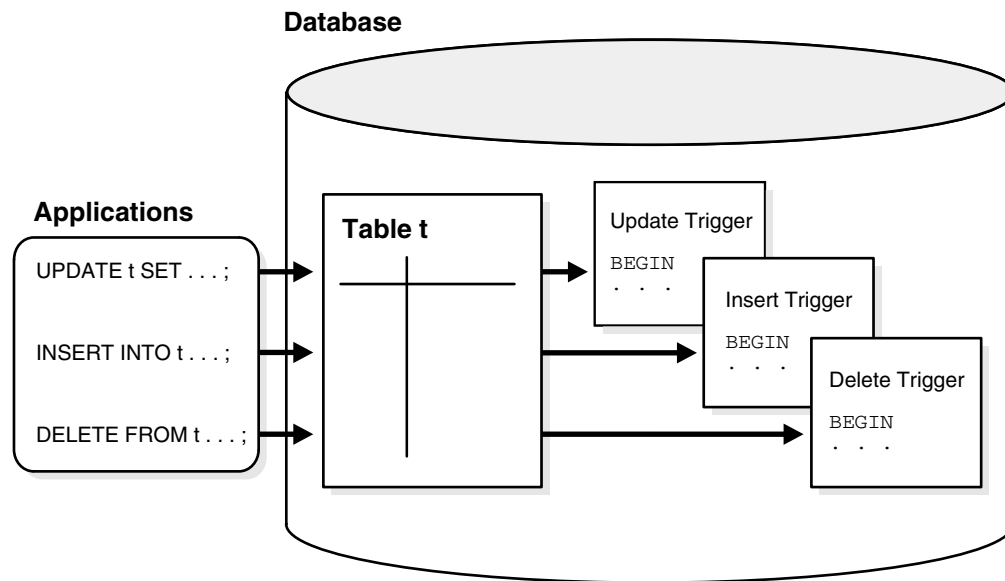
A trigger can also invoke a C procedure, which is useful for computationally intensive operations.

> **See Also:**
>
> - Chapter 24, "SQL" for information on the similarities of triggers to stored procedures
> - "The Triggering Event or Statement" on page 22-4

This section includes the following topics:

- How Triggers Are Used
- The Triggering Event or Statement

How Triggers Are Used

Triggers supplement the standard capabilities of Oracle Database to provide a highly customized database management system. For example, a trigger can restrict DML operations against a table to those issued during regular business hours. You can also use triggers to:

- Automatically generate derived column values
- Prevent invalid transactions
- Enforce complex security authorizations
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access

- Modify table data when DML statements are issued against views

- Publish information about database events, user events, and SQL statements to subscribing applications

> **See Also:** *Oracle Database PL/SQL Language Reference* for more information about triggers

This section includes the following topics:

- Some Cautionary Notes about Triggers

- Triggers Compared with Declarative Integrity Constraints

Some Cautionary Notes about Triggers

Although triggers are useful for customizing a database, use them only when necessary. Excessive use of triggers can result in complex interdependencies, which can be difficult to maintain in a large application. For example, when a trigger fires, a SQL statement within its trigger action potentially can fire other triggers, resulting in **cascading triggers**. This can produce unintended effects.

Triggers Compared with Declarative Integrity Constraints

You can use both triggers and integrity constraints to define and enforce any type of integrity rule. However, Oracle strongly recommends that you use triggers to constrain data input only in the following situations:

- To enforce referential integrity when child and parent tables are on different nodes of a distributed database

- To enforce complex business rules not definable using integrity constraints

- When a required referential integrity rule cannot be enforced using the following integrity constraints:

  - `NOT NULL, UNIQUE`

  - `PRIMARY KEY`

  - `FOREIGN KEY`

  - `CHECK`

  - `DELETE CASCADE`

  - `DELETE SET NULL`

> **See Also:** "How Oracle Database Enforces Data Integrity" on page 21-2 for more information about integrity constraints

Components of a Trigger

A trigger has three basic components, each explained in this section:

- The Triggering Event or Statement

- Trigger Restriction

- Trigger Action

Figure 22–2 represents each of these trigger components and is not meant to show exact syntax. The sections that follow explain each trigger component in greater detail.

Figure 22–2   The REORDER Trigger

REORDER Trigger


```
AFTER UPDATE OF parts_on_hand ON inventory          Triggering Statement


WHEN (new.parts_on_hand < new.reorder_point)        Trigger Restriction          Triggered Action


FOR EACH ROW
DECLARE                              /* a dummy variable for counting */
   NUMBER X;
BEGIN
   SELECT COUNT(*) INTO X          /* query to find out if part has already been */
   FROM pending_orders             /* reordered-if yes, x=1, if no, x=0 */
   WHERE part_no=:new.part_no;


IF x = 0
THEN                               /* part has not been reordered yet, so reorder */
   INSERT INTO pending_orders
   VALUES (new.part_no, new.reorder_quantity, sysdate);
 END IF;                           /* part has already been reordered */
END;
```

The Triggering Event or Statement

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire. A triggering event can be one or more of the following:

- An INSERT, UPDATE, or DELETE statement on a specific table (or view, in some cases)

- A CREATE, ALTER, or DROP statement on any schema object

- A database startup or instance shutdown

- A specific error message or any error message

- A user logon or logoff

For example, in Figure 22–2 on page 22-4, the triggering statement is:

```
... UPDATE OF parts_on_hand ON inventory ...
```

This statement means that when the parts_on_hand column of a row in the inventory table is updated, fire the trigger. When the triggering event is an UPDATE statement, you can include a column list to identify which columns must be updated to fire the trigger. You cannot specify a column list for INSERT and DELETE statements, because they affect entire rows of information.

A triggering event can specify multiple SQL statements:

```
... INSERT OR UPDATE OR DELETE OF inventory ...
```

This part means that when an INSERT, UPDATE, or DELETE statement is issued against the inventory table, fire the trigger. When multiple types of SQL statements can fire a trigger, you can use conditional predicates to detect the type of triggering statement. In this way, you can create a single trigger that runs different code based on the type of statement that fires the trigger.

Trigger Restriction

A trigger restriction specifies a Boolean expression that must be `true` for the trigger to fire. The trigger action is not run if the trigger restriction evaluates to `false` or `unknown`. In the example, the trigger restriction is:

```
new.parts_on_hand < new.reorder_point
```

Consequently, the trigger does not fire unless the number of available parts is less than a present reorder amount.

Trigger Action

A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be run when the following events occur:

- A triggering statement is issued.
- The trigger restriction evaluates to `true`.

Like stored procedures, a trigger action can:

- Contain SQL, PL/SQL, or Java statements
- Define PL/SQL language constructs such as variables, constants, cursors, exceptions
- Define Java language constructs
- Invoke stored procedures

If the triggers are row triggers, the statements in a trigger action have access to column values of the row being processed by the trigger. Correlation names provide access to the old and new values for each column.

Types of Triggers

This section describes the different types of triggers:

- Row Triggers and Statement Triggers
- BEFORE and AFTER Triggers
- Compound Triggers
- INSTEAD OF Triggers
- Triggers on System Events and User Events

Row Triggers and Statement Triggers

When you define a trigger, you can specify the number of times the trigger action is to be run:

- Once for every row affected by the triggering statement, such as a trigger fired by an `UPDATE` statement that updates many rows
- Once for the triggering statement, no matter how many rows it affects

This section includes the following topics:

- Row Triggers
- Statement Triggers

Row Triggers

A **row trigger** is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not run.

Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected. For example, Figure 22–2 on page 22-4 illustrates a row trigger that uses the values of each row affected by the triggering statement.

Statement Triggers

A **statement trigger** is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.

Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected. For example, use a statement trigger to:

- Make a complex security check on the current time or user

- Generate a single audit record

BEFORE and AFTER Triggers

When defining a trigger, you can specify the **trigger timing**—whether the trigger action is to be run before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

BEFORE and AFTER triggers fired by DML statements can be defined only on tables, not on views. However, triggers on the base tables of a view are fired if an INSERT, UPDATE, or DELETE statement is issued against the view. BEFORE and AFTER triggers fired by DDL statements can be defined only on the database or a schema, not on particular tables.

> **See Also:**
>
> - "INSTEAD OF Triggers" on page 22-8
>
> - "Triggers on System Events and User Events" on page 22-9 for information about how BEFORE and AFTER triggers can be used to publish information about DML and DDL statements

This section includes the following topics:

- BEFORE Triggers

- AFTER Triggers

- Trigger Type Combinations

BEFORE Triggers

BEFORE triggers run the trigger action before the triggering statement is run. This type of trigger is commonly used in the following situations:

- When the trigger action determines whether the triggering statement should be allowed to complete. Using a BEFORE trigger for this purpose, you can eliminate

unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action.

- To derive specific column values before completing a triggering INSERT or UPDATE statement.

AFTER Triggers

AFTER triggers run the trigger action after the triggering statement is run.

Trigger Type Combinations

Using the options listed previously, you can create four types of row and statement triggers:

- **BEFORE *statement* trigger**

  Before executing the triggering statement, the trigger action is run.

- **BEFORE *row* trigger**

  Before modifying each row affected by the triggering statement and before checking appropriate integrity constraints, the trigger action is run, if the trigger restriction was not violated.

- **AFTER *statement* trigger**

  After executing the triggering statement and applying any deferred integrity constraints, the trigger action is run.

- **AFTER *row* trigger**

  After modifying each row affected by the triggering statement and possibly applying appropriate integrity constraints, the trigger action is run for the current row provided the trigger restriction was not violated. Unlike BEFORE *row* triggers, AFTER *row* triggers lock rows.

You can have multiple triggers of the same type for the same statement for any given table. For example, you can have two BEFORE *statement* triggers for UPDATE statements on the employees table. Multiple triggers of the same type permit modular installation of applications that have triggers on the same tables. Also, Oracle Database materialized view logs use AFTER *row* triggers, so you can design your own AFTER *row* trigger in addition to the Oracle-defined AFTER *row* trigger.

You can create as many triggers of the preceding different types as you need for each type of DML statement, (INSERT, UPDATE, or DELETE).

> **See Also:** *Oracle Database PL/SQL Language Reference* for more information about trigger types

Compound Triggers

A compound trigger is a single trigger on a table that allows you to specify actions for each of four timing points:

- Before the firing statement
- Before each row that the firing statement affects
- After each row that the firing statement affects
- After the firing statement

The compound trigger body supports a common PL/SQL state that the code for each timing point can access. The common state is automatically destroyed when the firing statement completes, even when the firing statement causes an error.

The effect of the compound trigger is similar to what you could achieve with a simple trigger for each of the timing points for which you needed to code action, with an ancillary package to hold the state that these simple triggers would share. The obvious advantage of the compound trigger is that the required code is managed in a single compilation unit, but the more important advantage is that the lifetime of the compound trigger's state is automatically limited to the duration of the firing statement.

The compound trigger is useful when you want to accumulate facts that characterize the "for each row" changes and then act on them as a body at "after statement" time. Sometimes you are forced to use this approach (to avoid the mutating table error). Sometimes this approach gives better performance; for example, when maintaining a denormalized aggregate value in a master table in response to changes in a detail table, or when maintaining an audit table.

> **See Also:**   *Oracle Database PL/SQL Language Reference*

INSTEAD OF Triggers

INSTEAD OF triggers provide a transparent way of modifying views that cannot be modified directly through DML statements (INSERT, UPDATE, and DELETE). These triggers are invoked INSTEAD OF triggers because, unlike other types of triggers, Oracle Database fires the trigger instead of executing the triggering statement.

You can write normal INSERT, UPDATE, and DELETE statements against the view and the INSTEAD OF trigger is fired to update the underlying tables appropriately. INSTEAD OF triggers are activated for each row of the view that gets modified.

This section includes the following topics:

- Modify Views
- Views That Are Not Modifiable
- INSTEAD OF Triggers on Nested Tables

Modify Views

Modifying views can have ambiguous results:

- Deleting a row in a view could either mean deleting it from the base table or updating some values so that it is no longer selected by the view.
- Inserting a row in a view could either mean inserting a new row into the base table or updating an existing row so that it is projected by the view.
- Updating a column in a view that involves joins might change the semantics of other columns that are not projected by the view.

Object views present additional problems. For example, a key use of object views is to represent master/detail relationships. This operation inevitably involves joins, but modifying joins is inherently ambiguous.

As a result of these ambiguities, there are many restrictions on which views are modifiable. An INSTEAD OF trigger can be used on object views as well as relational views that are not otherwise modifiable.

A view is **inherently modifiable** if data can be inserted, updated, or deleted without using INSTEAD OF triggers and if it conforms to the restrictions listed as follows. Even

if the view is inherently modifiable, you might want to perform validations on the values being inserted, updated or deleted. `INSTEAD OF` triggers can also be used in this case. Here the trigger code performs the validation on the rows being modified and if valid, propagate the changes to the underlying tables.

`INSTEAD OF` triggers also enable you to modify object view instances on the client-side through OCI. To modify an object materialized by an object view in the client-side object cache and flush it back to the persistent store, you must specify `INSTEAD OF` triggers, unless the object view is inherently modifiable. However, it is not necessary to define these triggers for just pinning and reading the view object in the object cache.

> **See Also:**
>
> - *Oracle Database Object-Relational Developer's Guide*
>
> - *Oracle Call Interface Programmer's Guide*
>
> - *Oracle Database PL/SQL Language Reference* for more information about `INSTEAD OF` triggers

Views That Are Not Modifiable

If the view query contains any of the following constructs, the view is not inherently modifiable and therefore, you cannot perform inserts, updates, or deletes on the view:

- Set operators

- Aggregate functions

- `GROUP BY`, `CONNECT BY`, or `START WITH` clauses

- The `DISTINCT` operator

- Joins (however, some join views are updatable)

If a view contains pseudocolumns or expressions, you can only update the view with an `UPDATE` statement that does not refer to any of the pseudocolumns or expressions.

> **See Also:** "Updatable Join Views" on page 5-17

INSTEAD OF Triggers on Nested Tables

You cannot modify the elements of a nested table column in a view directly with the `TABLE` clause. However, you can do so by defining an `INSTEAD OF` trigger on the nested table column of the view. The triggers on the nested tables fire if a nested table element is updated, inserted, or deleted and handle the actual modifications to the underlying tables.

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference* for more information about triggers on nested tables
>
> - *Oracle Database SQL Language Reference* for information on the `CREATE TRIGGER` statement

Triggers on System Events and User Events

You can use triggers to publish information about database events to subscribers. Applications can subscribe to database events just as they subscribe to messages from other applications. These database events can include:

- System events

- – Database startup and shutdown

- – Data Guard role transitions

- – Server error message events

- User events

  - – User logon and logoff

  - – DDL statements (CREATE, ALTER, and DROP)

  - – DML statements (INSERT, DELETE, and UPDATE)

Triggers on system events can be defined at the database level or schema level. The DBMS_AQ package is one example of using database triggers to perform certain actions. For example, a database shutdown trigger is defined at the database level:

```
CREATE TRIGGER register_shutdown
  ON DATABASE
  SHUTDOWN
    BEGIN
    ...
    DBMS_AQ.ENQUEUE(...);
    ...
    END;
```

Triggers on DDL statements or logon/logoff events can also be defined at the database level or schema level. Triggers on DML statements can be defined on a table or view. A trigger defined at the database level fires for all users, and a trigger defined at the schema or table level fires only when the triggering event involves that schema or table.

This section includes the following topics:

- Event Publication

- Event Attributes

- System Events

- User Events

Event Publication

Event publication uses the publish-subscribe mechanism of Oracle Streams Advanced Queuing. A **queue** serves as a message repository for subjects of interest to various subscribers. Triggers use the DBMS_AQ package to enqueue a message when specific system or user events occur.

> **See Also:**
>
> - *Oracle Streams Advanced Queuing User's Guide* for information about the Oracle Streams Advanced Queuing implementation of Publish/Subscribe
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_AQ package

Event Attributes

Each event allows the use of attributes within the trigger text. For example, the database startup and shutdown triggers have attributes for the instance number and the database name, and the logon and logoff triggers have attributes for the user name. You can specify a function with the same name as an attribute when you create a

trigger if you want to publish that attribute when the event occurs. The attribute's value is then passed to the function or payload when the trigger fires. For triggers on DML statements, the `:OLD` column values pass the attribute's value to the `:NEW` column value.

System Events

System events that can fire triggers are related to instance startup and shutdown and error messages. Triggers created on startup and shutdown events have to be associated with the database. Triggers created on error events can be associated with the database or with a schema.

- `STARTUP` triggers fire when the database is opened by an instance. Their attributes include the system event, instance number, and database name.

- `SHUTDOWN` triggers fire just before the server starts shutting down an instance. You can use these triggers to make subscribing applications shut down completely when the database shuts down. For abnormal instance shutdown, these triggers cannot be fired. The attributes of `SHUTDOWN` triggers include the system event, instance number, and database name.

- `SERVERERROR` triggers fire when a specified error occurs, or when any error occurs if no error number is specified. Their attributes include the system event and error number.

- `DB_ROLE_CHANGE` triggers fire when a role transition (failover or switchover) occurs in a Data Guard configuration. The trigger notifies users when a role transition occurs, so that client connections can be processed on the new primary database and applications can continue to run.

User Events

User events that can fire triggers are related to user logon and logoff, DDL statements, and DML statements.

Triggers on LOGON and LOGOFF Events `LOGON` and `LOGOFF` triggers can be associated with the database or with a schema. Their attributes include the system event and user name, and they can specify simple conditions on `USERID` and `USERNAME`.

- `LOGON` triggers fire after a successful logon of a user.

- `LOGOFF` triggers fire at the start of a user logoff.

Triggers on DDL Statements DDL triggers can be associated with the database or with a schema. Their attributes include the system event, the type of schema object, and its name. They can specify simple conditions on the type and name of the schema object, as well as functions like `USERID` and `USERNAME`.

Triggers on DML Statements DML triggers for event publication are associated with a table. They can be either `BEFORE` or `AFTER` triggers that fire for each row on which the specified DML operation occurs. You cannot use `INSTEAD OF` triggers on views to publish events related to DML statements—instead, you can publish events using `BEFORE` or `AFTER` triggers for the DML operations on a view's underlying tables that are caused by `INSTEAD OF` triggers.

See Also:

- "Row Triggers" on page 22-6

- "BEFORE and AFTER Triggers" on page 22-6

- *Oracle Database PL/SQL Language Reference* for more information about event publication using triggers on system events and user events

Trigger Execution

A trigger is either enabled or disabled.

Table 22–1   Trigger Modes

Trigger Mode	Definition
Enabled	An enabled trigger runs its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to `true`.
Disabled	A disabled trigger does not run its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate to `true`.

For enabled triggers, Oracle Database automatically performs the following actions:

- Oracle Database runs triggers of each type in a planned firing sequence when more than one trigger is fired by a single SQL statement. First, statement level triggers are fired, and then row level triggers are fired.

- Oracle Database performs integrity constraint checking at a set point in time with respect to the different types of triggers and guarantees that triggers cannot compromise integrity constraints.

- Oracle Database provides read-consistent views for queries and constraints.

- Oracle Database manages the dependencies among triggers and schema objects referenced in the code of the trigger action

- Oracle Database uses two-phase commit if a trigger updates remote tables in a distributed database.

- Oracle Database fires multiple triggers in an unspecified, random order, if more than one trigger of the same type exists for a given statement; that is, triggers of the same type for the same statement are not guaranteed to fire in any specific order.

This section includes the following topics:

- The Execution Model for Triggers and Integrity Constraint Checking

- Data Access for Triggers

- Storage of PL/SQL Triggers

- Execution of Triggers

- Dependency Maintenance for Triggers

The Execution Model for Triggers and Integrity Constraint Checking

When a statement in a trigger body causes another trigger to fire, the triggers are said to be **cascading**. Oracle Database allows up to 32 triggers to cascade at simultaneously.

A relational database does not guarantee the order of rows processed by a SQL statement. Therefore, do not create triggers that depend on the order in which rows are processed.

> **See Also:** *Oracle Database PL/SQL Language Reference* for more information about creating triggers and the order in which they fire

Data Access for Triggers

When a trigger is fired, the tables referenced in the trigger action might be currently undergoing changes by SQL statements in other users' transactions. In all cases, the SQL statements running within triggers follow the common rules used for standalone SQL statements. In particular, if an uncommitted transaction has modified values that a trigger being fired either needs to read (query) or write (update), then the SQL statements in the body of the trigger being fired use the following guidelines:

- Queries see the current read-consistent materialized view of referenced tables and any data changed within the same transaction.

- Updates wait for existing data locks to be released before proceeding.

Storage of PL/SQL Triggers

Oracle Database stores PL/SQL triggers in compiled form, just like stored procedures. When a `CREATE TRIGGER` statement commits, the compiled PL/SQL code is stored in the database and the source code of the trigger is flushed from the shared pool.

> **See Also:** *Oracle Database PL/SQL Language Reference* for more information about compiling and storing triggers

Execution of Triggers

Oracle Database runs a trigger internally using the same steps used for subprogram execution. The only subtle difference is that a user has the right to fire a trigger if he or she has the privilege to run the triggering statement. Other than this, triggers are validated and run the same way as stored subprograms.

> **See Also:** *Oracle Database PL/SQL Language Reference* for more information about stored subprograms

Dependency Maintenance for Triggers

Like procedures, triggers depend on referenced objects. Oracle Database automatically manages the dependencies of a trigger on the schema objects referenced in its trigger action. The dependency issues for triggers are the same as those for stored procedures. Triggers are treated like stored procedures. They are inserted into the data dictionary.

> **See Also:** Chapter 6, "Schema Object Dependencies"

23

Information Integration

This chapter includes the following topics:

- Introduction to Oracle Information Integration
- Federated Access
- Information Sharing
- Data Comparison and Convergence at Oracle Databases
- Integrating Non-Oracle Systems

> **See Also:** *Oracle Database 2 Day + Data Replication and Integration Guide*

Introduction to Oracle Information Integration

As a company evolves, it becomes increasingly important for it to be able to share information among multiple databases and applications. Companies need to share OLTP updates, database events, and application messages, as customers place orders online, through the sales force, or even with a partner. This information must be routed to a variety of destinations, including heterogeneous replicated databases, message queuing systems, data warehouse staging areas, operational data stores, other applications, and a standby database.

There are three basic approaches to sharing information. You can consolidate the information into a single database, which eliminates the need for further integration. You can leave information distributed, and provide tools to federate that information, making it appear to be in a single virtual database. Or, you can share information, which lets you maintain the information in multiple data stores and applications. This chapter focuses on federating and sharing information.

> **See Also:** Chapter 16, "Business Intelligence" for more information on features to consolidate information

Oracle provides distributed SQL for federating distributed information. Distributed SQL synchronously accesses and updates data distributed among multiple databases, while maintaining location transparency and data integrity.

Oracle Streams is the asynchronous information sharing infrastructure in Oracle Database. Oracle Streams can mine the Oracle Database redo logs to capture data manipulation language (DML) and data definition language (DDL) changes to data, and it makes that changed data available to other applications and databases. Thus, Oracle Streams can provide an extremely flexible asynchronous replication solution, as well as an event notification framework. Because Streams supports applications that

explicitly enqueue and dequeue messages, it also provides a complete asynchronous messaging solution. That solution, Oracle Streams Advanced Queuing, can be used to exchange information with customers, partners, and suppliers, and to coordinate business processes.

Both Streams and distributed SQL can access and update data in non-Oracle systems using Oracle Database Gateways, Generic Connectivity, and the Messaging Gateway. Oracle Database can work with non-Oracle data sources, non-Oracle message queuing systems, and non-SQL applications, ensuring interoperability with other vendor's products and technologies. Each of the solutions are described in detail in the following sections.

A **distributed environment** is a network of disparate systems that seamlessly communicate with each other. Each system in the distributed environment is called a node. The system to which a user is directly connected is called the local system. Any additional systems accessed by this user are called remote systems. A distributed environment allows applications to access and exchange data from the local and remote systems. All the data can be simultaneously accessed and modified.

While a distributed environment enables increased access to a large amount of data across a network, it must also hide the location of the data and the complexity of accessing it across the network.

In order for a company to operate successfully in a distributed environment, it might need to do the following:

- Exchange data between Oracle Databases

- Communicate between applications

- Exchange information with customers, partners, and suppliers

- Replicate data between databases

- Communicate with non-Oracle databases

Federated Access

A homogeneous distributed database system is a network of two or more Oracle Databases that reside on one or more computers.

This section includes the following topics:

- Distributed SQL

- Location Transparency

- SQL and COMMIT Transparency

- Distributed Query Optimization

Distributed SQL

Distributed SQL enables applications and users to simultaneously access or modify the data in several databases as easily as they access or modify a single database.

An Oracle distributed database system can be transparent to users, making it appear as though it is a single Oracle Database. Companies can use this distributed SQL feature to make all its Oracle Databases look like one and thus reduce some of the complexity of the distributed system.

Oracle Database uses database links to enable users on one database to access objects in a remote database. A local user can access a link to a remote database without being a user on the remote database.

> **See Also:** *Oracle Database Administrator's Guide* for more information on database links

Location Transparency

An Oracle distributed database system lets application developers and administrators 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 the following:

- 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 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.

In addition to synonyms, developers can use views and stored procedures to establish location transparency for applications that work in a distributed database system.

> **See Also:**
>
> - Chapter 5, "Schema Objects" for more information on synonyms and views
>
> - Chapter 24, "SQL" for more information on stored procedures

SQL and COMMIT Transparency

The Oracle distributed database architecture also 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.

Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because Oracle Database must coordinate the committing or undo of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.

Oracle 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 tasks the other participating nodes to promise to commit or undo 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 undo. The two-phase commit mechanism is completely transparent, requiring no complex programming or other special operations to provide distributed transaction control.

> **See Also:** "The Two-Phase Commit Mechanism" on page 4-8

Distributed Query Optimization

Distributed query optimization 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 the Oracle Database optimizer 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 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:** *Oracle Database Performance Tuning Guide* for more information on the optimizer and hints

Information Sharing

At the heart of any integration is the sharing of data among various applications in the enterprise. **Replication** is the maintenance of database objects in two or more databases. It provides a solution to the scalability, availability, and performance issues facing many companies. For example, replication can improve the performance of a company's Web site. By locally replicating remote tables that are frequently queried by local users, such as the inventory table, the amount of data sent across the network is greatly reduced. By having local users access the local copies instead of one central copy, the distributed database does not need to send information across a network repeatedly, thus helping to maximize the performance of the database application. Oracle Streams provides powerful replication features that can be used to keep multiple copies of distributed objects synchronized.

Many companies have developed a variety of autonomous and distributed applications to automate business processes and manage business tasks. However, these applications need to communicate with each other, coordinating business processes and tasks in a consistent manner. They also need to exchange information efficiently with customers, partners, and suppliers over low-cost channels such as the Internet, while preserving a traceable history of events—a requirement previously satisfied through now obsolete paper forms.

For loose application coupling, Oracle offers Oracle Streams Advanced Queuing, which is built on top of the flexible Oracle Streams infrastructure. Oracle Streams Advanced Queuing provides a unified framework for processing events.

Events generated in applications, in workflow, or implicitly captured from redo logs or in database triggers can be captured and staged in a queue. These events can be consumed in a variety of ways. They can be applied automatically with a user-defined function or database table operation, or they can be dequeued explicitly. Also, notifications can be sent to the consuming application. These events can be transformed at any stage. If the consuming application is on a different database, then the events can be propagated to the appropriate database automatically. Operations on these events can be automatically audited, and the history can be retained for the user-specified duration.

This section includes the following topics:

- Oracle Streams
- Materialized Views

Oracle Streams

Oracle Streams enables the propagation and management of data, transactions, and events in a data stream either within a database, or from one database to another. The stream routes published information to subscribed destinations. As users' needs change, they can implement a new capability of Oracle Streams, without sacrificing existing capabilities.

Oracle Streams provides components that allow users to control what information is put into a stream, how the stream flows or is routed from node to node, what happens to messages in the stream as they flow into each node, and how the stream terminates. By specifying the configuration of the elements acting on the stream, a user can address specific requirements, such as message queuing or data replication.

Oracle Streams satisfies the information sharing requirements for a variety of usage scenarios. Oracle Streams Advanced Queuing provides the database-integrated message queuing and event management capabilities. In addition, Oracle includes tools to help users build event notification, replication, and data warehouse loading solutions using Oracle Streams.

Using the full power of Oracle Streams, you can create configurations that span multiple use cases, enabling new classes of applications. Most deployments and their associated metadata are compatible. For example, a system configured to load a data warehouse easily can be extended to enable bi-directional replication. A complete reconfiguration is not required.

This section includes the following topics:

- Oracle Streams Architecture
- Replication with Oracle Streams
- Oracle Streams Advanced Queuing
- Database Change Notification
- Change Data Capture
- Heterogeneous Environments
- Oracle Streams Use Cases

> **See Also:** *Oracle Streams Concepts and Administration*

Oracle Streams Architecture

The architecture of Oracle Streams is very flexible. As shown in Figure 23–1, Streams contains three basic elements.

- Capture
- Staging
- Consumption

Figure 23–1   Streams Information Flow
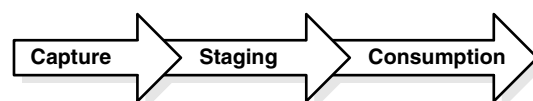
Capture Oracle Streams can capture database changes implicitly and explicitly and place these changes in a staging area. Database changes, such as DML and DDL

changes, can be implicitly captured. Rules determine which changes are captured. Information representing a captured change is formatted as a logical change record (LCR) and placed in the staging area.

Oracle Streams provides two components for implicit capture: capture processes and synchronous captures. Capture processes mine the redo logs to find database changes. Capture processes support mining the online redo log, as well as mining archived log files. In the case of online redo log mining, redo information is mined for change data at the same time it is written, reducing the latency of capture. Synchronous captures use an internal mechanism to capture DML changes as they are made to specified database objects.

Typically, capture processes are used to capture changes to a large number of database objects and to capture both DML and DDL changes. Synchronous captures are used to capture DML changes to a small number of database objects.

Oracle Streams also supports explicit capture. User applications can explicitly enqueue messages representing events into the staging area. These messages can be formatted as LCRs, which will allow them to be consumed by the apply process, or they can be formatted for consumption by another user application using an explicit dequeue.

Staging Once captured, messages are placed in a staging area. The staging area is a queue that stores and manages captured messages. LCRs and other types of messages are stored in a staging area until subscribers consume them. LCR staging provides a holding area with security, as well as auditing and tracking of LCR data.

Subscribers examine the contents of the staging area and determine whether they have an interest in the message representing a particular event. A subscriber can either be a user application, a propagation to another staging area, usually on another system, or an apply process. The subscriber optionally can evaluate a set of rules to determine whether the message meets the criteria set forth in the subscription. If so, then the message will be consumed by the subscriber.

If the subscriber is a user application, then the application dequeues the message from the staging area in order to consume the message. If the subscriber is a propagation to another staging area, then the message will be propagated to that staging area. If the subscriber is an apply process, then it will be dequeued and consumed by the apply process.

Messages in the staging area optionally may be propagated to other staging areas in the same database, or to staging areas in remote databases. To simplify network routing and reduce network traffic, messages need not be sent to all databases and applications. Rather, they can be directed through staging areas on one or more systems until they reach the subscribing system. Not all systems need subscribe to the messages, providing flexibility regarding what messages are applied at a particular system. A single staging area can stage messages from multiple databases, simplifying setup and configuration.

As messages enter the staging area, are propagated, or exit the staging area, they can be transformed. A transformation is a change in the form of an object participating in capture and apply or a change in the data it holds. Transformations can include changing the datatype representation of a particular column in a table at a particular database, adding a column to a table at one database only, or including a subset of the data in a table at a particular database.

Consumption Messages in a staging area can be consumed implicitly or explicitly. An apply process implicitly applies database changes encapsulated in messages to a database. An Oracle Streams apply process is flexible. It enables standard or custom apply of messages. A custom apply can manipulate the data or perform other actions

during apply. Support for explicit dequeue allows application developers to use Oracle Streams to reliably exchange messages. They can also notify applications of changes to data, by leveraging the change capture and propagation features of Oracle Streams.

Replication with Oracle Streams

Oracle Streams is an information sharing technology that automatically determines what information is relevant and shares that information with those who need it. This active sharing of information includes capturing and managing messages in the database, including messages that encapsulate DML and DDL changes, and propagating those messages to other databases and applications. Data changes can be applied directly to the replica database or can call a user-defined subprogram to perform alternative work at the destination database. For example, such a subprogram can populate a staging table used to load a data warehouse.

This section includes the following topics:

- Capturing DML and DDL Changes

- Propagating Changes Over a Directed Network

- Resolving Conflicts and Applying Changes

> **See Also:** *Oracle Streams Replication Administrator's Guide*

Capturing DML and DDL Changes Configuring Streams for replication begins with specifying an object or set of objects to be replicated. Using the one or both of the implicit capture mechanisms of Oracle Streams, changes made to these objects can be captured efficiently and replicated to one or more remote systems with little impact to the originating system. When a capture process is used, the capture process can extract both data changes (DML) and structure changes (DDL) from the redo log. When a synchronous capture is used, the synchronous capture uses an internal mechanism to capture DML changes when they occur. The captured changes are published to a staging area.

Log-based capture with a capture process leverages the fact that changes made to tables are logged in the redo log to guarantee recoverability in the event of a malfunction or media failure. Capturing changes directly from the redo log minimizes the overhead on the system. Oracle Database can read, analyze, and interpret redo information, which contains information about the history of activity on a database. Oracle Streams can mine the information and deliver change data to the capture process.

Capturing changes with a synchronous capture is best suited for environments that replicate DML changes to a relatively small number of database objects. Synchronous capture uses an internal mechanism to ensure that DML changes are captured when they happen.

Replicated databases utilizing Oracle Streams technology need not be identical. Participating databases can maintain different data structures, using Streams to transform the data into the appropriate format. Streams provides the ability to transform the stream at multiple points: during change capture, while propagating to another database, or during apply at the destination site. These transformations are user-defined functions registered within the Oracle Streams framework. For example, the transformation can be used to change the datatype representation of a particular column in a table, to change the name of a column in a table, or to change a table name.

The data at each site can be subsetted based on content as well. For example, the replica can use a rule which specifies that only the employees for a particular division

based on the department identifier column be contained within the table. Oracle Streams automatically manages the changes to ensure that the data within the replica matches the subset rule criteria.

Propagating Changes Over a Directed Network Messages in a staging area can be sent to staging areas in other databases. The directed network capability of Streams allows changes to be directed through intermediate databases as a pass-through. Changes at any database can be published and propagated to or through other databases anywhere on the network. By using the rules-based publish and subscribe capabilities of the staging area queues, database administrators can choose which changes are propagated to each destination database and can specify the route messages traverse on their way to a destination.

Thus, for example, a company could configure replication to capture all changes to a particular schema, propagate only changes to European customers to their European headquarters in London, apply only those changes relevant to the London office, and forward site-specific information to be applied at each field office.

This directed network approach is also friendly to Wide Area Networks (WAN), enabling changes to subsequent destinations to traverse the network once to a single site for later fan-out to other destinations, rather than sending to each destination directly.

Resolving Conflicts and Applying Changes Messages in a staging area can be consumed by an apply process, where the changes they represent are applied to database objects, or they can be consumed by an application. User-defined apply procedures enable total control over the messages to be applied.

Using custom apply, separate procedures can be defined for handling each type of DML operation (inserts, updates, or deletes) on a table. For example, using this custom apply capability, a user could write a procedure to skip the apply of all deletes for the `employees` table for employees with a salary greater than $100,000, based on a value for the employee in the `salary` table. Inserts and updates to the `employees` table would continue to be applied using the default apply engine, as would deletes for employees with salaries less than $100,000.

Custom apply could also be used to perform custom transformations of data. For example, changes to one table at the originating site might need to be applied to three different tables at the remote location.

The remote databases in a replication environment can be fully open for read/write, and need not be identical copies of the source database. Because the remote database can be updated by other means, an apply process detects conflicts before changes are applied. These conflicts also can be automatically resolved using built-in or custom resolution mechanisms.

Oracle Streams Advanced Queuing

Beyond database integration, Oracle Streams Advanced Queuing provides many features that make it the most robust and feature rich message queuing system. These features improve developer productivity and reduce the operational burden on administrators, which reduces the cost of building and maintaining Oracle-based distributed applications. These features are described in the following sections.

This section includes the following topics:

- Asynchronous Application Integration
- Extensible Integration Architecture

- Heterogeneous Application Integration

- Legacy Application Integration

- Standard-Based API Support

> **See Also:** *Oracle Streams Advanced Queuing User's Guide*

Asynchronous Application Integration Oracle Streams Advanced Queuing provides asynchronous integration of distributed applications. It offers several ways to enqueue messages. A capture process or synchronous capture can capture the messages implicitly, or applications and users can capture messages explicitly.

Messages can be enqueued with delay and expiration. Delay allows an enqueued message to be visible at a later date. Advanced Queuing also supports several ways to order messages before consumption. It supports first-in first-out ordering and priority-based ordering of messages.

Advanced Queuing also offers multiple ways to consume a message. Automatic apply lets users invoke a user-specified action for the message. Consuming applications can dequeue a message explicitly. Both blocking and nonblocking dequeue is supported. The consuming applications can choose to receive notifications either procedurally using PL/SQL, OCI, or Java callback functions. Alternatively, they can get notifications in an e-mail or by HTTP post. Consuming applications can also choose to perform automatic apply.

Extensible Integration Architecture Oracle Streams Advanced Queuing offers an extensible framework for developing and integrating distributed applications. Many applications are integrated with a distributed hub and spoke model with Oracle Database as the hub.

The distributed applications on an Oracle database communicate with queues in the same Oracle database server hub. The Oracle Database extensible framework lets multiple applications share the same queue, eliminating the need to add additional queues to support additional applications.

Also, Advanced Queuing supports multiconsumer queues, where a single message can be consumed by multiple applications. As additional applications are added, these applications can coordinate business transactions using the same queues and even the same messages in the Oracle database server hub. It offers the benefits of extensibility without losing guaranteed once and only once delivery of a message.

Advanced Queuing supports a content-based publish and subscribe model, where applications publish messages and consumers subscribe to the messages without knowledge of the publishing application. With such a model, it is possible to add consuming applications to a hub with no change required to existing applications.

If the distributed applications are running on different Oracle Databases, business communications can be automatically propagated to the appropriate Oracle Database. The propagation is managed automatically by the Oracle Streams Advanced Queuing system and is transparent to the application.

Heterogeneous Application Integration Traditionally, different applications had to use a common data model for communication. This data model was further restricted by the limited datatype support of the message-oriented middleware. Oracle Streams Advanced Queuing supports `ANYDATA` queues that can store messages of multiple datatypes.

Advanced Queuing provides applications with the full power of the Oracle type system. It includes support for scalar datatypes such as `NUMBER`, `DATE`, `VARCHAR`, and

so on, Oracle Database object types with inheritance, XMLType with additional operators for XML data, and `ANYDATA` support. In particular, with XMLType type support, application developers can make use of the full power of XML for extensibility and flexibility in business communications.

Oracle Streams Advanced Queuing also offers transformation capabilities. Applications with different data models can transform the messages while dequeuing or enqueuing the messages to or from their own data model. These transformation mappings are defined as SQL expressions, which can involve PL/SQL functions, Java functions, or external C callouts.

Legacy Application Integration The Oracle Messaging Gateway integrates Oracle Database applications with other message queuing systems, such as Websphere MQ (formerly called MQ Series) and Tibco. Because many legacy applications on mainframes communicate with Websphere MQ, there is a need for integrating these applications into an Oracle Database environment. The message gateway makes non-Oracle message queues appear as if they were Oracle Streams queues, and automatically propagates messages between Oracle Streams queues and Websphere MQ or Tibco queues.

Distributed applications spanning multiple partners can coordinate using the Internet access features of Oracle Streams Advanced Queuing. Using these features, a business partner or application can securely place an order into an Advanced Queuing queue over the Internet. Only authorized and authenticated business partners can perform these operations.

Advanced Queuing Internet operations utilize an XML-based protocol over Internet transports, such as HTTP(S), allowing messages to flow through firewalls without compromising security. Supporting the Internet for communications drastically reduces the cost of communications, and thus the cost of the entire solution.

Standard-Based API Support Oracle Streams Advanced Queuing supports industry-standard APIs: SQL, JMS, and SOAP. Database changes made using SQL are captured automatically as messages.

Similarly, the distributed messages and database changes can be applied to database tables, which can be seen using SQL. The messages can be enqueued and dequeued using industry-standard JMS. Advanced Queuing also has a SOAP-based XML API and supports OCI and OCCI to enqueue and dequeue messages.

Database Change Notification

Client applications can receive notifications when the result set of a registered query changes. For example, if the client registers a query of the `hr.employees` table, and if a user adds an employee, then the application can receive a database change notification when a new row is added to the table. A new query of `hr.employees` returns the changed result set. Database Change Notification is relevant in many development contexts, but is particularly useful to mid-tier applications that rely on cached data.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for more information on using Database Change Notification

Change Data Capture

Change Data Capture, a feature built on the Oracle Streams infrastructure, efficiently identifies and captures data that has been added to, updated, or removed from Oracle Database relational tables, and it makes the change data available for use by ETL tools

and applications. Using the Change Data Capture capabilities of Oracle Streams, it quickly identifies and processes only the data that has changed, not entire tables.

> **See Also:** *Oracle Database Data Warehousing Guide*

Heterogeneous Environments

Oracle Streams is an open information sharing solution, supporting heterogeneous replication between Oracle and non-Oracle systems. Using an Oracle Database Gateway, DML changes initiated at Oracle Databases can be applied to non-Oracle databases.

To implement capture and apply of DML changes from an Oracle Database source to a non-Oracle destination, an Oracle Database system functions as a proxy and runs the apply process that would normally be running at an Oracle Database destination site. The Oracle Database system then communicates with the non-Oracle system with an Oracle Database Gateway.

The changes are dequeued in Oracle Database itself and the local apply process applies the changes to a non-Oracle system across a network connection through an Oracle Database Gateway.

Users who want to propagate changes from a non-Oracle database to Oracle Database write an application to capture the changes made to the non-Oracle database. The application can capture the changes by reading from transaction logs or by using triggers. The application is then responsible for assembling and ordering these changes into transactions, converting them into the logical change record (LCR) format, and publishing them into the target Oracle Database staging area. These changes can be applied with a Streams apply process.

> **See Also:**
> - "Oracle Database Gateways" on page 23-14
> - *Oracle Streams Replication Administrator's Guide*

Oracle Streams Use Cases

Use Oracle Streams to create configurations that enable new classes of applications. In addition, all deployments and their associated metadata are compatible. For example, a replication installation easily can be extended to load a data warehouse or enable bi-directional replication—a complete reconfiguration is not required.

Suppose that a company uses Oracle Streams to maintain multiple copies of a corporate Web site for improved availability, scalability, and performance. Additional requirements could include a reporting database containing the most current information for analysts in a company headquarters office in New York to perform ad-hoc querying, as well as a disaster recovery database separately maintained from their New York office. Additionally, updatable materialized views can be used to support the field sales staff. A final requirement is to share data with existing applications that are hosted on a Sybase database.

Figure 23–2 illustrates this Streams configuration.

Figure 23–2   Streams Configuration
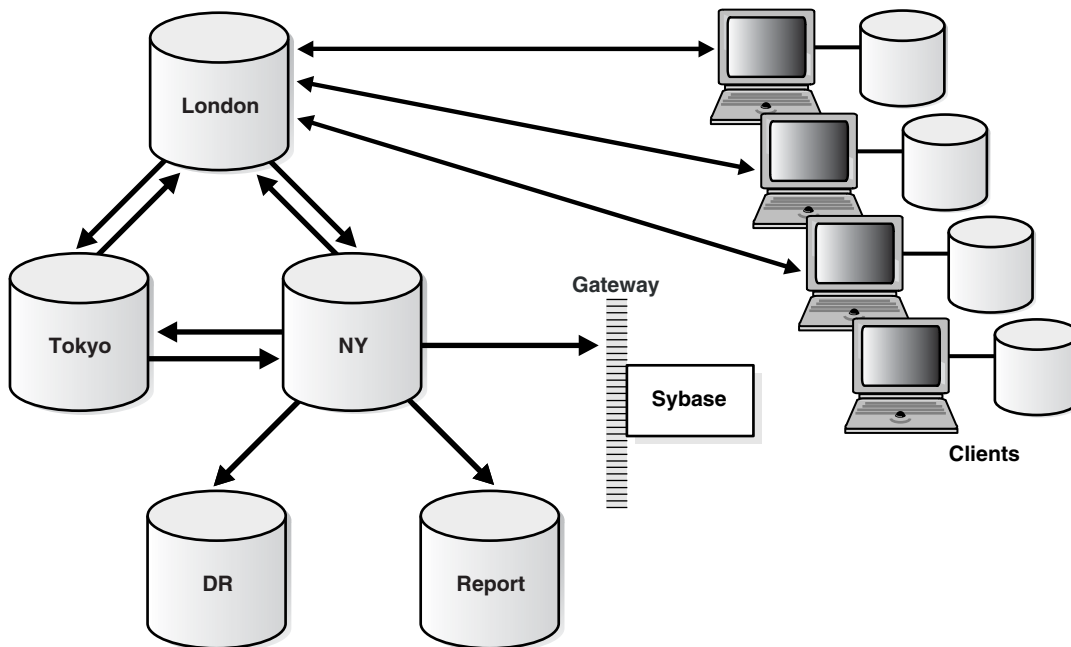
Oracle Streams is used to replicate data in an N-way configuration consisting of three regional sites: New York, London, and Tokyo. At each of these sites, Streams implicit capture collects any changes that occur for subscribed tables in each local region, and stages them locally in the queue. All changes captured in each region are then forwarded to each of the other region's databases. The goal is that all changes made at each database be reflected at every other database, providing complete data for the subscribed objects throughout the world.

Because the updates are applied automatically when received at each regional database, an Oracle Streams apply process is used to apply the changes. As changes are applied, Oracle Streams checks for and resolves any conflicts that are detected. Streams can also be used to exchange data for particular tables with non-Oracle databases. Utilizing the Oracle Database Gateway for Sybase, a Streams apply process applies the changes to a Sybase database using the same mechanisms as it does for Oracle Databases.

The databases for reporting and disaster recovery are hosted from the New York database site. The reporting database is a fully functional Oracle Database that has a read-only copy of the relevant application tables. The reporting site is not configured to capture changes on these application tables. Streams imposes no restrictions on the configuration or usage of this reporting database.

The London site also serves as the master site for several updatable materialized view sites. Each salesperson receives an updatable copy of just the portion of the data that he requires. These sites typically only connect once a day to upload their orders and download any changes since their last refresh.

Materialized Views

Oracle Streams is fully inter-operational with materialized views, or snapshots, which can be used to maintain updatable or read-only, point-in-time copies of data. They can be defined to contain a full copy of a table or a subset of the rows in the master table that satisfy a value-based selection criterion. There can be multitier materialized views as well, where one materialized view is based on another materialized view.

Materialized views are periodically updated, or refreshed, from their associated master tables through transactionally consistent batch updates.

Read-only materialized views can be used to periodically propagate the updated product catalog to the various sales offices, because the product catalog is only updated at the headquarters location.

Because materialized views do not require a dedicated connection, they are ideal for disconnected computing. For example, a company might choose to use updatable materialized views for the members of their sales force. A salesperson could enter orders into his or her laptop throughout the day, then simply connect to the regional sales office at the end of the day to upload these changes and download any updates.

> **See Also:** *Oracle Database Advanced Replication* for information about using materialized views for replication

Data Comparison and Convergence at Oracle Databases

Database objects can be shared at two or more Oracle Databases. One way to share database objects is to configure a replication environment that uses the features of Oracle Streams or materialized views. Typically, replication environments share database objects that contain data, such as tables, as well as other types of databases objects, such as indexes. When a change is made to a shared database object at one database, the change is transferred to and made at each of the other databases that share the database object. In this way, the replication environment keeps the shared database object synchronized at each database.

Sometimes, the data in these shared database objects can diverge at the databases that share the database object. Various factors can cause divergence, including network problems, computer system problems, and replication configuration errors.

The DBMS_COMPARISON package enables you to compare database objects at different databases and identify differences in them. This package also enables you converge the database objects so that they are consistent at different databases.

The DBMS_COMPARISON package can compare and converge the following types of database objects:

- Tables

- Single-table views

- Materialized views

- Synonyms for tables, single-table views, and materialized views

The results of comparisons are stored in several data dictionary views, including DBA_ COMPARISON, DBA_COMPARISION_SCAN, DBA_COMPARISON_COLUMNS, and DBA_ COMPARISON_ROW_DIF.

> **See Also:**
>
> - *Oracle Database 2 Day + Data Replication and Integration Guide*
>
> - *Oracle Database PL/SQL Packages and Types Reference*
>
> - *Oracle Database Reference*

Integrating Non-Oracle Systems

Oracle provides two solutions for integrating Oracle Database with non-Oracle databases--Generic Connectivity and Oracle Database Gateways. These solutions

enable Oracle Database clients to access non-Oracle data stores. They translate third party SQL dialects, data dictionaries, and datatypes into Oracle formats, thus making the non-Oracle data store appear to be a remote Oracle Database. These technologies enable companies to integrate seamlessly the different systems and provide a consolidated view of the company as a whole.

Generic Connectivity and Oracle Database Gateways can be used for synchronous access, using distributed SQL. In addition, Oracle Database Gateways can be used for asynchronous access, using Oracle Streams. Introducing an Oracle Database Gateway into an Oracle Streams environment enables replication of data from Oracle Database to a non-Oracle database.

Both Generic Connectivity and Oracle Database Gateways transparently access data in non-Oracle systems from an Oracle Database environment. As with an Oracle distributed database environment, location transparency can be extended to objects residing in non-Oracle systems as well. Therefore, users can create synonyms for the objects in the non-Oracle database and refer to them without having to specify its physical location. This transparency eliminates the need for application developers to customize their applications to access data from different non-Oracle systems, thus decreasing development efforts and increasing the mobility of the application. Instead of requiring applications to interoperate with non-Oracle systems using their native interfaces (which can result in intensive application-side processing), applications can be built upon a consistent Oracle Database interface for both Oracle Database and non-Oracle database systems.

This section includes the following topics:

- Generic Connectivity
- Oracle Database Gateways

Generic Connectivity

Generic Connectivity is a generic solution that uses an ODBC or OLEDB driver to access any ODBC or OLEDB compliant non-Oracle system. It provides data access to many data stores for which Oracle does not have a gateway solution. This enables transparent connectivity using industry standards, such as ODBC and OLEDB. Generic connectivity makes it possible to access low-end data stores, such as Foxpro, Access, dBase, and non-relational targets like Excel.

Oracle Database Gateways

In contrast to Generic Connectivity, which is a generic solution, Oracle Database Gateways are tailored solutions, specifically coded for the non-Oracle system. They provide an optimized solution, with more functionality and better performance than Generic Connectivity.

Generic Connectivity relies on industry standards, whereas Oracle Database Gateways access the non-Oracle systems using their native interface. Oracle Database Gateways are also end-to-end certified. Oracle Database has Oracle Database Gateways to many sources, including Sybase, DB2, Informix, and Microsoft SQL Server.

> **See Also:** *Oracle Database Heterogeneous Connectivity Administrator's Guide*

Part IV

Oracle Database Application Development

Part IV describes the languages and datatypes included with Oracle that can be used in application development. It contains the following chapters:

24

SQL

This chapter provides an overview of SQL.

This chapter includes the following topics:

- Introduction to SQL
- SQL Statements
- Cursors
- Shared SQL Areas
- Parsing
- Query Processing
- SQL Processing
- Overview of the Optimizer

> **See Also:** *Oracle Database SQL Language Reference*

Introduction to SQL

SQL is nonprocedural language that provides database access. It is nonprocedural in that users describe in SQL what they want done, and the SQL language compiler automatically generates a procedure to navigate the database and perform the desired task.

Oracle SQL includes many extensions to the ANSI/ISO standard SQL language, and Oracle tools and applications provide additional statements. The Oracle tools SQL*Plus and Oracle Enterprise Manager let you run any ANSI/ISO standard SQL statement against an Oracle database, as well as additional statements or functions that are available for those tools.

Although some Oracle tools and applications simplify or mask SQL use, all database operations are performed using SQL. Any other data access method circumvents the security built into Oracle Database and potentially compromises data security and integrity.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for detailed information about SQL statements and other parts of SQL (such as operators, functions, and format models)
> - *SQL*Plus User's Guide and Reference* for SQL*Plus statements, including their distinction from SQL statements

SQL Statements

All operations performed on the information in Oracle Database are run using SQL **statements**. A statement consists of identifiers, parameters, variables, names, datatypes, and SQL **reserved words**. SQL reserved words have special meaning in SQL and cannot be used for any other purpose. For example, SELECT and UPDATE are reserved words and cannot be used as table names.

A SQL statement is a computer program or instruction. The statement must be the equivalent of a complete SQL sentence, such as:

```
SELECT last_name, department_id FROM employees;
```

Only complete SQL statements can be run. A fragment such as the following generates an error indicating that more text is required before a SQL statement can run:

```
SELECT last_name
```

Oracle Database SQL statements are divided into the following categories:

- Data Manipulation Language Statements
- Data Definition Language Statements
- Transaction Control Statements
- Session Control Statements
- System Control Statements
- Embedded SQL Statements

> **See Also:** Chapter 22, "Triggers" for more information about using SQL statements in PL/SQL program units

Data Manipulation Language Statements

Data manipulation language (DML) statements query or manipulate data in existing schema objects. They enable you to:

- Retrieve or fetch data from one or more tables or views (SELECT); fetches can be scrollable (see "Scrollable Cursors" on page 24-5).
- Add new rows of data into a table or view (INSERT).
- Change column values in existing rows of a table or view (UPDATE).
- Update or insert rows conditionally into a table or view (MERGE).
- Remove rows from tables or views (DELETE).
- View the execution plan for a SQL statement (EXPLAIN PLAN).
- Lock a table or view, temporarily limiting other users' access (LOCK TABLE).

DML statements are the most frequently used SQL statements. Some examples of DML statements are:

```
SELECT last_name, manager_id, commission_pct + salary FROM employees;

INSERT INTO employees VALUES
    (1234, 'DAVIS', 'SALESMAN', 7698, '14-FEB-1988', 1600, 500, 30);

DELETE FROM employees WHERE last_name IN ('WARD','JONES');
```

DML Error Logging

When a DML statement encounters an error, the statement can continue processing while the error code and the associated error message text is logged to an error logging table. This is particularly helpful to long-running, bulk DML statements. After the DML operation completes, you can check the error logging table to correct rows with errors.

New syntax is added to the DML statements to provide the name of the error logging table, a statement tag, and a reject limit. The reject limit determines whether the statement should be aborted. For parallel DML operations, the reject limit is applied for each slave. The only values for the reject limit that are precisely enforced on parallel operations are zero and unlimited.

With data conversion errors, Oracle Database tries to provide a meaningful value to log for the column. For example, it could log the value of the first operand to the conversion operator that failed. If a value cannot be derived, then NULL is logged for the column.

> **See Also:**
>
> - "Description of SQL Statement Processing" on page 24-8
>
> - *Oracle Database Administrator's Guide* for more information on DML error logging
>
> - *Oracle Database Data Warehousing Guide* for examples using DML error logging
>
> - *Oracle Database SQL Language Reference* for the syntax for DML error logging

Data Definition Language Statements

Data definition language (DDL) statements define, alter the structure of, and drop schema objects. DDL statements enable you to:

- Create, alter, and drop schema objects and other database structures, including the database itself and database users (CREATE, ALTER, DROP).

- Change the names of schema objects (RENAME).

- Delete all the data in schema objects without removing the objects' structure (TRUNCATE).

- Grant and revoke privileges and roles (GRANT, REVOKE).

- Turn auditing options on and off (AUDIT, NOAUDIT).

- Add a comment to the data dictionary (COMMENT).

DDL statements implicitly commit the preceding commands and start new transactions. Some examples of DDL statements are:

```
CREATE TABLE plants
    (COMMON_NAME VARCHAR2 (15), LATIN_NAME VARCHAR2 (40));

DROP TABLE plants;

GRANT SELECT ON employees TO scott;

REVOKE DELETE ON employees FROM scott;
```

See Also:

-
-

Transaction Control Statements

Transaction control statements manage the changes made by DML statements and group DML statements into transactions. They enable you to:

- Make changes to a transaction permanent (COMMIT).
- Undo the changes in a transaction, since the transaction started or since a savepoint (ROLLBACK).
- Set a point to which you can roll back (SAVEPOINT).
- Establish properties for a transaction (SET TRANSACTION).

See Also:

Session Control Statements

Session control statements manage the properties of a particular user's session. For example, they enable you to:

- Alter the current session by performing a specialized function, such as enabling and disabling the SQL trace facility (ALTER SESSION).
- Enable and disable roles (groups of privileges) for the current session (SET ROLE).

System Control Statements

System control statements change the properties of the Oracle database instance. The only system control statement is ALTER SYSTEM. It enables you to change settings (such as the minimum number of shared servers), kill a session, and perform other tasks.

Embedded SQL Statements

Embedded SQL statements incorporate DDL, DML, and transaction control statements within a procedural language program. They are used with the Oracle precompilers. Embedded SQL statements enable you to:

- Define, allocate, and release cursors (DECLARE CURSOR, OPEN, CLOSE).
- Specify a database and connect to Oracle Database (DECLARE DATABASE, CONNECT).
- Assign variable names (DECLARE STATEMENT).
- Initialize descriptors (DESCRIBE).
- Specify how error and warning conditions are handled (WHENEVER).
- Parse and run SQL statements (PREPARE, EXECUTE, EXECUTE IMMEDIATE).
- Retrieve data from the database (FETCH).

Cursors

A **cursor** is a handle or name for a **private SQL area**—an area in memory that holds a parsed statement and other information for processing.

Although most Oracle Database users rely on the automatic cursor handling of the Oracle Database utilities, the programmatic interfaces offer application designers more control over cursors. In application development, a cursor is a named resource available to a program and can be used specifically to parse SQL statements embedded within the application.

Each user session can open multiple cursors up to the limit set by the initialization parameter OPEN_CURSORS. However, applications should close unneeded cursors to conserve system memory. If a cursor cannot be opened due to a limit on the number of cursors, then the database administrator can alter the OPEN_CURSORS initialization parameter.

Some statements (primarily DDL statements) require Oracle Database to implicitly issue recursive SQL statements, which also require recursive cursors. For example, a CREATE TABLE statement causes many updates to various data dictionary tables to record the new table and columns. Recursive calls are made for those recursive cursors; one cursor can run several recursive calls. These recursive cursors also use shared SQL areas.

Scrollable Cursors

Execution of a cursor puts the results of the query into a set of rows called the result set, which can be fetched sequentially or nonsequentially. Scrollable cursors are cursors in which fetches and DML operations do not need to be forward sequential only. Interfaces exist to fetch previously fetched rows, to fetch the *n*th row in the result set, and to fetch the *n*th row from the current position in the result set.

> **See Also:** *Oracle Call Interface Programmer's Guide* for more information about using scrollable cursors in OCI

Shared SQL Areas

Oracle Database automatically notices when applications send similar SQL statements to the database. The SQL area used to process the first occurrence of the statement is *shared*—that is, used for processing subsequent occurrences of that same statement. Therefore, only one shared SQL area exists for a unique statement. Because shared SQL areas are shared memory areas, any Oracle Database process can use a shared SQL area. The sharing of SQL areas reduces memory use on the database server, thereby increasing system throughput.

In evaluating whether statements are similar or identical, Oracle Database considers SQL statements issued directly by users and applications as well as recursive SQL statements issued internally by a DDL statement.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* and *Oracle Database Performance Tuning Guide* for more information about shared SQL

Parsing

Parsing is one stage in the processing of a SQL statement. When an application issues a SQL statement, the application makes a parse call to Oracle Database. During the parse call, Oracle Database:

- Checks the statement for syntactic and semantic validity.

- Determines whether the process issuing the statement has privileges to run it.

- Allocates a private SQL area for the statement.

Oracle Database also determines whether or not there is an existing shared SQL area containing the parsed representation of the statement in the library cache. If so, the user process uses this parsed representation and runs the statement immediately. If not, Oracle Database generates the parsed representation of the statement, and the user process allocates a shared SQL area for the statement in the library cache and stores its parsed representation there.

Note the difference between an application making a parse call for a SQL statement and Oracle Database actually parsing the statement.

- A **parse call** by the **application** associates a SQL statement with a private SQL area. After a statement has been associated with a private SQL area, it can be run repeatedly without your application making a parse call.

- A **parse operation** by Oracle Database allocates a shared SQL area for a SQL statement. Once a shared SQL area has been allocated for a statement, it can be run repeatedly without being reparsed.

Both parse calls and parsing can be expensive relative to execution, so perform them as seldom as possible.

> **See Also:** "Overview of PL/SQL" on page 1-36

Although parsing a SQL statement validates that statement, parsing only identifies errors that can be found *before statement execution*. Thus, some errors cannot be caught by parsing. For example, errors in data conversion or errors in data (such as an attempt to enter duplicate values in a primary key) and deadlocks are all errors or situations that can be encountered and reported only during the execution stage.

Query Processing

Queries are different from other types of SQL statements because, if successful, they return data as results. Whereas other statements simply return success or failure, a query can return one row or thousands of rows. The results of a query are *always* in **tabular format,** and the rows of the result are **fetched** (retrieved), either a row at a time or in groups.

Several issues relate only to query processing. Queries include not only explicit SELECT statements but also the implicit queries *(*subqueries*)* in other SQL statements. For example, each of the following statements requires a query as a part of its execution:

```
INSERT INTO table SELECT...

UPDATE table SET x = y WHERE...

DELETE FROM table WHERE...

CREATE table AS SELECT...
```

In particular, queries:

- Require **read consistency**
- Can use temporary segments for intermediate processing
- Can require the describe, define, and fetch stages of SQL statement processing.

SQL Processing

This section introduces the basics of SQL processing. It starts with a flow chart of typical SQL statement execution which generally covers most types of SQL statements. followed by a general description of the stages of SQL statement processing, and then a section indicating how the flow chart and description may differ for different types of SQL statements.

Topics include:

- Flow Chart of SQL Statement Execution
- Description of SQL Statement Processing
- Processing Other Types of SQL Statements

Flow Chart of SQL Statement Execution

Figure 24–1 outlines the stages commonly used to process and run a SQL statement. In some cases, Oracle Database can run these stages in a slightly different order. For example, the DEFINE stage could occur just before the FETCH stage, depending on how you wrote your code.

For many Oracle tools, several of the stages are performed automatically. Most users need not be concerned with or aware of this level of detail. However, this information could be useful when writing Oracle applications.

Figure 24–1 The Stages in Processing a SQL Statement
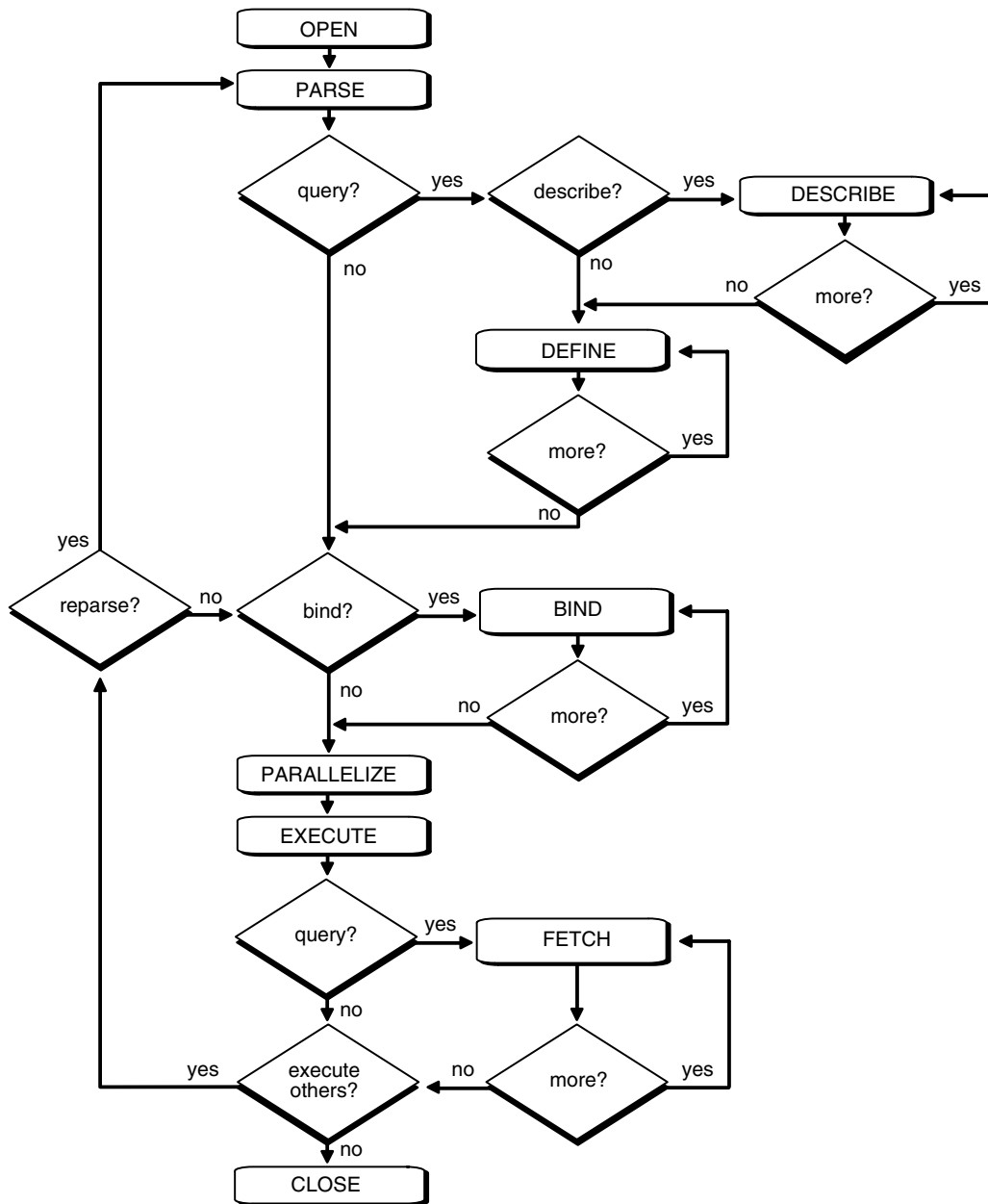
Description of SQL Statement Processing

This section provides an example of what happens during the execution of a SQL statement in each stage of processing. While this example specifically processes a DML statement, you can generalize it for other types of SQL statements. The subsequent section provides information on how execution of other types of SQL statements may differ from this description. See "Processing Other Types of SQL Statements" on page 24-11.

Assume that you are using a Pro*C program to increase the salary for all employees in a department. The program you are using has connected to Oracle Database and you are connected to the proper schema to update the employees table. You can embed the following SQL statement in your program:

```
EXEC SQL UPDATE employees SET salary = 1.10 * salary
    WHERE department_id = :department_id;
```

Department_id is a program variable containing a value for department number. When the SQL statement is run, the value of department_id is used, as provided by the application program.

The following lists the stages necessary for each type of statement processing, noting that Stage 7 is optional and Stages 4, 5, and 9 apply only to queries as indicated in Figure 24–1:

- Stage 1: Open or Create a Cursor

- Stage 2: Parse the Statement

- Stage 3: Determine if there is a Query

- Stage 4: Describe Results of a Query (Queries Only)

- Stage 5: Define Output of a Query (Queries Only)

- Stage 6: Bind Any Variables

- Stage 7: Parallelize the Statement (Optional)

- Stage 8: Run the Statement

- Stage 9: Fetch Rows of a Query (Queries Only)

- Stage 10: Close the Cursor

Stage 1: Open or Create a Cursor

A program interface call opens or creates a cursor. The cursor is created independent of any SQL statement: it is created in expectation of a SQL statement. In most applications, cursor creation is automatic. However, in precompiler programs, cursor creation can either occur implicitly or be explicitly declared.

Stage 2: Parse the Statement

During parsing, the SQL statement is passed from the user process to Oracle Database, and a parsed representation of the SQL statement is loaded into a shared SQL area. Many errors can be caught during this stage of statement processing.

> **See Also:**
> - "Parsing" on page 24-6
> - "Shared SQL Areas" on page 24-5

Stage 3: Determine if there is a Query

This stage determines if the SQL statement starts with a query.

> **See Also:** "Parsing" on page 24-6

Stage 4: Describe Results of a Query (Queries Only)

The describe stage is necessary only if the characteristics of a query's result are not known; for example, when a query is entered interactively by a user. In this case, the describe stage determines the characteristics (datatypes, lengths, and names) of a query's result.

Stage 5: Define Output of a Query (Queries Only)

In the define stage for queries, you specify the location, size, and datatype of variables defined to receive each fetched value. These variables are called **define variables**. Oracle Database performs datatype conversion if necessary. (See DEFINE on Figure 24–1, "The Stages in Processing a SQL Statement".)

Stage 6: Bind Any Variables

At this point, Oracle Database knows the meaning of the SQL statement but still does not have enough information to run the statement. Oracle Database needs values for any variables listed in the statement; in the example, Oracle Database needs a value for department_id. The process of obtaining these values is called **binding variables**.

A program must specify the location (memory address) where the value can be found. End users of applications may be unaware that they are specifying bind variables, because the Oracle Database utility can simply prompt them for a new value.

Because you specify the location (binding by reference), you need not rebind the variable before reexecution. You can change its value and Oracle Database looks up the value on each execution, using the memory address.

You must also specify a datatype and length for each value (unless they are implied or defaulted) if Oracle Database needs to perform datatype conversion.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide*
> - *Pro*C/C++ Programmer's Guide* (see Dynamic SQL Method 4)
>
> for more information about specifying a datatype and length for a value

Stage 7: Parallelize the Statement (Optional)

Oracle Database can parallelize queries (such as SELECT, INSERT, UPDATE, MERGE, DELETE), and some DDL operations such as index creation, creating a table with a subquery, and operations on partitions. Parallelization causes multiple server processes to perform the work of the SQL statement so it can complete faster.

> **See Also:** Chapter 16, "Business Intelligence"

Stage 8: Run the Statement

At this point, Oracle Database has all necessary information and resources, so the statement is run. If the statement is a query or an INSERT statement, no rows need to be locked because no data is being changed. If the statement is an UPDATE or DELETE statement, however, all rows that the statement affects are locked until the next COMMIT, ROLLBACK, or SAVEPOINT for the transaction. This ensures data integrity.

For some statements you can specify a number of executions to be performed. This is called **array processing**. Given *n* number of executions, the bind and define locations are assumed to be the beginning of an array of size *n*.

Stage 9: Fetch Rows of a Query (Queries Only)

In the fetch stage, rows are selected and ordered (if requested by the query), and each successive fetch retrieves another row of the result until the last row has been fetched.

Stage 10: Close the Cursor

The final stage of processing a SQL statement is closing the cursor.

Processing Other Types of SQL Statements

The following sections discuss how DDL, Transaction Control, and other SQL statements can differ from the process just described in

This section includes the following topics:

- DDL Statement Processing
- Transaction Control Processing
- Other Processing Types

DDL Statement Processing

The execution of DDL statements differs from the execution of DML statements and queries, because the success of a DDL statement requires write access to the data dictionary. For these statements, parsing (Stage 2) actually includes parsing, data dictionary lookup, and execution.

Transaction Control Processing

In general, only application designers using the programming interfaces to Oracle Database are concerned with the types of actions that are grouped together as one transaction. Transactions must be defined so that work is accomplished in logical units and data is kept consistent. A transaction consists of all of the necessary parts for one logical unit of work, no more and no less.

- Data in all referenced tables should be in a consistent state before the transaction begins and after it ends.
- Transactions should consist of only the SQL statements that make one consistent change to the data.

For example, a transfer of funds between two accounts (the transaction or logical unit of work) should include the debit to one account (one SQL statement) and the credit to another account (one SQL statement). Both actions should either fail or succeed together as a unit of work; the credit should not be committed without the debit. Other unrelated actions, such as a new deposit to one account, should not be included in the transfer of funds transaction.

Other Processing Types

Transaction management, session management, and system management SQL statements are processed using the parse and run stages. To rerun them, simply perform another execute.

Overview of the Optimizer

All SQL statements use the **optimizer**, a part of Oracle Database that determines the most efficient means of accessing the specified data. Oracle also provides techniques that you can use to make the optimizer perform its job better.

There are often many different ways to process a SQL DML (SELECT, INSERT, UPDATE, MERGE, or DELETE) statement; for example, by varying the order in which tables or indexes are accessed. The procedure Oracle Database uses to run a statement

can greatly affect how quickly the statement runs. The optimizer considers many factors among alternative access paths.

> **Note:** The optimizer might not make the same decisions from one version of Oracle Database to the next. In recent versions, the optimizer might make decisions based on better information available to it.

You can influence the optimizer's choices by setting the optimizer approach and goal. Objects with stale or no statistics are automatically analyzed. You can also gather statistics for the optimizer using the PL/SQL package DBMS_STATS.

Oracle Database 11*g* introduces new extended statistics including the following:

- multi-column statistics
- statistics for a functions on a column
- statistics on views

Also, you can now gather statistics without having them published. You can test the newly gathered statistics (pending statistics) before they are published.

Sometimes the application designer, who has more information about a particular application's data than is available to the optimizer, can choose a more effective way to run a SQL statement. The application designer can use hints in SQL statements to specify how the statement should be run.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about using DBMS_STATS
> - *Oracle Database Performance Tuning Guide* for more information about the optimizer

This section includes the following topics:

- SQL Plan Management (SPM)
- Execution Plans

SQL Plan Management (SPM)

With Oracle Database 11*g*, the optimizer automatically manages plans and ensures that only verified plans are used. SQL Plan Management (SPM) allows controlled plan evolution by only using a new plan after it has been verified to be perform better than the current plan.

Execution Plans

To run a DML statement, Oracle Database might need to perform many steps. Each of these steps either retrieves rows of data physically from the database or prepares them in some way for the user issuing the statement. The combination of the steps Oracle Database uses to run a statement is called an execution plan. An execution plan includes an access method for each table that the statement accesses and an ordering of the tables (the join order). The steps of the execution plan are not performed in the order in which they are numbered.

This section includes the following topics:

- Stored Outlines

- Editing Stored Outlines

Stored Outlines

Stored outlines are abstractions of an execution plan generated by the optimizer at the time the execution plan is created and are represented primarily as a set of hints. When the outline is subsequently used, these hints are applied at various stages of compilation. Outline data is stored in the OUTLN schema. You can tune execution plans by editing stored outlines.

Editing Stored Outlines

The outline is cloned into the user's schema at the onset of the outline editing session. All subsequent editing operations are performed on that clone until the user is satisfied with the edits and chooses to publicize them. In this way, any editing done by the user does not impact the rest of the user community, which would continue to use the public version of the outline until the edits are explicitly saved.

> **Note:**   Stored outlines are deprecated in Oracle Database 11*g*. Oracle highly recommends the use of SQL plan baselines instead of the stored outlines.
>
> See *Oracle Database Performance Tuning Guide* for details about execution plans and SQL plan baselines

25

Supported Application Development Languages

This chapter presents brief overviews of Oracle application development systems.

This chapter includes the following topics:

- Introduction to Oracle Application Development Languages
- Overview of C/C++ Programming Languages
- Overview of PL/SQL
- Overview of Java
- Overview of Microsoft Programming Languages
- Overview of Legacy Languages

> **See Also:** Chapter 24, "SQL"

Introduction to Oracle Application Development Languages

Oracle Database developers have a choice of languages for developing applications—C, C++, Java, COBOL, PL/SQL, Visual Basic, and C#. All language-specific standards are supported. Developers can choose the languages in which they are most proficient or one that is most suitable for a specific task. For example an application might use Java on the server side to create dynamic Web pages, PL/SQL to implement stored procedures in the database, and C++ to implement computationally intensive logic in the middle tier.

Oracle also provides the Pro* series of precompilers, which allow you to embed SQL and PL/SQL in your C, C++, COBOL, or FORTRAN application programs.

> **See Also:**
>
> - *Oracle Database Advanced Application Developer's Guide* for information on how to choose a programming environment
> - *Oracle Database Globalization Support Guide*
> - Chapter 26, "Oracle Data Types"

Overview of C/C++ Programming Languages

This section includes the following topics:

- Overview of Oracle Call Interface (OCI)

- Overview of Oracle C++ Call Interface (OCCI)
- Overview of the Oracle Type Translator
- Overview of Pro*C/C++ Precompiler

Overview of Oracle Call Interface (OCI)

Oracle Call Interface (OCI) is an application programming interface (API) that lets you create applications that use the native function calls of the C language to access an Oracle database and control all phases of SQL statement execution. OCI supports the data types, calling conventions, syntax, and semantics of C. OCI can directly access data in Oracle Database tables or can enqueue and dequeue data into or out of Oracle Streams.

OCI provides the following:

- Instant client, a way to deploy applications in a much reduced disk space.
- Thread management, connection pooling, globalization functions, and direct path loading of data (SQL*Loader Utility) from a C application.
- N-tiered authentication.
- Comprehensive support for application development using Oracle Database objects.
- Access to external databases.
- Applications that can service an increasing number of users and requests without additional hardware investments.

OCI lets you manipulate data and schemas in an Oracle database using the C host programming language. It provides a library of standard database access and retrieval functions in the form of a dynamic runtime library (OCI library) that can be linked in an application at runtime.

Query results can be cached in memory in the OCI client result cache. The OCI client can then use cached results for future executions of these queries. Because retrieving results from the result cache is faster than making a database call and rerunning the query, frequently run queries experience a significant performance improvement when their results are cached. The result cache on the OCI client is per process and shared between the different sessions.

> **See Also:** *Oracle Call Interface Programmer's Guide* for more information about the OCI client result cache

OCI supports object-relational features of Oracle Database. One important component is a set of calls that allows application programs to use a workspace called the **object cache**. The object cache is a memory block on the client side that allows programs to store entire objects and to navigate among them without round-trips to the server.

The object cache is completely under the control and management of the application programs using it. Oracle Database has no access to it. The application programs using it must maintain data coherency with the server and protect the workspace against simultaneous conflicting access.

Query results can be cached in memory in the OCI client result cache. The OCI client can then use cached results for future executions of these queries. Because retrieving results from the result cache is faster than making a database call and rerunning the query, frequently run queries experience a significant performance improvement when

their results are cached. The result cache on the OCI client is per process and shared between the different sessions.

> **See Also:** *Oracle Call Interface Programmer's Guide* for more information about the OCI client result cache

OCI provides functions to:

- Access objects on the server using SQL

- Access, manipulate and manage objects in the object cache by traversing pointers or `REF`s

- Convert Oracle Database dates, strings, and numbers to C data types

- Manage the size of the object cache's memory

- Create transient type descriptions. Transient type descriptions are not stored persistently in the database.

OCI improves concurrency by allowing individual objects to be locked. It improves performance by supporting complex object retrieval.

OCI developers can use the object type translator to generate the C structure data types corresponding to Oracle Database object types.

> **See Also:** *Oracle Call Interface Programmer's Guide*

Overview of Oracle C++ Call Interface (OCCI)

The Oracle C++ Call Interface (OCCI) is a C++ API that lets you use the object-oriented features, native classes, and methods of the C++ programing language to access Oracle Database. OCCI is built on top of OCI and combines its power and performance with the significantly more accessible interface of an object-oriented paradigm.

This section includes the following topics:

- OCCI Associative Relational and Object Interfaces

- OCCI Navigational Interface

OCCI Associative Relational and Object Interfaces

The associative relational API and object classes provide SQL access to the database. Through these interfaces, SQL is run on the server to create, manipulate, and fetch object or relational data. Applications can access any dataype on the server, including: large objects, objects and structured types, arrays, and references.

OCCI Navigational Interface

The navigational interface is a C++ interface that lets you seamlessly access and modify object-relational data in the form of C++ objects without using SQL. The C++ objects are transparently accessed and stored in the database as needed.

With the OCCI navigational interface, you can retrieve an object and navigate through references from that object to other objects. Server objects are materialized as C++ class instances in the application cache. An application can use OCCI object navigational calls to perform the following functions on the server's objects:

- Create, access, lock, delete, and flush objects

- Get references to the objects and navigate through them

> **See Also:** *Oracle C++ Call Interface Programmer's Guide*

Overview of the Oracle Type Translator

The Oracle type translator (OTT) is a program that automatically generates C language structure declarations corresponding to object types. It generates C++ class definitions for Oracle Database object types that can be used by OCCI applications for a native C++ object interface. OTT uses the Pro*C/C++ precompiler and the OCI server access package.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide*
> - *Oracle C++ Call Interface Programmer's Guide*
> - *Pro*C/C++ Programmer's Guide*

Overview of Pro*C/C++ Precompiler

An Oracle precompiler is a programming tool that lets you embed SQL statements in a high-level source program. The precompiler accepts the host program as input, translates the embedded SQL statements into standard Oracle Database run-time library calls, and generates a source program that you can compile, link, and run in the usual way. Oracle precompilers are available (but not on all systems) for C, C++, COBOL, and FORTRAN.

The Oracle Pro*C/C++ Precompiler lets you embed SQL statements in a C or C++ source file. Pro*C/C++ reads the source file as input and outputs a C or C++ source file that replaces the embedded SQL statements with Oracle Database runtime library calls, and is then compiled by the C or C++ compiler.

Pro*C/C++ lets you create highly customized applications. For example, you can create user interfaces that incorporate the latest windowing and mouse technology. You can also create applications that run in the background without the need for user interaction.

Furthermore, Pro*C/C++ helps you fine-tune your applications. It allows close monitoring of resource use, SQL statement execution, and various runtime indicators. With this information, you can change program parameters for maximum performance.

Although precompiling adds a step to the application development process, it saves time. The precompiler, not you, translates each embedded SQL statement into calls to the Oracle Database runtime library (SQLLIB). The Pro*C/C++ precompiler also analyzes host variables, defines mappings of structures into columns, and, with `SQLCHECK=FULL`, performs semantic analysis of the embedded SQL statements.

The Oracle Pro*C/C++ precompiler also allows programmers to use object data types in C and C++ programs. Pro*C/C++ developers can use the Object Type Translator to map Oracle Database object types and collections into C data types to be used in the Pro*C/C++ application.

Pro*C/C++ developers can also call OCI functions from their programs.

Pro*C/C++ provides compile time type checking of object types and collections and automatic type conversion from database types to C data types. Pro*C/C++ includes an `EXEC SQL` syntax to create and destroy objects and offers two ways to access objects in the server:

- SQL statements and PL/SQL functions or procedures embedded in Pro*C/C++ programs

- A simple interface to the object cache, where objects can be accessed by traversing pointers, then modified and updated on the server

   See Also:

   - "Overview of Microsoft Programming Languages" on page 25-30

   - *Pro*C/C++ Programmer's Guide* for a complete description of the Pro*C/C++ precompiler

Dynamic Creation and Access of Type Descriptions

Oracle provides a C API to enable dynamic creation and access of type descriptions. Additionally, you can create transient type descriptions, type descriptions that are not stored persistently in the database.

The C API enables creation and access of `OCIAnyData` and `OCIAnyDataSet`.

- The `OCIAnyData` type models a self descriptive (with regard to type) data instance of a given type.

- The `OCIAnyDataSet` type models a set of data instances of a given type.

Oracle also provides SQL data types (in Oracle's Open Type System) that correspond to these data types.

- `SYS.ANYTYPE` corresponds to `OCIType`

- `SYS.ANYDATA` corresponds to `OCIAnyData`

- `SYS.ANYDATASET` corresponds to `OCIAnyDataSet`

You can create database table columns and SQL queries on such data.

The C API uses the following terms:

- **Transient types** - Type descriptions (type metadata) that are not stored persistently in the database.

- **Persistent types** - SQL types created using the `CREATE TYPE` SQL statement. Their type descriptions are stored persistently in the database.

- **Self-descriptive data** - Data encapsulating type information along with the actual contents. The `ANYDATA` type (`OCIAnyData`) models such data. A data value of any SQL type can be converted to an `ANYDATA`, which can be converted back to the old data value. An incorrect conversion attempt results in an error.

- **Self-descriptive MultiSet** - Encapsulation of a set of data instances (all of the same type), along with their type description.

   See Also:

   - *Oracle Database Object-Relational Developer's Guide*

   - *Oracle Call Interface Programmer's Guide*

Overview of PL/SQL

PL/SQL is the Oracle procedural language extension to SQL. It provides a server-side, stored procedural language that is easy-to-use, seamless with SQL, robust, portable, and secure. The PL/SQL compiler and interpreter are embedded in Oracle Developer, providing developers with a consistent and leveraged development model on both the

client and the server side. In addition, PL/SQL stored procedures can be called from a number of Oracle Database clients, such as Pro*C or Oracle Call Interface, and from Oracle Reports and Oracle Forms.

PL/SQL enables you to mix SQL statements with procedural constructs. With PL/SQL, you can define and run PL/SQL program units such as procedures, functions, and packages. PL/SQL program units generally are categorized as anonymous blocks and stored procedures.

An **anonymous block** is a PL/SQL block that appears in your application and is not named or stored in the database. In many applications, PL/SQL blocks can appear wherever SQL statements can appear.

A **stored procedure** is a PL/SQL block that Oracle Database stores in the database and can be called by name from an application. When you create a stored procedure, Oracle Database parses the procedure and stores its parsed representation in the database. Oracle Database also lets you create and store functions (which are similar to procedures) and packages (which are groups of procedures and functions).

> **See Also:**
>
> "Overview of Java" on page 25-17
>
> Chapter 22, "Triggers"

This section includes the following topics:

- How PL/SQL Runs
- Language Constructs for PL/SQL
- PL/SQL Program Units
- Stored Procedures and Functions
- PL/SQL Packages
- PL/SQL Collections and Records
- PL/SQL Server Pages

How PL/SQL Runs

PL/SQL can run with either of the following:

- Interpreted Execution
- Native Execution

Interpreted Execution

In versions earlier than Oracle9*i*, PL/SQL source code was always compiled into a so-called bytecode representation, which is run by a portable virtual computer implemented as part of Oracle Database, and also in products such as Oracle Forms. Starting with Oracle9*i*, you can choose between native execution and interpreted execution

Native Execution

For best performance on computationally intensive program units, compile the source code of PL/SQL program units stored in the database directly to object code for the given platform. (This object code is linked into Oracle Database.)

See Also: *Oracle Database PL/SQL Language Reference*

The **PL/SQL engine** is the tool you use to define, compile, and run PL/SQL program units. This engine is a special component of many Oracle products, including Oracle Database.

While many Oracle products have PL/SQL components, this section specifically covers the program units that can be stored in Oracle Database and processed using Oracle Database PL/SQL engine. The PL/SQL capabilities of each Oracle tool are described in the appropriate tool's documentation.

Figure 25–1 illustrates the PL/SQL engine contained in Oracle Database.

Figure 25–1   The PL/SQL Engine and Oracle Database
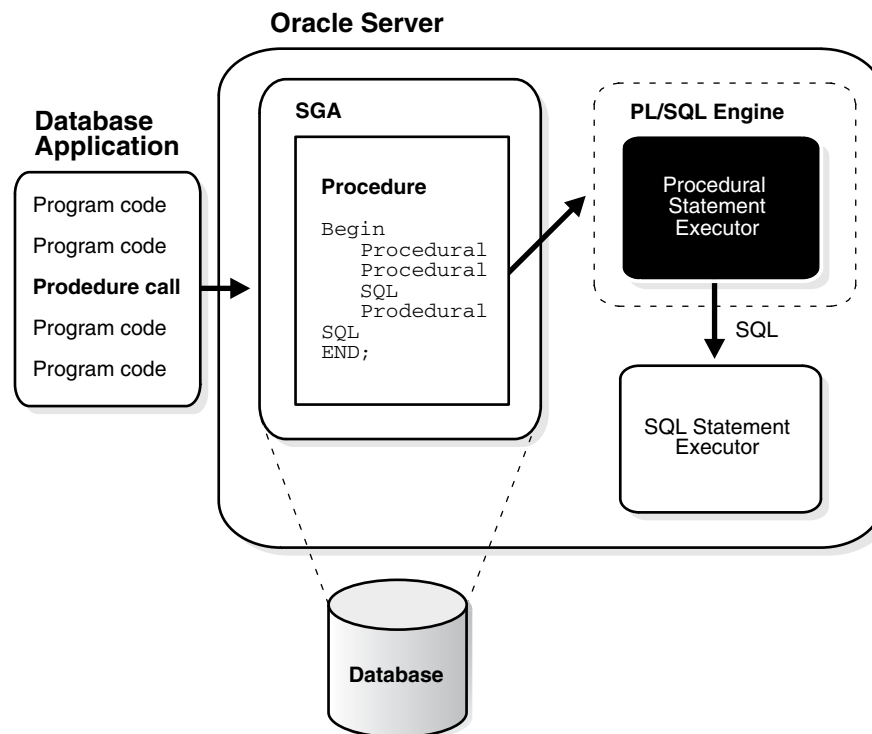
The program unit is stored in a database. When an application calls a procedure stored in the database, Oracle Database loads the compiled program unit into the shared pool in the system global area (SGA). The PL/SQL and SQL statement executors work together to process the statements within the procedure.

The following Oracle products contain a PL/SQL engine:

- Oracle Database
- Oracle Forms (version 3 and later)
- SQL*Menu (version 5 and later)
- Oracle Reports (version 2 and later)
- Oracle Graphics (version 2 and later)

You can call a stored procedure from another PL/SQL block, which can be either an anonymous block or another stored procedure. For example, you can call a stored procedure from Oracle Forms (version 3 or later).

Also, you can pass anonymous blocks to Oracle Database from applications developed with these tools:

- Oracle precompilers (including user exits)

- Oracle Call Interfaces (OCIs)

- SQL*Plus

- Oracle Enterprise Manager

Language Constructs for PL/SQL

PL/SQL blocks can include the following PL/SQL language constructs:

- Variables and constants

- Cursors

- Exceptions

> **See Also:** *Oracle Database PL/SQL Language Reference*

This section includes the following topics:

- Variables and Constants

- Cursors

- Exceptions

- Dynamic SQL in PL/SQL

Variables and Constants

Variables and constants can be declared within a procedure, function, or package. A variable or constant can be used in a SQL or PL/SQL statement to capture or provide a value when one is needed.

Some interactive tools, such as SQL*Plus, let you define variables in your current session. You can use such variables just as you would variables declared within procedures or packages.

Cursors

Cursors can be declared explicitly within a procedure, function, or package to facilitate record-oriented processing of Oracle Database data. Cursors also can be declared implicitly (to support other data manipulation actions) by the PL/SQL engine.

> **See Also:** "Scrollable Cursors" on page 24-5

Exceptions

PL/SQL lets you explicitly handle internal and user-defined error conditions, called **exceptions**, that arise during processing of PL/SQL code. Internal exceptions are caused by illegal operations, such as division by zero, or Oracle Database errors returned to the PL/SQL code. User-defined exceptions are explicitly defined and signaled within the PL/SQL block to control processing of errors specific to the application (for example, debiting an account and leaving a negative balance).

When an exception is raised, the execution of the PL/SQL code stops, and a routine called an exception handler is invoked. Specific exception handlers can be written for any internal or user-defined exception.

Dynamic SQL in PL/SQL

PL/SQL can run **dynamic SQL** statements whose complete text is not known until runtime. Dynamic SQL statements are stored in character strings that are entered into, or built by, the program at runtime. This enables you to create general purpose procedures. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.

You can write stored procedures and anonymous PL/SQL blocks that include dynamic SQL in two ways:

- By embedding dynamic SQL statements in the PL/SQL block
- By using the DBMS_SQL package

Additionally, you can issue DML or DDL statements using dynamic SQL. This helps solve the problem of not being able to statically embed DDL statements in PL/SQL. For example, you can choose to issue a DROP TABLE statement from within a stored procedure by using the EXECUTE IMMEDIATE statement or the PARSE procedure supplied with the DBMS_SQL package.

> **See Also:**
>
> - *Oracle Database Advanced Application Developer's Guide* for a comparison of the two approaches to dynamic SQL
> - *Oracle Database PL/SQL Language Reference* for details about dynamic SQL
> - *Oracle Database PL/SQL Packages and Types Reference*

PL/SQL Program Units

Oracle Database lets you access and manipulate database information using procedural schema objects called **PL/SQL program units**. Procedures, functions, and packages are all examples of PL/SQL program units.

Stored Procedures and Functions

A **procedure** or **function** is a schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or perform a set of related tasks. Procedures and functions permit the caller to provide parameters that can be input only, output only, or input and output values. Procedures and functions let you combine the ease and flexibility of SQL with the procedural functionality of a structured programming language.

Procedures and functions are identical except that functions always return a single value to the caller, while procedures do not. For simplicity, **procedure** as used in the remainder of this chapter means **procedure** *or* **function**.

You can run a procedure or function interactively by:

- Using an Oracle tool, such as SQL*Plus
- Calling it explicitly in the code of a database application, such as an Oracle Forms or precompiler application
- Calling it explicitly in the code of another procedure or trigger

See Also:

- *Pro*C/C++ Programmer's Guide* for information about how to call stored C or C++ procedures

- *Pro*COBOL Programmer's Guide* for information about how to call stored COBOL procedures

- Other programmer's guides for information about how to call stored procedures of specific kinds of application

Figure 25–2 illustrates a simple procedure that is stored in the database and called by several different database applications.

Figure 25–2   Stored Procedure
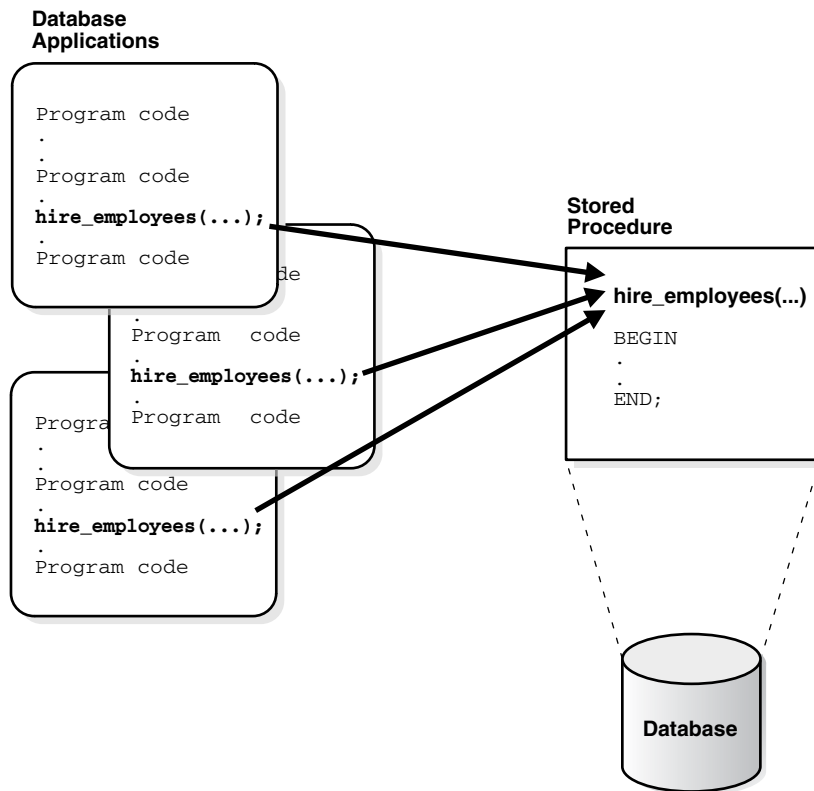
The following stored procedure example inserts an employee record into the `employees` table:

```
Procedure hire_employees (last_name VARCHAR2, job_id VARCHAR2, manager_id NUMBER,
hire_date DATE, salary NUMBER, commission_pct NUMBER, department_id NUMBER)

BEGIN
.
.
INSERT INTO employees VALUES (emp_sequence.NEXTVAL, last_name, job_id, manager_id,
hire_date, salary, commission_pct, department_id);
.
.
END
```

All of the database applications in this example call the `hire_employees` procedure. Alternatively, a privileged user can use Oracle Enterprise Manager or SQL*Plus to run the `hire_employees` procedure using a statement such as the following:

```
EXECUTE hire_employees ('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

This statement places a new employee record for `TSMITH` in the `employees` table.

> **See Also:** *Oracle Database PL/SQL Language Reference*

This section includes the following topics:

- Benefits of Procedures
- Procedure Guidelines
- Anonymous PL/SQL Blocks Compared with Stored Procedures
- Standalone Procedures
- Dependency Tracking for Stored Procedures
- External Procedures
- Table Functions

Benefits of Procedures

Stored procedures provide advantages in the following areas:

- Security with definer's rights procedures

  Stored procedures can help enforce data security. You can restrict the database operations that users can perform by allowing them to access data only through procedures and functions that run with the definer's privileges.

  For example, you can grant users access to a procedure that updates a table but not grant them access to the table itself. When a user invokes the procedure, the procedure runs with the privileges of the procedure's owner. Users who have only the privilege to run the procedure (but not the privileges to query, update, or delete from the underlying tables) can invoke the procedure, but they cannot manipulate table data in any other way.

  > **See Also:** "Dependency Tracking for Stored Procedures" on page 25-13

- Inherited privileges and schema context with invoker's rights procedures

  An invoker's rights procedure inherits privileges and schema context from the procedure that calls it. In other words, an invoker's rights procedure is not tied to a particular user or schema, and each invocation of an invoker's rights procedure operates in the current user's schema with the current user's privileges. Invoker's rights procedures make it easy for application developers to centralize application logic, even when the underlying data is divided among user schemas.

  For example, a user who runs an update procedure on the `employees` table as a manager can update salary, whereas a user who runs the same procedure as a clerk can be restricted to updating address data.

- Improved performance
  - The amount of information that must be sent over a network is small compared with issuing individual SQL statements or sending the text of an

entire PL/SQL block to Oracle Database, because the information is sent only once and thereafter invoked when it is used.

- A procedure's compiled form is readily available in the database, so no compilation is required at execution time.

- If the procedure is already present in the shared pool of the system global area (SGA), then retrieval from disk is not required, and execution can begin immediately.

■ Memory allocation

Because stored procedures take advantage the shared memory capabilities of Oracle Database, only a single copy of the procedure needs to be loaded into memory for execution by multiple users. Sharing the same code among many users results in a substantial reduction in Oracle Database memory requirements for applications.

■ Improved productivity

Stored procedures increase development productivity. By designing applications around a common set of procedures, you can avoid redundant coding and increase your productivity.

For example, procedures can be written to insert, update, or delete employee records from the `employees` table. These procedures can then be called by any application without rewriting the SQL statements necessary to accomplish these tasks. If the methods of data management change, only the procedures need to be modified, not all of the applications that use the procedures.

■ Integrity

Stored procedures improve the integrity and consistency of your applications. By developing all of your applications around a common group of procedures, you can reduce the likelihood of committing coding errors.

For example, you can test a procedure or function to guarantee that it returns an accurate result and, once it is verified, reuse it in any number of applications without testing it again. If the data structures referenced by the procedure are altered in any way, then only the procedure needs to be recompiled. Applications that call the procedure do not necessarily require any modifications.

Procedure Guidelines

Use the following guidelines when designing stored procedures:

■ Define procedures to complete a single, focused task. Do not define long procedures with several distinct subtasks, because subtasks common to many procedures can be duplicated unnecessarily in the code of several procedures.

■ Do not define procedures that duplicate the functionality already provided by other features of Oracle Database. For example, do not define procedures to enforce simple data integrity rules that you could easily enforce using declarative integrity constraints.

Anonymous PL/SQL Blocks Compared with Stored Procedures

A stored procedure is created and stored in the database as a schema object. Once created and compiled, it is a named object that can be run without recompiling. Additionally, dependency information is stored in the data dictionary to guarantee the validity of each stored procedure.

As an alternative to a stored procedure, you can create an anonymous PL/SQL block by sending an unnamed PL/SQL block to Oracle Database from an Oracle tool or an application. Oracle Database compiles the PL/SQL block and places the compiled version in the shared pool of the SGA, but it does not store the source code or compiled version in the database for reuse beyond the current instance. Shared SQL allows anonymous PL/SQL blocks in the shared pool to be reused and shared until they are flushed out of the shared pool.

In either case, by moving PL/SQL blocks out of a database application and into database procedures stored either in the database or in memory, you avoid unnecessary procedure recompilations by Oracle Database at runtime, improving the overall performance of the application and Oracle Database.

Standalone Procedures

Stored procedures not defined within the context of a package are called **standalone procedures**. Procedures defined within a package are considered a part of the package.

> **See Also:** "PL/SQL Packages" on page 25-14 for information about the advantages of packages

Dependency Tracking for Stored Procedures

A stored procedure depends on the objects referenced in its body. Oracle Database automatically tracks and manages such dependencies. For example, if you alter the definition of a table referenced by a procedure, then the procedure must be recompiled to validate that it will still work as designed. Usually, Oracle Database automatically administers such dependency management.

> **See Also:** Chapter 6, "Schema Object Dependencies" for more information about dependency tracking

External Procedures

A PL/SQL procedure executing on Oracle Database can call an external procedure or function that is written in the C programming language and stored in a shared library. The C routine runs in a separate address space from that of Oracle Database.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for more information about external procedures

Table Functions

Table functions are functions that can produce a set of rows as output. In other words, table functions return a collection type instance (nested table and VARRAY datatypes). You can use a table function in place of a regular table in the FROM clause of a SQL statement.

Oracle Database allows table functions to **pipeline** results (act like an Oracle Database rowsource) out of the functions. This can be achieved by either providing an implementation of the ODCITable interface, or using native PL/SQL instructions.

Pipelining helps to improve the performance of a number of applications, such as Oracle Warehouse Builder (OWB) and cartridges groups.

The ETL (Extraction-Transformation-Load) process in data warehouse building extracts data from an OLTP system. The extracted data passes through a sequence of transformations (written in procedural languages, such as PL/SQL) before it is loaded into a data warehouse.

Oracle Database also allows parallel execution of table and non-table functions. Parallel execution provides the following extensions:

- Functions can directly accept a set of rows corresponding to a subquery operand.

- A set of input rows can be partitioned among multiple instances of a parallel function. The function developer specifies how the input rows should be partitioned between parallel instances of the function.

Thus, table functions are similar to views. However, instead of defining the transform declaratively in SQL, you define it procedurally in PL/SQL. This is especially valuable for the arbitrarily complex transformations typically required in ETL.

> **See Also:**
>
> - "Overview of Extraction, Transformation, and Loading (ETL)" on page 16-5
>
> - *Oracle Database Data Cartridge Developer's Guide*
>
> - *Oracle Database PL/SQL Language Reference*

PL/SQL Packages

A **package** is a group of related procedures and functions, along with the cursors and variables they use, stored together in the database for continued use as a unit. Similar to standalone procedures and functions, packaged procedures and functions can be called explicitly by applications or users.

Oracle supplies many PL/SQL packages with Oracle Database to extend database functionality and provide PL/SQL access to SQL features. For example, the ULT_HTTP supplied package enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges. You can use the supplied packages when creating your applications or for ideas on creating your own stored procedures.

You create a package in two parts: the specification and the body. The package **specification** declares all public constructs of the package, and the **body** defines all constructs (public and private) of the package. The package body must be created in the same schema as the package. This separation of the two parts provides the following advantages:

- You have more flexibility in the development cycle. You can create specifications and reference public procedures without actually creating the package body.

- You can alter procedure bodies contained within the package body separately from their publicly declared specifications in the package specification. As long as the procedure specification does not change, objects that reference the altered procedures of the package are never marked invalid. That is, they are never marked as needing recompilation.

> **Note:** The package body and package specification always must be in the same schema.

Figure 25–3 illustrates a package that encapsulates a number of procedures used to manage an employee database.

Figure 25–3   A Stored Package
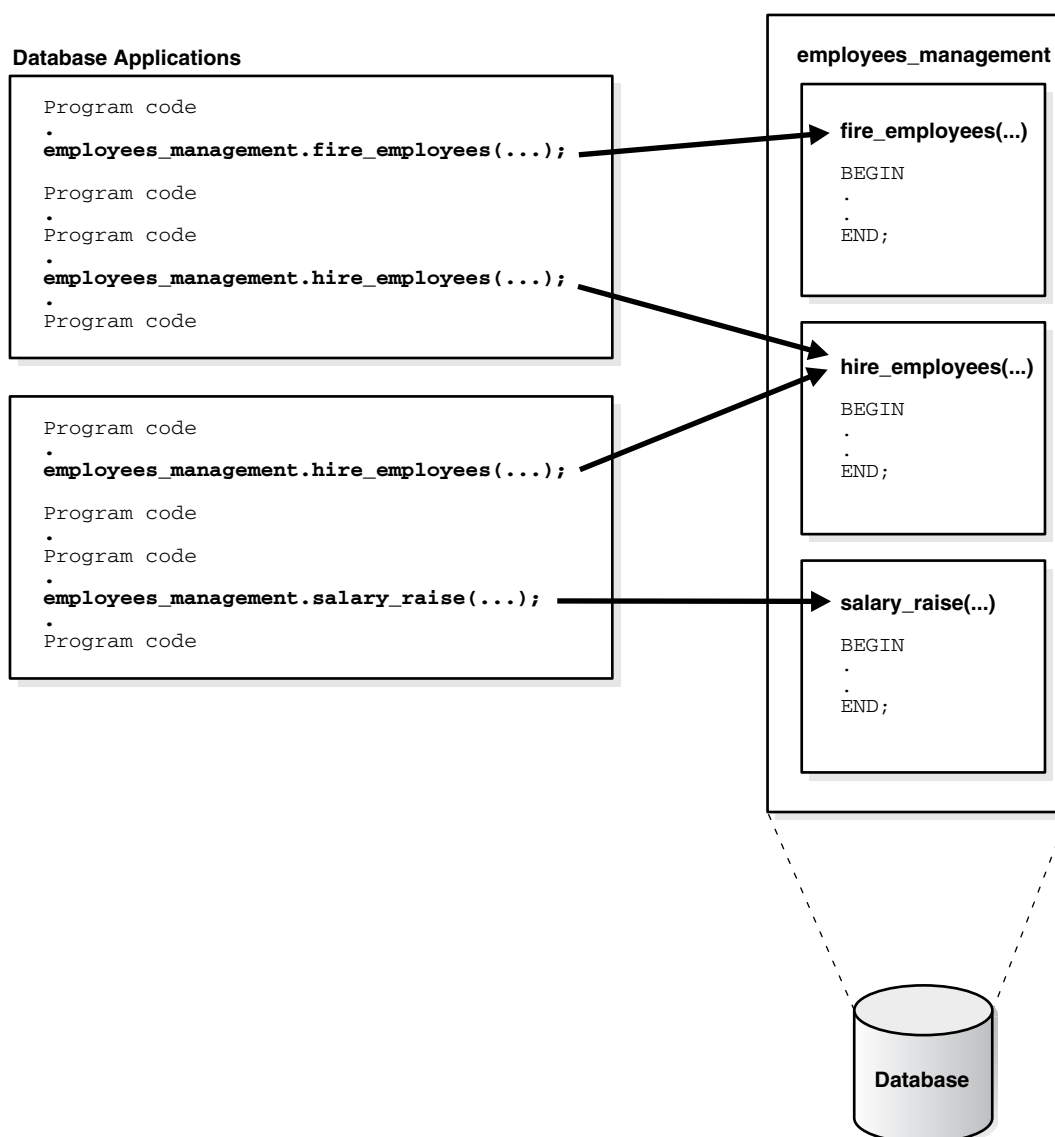
Database applications explicitly call packaged procedures as necessary. After being granted the privileges for the employees_management package, a user can explicitly run any of the procedures contained in it. For example, Oracle Enterprise Manager or SQL*Plus can issue the following statement to run the hire_employees package procedure:

```
EXECUTE employees_management.hire_employees ('TSMITH', 'CLERK', 1037, SYSDATE,
500, NULL, 20);
```

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference*
> - *Oracle Database PL/SQL Packages and Types Reference*

Benefits of Packages

Packages provide advantages in the following areas:

■ Encapsulation of related procedures and variables

Stored packages allow you to **encapsulate** or group stored procedures, variables, datatypes, and so on in a single named, stored unit in the database. This provides better organization during the development process. Encapsulation of procedural constructs also makes privilege management easier. Granting the privilege to use a package makes all constructs of the package accessible to the grantee.

■ Declaration of public and private procedures, variables, constants, and cursors

The methods of package definition allow you to specify which variables, cursors, and procedures are public and private. Public means that it is directly accessible to the user of a package. Private means that it is hidden from the user of a package.

For example, a package can contain 10 procedures. You can define the package so that only three procedures are public and therefore available for execution by a user of the package. The remainder of the procedures are private and can only be accessed by the procedures within the package. Do not confuse public and private package variables with grants to PUBLIC.

> **See Also:** Chapter 20, "Database Security" for more information about grants to PUBLIC

■ Better performance

An entire package is loaded into memory when a procedure within the package is called for the first time. This load is completed in one operation, as opposed to the separate loads required for standalone procedures. Therefore, when calls to related packaged procedures occur, no disk I/O is necessary to run the compiled code already in memory.

A package body can be replaced and recompiled without affecting the specification. As a result, schema objects that reference a package's constructs (always through the specification) need not be recompiled unless the package specification is also replaced. By using packages, unnecessary recompilations can be minimized, resulting in less impact on overall database performance.

PL/SQL Collections and Records

Many programming techniques use collection types such as arrays, bags, lists, nested tables, sets, and trees. To support these techniques in database applications, PL/SQL provides the datatypes TABLE and VARRAY, which allow you to declare index-by tables, nested tables, and variable-size arrays.

This section includes the following topics:

■ Collections

■ Records

Collections

A collection is an ordered group of elements, all of the same type. Each element has a unique subscript that determines its position in the collection.

Collections work like the arrays found in most third-generation programming languages. Also, collections can be passed as parameters. So, you can use them to move columns of data into and out of database tables or between client-side applications and stored subprograms.

Records

You can use the `%ROWTYPE` attribute to declare a record that represents a row in a table or a row fetched from a cursor. But, with a user-defined record, you can declare fields of your own.

Records contain uniquely named fields, which can have different datatypes. Suppose you have various data about an employee such as name, salary, and hire date. These items are dissimilar in type but logically related. A record containing a field for each item lets you treat the data as a logical unit.

> **See Also:** *Oracle Database PL/SQL Language Reference* for detailed information on using collections and records

PL/SQL Server Pages

PL/SQL Server Pages (PSP) are server-side Web pages (in HTML or XML) with embedded PL/SQL scripts marked with special tags. To produce dynamic Web pages, developers have usually written CGI programs in C or Perl that fetch data and produce the entire Web page within the same program. The development and maintenance of such dynamic pages is costly and time-consuming.

Scripting fulfills the demand for rapid development of dynamic Web pages. Small scripts can be embedded in HTML pages without changing their basic HTML identity. The scripts contain the logic to produce the dynamic portions of HTML pages and are run when the pages are requested by the users.

The separation of HTML content from application logic makes script pages easier to develop, debug, and maintain. The simpler development model, along the fact that scripting languages usually demand less programming skill, enables Web page writers to develop dynamic Web pages.

There are two kinds of embedded scripts in HTML pages: client-side scripts and server-side scripts. **Client-side scripts** are returned as part of the HTML page and are run in the browser. They are mainly used for client-side navigation of HTML pages or data validation. **Server-side scripts**, while also embedded in the HTML pages, are run on the server side. They fetch and manipulate data and produce HTML content that is returned as part of the page. PSP scripts are server-side scripts.

A PL/SQL gateway receives HTTP requests from an HTTP client, invokes a PL/SQL stored procedure as specified in the URL, and returns the HTTP output to the client. A PL/SQL Server Page is processed by a PSP compiler, which compiles the page into a PL/SQL stored procedure. When the procedure is run by the gateway, it generates the Web page with dynamic content. PSP is built on one of two existing PL/SQL gateways:

- PL/SQL cartridge of Oracle Application Server
- WebDB

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for more information about PL/SQL Server Pages

Overview of Java

Java is an object-oriented programming language efficient for application-level programs. It includes the following features:

- A Java Virtual Machine (JVM), which provides the fundamental basis for platform independence

- Automatic storage management techniques, such as gathering scattered memory into contiguous memory space

- Language syntax that borrows from C and enforces strong typing

This section contains the following topics:

- Java and Object-Oriented Programming Terminology

- Class Hierarchy

- Interfaces

- Polymorphism

- Overview of the Java Virtual Machine (JVM)

- Why Use Java in Oracle Database?

- Oracle's Java Application Strategy

Java and Object-Oriented Programming Terminology

This section covers some basic terminology of Java application development in the Oracle Database environment.

This section includes the following topics:

- Classes

- Attributes

- Methods

Classes

All object-oriented programming languages support the concept of a class. As with a table definition, a class provides a template for objects that share common characteristics. Each class can contain the following:

- Attributes—static or instance variables that each object of a particular class possesses

- Methods—you can invoke methods defined by the class or inherited by any classes extended from the class

When you create an object from a class, you are creating an instance of that class. The instance contains the fields of an object, which are known as its data, or state.

Figure 25–4 shows an example of an `Employee` class defined with two attributes: last name (`lastName`) and employee identifier (`ID`).

Figure 25–4   Classes and Instances
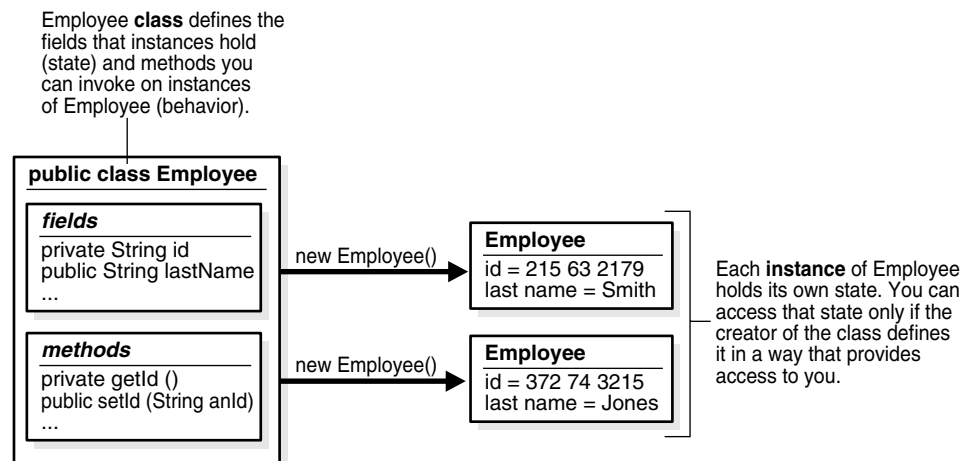
When you create an instance, the attributes store individual and private information relevant only to the employee. That is, the information contained within an employee instance is known only for that single employee. The example in Figure 25–4 shows two instances of employee—Smith and Jones. Each instance contains information relevant to the individual employee.

Attributes

Attributes within an instance are known as fields. Instance fields are analogous to the fields of a relational table row. The class defines the fields, as well as the type of each field. You can declare fields in Java to be static, public, private, protected, or default access.

- Public, private, protected, or default access fields are created within each instance.

- Static fields are like global variables in that the information is available to all instances of the employee class.

The language specification defines the rules of visibility of data for all fields. Rules of visibility define under what circumstances you can access the data in these fields.

Methods

The class also defines the methods you can invoke on an instance of that class. Methods are written in Java and define the behavior of an object. This bundling of state and behavior is the essence of encapsulation, which is a feature of all object-oriented programming languages. If you define an Employee class, declaring that each employee's id is a private field, other objects can access that private field only if a method returns the field. In this example, an object could retrieve the employee's identifier by invoking the Employee.getId method.

In addition, with encapsulation, you can declare that the Employee.getId method is private, or you can decide not to write an Employee.getId method. Encapsulation helps you write programs that are reusable and not misused. Encapsulation makes public only those features of an object that are declared public; all other fields and methods are private. Private fields and methods can be used for internal object processing.

Class Hierarchy

Java defines classes within a large hierarchy of classes. At the top of the hierarchy is the Object class. All classes in Java inherit from the `Object` class at some level, as you walk up through the inheritance chain of superclasses. When it is said that Class B inherits from Class A, each instance of Class B contains all the fields defined in class B, as well as all the fields defined in Class A. For example, in Figure 25–5, the `FullTimeEmployee` class contains the `id` and `lastName` fields defined in the `Employee` class, because it inherits from the Employee class. In addition, the `FullTimeEmployee` class adds another field, `bonus`, which is contained only within `FullTimeEmployee`.

You can invoke any method on an instance of Class B that was defined in either Class A or B. In our employee example, the `FullTimeEmployee` instance can invoke methods defined only within its own class, or methods defined within the `Employee` class.

Figure 25–5   Using Inheritance to Localize Behavior and State
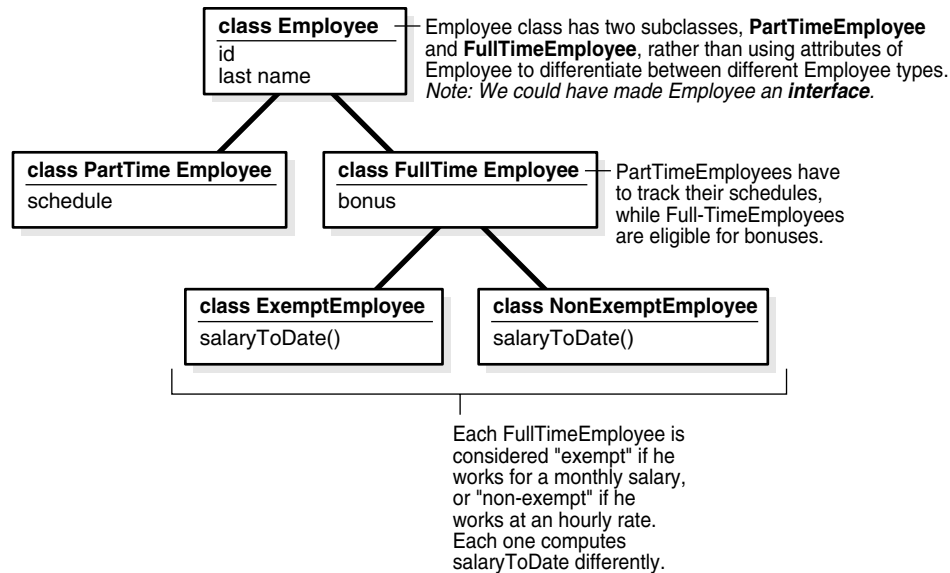
Instances of Class B are substitutable for instances of Class A, which makes inheritance another powerful construct of object-oriented languages for improving code reuse. You can create new classes that define behavior and state where it makes sense in the hierarchy, yet make use of pre-existing functionality in class libraries.

Interfaces

Java supports only single inheritance; that is, each class has one and only one class from which it inherits. If you must inherit from more than one source, Java provides the equivalent of multiple inheritance, without the complications and confusion that usually accompany it, through interfaces. Interfaces are similar to classes; however, interfaces define method signatures, not implementations. The methods are implemented in classes declared to implement an interface. Multiple inheritance occurs when a single class simultaneously supports many interfaces.

Polymorphism

Assume in our `Employee` example that the different types of employees must be able to respond with their compensation to date. Compensation is computed differently for different kinds of employees.

- `FullTimeEmployees` are eligible for a bonus

- `NonExemptEmployees` get overtime pay

In traditional procedural languages, you would write a long switch statement, with the different possible cases defined.

```
switch (employee.type) {
case: Employee
return employee.salaryToDate;
case: FullTimeEmployee
return employee.salaryToDate + employee.bonusToDate;
...
```

If you add a new kind of employee, then you must update your switch statement. If you modify your data structure, then you must modify all switch statements that use it.

In an object-oriented language such as Java, you implement a method, `compensationToDate`, for each subclass of `Employee` class that requires any special treatment beyond what is already defined in `Employee` class. For example, you could implement the `compensationToDate` method of `NonExemptEmployee`, as follows:

```
private float compensationToDate() {
return super.compensationToDate() + this.overtimeToDate();
}
```

Implement `FullTimeEmployee`'s method as follows:

```
private float compensationToDate() {
return super.compensationToDate() + this.bonusToDate();
}
```

The common usage of the method name `compensationToDate` lets you invoke the identical method on different classes and receive different results, without knowing the type of employee you are using. You do not have to write a special method to handle `FullTimeEmployees` and `PartTimeEmployees`. This ability for the different objects to respond to the identical message in different ways is known as polymorphism.

In addition, you could create an entirely new class that does not inherit from `Employee` at all—`Contractor`—and implement a `compensationToDate` method in it. A program that calculates total payroll to date would iterate over all people on payroll, regardless of whether they were full-time, part-time, or contractors, and add up the values returned from invoking the `compensationToDate` method on each. You can safely make changes to the individual `compensationToDate` methods with the knowledge that callers of the methods will work correctly. For example, you can safely add new fields to existing classes.

Overview of the Java Virtual Machine (JVM)

As with other high-level computer languages, Java source compiles to low-level instructions. In Java, these instructions are known as bytecodes (because their size is uniformly one byte of storage). Most other languages—such as C—compile to computer-specific instructions, such as instructions specific to an Intel or HP processor.

Java source compiles to a standard, platform-independent set of bytecodes, which interacts with a Java Virtual Machine (JVM). The JVM is a separate program that is optimized for the specific platform on which you run your Java code.

Figure 25–6 illustrates how Java can maintain platform independence. Java source is compiled into bytecodes, which are platform independent. Each platform has installed a JVM that is specific to its operating system. The Java bytecodes from your source get interpreted through the JVM into appropriate platform dependent actions.

Figure 25–6   Java Component Structure
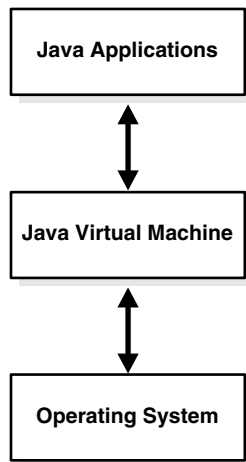
When you develop a Java program, you use predefined core class libraries written in the Java language. The Java core class libraries are logically divided into packages that provide commonly-used functionality, such as basic language support (`java.lang`), I/O (`java.io`), and network access (`java.net`). Together, the JVM and core class libraries provide a platform on which Java programmers can develop with the confidence that any hardware and operating system that supports Java will execute their program. This concept is what drives the "write once, run anywhere" idea of Java.

Figure 25–7 illustrates how Oracle's Java applications sit on top of the Java core class libraries, which in turn sit on top of the JVM. Because Oracle's Java support system is located within the database, the JVM interacts with Oracle Database libraries, instead of directly with the operating system.

Figure 25–7   Java Component Structure
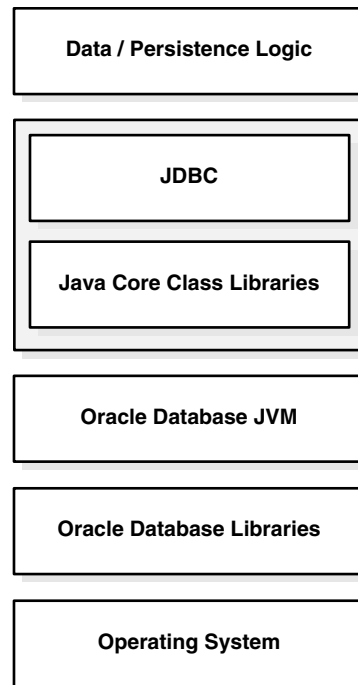
Sun Microsystems furnishes publicly available specifications for both the Java language and the JVM. The Java Language Specification (JLS) defines things such as syntax and semantics; the JVM specification defines the necessary low-level behavior for the computer that runs the bytecodes. In addition, Sun Microsystems provides a compatibility test suite for JVM implementors to determine if they have complied with the specifications. This test suite is known as the Java Compatibility Kit (JCK). Oracle's JVM implementation complies fully with JCK. Part of the overall Java strategy is that an openly specified standard, together with a simple way to verify compliance with that standard, allows vendors to offer uniform support for Java across all platforms.

Why Use Java in Oracle Database?

You can write and load Java applications within the database, because it is a safe language. Java prevents anyone from tampering with the operating system that the Java code resides in. Some languages, such as C, can introduce security problems within the database; Java, because of its design, is a safe language to allow within the database.

Although Java presents many advantages to developers, providing an implementation of a JVM that supports Java server applications in a scalable manner is a challenge. This section discusses some of these challenges.

- Multithreading

- Automated Storage Management

- Footprint

- Performance

- Dynamic Class Loading

Multithreading

Multithreading support is often cited as one of the key scalability features of Java. Certainly, the Java language and class libraries make it simpler to write shared server applications in Java than many other languages, but it is still a daunting task in any language to write reliable, scalable shared server code.

As a database server, Oracle Database efficiently schedules work for thousands of users. The Oracle JVM uses the facilities of the RDBMS server to concurrently schedule Java execution for thousands of users. Although Oracle Database supports Java language level threads required by the JLS and JCK, using threads within the scope of the database does not increase scalability. Using the embedded scalability of the database eliminates the need for writing shared server Java servers. You should use the database's facilities for scheduling users by writing single-threaded Java applications. The database takes care of the scheduling between each application; thus, you achieve scalability without having to manage threads. You can still write shared server Java applications, but multiple Java threads does not increase your server's performance.

One difficulty multithreading imposes on Java is the interaction of threads and automated storage management, or garbage collection. The garbage collector executing in a generic JVM has no knowledge of which Java language threads are executing or how the underlying operating system schedules them.

- Non-Oracle model—A single user maps to a single Java language level thread; the same single garbage collector manages all garbage from all users. Different techniques typically deal with allocation and collection of objects of varying lifetimes and sizes. The result in a heavily shared server application is, at best, dependent upon operating system support for native threads, which can be unreliable and limited in scalability. High levels of scalability for such implementations have not been convincingly demonstrated.

- Oracle JVM model—Even when thousands of users connect to the server and run the same Java code, each user experiences it as if he is executing his own Java code on his own Java Virtual Machine. The responsibility of the Oracle JVM is to make use of operating system processes and threads, using the scalable approach of the Oracle RDBMS. As a result of this approach, the JVM's garbage collector is more reliable and efficient because it never collects garbage from more than one user at any time.

Automated Storage Management

Garbage collection is a major feature of Java's automated storage management, eliminating the need for Java developers to allocate and free memory explicitly. Consequently, this eliminates a large source of memory leaks that commonly plague C and C++ programs. There is a price for such a benefit: garbage collection contributes to the overhead of program execution speed and footprint. Although many papers have been written qualifying and quantifying the trade-off, the overall cost is reasonable, considering the alternatives.

Garbage collection imposes a challenge to the JVM developer seeking to supply a highly scalable and fast Java platform. The Oracle JVM meets these challenges in the following ways:

- The Oracle JVM uses the Oracle Database scheduling facilities, which can manage multiple users efficiently.

- Garbage collection is performs consistently for multiple users because garbage collection is focused on a single user within a single session. The Oracle JVM enjoys a huge advantage because the burden and complexity of the memory

manager's job does not increase as the number of users increases. The memory manager performs the allocation and collection of objects within a single session—which typically translates to the activity of a single user.

- The Oracle JVM uses different garbage collection techniques depending on the type of memory used. These techniques provide high efficiency and low overhead.

Footprint

The footprint of an executing Java program is affected by many factors:

- Size of the program itself—how many classes and methods and how much code they contain.

- Complexity of the program—the amount of core class libraries that the Oracle JVM uses as the program runs, as opposed to the program itself.

- Amount of state the JVM uses—how many objects the JVM allocates, how large they are, and how many must be retained across calls.

- Ability of the garbage collector and memory manager to deal with the demands of the executing program, which is often non-deterministic. The speed with which objects are allocated and the way they are held on to by other objects influences the importance of this factor.

From a scalability perspective, the key to supporting many concurrent clients is a minimum user session footprint. The Oracle JVM keeps the user session footprint to a minimum by placing all read-only data for users, such as Java bytecodes, in shared memory. Appropriate garbage collection algorithms are applied against call and session memories to maintain a small footprint for the user's session. The Oracle JVM uses three types of garbage collection algorithms to maintain the user's session memory:

- Generational scavenging for short-lived objects

- Mark and lazy sweep collection for objects that exist for the life of a single call

- Copying collector for long-lived objects—objects that live across calls within a session

Performance

Oracle JVM performance is enhanced by implementing a native compiler. Java runs platform-independent bytecodes on top of a JVM, which in turn interacts with the specific hardware platform. Any time you add levels within software, your performance is degraded. Because Java requires going through an intermediary to interpret platform-independent bytecodes, a degree of inefficiency exists for Java applications that does not exists within a platform-dependent language, such as C. To address this issue, several JVM suppliers create native compilers. Native compilers translate Java bytecodes into platform-dependent native code, which eliminates the interpreter step and improves performance.

Table 25–1 describes two methods for native compilation.

Table 25–1   Native Compilation Methods

Method	Description
Just-In-Time (JIT) Compilation	JIT compilers quickly compile Java bytecodes to native (platform-specific) computer code during runtime. This does not produce an executable to be run on the platform; instead, it provides platform-dependent code from Java bytecodes that is run directly after it is translated. This should be used for Java code that is run frequently, which will be run at speeds closer to languages such as C.
Static Compilation	Static compilation translates Java bytecodes to platform-independent C code before runtime. Then a standard C compiler compiles the C code into an executable for the target platform. This approach is more suitable for Java applications that are modified infrequently. This approach takes advantage of the mature and efficient platform-specific compilation technology found in modern C compilers.

Oracle Database uses static compilation to deliver its core Java class libraries: the ORB and JDBC code in natively compiled form. It is applicable across all the platforms Oracle supports, whereas a JIT approach requires low-level, processor-dependent code to be written and maintained for each platform. You can use this native compilation technology with your own Java code.

Dynamic Class Loading

Another strong feature of Java is dynamic class loading. The class loader loads classes from the disk (and places them in the JVM-specific memory structures necessary for interpretation) only as they are used during program execution. The class loader locates the classes in the CLASSPATH and loads them during program execution. This approach, which works well for applets, poses the problems listed in Table 25–2 in a server environment.

Table 25–2   Problems Associated with Dynamic Class Loading

Problem	Description	Solution
Predictability	The class loading operation places a severe penalty on first-time execution. A simple program can cause the Oracle JVM to load many core classes to support its needs. A programmer cannot easily predict or determine the number of classes loaded.	The Oracle JVM loads classes dynamically, just as with any other Java Virtual Machine. The same one-time class loading speed hit is encountered. However, because the classes are loaded into shared memory, no other users of those classes will cause the classes to load again—they will simply use the same pre-loaded classes.

Table 25–2   (Cont.) Problems Associated with Dynamic Class Loading

Problem	Description	Solution
Reliability	A benefit of dynamic class loading is that it supports program updating. For example, you would update classes on a server, and clients who download the program and load it dynamically see the update whenever they next use the program. Server programs tend to emphasize reliability. As a developer, you must know that every client runs a specific program configuration. You do not want clients to inadvertently load some classes that you did not intend them to load.	Oracle Database separates the upload and resolve operation from the class loading operation at runtime. You upload Java code you developed to the server using the `loadjava` utility. Instead of using `CLASSPATH`, you specify a resolver at installation time. The resolver is analogous to `CLASSPATH`, but lets you specify the schemas in which the classes reside. This separation of resolution from class loading means you always know what program users run.

Oracle's Java Application Strategy

One appeal of Java is its ubiquity and the growing number of programmers capable of developing applications using it. Oracle furnishes enterprise application developers with an end-to-end Java solution for creating, deploying, and managing Java applications. The total solution consists of client-side and server-side programmatic interfaces, tools to support Java development, and a Java Virtual Machine integrated with Oracle Database. All these products are compatible with Java standards.

In addition to the Oracle JVM, the Java programming environment consists of the following:

- Java stored procedures as the Java equivalent and companion for PL/SQL. Java stored procedures are tightly integrated with PL/SQL. You can call a Java stored procedure from a PL/SQL package; you can call PL/SQL procedures from a Java stored procedure.

- SQL data can be accessed through the JDBC programming interface.

- Tools and scripts used in assisting in development, class loading, and class management.

This section includes the following topics:

- Java Stored Procedures

- PL/SQL Integration and Oracle Database Functionality

- JDBC

- JPublisher

- Java Messaging Service

Java Stored Procedures

A Java stored procedure is a program you write in Java to run in the server, exactly as a PL/SQL stored procedure. You invoke it directly with products like SQL*Plus, or indirectly with a trigger. You can access it from any Oracle Net client—OCI, precompiler, or JDBC.

In addition, you can use Java to develop powerful programs independently of PL/SQL. Oracle Database provides a fully-compliant implementation of the Java programming language and JVM.

> **See Also:** *Oracle Database Java Developer's Guide* explains how to write stored procedures in Java, how to access them from PL/SQL, and how to access PL/SQL functionality from Java.

PL/SQL Integration and Oracle Database Functionality

You can invoke existing PL/SQL programs from Java and invoke Java programs from PL/SQL. This solution protects and leverages your existing investment while opening up the advantages and opportunities of Java-based Internet computing.

JDBC

Java database connectivity (JDBC) is an application programming interface (API) for Java developers to access SQL data. It is available on client and server, so you can deploy the same code in either place.

Oracle's JDBC allows access to objects and collection types defined in the database from Java programs through dynamic SQL. Dynamic SQL means that the embedded SQL statement to be run is not known before the application is run, and requires input to build the statement. It provides for translation of types defined in the database into Java classes through default or customizable mappings, and it also enables you to monitor, trace, and correlate resource consumption of Java and J2EE applications down to the database operation level.

Core Java class libraries provide only one JDBC API. JDBC is designed, however, to allow vendors to supply drivers that offer the necessary specialization for a particular database. Oracle delivers the following three distinct JDBC drivers.

Table 25–3 JDBC Drivers

Driver	Description
JDBC Thin Driver	You can use the JDBC Thin driver to write 100% pure Java applications and applets that access Oracle SQL data. The JDBC Thin driver is especially well-suited to Web browser-based applications and applets, because you can dynamically download it from a Web page just like any other Java applet.
JDBC Oracle Call Interface Driver	The JDBC Oracle Call Interface (OCI) driver accesses Oracle-specific native code (that is, non-Java) libraries on the client or middle tier, providing a richer set of functionality and some performance boost compared to the JDBC Thin driver, at the cost of significantly larger size and client-side installation.
JDBC Server-side Internal Driver	Oracle Database uses the server-side internal driver when Java code runs on the server. It allows Java applications running in the server's JVM to access locally defined data (that is, on the same computer and in the same process) with JDBC. It provides a further performance boost because of its ability to use underlying Oracle RDBMS libraries directly, without the overhead of an intervening network connection between your Java code and SQL data. By supporting the same Java-SQL interface on the server, Oracle Database does not require you to rework code when deploying it.

> **See Also:**
>
> - *Oracle Database JDBC Developer's Guide and Reference*
> - *Oracle Database Advanced Application Developer's Guide* for examples of JDBC programs

SQLJ

SQLJ allows developers to use object datatypes in Java programs. Developers can use JPublisher to map Oracle object and collection types into Java classes to be used in the application.

SQLJ provides access to server objects using SQL statements embedded in the Java code. SQLJ provides compile-time type checking of object types and collections in the SQL statements. The syntax is based on an ANSI standard (SQLJ Consortium).

You can specify Java classes as SQL user-defined object types. You can define columns or rows of this SQLJ type. You can also query and manipulate the objects of this type as if they were SQL primitive types. Additionally, you can do the following:

- Make the static fields of a class visible in SQL

- Allow the user to call a Java constructor

- Maintain the dependency between the Java class and its corresponding type

JPublisher

Java Publisher (JPublisher) is a utility, written entirely in Java, that generates Java classes to represent the following user-defined database entities in your Java program:

- SQL object types

- Object reference types ("REF types")

- SQL collection types (VARRAY types or nested table types)

- PL/SQL packages

JPublisher lets you to specify and customize the mapping of these entities to Java classes in a strongly typed paradigm.

> **See Also:** *Oracle Database JPublisher User's Guide*

Java Messaging Service

Java Messaging Service (JMS) is a messaging standard developed by Sun Microsystems along with Oracle, IBM, and other vendors. It defines a set of interfaces for JMS applications and specifies the behavior implemented by JMS providers. JMS provides a standard-based API to enable asynchronous exchange of business events within the enterprise, as well as with customers and partners. JMS facilitates reliable communication between loosely coupled components in a distributed environment, significantly simplifying the effort required for enterprise integration. The combination of Java technology with enterprise messaging enables development of portable applications.

Oracle Java Messaging Service is a Java API for Oracle Streams, based on the JMS standard. Multiple client applications can send and receive messages of any type through a central JMS provider (Oracle Streams). The JMS client consists of the Java application as well as a messaging client runtime library that implements the JMS interface and communicates with Oracle Streams.

Java Messaging Oracle JMS supports the standard JMS interfaces and has extensions to support other Streams features that are not a part of the standard. It can be used to enqueue and dequeue messages in the queue available with Oracle Streams. Oracle JMS includes the standard JMS features:

- Point-to-point communication using queues

- Publish-subscribe communication using topics

- Synchronous and asynchronous message exchange

- Subject-based routing

Oracle Streams also provides extensions to the standard JMS features:

- Point-to-multipoint communication using a recipient list for specifying the applications to receive the messages

- Administrative API to create the queue tables, queues and subjects

- Automatic propagation of messages between queues on different databases, enabling the application to define remote subscribers

- Transacted session support, allowing both JMS and SQL operations in one transaction

- Message retention after message is consumed

- Exception handling

- Delay specification before a message is visible

Overview of Microsoft Programming Languages

Oracle offers a variety of data access methods from COM-based programming languages, such as Visual Basic and Active Server Pages. These include Oracle Objects for OLE (OO40) and the Oracle Provider for OLE DB. The latter can be used with Microsoft's ActiveX Data Objects (ADO). Server-side programming to COM Automation servers, such as Microsoft Office, is available through the COM Automation Feature. More traditional ODBC access is available through Oracle's ODBC Driver. C/C++ applications can also use the Oracle Call Interface (OCI). These data access drivers have been engineered to provide superior performance with Oracle Database and expose the database's advanced features which may not be available in third-party drivers.

Oracle also provides optimum .NET data access support through the Oracle Data Provider for .NET, allowing .NET to access advanced Oracle features. Oracle also supports OLE DB .NET and ODBC .NET.

This section includes the following topics:

- Open Database Connectivity

- Overview of Oracle Objects for OLE

- Oracle Data Provider for .NET

Open Database Connectivity

Open database connectivity (ODBC), is a database access protocol that lets you connect to a database and then prepare and run SQL statements against the database. In conjunction with an ODBC driver, an application can access any data source including data stored in spreadsheets, like Excel. Because ODBC is a widely accepted standard API, applications can be written to comply to the ODBC standard. The ODBC driver performs all mappings between the ODBC standard and the particular database the application is accessing. Using a data source-specific driver, an ODBC compliant program can access any data source without any more development effort.

Oracle provides the ODBC interface so that applications of any type that are ODBC compliant can access Oracle Database using the ODBC driver provided by Oracle. For

example, an application written in Visual Basic can use ODBC to access Oracle Database.

Overview of Oracle Objects for OLE

Oracle Objects for OLE (OO4O) allows easy access to data stored in Oracle Databases with any programming or scripting language that supports the Microsoft COM Automation and ActiveX technology. This includes Visual Basic, Visual C++, Visual Basic For Applications (VBA), IIS Active Server Pages (VBScript and JavaScript), and others.

OO4O consists of the following software layers:

- OO4O Automation Server
- Oracle Data Control
- The Oracle Objects for OLE C++ Class Library

OO4O Automation Server

The OO4O Automation Server is a set of COM Automation objects for connecting to Oracle Database servers, executing SQL statements and PL/SQL blocks, and accessing the results.

OO4O provides key features for accessing Oracle Databases in environments ranging from the typical two-tier client/server applications, such as those developed in Visual Basic or Excel, to application servers deployed in multitiered application server environments, such as Web server applications in Microsoft Internet Information Server (IIS) or Microsoft Transaction Server.

Oracle Data Control

The Oracle Data Control (ODC) is an ActiveX Control designed to simplify the exchange of data between Oracle Database and visual controls, such edit, text, list, and grid controls in Visual Basic and other development tools that support custom controls.

ODC acts an agent to handle the flow of information from Oracle Database and a visual data-aware control, such as a grid control, that is bound to it. The data control manages various user interface (UI) tasks such as displaying and editing data. It also runs and manages the results of database queries.

> **See Also:** *Oracle Objects for OLE Developer's Guide*

The Oracle Objects for OLE C++ Class Library

The Oracle Objects for OLE C++ Class Library is a collection of C++ classes that provide programmatic access to the Oracle Object Server. Although the class library is implemented using OLE Automation, neither the OLE development kit nor any OLE development knowledge is necessary to use it. This library helps C++ developers avoid writing COM client code for accessing the OO4O interfaces.

> **See Also:** Oracle Objects for OLE C++ Class Library help from the Start menu

Oracle Data Provider for .NET

Oracle Data Provider for .NET (ODP.NET) is an implementation of a data provider for Oracle Database. ODP.NET uses Oracle native APIs for fast and reliable access to

Oracle Database data and features from any .NET application. ODP.NET also uses and inherits classes and interfaces available in the Microsoft .NET Framework Class Library.

Following the .NET Framework, ODP.NET uses the ADO.NET model, which allows native providers to expose provider-specific features and data types.

Using ODP.NET, developers can write programs in Visual Basic .NET, C#, and other .NET languages.

See Also: *Oracle Data Provider for .NET Developer's Guide*

Overview of Legacy Languages

This section contains the following topics:

- Overview of Pro*COBOL Precompiler
- Overview of Pro*FORTRAN Precompiler

Overview of Pro*COBOL Precompiler

The Pro*COBOL Precompiler is a programming tool that lets you embed SQL statements in a host COBOL program. Pro*COBOL reads the source file as input and outputs a COBOL source file that replaces the embedded SQL statements with Oracle runtime library calls, and is then compiled by the COBOL compiler.

Like the Pro*C/C++ Precompiler, Pro*COBOL lets you create highly customized applications. For example, you can create user interfaces that incorporate the latest windowing and mouse technology. You can also create applications that run in the background without the need for user interaction.

Furthermore, with Pro*COBOL you can fine-tune your applications. It enables close monitoring of resource usage, SQL statement execution, and various run-time indicators. With this information, you can adjust program parameters for maximum performance.

See Also: *Pro*COBOL Programmer's Guide*

Overview of Pro*FORTRAN Precompiler

The Oracle Pro*FORTRAN Precompiler lets you embed SQL in a host FORTRAN program.

Pro*FORTRAN is not supported on Windows.

See Also: *Pro*FORTRAN Supplement to the Oracle Precompilers Guide*

26

Oracle Data Types

This chapter discusses the Oracle built-in datatypes, their properties, and how they map to non-Oracle datatypes.

This chapter includes the following topics:

- Introduction to Oracle Datatypes
- Overview of Character Datatypes
- Overview of Numeric Datatypes
- Overview of DATE Datatype
- Overview of LOB Datatypes
- Overview of RAW and LONG RAW Datatypes
- Overview of ROWID and UROWID Datatypes
- Overview of ANSI, DB2, and SQL/DS Datatypes
- Overview of XML Datatypes
- Overview of URI Datatypes
- Overview of Object Datatypes and Object Views
- Data Conversion

Introduction to Oracle Datatypes

Each column value and constant in a SQL statement has a **datatype**, which is associated with a specific storage format, constraints, and a valid range of values. When you create a table, you must specify a datatype for each of its columns.

Oracle provides the following categories of built-in datatypes:

- Overview of Character Datatypes
- Overview of Numeric Datatypes
- Overview of DATE Datatype
- Overview of LOB Datatypes
- Overview of RAW and LONG RAW Datatypes
- Overview of ROWID and UROWID Datatypes

> **Note:** PL/SQL has additional datatypes for constants and variables, which include `BOOLEAN`, reference types, composite types (collections and records), and user-defined subtypes.

See Also:

- *Oracle Database PL/SQL Language Reference* for more information about PL/SQL datatypes

- *Oracle Database Advanced Application Developer's Guide* for information about how to use the built-in datatypes

The following sections that describe each of the built-in datatypes in more detail.

Overview of Character Datatypes

The character datatypes store character (alphanumeric) data in strings, with byte values corresponding to the character encoding scheme, generally called a character set or code page.

The database's character set is established when you create the database. Examples of character sets are 7-bit ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code), Code Page 500, Japan Extended UNIX, and Unicode UTF-8. Oracle supports both single-byte and multibyte encoding schemes.

See Also:

- *Oracle Database Advanced Application Developer's Guide* for information about how to select a character datatype

- *Oracle Database Globalization Support Guide* for more information about converting character data

This section includes the following topics:

- CHAR Datatype
- VARCHAR2 and VARCHAR Datatypes
- Length Semantics for Character Datatypes
- NCHAR and NVARCHAR2 Datatypes
- Use of Unicode Data in Oracle Database
- LOB Character Datatypes
- LONG Datatype

CHAR Datatype

The `CHAR` datatype stores fixed-length character strings. When you create a table with a `CHAR` column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes for the `CHAR` column width. The default is 1 byte. Oracle then guarantees that:

- When you insert or update a row in the table, the value for the `CHAR` column has the fixed length.

- If you give a shorter value, then the value is blank-padded to the fixed length.

- If a value is too large, Oracle Database returns an error.

Oracle Database compares CHAR values using blank-padded comparison semantics.

> **See Also:** *Oracle Database SQL Language Reference* for details about blank-padded comparison semantics

VARCHAR2 and VARCHAR Datatypes

The VARCHAR2 datatype stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column. For each row, Oracle Database stores each value in the column as a variable-length field unless a value exceeds the column's maximum length, in which case Oracle Database returns an error. Using VARCHAR2 and VARCHAR saves on space used by the table.

For example, assume you declare a column VARCHAR2 with a maximum size of 50 characters. In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece stores only the 10 characters (10 bytes), not 50.

Oracle Database compares VARCHAR2 values using nonpadded comparison semantics.

> **See Also:** *Oracle Database SQL Language Reference* for details about nonpadded comparison semantics

VARCHAR Datatype

The VARCHAR datatype is synonymous with the VARCHAR2 datatype. To avoid possible changes in behavior, always use the VARCHAR2 datatype to store variable-length character strings.

Length Semantics for Character Datatypes

Globalization support allows the use of various character sets for the character datatypes. Globalization support lets you process single-byte and multibyte character data and convert between character sets. Client sessions can use client character sets that are different from the database character set.

Consider the size of characters when you specify the column length for character datatypes. You must consider this issue when estimating space for tables with columns that contain character data.

The length semantics of character datatypes can be measured in bytes or characters.

- **Byte semantics** treat strings as a sequence of bytes. This is the default for character datatypes.

- **Character semantics** treat strings as a sequence of characters. A character is technically a codepoint of the database character set.

For single byte character sets, columns defined in character semantics are basically the same as those defined in byte semantics. Character semantics are useful for defining varying-width multibyte strings; it reduces the complexity when defining the actual length requirements for data storage. For example, in a Unicode database (UTF8), you need to define a VARCHAR2 column that can store up to five Chinese characters together with five English characters. In byte semantics, this would require (5*3 bytes) + (1*5 bytes) = 20 bytes; in character semantics, the column would require 10 characters.

VARCHAR2(20 BYTE) and SUBSTRB(*<string>*,1,20) use byte semantics.
VARCHAR2(10 CHAR) and SUBSTR(*<string>*,1,10) use character semantics.

The parameter NLS_LENGTH_SEMANTICS decides whether a new column of character datatype uses byte or character semantics. The default length semantic is byte. If all character datatype columns in a database use byte semantics (or all use character semantics) then users do not have to worry about which columns use which semantics. The BYTE and CHAR qualifiers shown earlier should be avoided when possible, because they lead to mixed-semantics databases. Instead, the NLS_LENGTH_ SEMANTICS initialization parameter should be set appropriately in the server parameter file (SPFILE) or initialization parameter file, and columns should use the default semantics.

> **See Also:**
>
> - "Use of Unicode Data in Oracle Database" on page 26-5
>
> - *Oracle Database Globalization Support Guide* for information about Oracle's globalization support feature
>
> - *Oracle Database Advanced Application Developer's Guide* for information about setting length semantics and choosing the appropriate Unicode character set.
>
> - *Oracle Database Upgrade Guide* for information about migrating existing columns to character semantics

NCHAR and NVARCHAR2 Datatypes

NCHAR and NVARCHAR2 are Unicode datatypes that store Unicode character data. The character set of NCHAR and NVARCHAR2 datatypes can only be either AL16UTF16 or UTF8 and is specified at database creation time as the national character set. AL16UTF16 and UTF8 are both Unicode encoding.

- The NCHAR datatype stores fixed-length character strings that correspond to the national character set.

- The NVARCHAR2 datatype stores variable length character strings.

When you create a table with an NCHAR or NVARCHAR2 column, the maximum size specified is always in character length semantics. Character length semantics is the default and only length semantics for NCHAR or NVARCHAR2.

For example, if national character set is UTF8, then the following statement defines the maximum byte length of 90 bytes:

```
CREATE TABLE tab1 (col1 NCHAR(30));
```

This statement creates a column with maximum character length of 30. The maximum byte length is the multiple of the maximum character length and the maximum number of bytes in each character.

This section includes the following topics:

- NCHAR

- NVARCHAR2

NCHAR

The maximum length of an NCHAR column is 2000 bytes. It can hold up to 2000 characters. The actual data is subject to the maximum byte limit of 2000. The two size constraints must be satisfied simultaneously at run time.

NVARCHAR2

The maximum length of an `NVARCHAR2` column is 4000 bytes. It can hold up to 4000 characters. The actual data is subject to the maximum byte limit of 4000. The two size constraints must be satisfied simultaneously at run time.

> **See Also:** *Oracle Database Globalization Support Guide* for more information about the `NCHAR` and `NVARCHAR2` datatypes

Use of Unicode Data in Oracle Database

Unicode is an effort to have a unified encoding of every character in every language known to man. It also provides a way to represent privately-defined characters. A database column that stores Unicode can store text written in any language.

Oracle Database users deploying globalized applications have a strong need to store Unicode data in Oracle Databases. They need a datatype which is guaranteed to be Unicode regardless of the database character set.

Oracle Database supports a reliable Unicode datatype through `NCHAR`, `NVARCHAR2`, and `NCLOB`. These datatypes are guaranteed to be Unicode encoding and always use character length semantics. The character sets used by `NCHAR`/`NVARCHAR2` can be either `UTF8` or `AL16UTF16`, depending on the setting of the national character set when the database is created. These datatypes allow character data in Unicode to be stored in a database that may or may not use Unicode as database character set.

Implicit Type Conversion

In addition to all the implicit conversions for `CHAR`/`VARCHAR2`, Oracle Database also supports implicit conversion for `NCHAR`/`NVARCHAR2`. Implicit conversion between `CHAR`/`VARCHAR2` and `NCHAR`/`NVARCHAR2` is also supported.

LOB Character Datatypes

The LOB datatypes for character data are `CLOB` and `NCLOB`. They can store up to 8 terabytes of character data (`CLOB`) or national character set data (`NCLOB`).

> **See Also:** "Overview of LOB Datatypes" on page 26-11

LONG Datatype

> **Note:** Do not create tables with `LONG` columns. Use LOB columns (`CLOB`, `NCLOB`) instead. `LONG` columns are supported only for backward compatibility.
>
> Oracle also recommends that you convert existing `LONG` columns to LOB columns. LOB columns are subject to far fewer restrictions than `LONG` columns. Further, LOB functionality is enhanced in every release, whereas `LONG` functionality has been static for several releases.

Columns defined as `LONG` can store variable-length character data containing up to 2 gigabytes of information. `LONG` data is text data that is to be appropriately converted when moving among different systems.

`LONG` datatype columns are used in the data dictionary to store the text of view definitions. You can use `LONG` columns in `SELECT` lists, `SET` clauses of `UPDATE` statements, and `VALUES` clauses of `INSERT` statements.

See Also:

- *Oracle Database Advanced Application Developer's Guide* for information about the restrictions on the `LONG` datatype

- "Overview of RAW and LONG RAW Datatypes" on page 26-13 for information about the `LONG RAW` datatype

Overview of Numeric Datatypes

The numeric datatypes store positive and negative fixed and floating-point numbers, zero, infinity, and values that are the undefined result of an operation (that is, is "not a number" or NAN).

This section includes the following topics:

- NUMBER Datatype
- Floating-Point Numbers

NUMBER Datatype

The `NUMBER` datatype stores fixed and floating-point numbers. Numbers of virtually any magnitude can be stored and are guaranteed portable among different systems operating Oracle Database, up to 38 digits of precision.

The following numbers can be stored in a `NUMBER` column:

- Positive numbers in the range 1×10^{-130} to $9.99...9 \times 10^{125}$ with up to 38 significant digits

- Negative numbers from -1×10^{-130} to $9.99...99 \times 10^{125}$ with up to 38 significant digits

- Zero

- Positive and negative infinity (generated only by importing from an Oracle Database, Version 5)

For numeric columns, you can specify the column as:

```
column_name NUMBER
```

Optionally, you can also specify a **precision** (total number of digits) and **scale** (number of digits to the right of the decimal point):

```
column_name NUMBER (precision, scale)
```

If a precision is not specified, the column stores values as given. If no scale is specified, the scale is zero.

Oracle guarantees portability of numbers with a precision equal to or less than 38 digits. You can specify a scale and no precision:

```
column_name NUMBER (*, scale)
```

In this case, the precision is 38, and the specified scale is maintained.

When you specify numeric fields, it is a good idea to specify the precision and scale. This provides extra integrity checking on input.

Table 26–1 shows examples of how data would be stored using different scale factors.

Table 26–1   How Scale Factors Affect Numeric Data Storage

Input Data	Specified As	Stored As
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	(not accepted, exceeds precision)
7,456,123.89	NUMBER(7,-2)	7456100

If you specify a negative scale, Oracle Database rounds the actual data to the specified number of places to the left of the decimal point. For example, specifying (7,-2) means Oracle Database rounds to the nearest hundredths, as shown in Table 26–1.

For input and output of numbers, the standard Oracle Database default decimal character is a period, as in the number 1234.56. The decimal is the character that separates the integer and decimal parts of a number. You can change the default decimal character with the initialization parameter NLS_NUMERIC_CHARACTERS. You can also change it for the duration of a session with the ALTER SESSION statement. To enter numbers that do not use the current default decimal character, use the TO_NUMBER function.

Internal Numeric Format

Oracle Database stores numeric data in variable-length format. Each value is stored in scientific notation, with 1 byte used to store the exponent and up to 20 bytes to store the mantissa. The resulting value is limited to 38 digits of precision. Oracle Database does not store leading and trailing zeros. For example, the number 412 is stored in a format similar to 4.12×10^2, with 1 byte used to store the exponent(2) and 2 bytes used to store the three significant digits of the mantissa(4,1,2). Negative numbers include the sign in their length.

Taking this into account, the column size in bytes for a particular numeric data value NUMBER(p), where p is the precision of a given value, can be calculated using the following formula:

```
ROUND((length(p)+s)/2))+1
```

where s equals zero if the number is positive, and s equals 1 if the number is negative.

Zero and positive and negative infinity (only generated on import from Oracle Database, Version 5) are stored using unique representations. Zero and negative infinity each require 1 byte; positive infinity requires 2 bytes.

Floating-Point Numbers

Oracle Database provides two numeric datatypes exclusively for floating-point numbers: BINARY_FLOAT and BINARY_DOUBLE. They support all of the basic functionality provided by the NUMBER datatype. However, while NUMBER uses decimal precision, BINARY_FLOAT and BINARY_DOUBLE use binary precision. This enables faster arithmetic calculations and usually reduces storage requirements.

BINARY_FLOAT and BINARY_DOUBLE are approximate numeric datatypes. They store approximate representations of decimal values, rather than exact representations. For example, the value 0.1 cannot be exactly represented by either BINARY_DOUBLE or BINARY_FLOAT. They are frequently used for scientific computations. Their behavior is similar to the datatypes FLOAT and DOUBLE in Java and XMLSchema.

This section includes the following topics:

- BINARY_FLOAT Datatype
- BINARY_DOUBLE Datatype

BINARY_FLOAT Datatype

BINARY_FLOAT is a 32-bit, single-precision floating-point number datatype. Each BINARY_FLOAT value requires 5 bytes, including a length byte.

BINARY_DOUBLE Datatype

BINARY_DOUBLE is a 64-bit, double-precision floating-point number datatype. Each BINARY_DOUBLE value requires 9 bytes, including a length byte.

> **Note:** BINARY_DOUBLE and BINARY_FLOAT implement most of the Institute of Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic, IEEE Standard 754-1985 (IEEE754). For a full description of the Oracle Database implementation of floating-point numbers and its differences from IEEE754, see the *Oracle Database SQL Language Reference*

Overview of DATE Datatype

The DATE datatype stores point-in-time values (dates and times) in a table. The DATE datatype stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight).

Oracle Database can store dates in the Julian era, ranging from January 1, 4712 BCE through December 31, 4712 CE (Common Era, or 'AD'). Unless BCE ('BC' in the format mask) is specifically used, CE date entries are the default.

Oracle Database uses its own internal format to store dates. Date data is stored in fixed-length fields of seven bytes each, corresponding to century, year, month, day, hour, minute, and second.

For input and output of dates, the standard Oracle date format is DD-MON-YY, as follows:

```
'13-NOV-92'
```

You can change this default date format for an instance with the parameter NLS_DATE_FORMAT. You can also change it during a user session with the ALTER SESSION statement. To enter dates that are not in standard Oracle date format, use the TO_DATE function with a format mask:

```
TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')
```

Oracle Database stores time in 24-hour format—HH:MI:SS. By default, the time in a date field is 00:00:00 A.M. (midnight) if no time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month. To enter the time

portion of a date, use the `TO_DATE` function with a format mask indicating the time portion, as in:

```
INSERT INTO birthdays (bname, bday) VALUES
    ('ANDY',TO_DATE('13-AUG-66 12:56 A.M.','DD-MON-YY HH:MI A.M.'));
```

This section includes the following topics:

- Use of Julian Dates

- Date Arithmetic

- Centuries and the Year 2000

- Daylight Savings Support

- Time Zones

Use of Julian Dates

Julian dates allow continuous dating by the number of days from a common reference. (The reference is 01-01-4712 years BCE, so current dates are somewhere in the 2.4 million range.) A Julian date is nominally a noninteger, the fractional part being a portion of a day. Oracle Database uses a simplified approach that results in integer values. Julian dates can be calculated and interpreted differently. The calculation method used by Oracle Database results in a seven-digit number (for dates most often used), such as 2449086 for 08-APR-93.

> **Note:** Oracle Julian dates might not be compatible with Julian dates generated by other date algorithms.

The format mask `'J'` can be used with date functions (`TO_DATE` or `TO_CHAR`) to convert date data into Julian dates. For example, the following query returns all dates in Julian date format:

```
SELECT TO_CHAR (hire_date, 'J') FROM employees;
```

You must use the `TO_NUMBER` function if you want to use Julian dates in calculations. You can use the `TO_DATE` function to enter Julian dates:

```
INSERT INTO employees (hire_date) VALUES (TO_DATE(2448921, 'J'));
```

Date Arithmetic

Oracle date arithmetic takes into account the anomalies of the calendars used throughout history. For example, the switch from the Julian to the Gregorian calendar, 15-10-1582, eliminated the previous 10 days (05-10-1582 through 14-10-1582). The year 0 does not exist.

You can enter missing dates into the database, but they are ignored in date arithmetic and treated as the next "real" date. For example, the next day after 04-10-1582 is 15-10-1582, and the day following 05-10-1582 is also 15-10-1582.

> **Note:** This discussion of date arithmetic might not apply to all countries' date standards (such as those in Asia).

Centuries and the Year 2000

Oracle Database stores year data with the century information. For example, Oracle Database stores 1996 or 2001, and not simply 96 or 01. The DATE datatype always stores a four-digit year internally, and all other dates stored internally in the database have four digit years. Oracle Database utilities such as import, export, and recovery also deal with four-digit years.

Daylight Savings Support

Oracle Database provides daylight savings support for DATETIME datatypes in the server. You can insert and query DATETIME values based on local time in a specific region. The DATETIME datatypes TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE are time-zone aware.

> **See Also:**
> - *Oracle Database Advanced Application Developer's Guide* for more information about centuries and date format masks
> - *Oracle Database SQL Language Reference* for information about date format codes

Time Zones

You can include the time zone in your date/time data and provides support for fractional seconds. Three new datatypes are added to DATE, with the differences listed in Table 26–2.

Table 26–2   Time Zone Datatypes

Datatype	Time Zone	Fractional Seconds
DATE	No	No
TIMESTAMP	No	Yes
TIMESTAMP WITH TIME ZONE	Explicit	Yes
TIMESTAMP WITH LOCAL TIME ZONE	Relative	Yes

TIMESTAMP WITH LOCAL TIME ZONE is stored in the database time zone. When a user selects the data, the value is adjusted to the user's session time zone.

For example, a San Francisco database has system time zone = -8:00. When a New York client (session time zone = -5:00) inserts into or selects from the San Francisco database, TIMESTAMP WITH LOCAL TIME ZONE data is adjusted as follows:

- The New York client inserts TIMESTAMP'1998-1-23 6:00:00-5:00' into a TIMESTAMP WITH LOCAL TIME ZONE column in the San Francisco database. The inserted data is stored in San Francisco as binary value 1998-1-23 3:00:00.

- When the New York client selects that inserted data from the San Francisco database, the value displayed in New York is '1998-1-23 6:00:00'.

- A San Francisco client, selecting the same data, see the value '1998-1-23 3:00:00'.

> **Note:** To avoid unexpected results in your DML operations on
> datatime data, you can verify the database and session time zones by
> querying the built-in SQL functions DBTIMEZONE and
> SESSIONTIMEZONE. If the database time zone or the session time
> zone has not been set manually, Oracle Database uses the operating
> system time zone by default. If the operating system time zone is not a
> valid Oracle time zone, Oracle Database uses UTC as the default
> value.

> **See Also:** *Oracle Database SQL Language Reference* for details about
> the syntax of creating and entering data in time stamp columns

Overview of LOB Datatypes

The LOB datatypes BLOB, CLOB, NCLOB, and BFILE enable you to store and
manipulate large blocks of unstructured data (such as text, graphic images, video clips,
and sound waveforms) in binary or character format. They provide efficient, random,
piece-wise access to the data. Oracle recommends that you always use LOB datatypes
over LONG datatypes. You can perform parallel queries (but not parallel DML or DDL)
on LOB columns.

LOB datatypes differ from LONG and LONG RAW datatypes in several ways. For
example:

- A table can contain multiple LOB columns but only one LONG column.

- A table containing one or more LOB columns can be partitioned, but a table
  containing a LONG column cannot be partitioned.

- The maximum size of a LOB is 128 terabytes depending on database block size,
  and the maximum size of a LONG is only 2 gigabytes.

- LOBs support random access to data, but LONGs support only sequential access.

- LOB datatypes (except NCLOB) can be attributes of a user-defined object type but
  LONG datatypes cannot.

- Temporary LOBs that act like local variables can be used to perform
  transformations on LOB data. Temporary internal LOBs (BLOBs, CLOBs, and
  NCLOBs) are created in a temporary tablespace and are independent of tables. For
  LONG datatypes, however, no temporary structures are available.

- Tables with LOB columns can be replicated, but tables with LONG columns cannot.

SQL statements define LOB columns in a table and LOB attributes in a user-defined
object type. When defining LOBs in a table, you can explicitly specify the tablespace
and storage characteristics for each LOB.

LOB datatypes can be stored inline (within a table), out-of-line (within a tablespace,
using a LOB locator), or in an external file (BFILE datatypes). With compatibility set to
Oracle9*i* or higher, you can use LOBs with SQL VARCHAR operators and functions.

See Also:

- *Oracle Database SQL Language Reference* for a list of differences between the LOB datatypes and the LONG and LONG RAW datatypes

- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOB storage and LOB locators

This section includes the following topics:

- BLOB Datatype

- CLOB and NCLOB Datatypes

- BFILE Datatype

BLOB Datatype

The BLOB datatype stores unstructured binary data in the database. BLOBs can store up to 128 terabytes of binary data.

BLOBs participate fully in transactions. Changes made to a BLOB value by the DBMS_LOB package, PL/SQL, or the OCI can be committed or rolled back. However, BLOB locators cannot span transactions or sessions.

CLOB and NCLOB Datatypes

The CLOB and NCLOB datatypes store up to 128 terabytes of character data in the database. CLOBs store database character set data, and NCLOBs store Unicode national character set data. Storing varying-width LOB data in a fixed-width Unicode character set internally enables Oracle Database to provide efficient character-based random access on CLOBs and NCLOBs.

CLOBs and NCLOBs participate fully in transactions. Changes made to a CLOB or NCLOB value by the DBMS_LOB package, PL/SQL, or the OCI can be committed or rolled back. However, CLOB and NCLOB locators cannot span transactions or sessions. You cannot create an object type with NCLOB attributes, but you can specify NCLOB parameters in a method for an object type.

See Also: *Oracle Database Globalization Support Guide* for more information about national character set data and Unicode

BFILE Datatype

The BFILE datatype stores unstructured binary data in operating-system files outside the database. A BFILE column or attribute stores a file locator that points to an external file containing the data. The amount of BFILE data that can be stored is limited by the operating system.

BFILEs are read only; you cannot modify them. They support only random (not sequential) reads, and they do not participate in transactions. The underlying operating system must maintain the file integrity, security, and durability for BFILEs. The database administrator must ensure that the file exists and that Oracle Database processes have operating-system read permissions on the file.

Overview of RAW and LONG RAW Datatypes

> **Note:** The LONG RAW datatype is provided for backward compatibility with existing applications. For new applications, use the BLOB and BFILE datatypes for large amounts of binary data.
>
> Oracle also recommends that you convert existing LONG RAW columns to LOB columns. LOB columns are subject to far fewer restrictions than LONG columns. Further, LOB functionality is enhanced in every release, whereas LONG RAW functionality has been static for several releases.

The RAW and LONG RAW datatypes are used for data that is not to be interpreted (not converted when moving data between different systems) by Oracle Database. These datatypes are intended for binary data or byte strings. For example, LONG RAW can be used to store graphics, sound, documents, or arrays of binary data. The interpretation depends on the use.

RAW is a variable-length datatype like the VARCHAR2 character datatype, except Oracle Net Services (which connects user sessions to the instance) and the Import and Export utilities do not perform character conversion when transmitting RAW or LONG RAW data. In contrast, Oracle Net Services and Import/Export automatically convert CHAR, VARCHAR2, and LONG data between the database character set and the user session character set, if the two character sets are different.

When Oracle Database automatically converts RAW or LONG RAW data to and from CHAR data, the binary data is represented in hexadecimal form with one hexadecimal character representing every four bits of RAW data. For example, one byte of RAW data with bits 11001011 is displayed and entered as 'CB'.

LONG RAW data cannot be indexed, but RAW data can be indexed.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for information about other restrictions on the LONG RAW datatype

Overview of ROWID and UROWID Datatypes

Oracle Database uses a ROWID datatype to store the address (**rowid**) of every row in the database.

- **Physical rowids** store the addresses of rows in ordinary tables (excluding index-organized tables), clustered tables, table partitions and subpartitions, indexes, and index partitions and subpartitions.

- **Logical rowids** store the addresses of rows in index-organized tables.

A single datatype called the **universal rowid**, or UROWID, supports both logical and physical rowids, as well as rowids of foreign tables such as non-Oracle tables accessed through a gateway.

A column of the UROWID datatype can store all kinds of rowids. The value of the COMPATIBLE initialization parameter (for file format compatibility) must be set to 8.1 or higher to use UROWID columns.

> **See Also:**

This section includes the following topics:

- [The ROWID Pseudocolumn](#)
- [Physical Rowids](#)
- [Logical Rowids](#)
- [Rowids in Non-Oracle Databases](#)

The ROWID Pseudocolumn

Each table in an Oracle database internally has a **pseudocolumn** named ROWID. This pseudocolumn is not evident when listing the structure of a table by executing a SELECT * FROM ... statement, or a DESCRIBE ... statement using SQL*Plus, nor does the pseudocolumn take up space in the table. However, each row's address can be retrieved with a SQL query using the reserved word ROWID as a column name, for example:

```
SELECT ROWID, last_name FROM employees;
```

You cannot set the value of the pseudocolumn ROWID in INSERT or UPDATE statements, and you cannot delete a ROWID value. Oracle Database uses the ROWID values in the pseudocolumn ROWID internally for the construction of indexes.

You can reference rowids in the pseudocolumn ROWID like other table columns (used in SELECT lists and WHERE clauses), but rowids are not stored in the database, nor are they database data. However, you can create tables that contain columns having the ROWID datatype, although Oracle does not guarantee that the values of such columns are valid rowids. The user must ensure that the data stored in the ROWID column truly is a valid ROWID.

> **See Also:**

Physical Rowids

Physical rowids provide the fastest possible access to a row of a given table. They contain the physical address of a row (down to the specific block) and allow you to retrieve the row in a single block access. Oracle guarantees that as long as the row exists, its rowid does not change. These performance and stability qualities make rowids useful for applications that select a set of rows, perform some operations on them, and then access some of the selected rows again, perhaps with the purpose of updating them.

Every row in a nonclustered table is assigned a unique rowid that corresponds to the physical address of a row's row piece (or the initial row piece if the row is chained among multiple row pieces). In the case of clustered tables, rows in different tables that are in the same data block can have the same rowid.

A row's assigned rowid remains unchanged unless the row is exported and imported using the Import and Export utilities. When you delete a row from a table and then commit the encompassing transaction, the deleted row's associated rowid can be assigned to a row inserted in a subsequent transaction.

A physical rowid datatype has one of two formats:

- The **extended rowid** format supports tablespace-relative data block addresses and efficiently identifies rows in partitioned tables and indexes as well as nonpartitioned tables and indexes. Tables and indexes created by an Oracle8*i* (or higher) server always have extended rowids.

- A **restricted rowid** format is also available for backward compatibility with applications developed with Oracle Database Version 7 or earlier releases.

This section includes the following topics:

- Extended Rowids

- Restricted Rowids

- Examples of Rowid Use

- How Rowids Are Used

Extended Rowids

Extended rowids use a base 64 encoding of the physical address for each row selected. The encoding characters are `A-Z`, `a-z`, `0-9`, `+`, and `/`. For example, the following query:

```
SELECT ROWID, last_name FROM employees WHERE department_id = 20;
```

can return the following row information:

```
ROWID              LAST_NAME
------------------ ----------
AAAAaoAATAAABrXAAA BORTINS
AAAAaoAATAAABrXAAE RUGGLES
AAAAaoAATAAABrXAAG CHEN
AAAAaoAATAAABrXAAN BLUMBERG
```

An extended rowid has a four-piece format, `OOOOOOFFFBBBBBBRRR`:

- `OOOOOO`: The **data object number** that identifies the database segment (`AAAAao` in the example). Schema objects in the same segment, such as a cluster of tables, have the same data object number.

- `FFF`: The tablespace-relative **datafile number** of the datafile that contains the row (file `AAT` in the example).

- `BBBBBB`: The **data block** that contains the row (block `AAABrX` in the example). Block numbers are relative to their datafile, **not** tablespace. Therefore, two rows with identical block numbers could reside in two different datafiles of the same tablespace.

- `RRR`: The **row** in the block.

You can retrieve the data object number from data dictionary views `USER_OBJECTS`, `DBA_OBJECTS`, and `ALL_OBJECTS`. For example, the following query returns the data object number for the `employees` table in the `SCOTT` schema:

```
SELECT DATA_OBJECT_ID FROM DBA_OBJECTS
    WHERE OWNER = 'SCOTT' AND OBJECT_NAME = 'EMPLOYEES';
```

You can also use the `DBMS_ROWID` package to extract information from an extended rowid or to convert a rowid from extended format to restricted format (or vice versa).

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for information about the `DBMS_ROWID` package

Restricted Rowids

Restricted rowids use a binary representation of the physical address for each row selected. When queried using SQL*Plus, the binary representation is converted to a `VARCHAR2`/hexadecimal representation. The following query:

```
SELECT ROWID, last_name FROM employees
    WHERE department_id = 30;
```

can return the following row information:

```
ROWID             ENAME
----------------- ----------
00000DD5.0000.0001 KRISHNAN
00000DD5.0001.0001 ARBUCKLE
00000DD5.0002.0001 NGUYEN
```

As shown, a restricted rowid's VARCHAR2/hexadecimal representation is in a three-piece format, **block.row.file**:

- The **data block** that contains the row (block DD5 in the example). Block numbers are relative to their datafile, **not** tablespace. Two rows with identical block numbers could reside in two different datafiles of the same tablespace.

- The **row** in the block that contains the row (rows 0, 1, 2 in the example). Row numbers of a given block always start with 0.

- The **datafile** that contains the row (file 1 in the example). The first datafile of every database is always 1, and file numbers are unique within a database.

Examples of Rowid Use

You can use the function SUBSTR to break the data in a rowid into its components. For example, you can use SUBSTR to break an extended rowid into its four components (database object, file, block, and row):

```
SELECT ROWID,
       SUBSTR(ROWID,1,6) "OBJECT",
       SUBSTR(ROWID,7,3) "FIL",
       SUBSTR(ROWID,10,6) "BLOCK",
       SUBSTR(ROWID,16,3) "ROW"
       FROM products;

ROWID             OBJECT  FIL  BLOCK   ROW
----------------- ------  ---  ------  ----
AAAA8mAALAAAAQkAAA  AAAA8m  AAL  AAAAQk  AAA
AAAA8mAALAAAAQkAAF  AAAA8m  AAL  AAAAQk  AAF
AAAA8mAALAAAAQkAAI  AAAA8m  AAL  AAAAQk  AAI
```

Or you can use SUBSTR to break a restricted rowid into its three components (block, row, and file):

```
SELECT ROWID, SUBSTR(ROWID,15,4) "FILE",
       SUBSTR(ROWID,1,8) "BLOCK",
       SUBSTR(ROWID,10,4) "ROW"
       FROM products;

ROWID             FILE  BLOCK     ROW
----------------- ----  --------  ----
00000DD5.0000.0001  0001  00000DD5  0000
00000DD5.0001.0001  0001  00000DD5  0001
00000DD5.0002.0001  0001  00000DD5  0002
```

Rowids can be useful for revealing information about the physical storage of a table's data. For example, if you are interested in the physical location of a table's rows (such as for table striping), the following query of an extended rowid tells how many datafiles contain rows of a given table:

```
SELECT COUNT(DISTINCT(SUBSTR(ROWID,7,3))) "FILES" FROM tablename;
```

```
    FILES
--------
       2
```

How Rowids Are Used

Oracle Database uses rowids internally for the construction of indexes. Each key in an index is associated with a rowid that points to the associated row's address for fast access. End users and application developers can also use rowids for several important functions:

- Rowids are the fastest means of accessing particular rows.
- Rowids can be used to see how a table is organized.
- Rowids are unique identifiers for rows in a given table.

Before you use rowids in DML statements, they should be verified and guaranteed not to change. The intended rows should be locked so they cannot be deleted. Under some circumstances, requesting data with an invalid rowid could cause a statement to fail.

You can also create tables with columns defined using the `ROWID` datatype. For example, you can define an exception table with a column of datatype `ROWID` to store the rowids of rows in the database that violate integrity constraints. Columns defined using the `ROWID` datatype behave like other table columns: values can be updated, and so on. Each value in a column defined as datatype `ROWID` requires six bytes to store pertinent column data.

Logical Rowids

Rows in index-organized tables do not have permanent physical addresses—they are stored in the index leaves and can move within the block or to a different block as a result of insertions. Therefore their row identifiers cannot be based on physical addresses. Instead, Oracle provides index-organized tables with logical row identifiers, called **logical rowids**, that are based on the table's primary key. Oracle Database uses these logical rowids for the construction of secondary indexes on index-organized tables.

Each logical rowid used in a secondary index includes a **physical guess**, which identifies the block location of the row in the index-organized table at the time the guess was made; that is, when the secondary index was created or rebuilt.

Oracle Database can use guesses to probe into the leaf block directly, bypassing the full key search. This ensures that rowid access of nonvolatile index-organized tables gives comparable performance to the physical rowid access of ordinary tables. In a volatile table, however, if the guess becomes stale the probe can fail, in which case a primary key search must be performed.

The values of two logical rowids are considered equal if they have the same primary key values but different guesses.

This section includes the following topics:

- Comparison of Logical Rowids with Physical Rowids
- Guesses in Logical Rowids

Comparison of Logical Rowids with Physical Rowids

Logical rowids are similar to the physical rowids in the following ways:

- Logical rowids are accessible through the `ROWID` pseudocolumn.

  You can use the `ROWID` pseudocolumn to select logical rowids from an index-organized table. The `SELECT ROWID` statement returns an opaque structure, which internally consists of the table's primary key and the physical guess (if any) for the row, along with some control information.

  You can access a row using predicates of the form `WHERE ROWID = value`, where *value* is the opaque structure returned by `SELECT ROWID`.

- Access through the logical rowid is the fastest way to get to a specific row, although it can require more than one block access.

- A row's logical rowid does not change as long as the primary key value does not change. This is less stable than the physical rowid, which stays immutable through all updates to the row.

- Logical rowids can be stored in a column of the `UROWID` datatype

One difference between physical and logical rowids is that logical rowids cannot be used to see how a table is organized.

> **Note:** An opaque type is one whose internal structure is not known to the database. The database provides storage for the type. The type designer can provide access to the contents of the type by implementing functions, typically 3GL routines.

> **See Also:** "Overview of ROWID and UROWID Datatypes" on page 26-13

Guesses in Logical Rowids

When a row's physical location changes, the logical rowid remains valid even if it contains a guess, although the guess could become stale and slow down access to the row. Guess information cannot be updated dynamically. For secondary indexes on index-organized tables, however, you can rebuild the index to obtain fresh guesses. Note that rebuilding a secondary index on an index-organized table involves reading the base table, unlike rebuilding an index on an ordinary table.

Collect index statistics with the `DBMS_STATS` package or `ANALYZE` statement to keep track of the staleness of guesses, so Oracle Database does not use them unnecessarily. This is particularly important for applications that store rowids with guesses persistently in a `UROWID` column, then retrieve the rowids later and use them to fetch rows.

When you collect index statistics with the `DBMS_STATS` package or `ANALYZE` statement, Oracle Database checks whether the existing guesses are still valid and records the percentage of stale/valid guesses in the data dictionary. After you rebuild a secondary index (recomputing the guesses), collect index statistics again.

In general, logical rowids without guesses provide the fastest possible access for a highly volatile table. If a table is static or if the time between getting a rowid and using

it is sufficiently short to make row movement unlikely, logical rowids with guesses provide the fastest access.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about collecting statistics

Rowids in Non-Oracle Databases

Oracle Database applications can be run against non-Oracle database servers using SQL*Connect. The format of rowids varies according to the characteristics of the non-Oracle system. Furthermore, no standard translation to VARCHAR2/hexadecimal format is available. Programs can still use the ROWID datatype. However, they must use a nonstandard translation to hexadecimal format of length up to 256 bytes.

Rowids of a non-Oracle database can be stored in a column of the UROWID datatype.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for details on handling rowids with non-Oracle systems
> - "Overview of ROWID and UROWID Datatypes" on page 26-13

Overview of ANSI, DB2, and SQL/DS Datatypes

SQL statements that create tables and clusters can also use ANSI datatypes and datatypes from IBM's products SQL/DS and DB2. Oracle Database recognizes the ANSI or IBM datatype name that differs from the Oracle datatype name, records it as the name of the datatype of the column, and then stores the column's data in an Oracle datatype based on the conversions.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the conversions

Overview of XML Datatypes

Oracle provides the XMLType datatype to handle XML data.

XMLType Datatype

XMLType can be used like any other user-defined type. XMLType can be used as the datatype of columns in tables and views. Variables of XMLType can be used in PL/SQL stored procedures as parameters, return values, and so on. You can also use XMLType in PL/SQL, SQL and Java, and through JDBC and OCI.

A number of useful functions that operate on XML content have been provided. Many of these are provided both as SQL functions and as member functions of XMLType. For example, function extract extracts a specific node(s) from an XMLType instance. You can use XMLType in SQL queries in the same way as any other user-defined datatypes in the system.

See Also:

- *Oracle XML Developer's Kit Programmer's Guide*

- *Oracle XML DB Developer's Guide*

- *Oracle Streams Advanced Queuing User's Guide* for information about using `XMLType` with Advanced Queuing

- Chapter 1, "Introduction to Oracle Database"

Overview of URI Datatypes

A URI, or uniform resource identifier, is a generalized kind of URL. Like a URL, it can reference any document, and can reference a specific part of a document. It is more general than a URL because it has a powerful mechanism for specifying the relevant part of the document. By using `UriType`, you can do the following:

- Create table columns that point to data inside or outside the database.

- Query the database columns using functions provided by `UriType`.

See Also: *Oracle XML DB Developer's Guide*

Overview of Object Datatypes and Object Views

Object types and other user-defined datatypes let you define datatypes that model the structure and behavior of the data in their applications. An object view is a virtual object table.

See Also: *Oracle Database Object-Relational Developer's Guide*

Data Conversion

In some cases, Oracle Database supplies data of one datatype where it expects data of a different datatype. This is allowed when Oracle Database can automatically convert the data to the expected datatype.

See Also: *Oracle Database SQL Language Reference* for the rules for implicit datatype conversions

Glossary

ADDM

See **Automatic Database Diagnostic Monitor (ADDM)**

ADR

See **Automatic Diagnostic Repository**

ADRCI

A command-line tool that is part of the fault diagnosability infrastructure introduced in Oracle Database 11*g*.

AFTER trigger

When defining a trigger, you can specify the trigger timing—whether the trigger action is to be executed before or after the triggering statement.

`AFTER` triggers execute the trigger action after the triggering statement is run.

`BEFORE` and `AFTER` apply to both statement and row triggers.

See also: **trigger**

architecture

See **Oracle architecture**

ARCHIVELOG mode

The mode of the database in which Oracle Database copies filled online redo logs to disk. Specify the mode at database creation or by using the `ALTER DATABASE` statement. You can enable automatic archiving either dynamically using the `ALTER SYSTEM` statement or by setting the initialization parameter `LOG_ARCHIVE_START` to `TRUE`.

Running the database in `ARCHIVELOG` mode has several advantages over `NOARCHIVELOG` mode. You can:

- Back up the database while it is open and being accessed by users.

- Recover the database to any desired point in time.

To protect the `ARCHIVELOG` mode database in case of failure, back up the archived logs.

ASM

See **Automatic Storage Management (ASM)**

Automatic Database Diagnostic Monitor (ADDM)

This lets Oracle Database diagnose its own performance and determine how identified problems could be resolved. It runs automatically after each AWR statistics capture, making the performance diagnostic data readily available.

Automatic Diagnostic Repository

The Automatic Diagnostic Repository (ADR) is a system-wide tracing and logging central repository. The repository is a file-based hierarchical datastore for depositing diagnostic information, including network tracing and logging information.

Automatic Storage Management (ASM)

A vertical integration of both the file system and the volume manager built specifically for Oracle Database files. It extends the concept of stripe and mirror everything to optimize performance, while removing the need for manual I/O tuning.

Automatic Storage Management disk

Storage is added and removed from Automatic Storage Management disk groups in units of Automatic Storage Management disks.

Automatic Storage Management file

Oracle Database file stored in an Automatic Storage Management disk group. When a file is created, certain file attributes are permanently set. Among these are its redundancy level (`MIRROR`, `HIGH`, or `UNPROTECTED`) and its striping policy. Automatic Storage Management files are not visible from the operating system or its utilities, but they are visible to database instances, RMAN, and other Oracle-supplied tools such as `ASMCMD`.

Automatic Storage Management instance

An Oracle database instance that mounts Automatic Storage Management disk groups and performs management functions necessary to make Automatic Storage Management files available to database instances. Automatic Storage Management instances do not mount databases.

See also: **instance**

Automatic Storage Management template

Collections of attributes used by Automatic Storage Management during file creation. Templates simplify file creation by mapping complex file attribute specifications into a single name. A default template exists for each Oracle Database file type. Users can modify the attributes of the default templates or create new templates.

automatic undo management mode

A mode of the database in which it automatically manages undo space in a dedicated undo tablespace. In this mode, the database also automatically tunes the undo retention period. This is the default mode for new database installations for Oracle Database 11*g* and later.

See also: **manual undo management mode**

Automatic Workload Repository (AWR)

A built-in repository in every Oracle Database. At regular intervals, the Oracle Database makes a snapshot of all its vital statistics and workload information and stores them here.

AWR

See **Automatic Workload Repository (AWR)**

background process

Background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. The background processes asynchronously perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

Oracle Database creates a set of background processes for each instance.

See also: **instance**, **process**, **Oracle process**, **user process**

BEFORE trigger

When defining a trigger, you can specify the trigger timing—whether the trigger action is to be executed before or after the triggering statement.

BEFORE triggers execute the trigger action before the triggering statement is run.

BEFORE and AFTER apply to both statement and row triggers.

See also: **trigger**

buffer cache

The portion of the SGA that holds copies of Oracle Database data blocks. All user processes concurrently connected to the instance share access to the buffer cache.

The buffers in the cache are organized in two lists: the dirty list and the least recently used (LRU) list. The dirty list holds dirty buffers, which contain data that has been modified but has not yet been written to disk. The least recently used (LRU) list holds free buffers (unmodified and available), pinned buffers (currently being accessed), and dirty buffers that have not yet been moved to the dirty list.

See also: **system global area (SGA)**

byte semantics

The length of string is measured in bytes.

cache recovery

The part of instance recovery where Oracle Database applies all committed and uncommitted changes in the redo log files to the affected data blocks. Also known as the *rolling forward* phase of instance recovery.

character semantics

The length of string is measured in characters.

CHECK constraint

A CHECK integrity constraint on a column or set of columns requires that a specified condition be true or unknown for every row of the table. If a DML statement results in the condition of the CHECK constraint evaluating to false, then the statement is rolled back.

checkpoint

A data structure that defines an SCN in the redo thread of a database. Checkpoints are recorded in the **control file** and each datafile header, and are a crucial element of recovery.

client

In client/server architecture, the front-end database application, which interacts with a user through the keyboard, display, and pointing device such as a mouse. The client portion has no data access responsibilities. It concentrates on requesting, processing, and presenting data managed by the server portion.

See also: **client/server architecture**, **server**

client/server architecture

Software architecture based on a separation of processing between two CPUs, one acting as the client in the transaction, requesting and receiving services, and the other as the server that provides services in a transaction.

cluster

Optional structure for storing table data. Clusters are groups of one or more tables physically stored together because they share common columns and are often used together. Because related rows are physically stored together, disk access time improves.

column

Vertical space in a database table that represents a particular domain of data. A column has a column name and a specific datatype. For example, in a table of employee information, all of the employees' dates of hire would constitute one column.

See also: **row**, **table**

commit

Make permanent changes to data (inserts, updates, deletes) in the database. Before changes are committed, both the old and new data exist so that changes can be stored or the data can be restored to its prior state.

See also: **rolling back**

concurrency

Simultaneous access of the same data by many users. A multiuser database management system must provide adequate concurrency controls, so that data cannot be updated or changed improperly, compromising data integrity.

See also: **data consistency**

connection

Communication pathway between a user process and an Oracle database instance.

See also: **session**, **user process**

consistent backup

A **whole database backup** that you can open with the `RESETLOGS` option without performing media recovery. In other words, you do not need to apply redo to datafiles in this backup for it to be consistent. All datafiles in a consistent backup must:

- Have the same checkpoint **system change number (SCN)** in their headers, unless they are datafiles in tablespaces that are read only or offline normal (in which case they will have a clean SCN that is earlier than the checkpoint SCN)

- Contain no changes past the checkpoint SCN, that is, are not fuzzy

- Match the datafile checkpoint information stored in the control file

You can only take consistent backups after you have made a clean shutdown of the database. The database must not be opened until the backup has completed.

See also: **inconsistent backup**

control file

A file that records the physical structure of a database and contains the database name, the names and locations of associated databases and redo log files, the time stamp of the database creation, the current log sequence number, and checkpoint information.

See also: **physical structures**, **redo log**

Data Recovery Advisor

An Oracle Database infrastructure that automatically diagnoses persistent data failures, presents repair options to the user, and executes repairs at the user's request. The purpose of Data Recovery Advisor is to reduce the **mean time to recover (MTTR)** and improve manageability and reliability of Oracle Database by providing a centralized tool for automated data repair.

database

Collection of data that is treated as a unit. The purpose of a database is to store and retrieve related information. Each Oracle database instance accesses only one database.

database buffer

One of several types of memory structures that stores information within the system global area. Database buffers store the most recently used blocks of data.

See also: **system global area (SGA)**

database buffer cache

Memory structure in the system global area that stores the most recently used blocks of data.

See also: **system global area (SGA)**

database link

A named schema object that describes a path from one database to another. Database links are implicitly used when a reference is made to a global object name in a distributed database.

database writer process (DBW*n*)

An Oracle background process that writes the contents of buffers to datafiles. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk.

See also: **buffer cache**

datafile

A physical file on disk that was created by Oracle Database and contains data structures such as tables and indexes. A datafile can belong to only one database, and is located either in an operating system file system or in an Automatic Storage Management disk group.

See also: **index**, **physical structures**

datafile copy

A copy of a datafile on disk produced by either:

- The Recovery Manager `COPY` command

- An operating system utility

data block

Smallest logical unit of data storage in Oracle Database. Also called logical blocks, Oracle blocks, or pages. One data block corresponds to a specific number of bytes of physical database space on disk.

See also: **extent**, **segment**

data consistency

In a multiuser environment, where many users can access data at the same time (concurrency), data consistency means that each user sees a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.

See also: **concurrency**

data dictionary

The central set of tables and views that are used as a read-only reference about a particular database. A data dictionary stores such information as:

- The logical and physical structure of the database

- Valid users of the database

- Information about integrity constraints

- How much space is allocated for a schema object and how much of it is in use

A data dictionary is created when a database is created and is automatically updated when the structure of the database is updated.

data integrity

Business rules that dictate the standards for acceptable data. These rules are applied to a database by using integrity constraints and triggers to prevent the entry of invalid information into tables.

See also: **integrity constraint**, **trigger**

data segment

Each nonclustered table has a data segment. All of the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment.

Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.

See also: **cluster**, **extent**, **segment**

dedicated server

A database server configuration in which a server process handles requests for a single user process.

See also: **shared server**

define variables

Variables defined (location, size, and datatype) to receive each fetched value.

disk group

One or more Automatic Storage Management disks managed as a logical unit. Automatic Storage Management disks can be added or dropped from a disk group while preserving the contents of the files in the group, and with only a minimal amount of automatically initiated I/O required to redistribute the data evenly. All I/O to a disk group is automatically spread across all the disks in the group.

dispatcher processes (D*nnn*)

Optional background processes, present only when a shared server configuration is used. At least one dispatcher process is created for every communication protocol in use (D000, . . ., D*nnn*). Each dispatcher process is responsible for routing requests from connected user processes to available shared server processes and returning the responses back to the appropriate user processes.

See also: **shared server**

distributed processing

Software architecture that uses more than one computer to divide the processing for a set of related jobs. Distributed processing reduces the processing load on a single computer.

DDL

Data definition language. Includes statements like `CREATE/ALTER TABLE/INDEX`, which define or change data structure.

DML

Data manipulation language. Includes statements like `INSERT`, `UPDATE`, and `DELETE`, which change data in tables.

DOP

The degree of parallelism of an operation.

extent

Second level of logical database storage. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information.

See also: **data block**, **segment**

failure group

A subset of disks within an Automatic Storage Management (ASM) disk group that share a common resource whose failure must be tolerated. Failure groups are used to determine which ASM disks to use for storing redundant copies of data. For example, if, in a particular disk group, disks 1 through 4 are on disk controller A and disks 5 through 8 are on disk controller B, disks 1 through 4 can be assigned to failure group A, and disks 4 through 8 can be assigned to failure group B. For file extents on disks in failure group A, ASM always stores redundant copies of the extents on disks in failure group B. This way, if disk controller A fails, there is always at least one copy of every file extent available. It is up to the storage administrator to define failure groups based on current storage hardware configuration. If no specific failure group assignments are made, each disk in a disk group is automatically placed in its own failure group.

flash recovery area

An optional disk location that you can use to store recovery-related files such as control file and online redo log copies, archived logs, flashback logs, and RMAN

backups. Oracle Database manages the files in the flash recovery area automatically. You can specify the disk quota, which is the maximum size of the flash recovery area.

foreign key

Integrity constraint that requires each value in a column or set of columns to match a value in a related table's `UNIQUE` or `PRIMARY KEY`.

`FOREIGN KEY` integrity constraints also define referential integrity actions that dictate what Oracle Database should do with dependent data if the data it references is altered.

See also: **integrity constraint**, **primary key**

inconsistent backup

A backup in which some of the files in the backup contain changes that were made after the files were checkpointed. This type of backup needs recovery before it can be made consistent. Inconsistent backups are usually created by taking online database backups; that is, the database is open while the files are being backed up. You can also make an inconsistent backup by backing up datafiles while a database is closed, either:

- Immediately after an Oracle database instance failed (or all instances in an Oracle Real Application Clusters configuration)

- After shutting down the database using `SHUTDOWN ABORT`

Note that inconsistent backups are only useful if the database is in `ARCHIVELOG` mode.

See also: **consistent backup**, **online backup**, **system change number (SCN)**, **whole database backup**

index

Optional structure associated with tables and clusters. You can create indexes on one or more columns of a table to speed access to data on that table.

See also: **cluster**

indextype

An object that registers a new indexing scheme by specifying the set of supported operators and routines that manage a domain index.

index segment

Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.

See also: **index**, **segment**

initialization parameter file

A text file that contains initialization parameter settings. In contrast to the **server parameter file**, this parameter file is not binary and does not need to be located on the database host. The text-based initialization parameter file can be read by the database server, but it is not written to by the server. You can use a text editor to alter the file.

instance

A system global area (SGA) and the Oracle Database background processes constitute an Oracle database instance. Every time a database is started, a system global area is allocated and Oracle Database background processes are started. The SGA is deallocated when the instance shuts down.

See also: **background process**, **system global area (SGA)**, **Automatic Storage Management instance**

integrity

See **data integrity**

integrity constraint

Declarative method of defining a rule for a column of a table. Integrity constraints enforce the business rules associated with a database and prevent the entry of invalid information into tables.

key

Column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database.

See also: **integrity constraint**, **foreign key**, **primary key**

large pool

Optional area in the system global area that provides large memory allocations for Oracle Database backup and restore operations, I/O server processes, and session memory for the shared server and Oracle XA.

See also: **system global area (SGA)**, **process**, **shared server**, **Oracle XA**

logical backups

Backups in which an Oracle export utility uses SQL to read database data and export it into a binary file at the operating system level. You can then import the data back into a database using Oracle utilities. Backups taken with Oracle export utilities differ in the following ways from RMAN backups:

- Database logical objects are exported independently of the files that contain those objects.

- Logical backups can be imported into a different database, even on a different platform. RMAN backups are not portable between databases or platforms.

See also: **physical backups**

logical structures

Logical structures of Oracle Database include tablespaces, schema objects, data blocks, extents, and segments. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

See also: **physical structures**

LogMiner

A utility that lets administrators use SQL to read, analyze, and interpret log files. It can view any redo log file, online or archived. The Oracle Enterprise Manager application Oracle LogMiner Viewer adds a GUI-based interface.

log writer process (LGWR)

The log writer process (LGWR) is responsible for redo log buffer management—writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

See also: **redo log**

manual undo management mode

A mode of the database in which undo blocks are stored in user-managed rollback segments. In **automatic undo management mode**, undo blocks are stored in a system-managed, dedicated undo tablespaces.

See also: **automatic undo management mode**

materialized view

A materialized view provides access to table data by storing the results of a query in a separate schema object.

See also: **view**

mean time to recover (MTTR)

The desired time required to perform instance or media recovery on the database. For example, you may set 10 minutes as the goal for media recovery from a disk failure. A variety of factors influence MTTR for media recovery, including the speed of detection, the type of method used to perform media recovery, and the size of the database.

mounted database

An **instance** that is started and has the control file associated with the database open. You can mount a database without opening it; typically, you put the database in this state for maintenance or for restore and recovery operations.

NOT NULL constraint

Data integrity constraint that requires a column of a table contain no null values.

See also: **NULL value**

NULL value

Absence of a value in a column of a row. Nulls indicate missing, unknown, or inapplicable data. A null should not be used to imply any other value, such as zero.

object type

An object type consists of two parts: a spec and a body. The type body always depends on its type spec.

online backup

A backup of one or more datafiles taken while a database is open and the datafiles are online. When you make a user-managed backup while the database is open, you must put the tablespaces in backup mode by issuing an `ALTER TABLESPACE BEGIN BACKUP` command. When you make an RMAN backup while the database is open, however, you do not need to put the tablespaces in backup mode.

online redo log

The online redo log is a set of two or more files that record all changes made to Oracle Database datafiles and control files. Whenever a change is made to the database, Oracle Database generates a redo record in the redo buffer. The LGWR process flushes the contents of the redo buffer into the redo log.

See also: **redo log**

operator

In memory management, the term operator refers to a data flow operator, such as a sort, hash join, or bitmap merge.

Oracle architecture

Memory and process structures used by Oracle Database to manage a database.

See also: **database**, **process**, **server**

Oracle Clusterware

A portable cluster management solution that is integrated with Oracle Real Application Clusters 10*g* to monitor and restart Oracle components. Oracle Clusterware can be programmed to manage and monitor any application running in the cluster. Within a cluster managed by Oracle Clusterware, you can run both single instance and Oracle Real Application Clusters databases.

Oracle Enterprise Manager

An Oracle system management tool that provides an integrated solution for centrally managing your heterogeneous environment. It combines a graphical console, Oracle Management Servers, Oracle Intelligent Agents, common services, and administrative tools for managing Oracle products.

Oracle process

Oracle processes run Oracle Database code. They include server processes and background processes.

See also: **process**, **server process**, **background process**, **user process**

Oracle RAC

See also: **Oracle Real Application Clusters**

Oracle Real Application Clusters

Option that allows multiple concurrent instances to share a single physical database.

See also: **instance**

Oracle XA

The Oracle XA library is an external interface that allows global transactions to be coordinated by a transaction manager other than Oracle Database.

partition

A smaller and more manageable piece of a table or index. Tables are partitioned based on a partitioning key. For example, a sales history table may be partitioned by sales date, and there could be one partition for every calendar quarter. For large tables, partitions improve query performance and make table administration easier.

physical backups

Physical database files that have been copied from one place to another. The files can be datafiles, archived redo logs, or control files. You can make physical backups using Recovery Manager or with operating system commands such as the UNIX `cp`.

See also: **logical backups**

physical structures

Physical database structures of Oracle Database include datafiles, redo log files, and control files.

See also: **logical structures**

PL/SQL

Oracle's procedural language extension to SQL. PL/SQL enables you to mix SQL statements with procedural constructs. With PL/SQL, you can define and execute PL/SQL program units such as procedures, functions, and packages.

See also: **SQL**

primary key

The column or set of columns included in the definition of a table's PRIMARY KEY constraint. A primary key's values uniquely identify the rows in a table. Only one primary key can be defined for each table.

See also: **PRIMARY KEY constraint**

PRIMARY KEY constraint

Integrity constraint that disallows duplicate values and nulls in a column or set of columns.

See also: **integrity constraint**, **key**

priority inversion

Priority inversion occurs when a high priority job is run with lower amount of resources than a low priority job. Thus the expected priority is "inverted."

process

Each process in an Oracle database instance performs a specific job. By dividing the work of Oracle Database and database applications into several processes, multiple users and applications can connect to a single database instance simultaneously.

See also: **Oracle process**, **user process**

program global area (PGA)

A memory buffer that contains data and control information for a server process. A PGA is created by Oracle Database when a server process is started. The information in a PGA depends on the Oracle Database configuration.

query block

A self-contained DML against a table. A query block can be a top-level DML or a subquery.

See also: **DML**

read consistency

In a multiuser environment, the Oracle Database read consistency ensures that

- The set of data seen by a statement remains constant throughout statement execution (statement-level read consistency).

- Readers and writers of database data do not wait for other writers or other readers of the same data. Writers of database data wait only for other writers who are updating identical rows in concurrent transactions.

See also: **concurrency**, **data consistency**

read-only database

A database opened with the `ALTER DATABASE OPEN READ ONLY` command. As their name suggests, read-only databases are for queries only and cannot be modified. Oracle Database allows a **standby database** to be run in read-only mode, which means that it can be queried while still serving as an up-to-date emergency replacement for the primary database.

Recovery Manager (RMAN)

A utility that backs up, restores, and recovers Oracle Databases. You can use it with or without the central information repository called a recovery catalog. If you do not use a recovery catalog, RMAN uses the database's control file to store information necessary for backup and recovery operations. You can use RMAN in conjunction with a media manager to back up files to tertiary storage.

recovery point objective (RPO)

The maximum amount of data an IT-based organization is willing to lose as a result of a system failure. RPO is a balance point between the costs of more complete recovery and the harm done to the organization by loss of data. It indicates the data-loss tolerance of a business process or an organization in general. It is often measured in terms of time, such as five hours or two days worth of data loss.

recovery time objective (RTO)

The maximum amount of time that an IT-based organization is willing to be down after a system failure. RTO is the balance point between the costs of faster recovery and the costs of downtime to the organization. RTO indicates the downtime tolerance of a business process or an organization in general.

redo log

A set of files that protect altered database data in memory that has not been written to the datafiles. The redo log can consist of two parts: the online redo log and the archived redo log.

See also: **online redo log**

redo log buffer

Memory structure in the system global area that stores redo entries—a log of changes made to the database. The redo entries stored in the redo log buffers are written to an online redo log file, which is used if database recovery is necessary.

See also: **system global area (SGA)**

redo thread

The redo generated by an instance. If the database runs in a single instance configuration, then the database has only one thread of redo. If you run in an Oracle Real Application Clusters configuration, then you have multiple redo threads, one for each instance.

referential integrity

A rule defined on a key (a column or set of columns) in one table that guarantees that the values in that key match the values in a key in a related table (the referenced value). Referential integrity includes the rules that dictate what types of data manipulation are allowed on referenced values and how these actions affect dependent values.

See also: **key**

RMAN

See also: **Recovery Manager (RMAN)**

rollback segment

Logical database structure created by the database administrator to temporarily store undo information. Rollback segments store old data changed by SQL statements in a transaction until it is committed. Oracle has now deprecated this method of storing undo.

See also: **commit**, **logical structures**, **segment**, **automatic undo management mode**

rolling back

The use of rollback segments to undo uncommitted transactions applied to the database during the **rolling forward** stage of recovery.

See also: **commit**, **rolling forward**

rolling forward

The application of redo records or incremental backups to datafiles and control files in order to recover changes to those files.

See also: **rolling back**

row

Set of attributes or values pertaining to one entity or record in a table. A row is a collection of column information corresponding to a single record.

See also: **column**, **table**

ROWID

A globally unique identifier for a row in a database. It is created at the time the row is inserted into a table, and destroyed when it is removed from a table.

schema

Collection of database objects, including logical structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. A schema has the name of the user who controls it.

See also: **logical structures**

segment

Third level of logical database storage. A segment is a set of extents, each of which has been allocated for a specific data structure, and all of which are stored in the same tablespace.

See also: **extent**, **data block**

sequence

A sequence generates a serial list of unique numbers for numeric columns of a database's tables.

server

In a client/server architecture, the computer that runs Oracle software and handles the functions required for concurrent, shared data access. The server receives and processes the SQL and PL/SQL statements that originate from client applications.

See also: **client**, **client/server architecture**

server parameter file

A binary file containing initialization parameter settings that is maintained on the Oracle Database host. You cannot manually edit this file with a text editor. A server parameter file is initially built from a text initialization parameter file by means of the `CREATE SPFILE` statement or created directly with the Database Configuration Assistant.

server process

Server processes handle requests from connected user processes. A server process is in charge of communicating with the user process and interacting with Oracle Database to carry out requests of the associated user process.

See also: **process**, **user process**

service level agreement (SLA)

An agreement between a service provider and a service consumer, usually specifying what service is provided, maximum allowable interruptions in service, who will measure service delivery, and what happens if the provider fails to meet the terms of the agreement.

session

Specific connection of a user to an Oracle database instance through a user process. A session lasts from the time the user connects until the time the user disconnects or exits the database application.

See also: **connection**, **instance**, **user process**

shared pool

Portion of the system global area that contains shared memory constructs such as shared SQL areas. A shared SQL area is required to process every unique SQL statement submitted to a database.

See also: **system global area (SGA)**, **SQL**

shared server

A database server configuration that allows many user processes to share a small number of server processes, minimizing the number of server processes and maximizing the use of available system resources.

See also: **dedicated server**

SQL

Structured Query Language, a nonprocedural language to access data. Users describe in SQL what they want done, and the SQL language compiler automatically generates a procedure to navigate the database and perform the task. Oracle SQL includes many extensions to the ANSI/ISO standard SQL language.

See also: **SQL*Plus**, **PL/SQL**

SQL*Plus

Oracle tool used to run SQL statements against Oracle Database.

See also: **SQL**, **PL/SQL**

standby database

A copy of a production database that you can use for disaster protection. You can update the standby database with archived redo logs from the production database in order to keep it current. If a disaster destroys the production database, you can activate the standby database and make it the new production database.

subtype

In the hierarchy of user-defined datatypes, a subtype is always a dependent on its supertype.

supertype

See: **subtype**

synonym

An alias for a table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym.

system change number (SCN)

A stamp that defines a committed version of a database at a point in time. Oracle Database assigns every committed transaction a unique SCN.

system global area (SGA)

A group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, then the data in the instance's SGA is shared among the users. Consequently, the SGA is sometimes referred to as the shared global area.

See also: **instance**

table

Basic unit of data storage in Oracle Database. Table data is stored in rows and columns.

See also: **column**, **row**

tablespace

A database storage unit that groups related logical structures together.

See also: **logical structures**

tempfile

A file that belongs to a temporary tablespace, and is created with the `TEMPFILE` option. Temporary tablespaces cannot contain permanent database objects such as tables, and are typically used for sorting.

temporary segment

Temporary segments are created by Oracle Database when a SQL statement needs a temporary database area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the system for future use.

See also: **extent**, **segment**

transaction

Logical unit of work that contains one or more SQL statements. All statements in a transaction are committed or rolled back together.

See also: **commit**, **rolling back**

transaction recovery

Transaction recovery involves rolling back all uncommitted transactions of a failed instance. These are "in-progress" transactions that did not commit and that Oracle Database needs to undo. It is possible for uncommitted transactions to get saved to disk. In this case, Oracle Database uses undo data to reverse the effects of any changes that were written to the datafiles but not yet committed.

trigger

Stored database procedure automatically invoked whenever a table or view is modified, for example by INSERT, UPDATE, or DELETE operations.

Unicode

A way of representing all the characters in all the languages in the world. Characters are defined as a sequence of codepoints, a base codepoint followed by any number of surrogates. There are 64K codepoints.

Unicode column

A column of type NCHAR, NVARCHAR2, or NCLOB guaranteed to hold Unicode.

UNIQUE KEY constraint

A data integrity constraint requiring that every value in a column or set of columns (key) be unique—that is, no two rows of a table have duplicate values in a specified column or set of columns.

See also: **integrity constraint**, **key**

user name

The name by which a user is known to Oracle Database and to other users. Every user name is associated with a password, and both must be entered to connect to Oracle Database.

user process

User processes execute the application or Oracle tool code.

See also: **process**, **Oracle process**

UTC

Coordinated Universal Time, previously called Greenwich Mean Time, or GMT.

view

A view is a custom-tailored presentation of the data in one or more tables. A view can also be thought of as a "stored query." Views do not actually contain or store data; they derive their data from the tables on which they are based.

Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view affect it's base tables.

whole database backup

A backup of the control file and all datafiles that belong to a database.

Index

T