Cognizant
Passion for making a difference

Handout: Oracle 10g SQL

Version: ORACLE10gSQL/Handout/0308/1.0
Date: 31-03-08

Cognizant
500 Glen Pointe Center West
Teaneck, NJ 07666
Ph: 201-801-0233
www.cognizant.com

TABLE OF CONTENTS

Introduction

About this Module

The module provides an overview about the basics of SQL statements in Oracle.

Target Audience

This module is designed for the entry level trainees.

Module Objectives

After completing this module, you will be able to:

- Write statements using DDL, DML and TCL statements
- Write queries using all clauses and fetch data from one or more tables or views
- Write queries using all kinds of Joins
- Write simple and complex sub queries
- Write queries in ANSI syntax
- Write Static and Dynamic SQL statements

Pre-requisite

The trainees need to have an idea of the following:

- General Programming and Logic
- Structured Query Language
- Relational DBMS

Session 02: RDBMS Overview and Oracle Database

Learning Objectives

After completing this session, you will be able to:

- Explain the RDBMS concepts
- List the basic components of Oracle
- Describe the importance of SQL*Plus & PL*SQL in Oracle database

Understand the Database

Data base is a persistent repository of logically related data. The purpose of a database is to store and retrieve related information. Data base Provides platform for easy Data management, facilitate data sharing and Reduces data redundancy .

Understand the RDBMS Concepts

DBMS is a software system that enables users to define, create and maintain the database and provides controlled access to database. RDBMS stands for Relational Database Management System is a database which stores data in the form of tables and the relationship among the data is also stored in the form of tables. Relationship between the tables made the representation, loading and retrieval of the data easy.

RDBMS provides a set-oriented database language. For most RDBMS, this set-oriented database language is SQL. *Set oriented* means that SQL processes sets of data in groups.

Codd's Rule

Dr. E.F. Codd, an IBM researcher, first developed the relational data model in 1970. In 1985, Dr. Codd published a list of 12 rules that concisely define an ideal relational database, which have provided a guideline for the design of all relational database systems ever since.

Rule 1: The Information Rule

All data should be presented to the user in table form.

Rule 2: Guaranteed Access Rule

All data should be accessible without ambiguity. This can be accomplished through a combination of the table name, primary key, and column name.

Rule 3: Systematic Treatment of Null Values

A field should be allowed to remain empty. This involves the support of a null value, which is distinct from an empty string or a number with a value of zero. Of course, this can't apply to primary keys. In addition, most database implementations support the concept of a nun- null field constraint that prevents null values in a specific table column.

Rule 4: Dynamic On-Line Catalog Based on the Relational Model

A relational database must provide access to its structure through the same tools that are used to access the data. This is usually accomplished by storing the structure definition within special system tables.

Rule 5: Comprehensive Data Sublanguage Rule

The database must support at least one clearly defined language that includes functionality for data definition, data manipulation, data integrity, and database transaction control. All commercial relational databases use forms of the standard SQL (Structured Query Language) as their supported comprehensive language.

Rule 6: View Updating Rule

Data can be presented to the user in different logical combinations, called views. Each view should support the same full range of data manipulation that direct-access to a table has available. In practice, providing update and delete access to logical views is difficult and is not fully supported by any current database.

Rule 7: High-level Insert, Update, and Delete

Data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.

Rule 8: Physical Data Independence

The user is isolated from the physical method of storing and retrieving information from the database. Changes can be made to the underlying architecture (hardware, disk storage methods) without affecting how the user accesses it.

Rule 9: Logical Data Independence

How a user views data should not change when the logical structure (tables structure) of the database changes. This rule is particularly difficult to satisfy. Most databases rely on strong ties between the user view of the data and the actual structure of the underlying tables.

Rule 10: Integrity Independence

The database language (like SQL) should support constraints on user input that maintain database integrity. This rule is not fully implemented by most major vendors. At a minimum, all databases do preserve two constraints through SQL.

No component of a primary key can have a null value. (Refer rule 3)

If a foreign key is defined in one table, any value in it must exist as a primary key in another table.

Rule 11: Distribution Independence

A user should be totally unaware of whether or not the database is distributed (whether parts of the database exist in multiple locations). A variety of reasons make this rule difficult to implement; I will spend time addressing these reasons when we discuss distributed databases.

Rule 12: No subversion Rule

There should be no way to modify the database structure other than through the multiple row database language (like SQL). Most databases today support administrative tools that allow some direct manipulation of the data structure.

Basic ORACLE Architecture

An Oracle 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 solving the problems of information management. In general, a server reliably manages a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

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

Oracle Grid Architecture

Grid computing is a new 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. Grid computing also enables the use of smaller individual hardware components

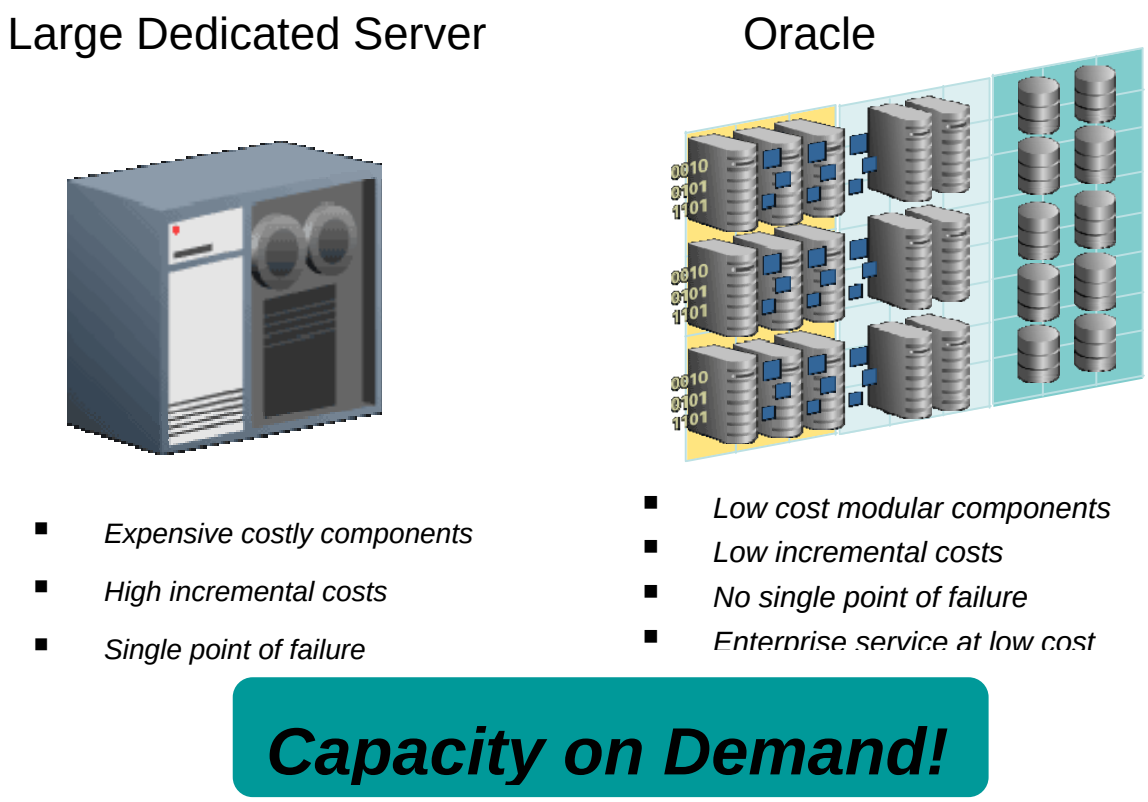


Figure 1: Oracle grid Architecture

Server Virtualization

Oracle Real Application Clusters 10g (RAC) enable a single database to run across multiple clustered nodes in a grid, pooling the processing resources of several standard machines.

Storage Virtualization

The Oracle Automatic Storage Management (ASM) feature of Oracle Database 10g provides a virtualization layer between the database and storage so that multiple disks can be treated as a single disk group and disks can be dynamically added or removed while keeping databases online.

Grid Management

The Grid Control feature of Oracle Enterprise Manager 10g provides a single console to manage multiple systems together as a logical group.

Application Architecture

Client/Server Architecture:

An Oracle database system can easily take advantage of distributed processing by using its client/server architecture. The client runs on a different computer than the database server, generally on a PC.

Multitier Architecture has the following components:

1. A client or initiator process that starts an operation
2. One or more application servers it can serve as an interface between clients and multiple database servers, including providing an additional level of security.
3. An end or database server that stores most of the data used in the operation

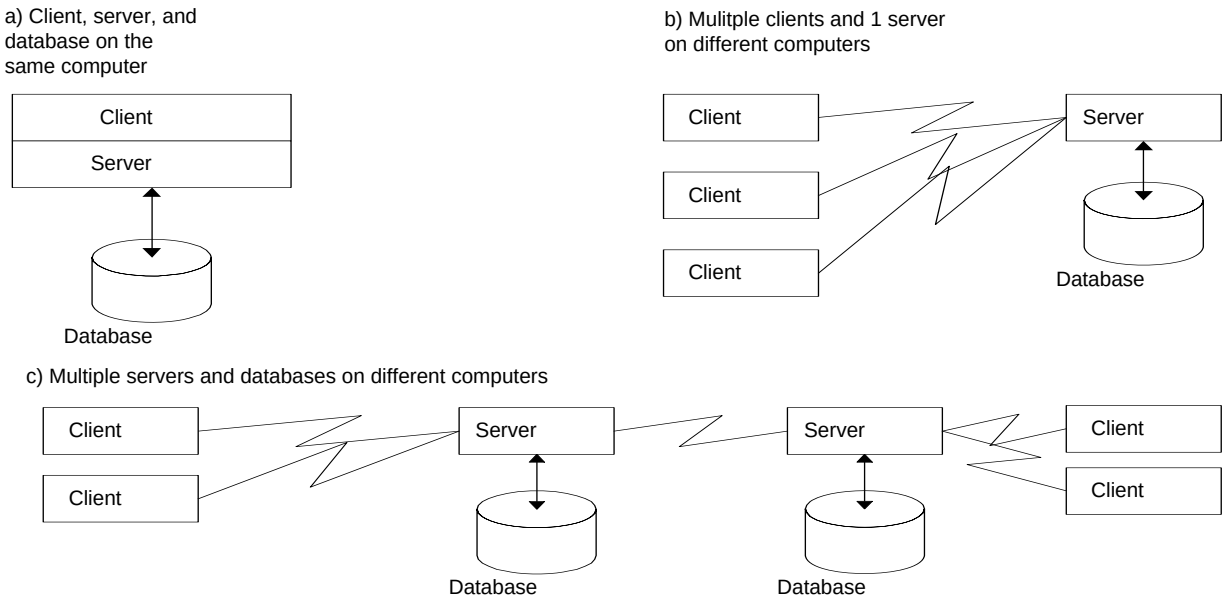


Figure 2

Logical Database Structures

The logical structures of an Oracle database include schema objects, data blocks, extents, segments, and table spaces.

Schemas and 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. Schema objects include structures like tables, views, and indexes.

Some of the most common schema objects are defined in the following section.

Tables

Tables are the basic unit of data storage in an Oracle database. Database tables hold all user-accessible data. Each table has columns and rows. Oracle stores each row of a database table containing data for less than 256 columns as one or more row pieces. A table that has an employee database, for example, can have a column called employee number, and each row in that column is an employee's number.

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 actually contain data. Rather, they derive their data from the tables on which they are based, referred to as the base tables of the views.

Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view actually affect the base tables of the view.

Views provide an additional level of 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.

Indexes

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

When processing a request, Oracle 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 dollars) or a specific row.

Indexes are created on one or more columns of a table. After it is created, an index is automatically maintained and used by Oracle. Changes to table data (such as adding new rows, updating rows, or deleting rows) are automatically incorporated into all relevant indexes with complete transparency to the users.

Segments

The logical storage structures, including data blocks, extents, and segments, enable Oracle to have fine-grained control of disk space use.

Oracle 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 initialization parameter DB_BLOCK_SIZE. A database uses and allocates free database space in Oracle 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 certain logical structure. The following table describes the different types of segments.

Segment	Description
Data segment	Each table has a data segment. All table data is stored in the extents of the 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.
Index segment	Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
Temporary segment	Temporary segments are created by Oracle when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the extents in the temporary segment are returned to the system for future use.
Rollback segment	The information in a rollback segment is used during database recovery: To generate read-consistent database information To roll back uncommitted transactions for users

Oracle dynamically allocates space when the existing extents of a segment become full. In other words, when the extents of a segment are full, Oracle 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.

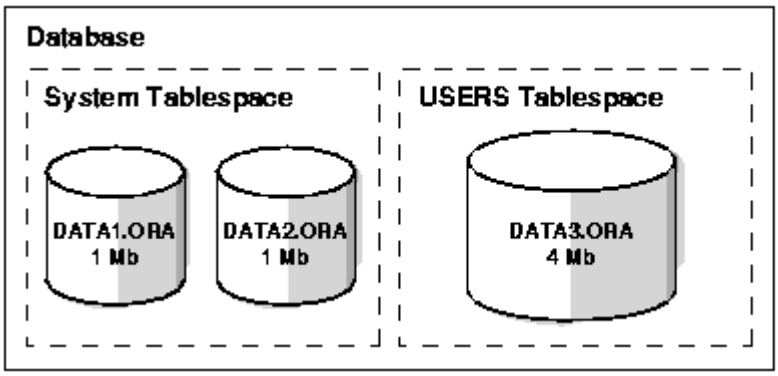
Table spaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group together all application objects to simplify some administrative operations.

Databases, Tablespaces, and Datafiles

The relationship between databases, tablespaces, and datafiles (datafiles are described in the next section) is illustrated in Figure 1.

Figure 3: Database, Tablespaces, and Datafiles



This figure illustrates the following:

- Each database is logically divided into one 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. (The SYSTEM tablespace has 2-megabyte (Mb) storage capacity, and USERS tablespace has 4 MB).
- The combined storage capacity of a database's tablespaces is the total storage capacity of the database (6 Mb).

Physical Database Structures

The following sections explain the physical database structures of an Oracle database, including Datafiles, Control Files, Redo Log Files, Archive Log Files, Parameter Files, Alert and Trace Log Files and Backup Files

Datafiles

Every Oracle database has one or more physical datafiles. The datafiles 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.

The characteristics of datafiles are:

- A datafile can be associated with only one database.
- Datafiles can have certain characteristics set to let them automatically extend when the database runs out of space.

- One or more datafiles form a logical unit of database storage called a tablespace, as discussed earlier in this chapter.
- Data in a datafile is read, as needed, during normal database operation and stored in the memory cache of Oracle. For example, assume that a user wants to access some data in a table of a database. 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 database writer process (DBWn) background process.

Redo Log Files

Every Oracle database has a set of two or more redo log files. The set of redo log files is collectively known as the redo log for the database. A redo log is made up of redo entries (also called redo records).

The primary function of the redo log is to 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.

The information in a redo log file is used only to recover the database from a system or media failure that prevents database data from being written to the datafiles. For example, if an unexpected power outage terminates database operation, then data in memory cannot be written to the datafiles, and the data is lost. However, lost data can be recovered when the database is opened, after power is restored. By applying the information in the most recent redo log files to the database datafiles, Oracle restores the database to the time at which the power failure occurred.

The process of applying the redo log during a recovery operation is called rolling forward.

Control Files

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

- Database name
- Names and locations of datafiles and redo log files
- Time stamp of database creation
- Use of Control Files

Every time an instance of an Oracle database is started, its control file identifies the database and redoes 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 data file or redo log file is created), then the control file is automatically modified by Oracle to reflect the change. A control file is also used in database recovery.

Introduction to Oracle Memory Structures

Oracle uses memory to store information such as the following:

Program code

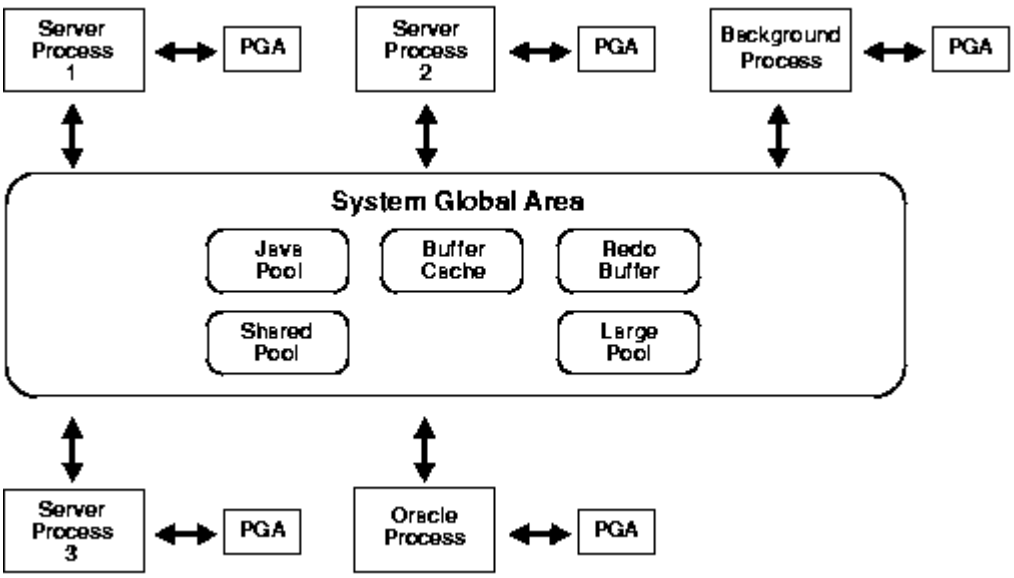
- Information about a connected session, even if it is not currently active
- Information needed during program execution (for example, the current state of a query from which rows are being fetched)
- Information that is shared and communicated among Oracle processes (for example, locking information)
- Cached data that is also permanently stored on peripheral memory (for example, data blocks and redo log entries)

The basic memory structures associated with Oracle include:

- **System Global Area (SGA),** which is shared by all server and background processes and holds the following:
  - Database buffer cache
  - Redo log buffer
  - Shared pool
  - Large pool (if configured)
- **Program Global Areas (PGA),** which is private to each server and background process; there is one PGA for each process. The PGA holds the following:
  - Stack areas
  - Data areas

Figure 4 illustrates the relationships among these memory structures.

Figure 4 Oracle Memory Structures



System Global Area (SGA) Overview

A system global area (SGA) is 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 called the shared global area.

An SGA and Oracle processes constitute an Oracle instance. Oracle 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 users connected to a multiple-process database instance can read information contained within the instance's SGA, and several processes write to the SGA during execution of Oracle.

The SGA contains the following data structures:

- Database buffer cache
- Redo log buffer
- Shared pool
- Java pool
- Large pool (optional)
- Data dictionary cache
- Other miscellaneous information

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.

Database Buffer Cache

The database buffer cache is the portion of the SGA that holds copies of data blocks read from data files. All user processes concurrently connected to the instance share access to the database buffer cache. The first time an Oracle 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 data file 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.

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 server 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 online redo log file (or group of files) on disk.

Shared Pool

The shared pool portion of the SGA contains three major areas: library cache, dictionary cache, buffers for parallel execution messages, and control structures.

Library Cache

The library cache includes the shared SQL areas, private SQL areas (in the case of a multiple transaction server), 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

A shared SQL area contains the parse tree and execution plan for a given SQL statement. Oracle 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 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 can de allocate 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 de-allocates a shared SQL area, the associated SQL statement must be reparsed and reassigned to another shared SQL area at its next execution.

PL/SQL Program Units and the Shared Pool

Oracle processes PL/SQL program units (procedures, functions, packages, anonymous blocks, and database triggers) much the same way it processes individual SQL statements. Oracle allocates a shared area to hold the parsed, compiled form of a program unit. Oracle 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.

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 accesses the data dictionary frequently during SQL statement parsing. This access is essential to the continuing operation of Oracle.

The data dictionary is accessed so often by Oracle 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 user processes share these two caches for access to data dictionary information.

Program Global Areas (PGA) Overview

A program global area (PGA) is a memory region, which contains data and control information for a server process. It is a non-shared memory created by Oracle when a server process is started. Access to it is exclusive to that server process and is read and written only by Oracle code acting on behalf of it.

The PGA holds information like sorting data, cursors and so on.

Introduction to Processes

All connected Oracle 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 pre-compiler program) or an Oracle tool (such as SQL*Plus), which issues SQL statements to an Oracle database.

Oracle server code: Each user has some Oracle server 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.

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 instance performs a specific job. By dividing the work of Oracle 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 system can be categorized into two major groups:
- User processes run the application or Oracle tool code.
- Oracle processes run the Oracle server code. They include server processes and background processes.

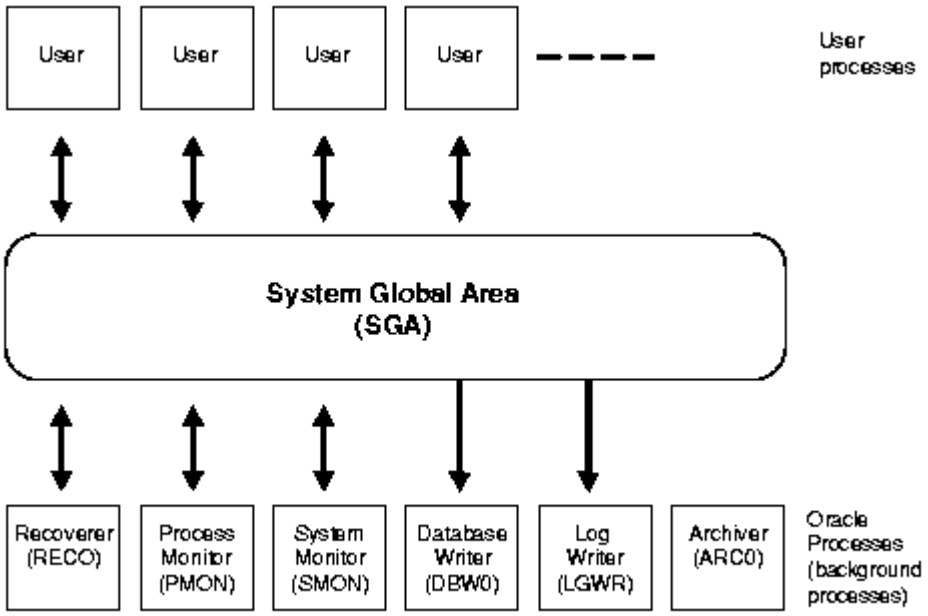
The process structure varies for different Oracle configurations, depending on the operating system and the choice of Oracle options. The code for connected users can be configured as a dedicated server or a shared server.

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 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 server code. Each server process that runs Oracle server code (a shared server process) can serve multiple user processes.

Figure 5 illustrates a dedicated server configuration. Each connected user has a separate user process, and several background processes run Oracle.

Figure 5: An Oracle Instance



User Processes Overview

When a user runs an application program (such as a Pro*C program) or an Oracle tool (such as Enterprise Manager or SQL*Plus), Oracle 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 instance. A communication pathway is established using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle) or network software (when different computers run the database application and Oracle, and communicate through a network).

A session is a specific connection of a user to an Oracle instance through a user process. For example, when a user starts SQL*Plus, the user must provide a valid username 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 user using the same username. For example, a user with the username/password of SCOTT/TIGER can connect to the same Oracle instance several times.

In configurations without the shared server, Oracle creates a server process on behalf of each user session. However, with the shared server, many user sessions can share a single server process.

Oracle Processes Overview

This section describes the two types of processes that run the Oracle server code (server processes and background processes). It also describes the trace files and alert file, which record database events for the Oracle processes.

Server Processes

Oracle creates server processes to handle the requests of user processes connected to the instance. In some situations when the application and Oracle operate on the same machine, 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 operate on different machines, a user process always communicates with Oracle 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

Background Processes

To maximize performance and accommodate many users, a multi process Oracle system uses some additional Oracle processes called background processes.

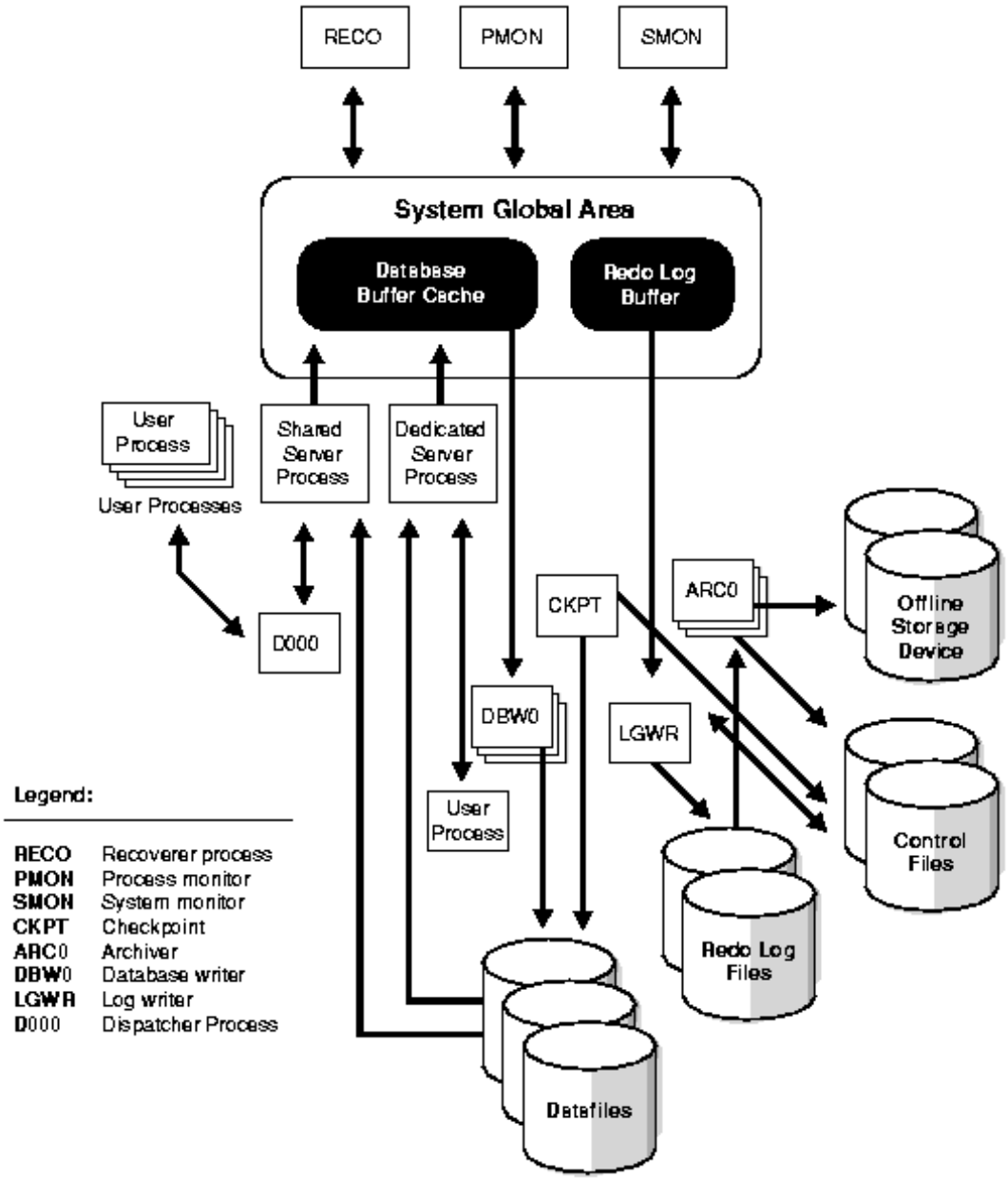
An Oracle instance can have many background processes; not all are always present.

The background processes in an Oracle instance include the following:

- Database Writer Process (DBWn)
- Log Writer Process (LGWR)
- Checkpoint Process (CKPT)
- System Monitor Process (SMON)
- Process Monitor Process (PMON)
- Recoverer Process (RECO)
- Job Queue Processes
- Archiver Processes (ARCn)
- Queue Monitor Processes (QMNn) (RECO)

On many operating systems, background processes are created automatically when an instance is started.

The following figure illustrates how each background process interacts with the different parts of an Oracle database, and the rest of this section describes each process.



Database Writer Process (DBWn)

The database writer process (DBWn) writes the contents of buffers to datafiles. The DBWn 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 DBWn 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 DBWn 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. DBWn manages the buffer cache so that user processes can always find free buffers.

By writing cold, dirty buffers to disk, DBWn 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.

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.

Checkpoint Process (CKPT)

When a checkpoint occurs, Oracle 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; DBWn always performs that work.

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.

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. Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

Archiver Processes (ARCn)

The archiver process (ARCn) copies online redo log files to a designated storage device after a log switch has occurred. ARCn processes are present only when the database is in ARCHIVELOG mode, and automatic archiving is enabled.

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.

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 instance.

Introduction to SQL * PLUS

SQL * PLUS is an Oracle tool. It Recognizes and submits SQL statements residing in the SQL buffer (memory area) for execution to the Oracle server. It has its own sets of commands.SQL * PLUS can be used to format query results. It has Capability to access local and remote databases. With the help of SQL * PLUS we Can control environmental settings

Categories of SQL * PLUS statements

- Environment: Affects the general behavior of the SQL statements for the session.
- Format: Formats query results.
- File Manipulation: saves loads and runs script files.
- Execution: Sends SQL statements from SQL buffer to the Oracle server
- Edit: Modifies SQL statements in the buffer
- Interaction: Enables creation and passing of variables to SQL statements, print variable values on the screen
- Miscellaneous: Commands to connect to the database, display table structures and column definitions

SQL *PLUS Editor

- SQL *PLUS does not have a built-in editor
- EDIT (ED) command of SQL *PLUS enables you to invoke editor
- The editor can be used to edit the contents of SQL buffer
- Upon invoking the editor, the contents of the buffer are written into a file called afiedt.buf
- The editor is not invoked when the buffer is empty

SQL *PLUS Edit Commands

- SAVE or SAV: stores the contents of the buffer in an operating system file.
- @ or RUN <file name>: Executing the contents of a script file.
- GET <file name>: Write the contents of the previously stored file in the SQL buffer

LOGGING to the database with the help of SQL *PLUS

- Ensure the Oracle Server has been installed.
- Ensure SQL *PLUS has been installed in the client machine.
- The server and the client are connected with the help of a network.

- You have got a valid username, password and host string

Introduction to iSQL * PLUS

iSQL * PLUS is an Oracle tool that recognizes and submits SQL statements to the Oracle server for execution and contains its own command language.

Features of iSQL * PLUS

- Accessed from a browser
- Accepts ad hoc entry of statements
- Provides online editing for Modifying SQL Statements
- Controls environmental setting
- Formats query result into a basic report.
- Access local and remote databases

Summary

- A database-management system (DBMS) consists of a collection of interrelated data and a set of programs to access those data.
- DBMS acts as an interface between application programs and physical data files.
- An Oracle server is a database management system that provides an open, comprehensive, integrated approach to information management.
- Oracle database is a collection of data that is treated as a unit which consists of three file types.

Test Your Understanding

1. Review architectural components.
2. What are the structures involved in connecting a user to an Oracle Instance.
3. What are the different background processes of Oracle?

Session 03: Basic Element of SQL

Learning Objectives

After completing this session, you will be able to:

- Explain the various data types and literals
- Describe the NULL and comments

Data Type

The data type of a value associates a fixed set of properties with the value. These properties cause Oracle to treat values of one data type differently from values of another. A data type is either scalar or non scalar. A scalar type contains an atomic value, whereas a non-scalar (sometimes called a "collection") contains a set of values.

Data type can be classified as Oracle built- in data types and user defined data types.

Oracle built-in Data types:

Oracle built-in data types can be further classified into following types.

- Character data types
- Numeric data types.
- Date data types.
- BLOB data type.
- CLOB data type.

Character Data Types

Character data types: Character data types store alphanumeric data. These data types are used for character data.

- CHAR Datatype
- VARCHAR2 Datatype
- NCHAR Datatype
- NVARCHAR2 Datatype

CHAR Data type: The CHAR data type specifies a fixed-length character string. Oracle ensures that all values stored in a CHAR column have the length specified by size. If you insert a value that is shorter than the column length, then Oracle blank-pads the value to column length. If you try to insert a value that is too long for the column, then Oracle returns an error.

Example:

```
CHAR (10)
```

The default length for a CHAR column is 1 byte and the maximum allowed is 2000 bytes

Varchar2 Data type: The VARCHAR2 data type specifies a variable-length character string.

Example:

```
Varchar2 (10)
```

The maximum length of VARCHAR2 data is 4000

NCHAR Data type: The NCHAR data type is a Unicode-only data type. When you create a table with an NCHAR column, you define the column length in characters. You define the national character set when you create your database. The maximum length of a column is determined by the national character set definition. Width specifications of character data type NCHAR refer to the number of characters. The maximum column size allowed is 2000 bytes.

Example:

```
NCHAR (10)
```

NVARCHAR2 Data type: The NVARCHAR2 data type is a Unicode-only data type. When you create a table with an NVARCHAR2 column, you supply the maximum number of characters it can hold. Oracle subsequently stores each value in the column exactly as you specify it, provided the value does not exceed the maximum length of the column. The maximum length of the column is determined by the national character set definition. Width specifications of character data type NVARCHAR2 refer to the number of characters. The maximum column size allowed is 4000 bytes.

Example: `NVARCHAR2 (10)`

Numeric Data Types

The NUMBER data type stores zero as well as positive and negative fixed numbers with absolute values from 1.0 x 10-130 to (but not including) 1.0 x 10126. If you specify an arithmetic expression whose value has an absolute value greater than or equal to 1.0 x10126, then Oracle returns an error. Each NUMBER value requires from 1 to 22 bytes.

Example:

```
Number (p, s)   Number (6, 2)
```

Where:

- p is the precision, or the total number of significant decimal digits
- s is the scale, or the number of digits from the decimal point to the least significant digit

Date Data Types

Date Data types: The date time data types are

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE

Date: The DATE data type stores date and time information. Although date and time information can be represented in both character and number data types, the DATE data type has special associated properties. For each DATE value, Oracle stores the following information: century, year, month, date, hour, minute, and second.

The following is an example of an ANSI date literal:

```
DATE '1998-12-25'
```

Time Stamp: The TIMESTAMP data type is an extension of the DATE data type. It stores year, month, day, hour, minute, and second values. It also stores fractional seconds, which are not stored by the DATE data type.

Specify the TIMESTAMP data type as follows:

```
TIMESTAMP [(fractional_seconds_precision)]
```

fractional_seconds_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field. It can be a number in the range 0 to 9. The default is 6.For example, '26-JUN-02 09:39:16.78' shows 16.78 seconds. The fractional seconds precision is 2 because there are 2 digits in '78'.

You can specify the TIMESTAMP literal in a format like the following:

TIMESTAMP 'YYYY-MM-DD HH24:MI: SS.FF' Using the example format, specify TIMESTAMP as a literal as follows: TIMESTAMP '1997-01-31 09:26:50.12‘

TIME STAMP WITH TIME ZONE: TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time zone offset or time zone region name in its value. The time zone offset is the difference (in hours and minutes) between local time and UTC (Coordinated Universal Time, formerly Greenwich Mean Time).

Specify the TIMESTAMP WITH TIME ZONE datatype as follows:

```
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE
```

fractional_seconds_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

You can specify TIMESTAMP WITH TIME ZONE as a literal as follows:

```
TIMESTAMP '1997-01-31 09:26:56.66 +02:00'
```

The following expression specifies US/Pacific for the time zone region:

```
TIMESTAMP '1999-01-15 8:00:00 US/Pacific'
```

TIMESTAMP WITH LOCAL TIME ZONE: Specify the TIMESTAMP WITH LOCAL TIME ZONE datatype as follows:

```
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE
```

fractional_seconds_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

The default date format for TIMESTAMP WITH LOCAL TIME ZONE is determined by the value of the NLS_TIMESTAMP_FORMAT initialization parameter.

BLOB Data type: The BLOB datatype stores unstructured binary data in the database. BLOBs can store up to 8 terabytes of binary data.

CLOB Data type: The CLOB datatype store up to 8 terabytes of character data in the database.

Literals

Literals refer to a fixed data value. Literals can be classified as following types.

- Text literals
- Numeric literals
- Datetime literals

Text literals: Text, character, and string literals are always surrounded by single quotation marks. A text literal can have a maximum length of 4000 bytes.

```
Example: ‘Hello‘, 'ORACLE.dbs‘ ,'Jackie''s raincoat'
'09-MAR-98‘, N'nchar literal
```

Numeric Literals: Numeric literals are use to specify fixed and floating point numbers. Numeric literals can be classified as Integer literal and floating point literal.

An integer can store a maximum of 38 digits of precision.

Here are some valid integers:7 ,+255

Here are some valid NUMBER literals: 25,+6.34,0.5,25e-03,-1

Here are some valid floating-point number literals:25f ,+6.34F,0.5d,-1D

Date literals: You can specify a DATE value as a string literal, or you can convert a character or numeric value to a date value with the TO_DATE function.

Example:

```
DATE '1998-12-25'

TO_DATE ('98-DEC-25 17:30','YY-MON-DD HH24:MI')

TIMESTAMP '1997-01-31 09:26:50.124'

TIMESTAMP '1997-01-31 09:26:56.66 +02:00'

TIMESTAMP '1999-04-15 8:00:00 US/Pacific'
```

NULL Values

What is a Null value?

If a column in a row has no value, then the column is said to be **null**, or to contain null. Nulls can appear in columns of any datatype that are not restricted by `NOT NULL` or `PRIMARY KEY` integrity constraints. Zero is not equivalent to null.

What does the NULL value do?

Although `NULL` value means inaction, it can, however, improve readability. In a construct allowing alternative actions, the `NULL` statement serves as a placeholder. It tells Readers that the associated alternative has not been overlooked, but that indeed no action is necessary. Null represents a lack of data; a null cannot be equal or unequal to any value or to another null. Nulls value can be tested by using the comparison conditions `IS NULL` and `IS NOT NULL`.

An example of the usage of NULL value

You can use the NVL function to return a value when a null occurs

Example: NVL (Rate_of_interest, 0) returns 0 if Rate_of_interest is null or the value of Rate_of_interest if it is not null.

Comments

Comments can make your application easier for you to read and maintain. For example, you can include a comment in a statement that describes the purpose of the statement within your application

You can include a comment in a statement in two ways:

- Begin the comment with a slash and an asterisk (/*). Proceed with the text of the comment. This text can span multiple lines. End the comment with an asterisk and a slash (*/). The opening and terminating characters need not be separated from the text by a space or a line break.
- Begin the comment with -- (two hyphens). Proceed with the text of the comment. This text cannot extend to a new line. End the comment with a line break.

Example:

```
SELECT  loan_amount  --Amount in $
                FROM
                         daily_pipeline_loans_fact
         /* select all the loans where loan amount is exceed
             10000*/
               WHERE loan_amount >10000
```

Summary

- Data type can be classified as Oracle built- in data types and user defined data types.
- Oracle built-in data types can be further classified as character data types, numeric data types, date data types, BLOB data type, CLOB data type.
- Character data types store alphanumeric data.
- The DATE data type stores date and time information.
- Literals can be classified as text literals, numeric literals, datetime literals.

Test your Understanding

1. What are the different Oracle built-in data types?
2. What are the different character data types provide by Oracle?
3. What is the maximum length of Varchar2?
4. What is the difference between char and Varchar2?
5. What is the difference between char and nchar?
6. What is the different date data types provide by oracle?
7. Two null values are equal. True or False.
8. Null equivalent to zero. True or False.
9. What is the use of ISNULL and IS NOT NULL function?

Session 04: SQL Statements

Learning Objectives

After completing this session, you will be able to:

- Write SQL Statements
- Create tables
- Maintain tables
- `INSERT` into table
- `UPDATE` rows from table
- `DELETE` rows from table
- Grant rights to Oracle database
- Revoke rights to Oracle database

Understand the features of SQL

What is SQL?

The Structured Query Language (SQL) is a set of statements with which all programs and users access data in an Oracle Database.

Who Developed SQL and When?

IBM developed the SQL, popularly pronounced as “SEQUEL” and implemented it in 1979. Actually, in doing so, IBM implemented the model on Relational Database Management Systems (RDBMS) developed by Dr. E.F. Codd. Today, SQL is the most popular RDBMS language.

What can SQL do?

With SQL*Plus, you can execute SQL commands and PL/SQL blocks, additionally you can perform the following tasks:

- Enter, edit, store, retrieve and run SQL commands and PL/SQL blocks
- Format, perform calculations on, store, print and create web output of query results
- List column definitions for any table
- Access and copy data between SQL databases
- Send messages to and accept responses from an end user
- Perform database administration

SQL Statements

SQL statements can be classified into following categories:

Statements	Description
SELECT	Retrieves data from the database.
INSERT UPDATE DELETE MERGE	Enter new rows, changes existing rows, delete unwanted rows. Collectively known as data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Set up, change, and remove data structure from the tables. Collectively known as data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Manage the change made by DML statements. Collectively known as transaction control language(TCL)
GRANT REVOKE	Grant or remove access to both database and its objects. Collectively Known as data control language (DCL)

Data Definition Language (DDL)

Data Definition Language is a set of SQL commands used to create, modify and delete database structures.

- The CREATE TABLE is used to create and define a new relational table
- The DROP TABLE is used to delete a table and all data within the specified table
- The ALTER TABLE is used to change an existing table definition
  - Cannot delete an existing column
  - Cannot change an existing table constraint definition
- The TRUNCATE TABLE is used to delete all the rows within the specified table
- Rename table is used to change the name of an existing tables to the specified name.

CREATE TABLE: To create a table the user must have Create table privileges and a storage area in which to create the table

Rules for naming a table:

- The name must begin with a letter, A-Z or a-z.
- May contain letters, numeric and special character _(underscore). The $ and # are also legal but discouraged;
- The name is the same whether upper or lower case letters are used
- It may be up to 30 char length
- The name must not duplicate the name of any other object in your account.

- The name must not be a sequel reserved word

Syntax:

```
CREATE TABLE [SCHEMA.]Tablename (column_name data type [DEFAULT expr],
column_name data type [default expr], …)
```

In syntax

- SCHEMA is same as the owner's name.
- Table name is the name of the table.
- DEFAULT expr specifies a default value
- Column_name is the name of the column.
- Data type is the column's data type and its length.

Example:

```
CREATE TABLE product_dim
                                          (product_id       NUMBER(5),
                                           product_name VARCHAR2(10))
```

Create table with constraints:

Constraint is a rule that restricts the values for a column on which it is defined. Constraints check data as it is entered or updated in the database and prevent data that does not conform to the constraint's rule from being entered.

Constraints are classified into two types:

- Column level constraints.
- Table level constraints.

NOT NULL: NOT NULL specifies that a column must have some value.

NULL: NULL (default) allows NULL values in the column**.**

DEFAULT: Specifies some default value if no value entered during INSERT

UNIQUE: Specifies that column(s) must have unique values

PRIMARY KEY: Specifies that column(s) must have unique values. Index is automatically generated for column

FOREIGN KEY : Specifies that column(s) are defined primary key in another table. Used for referential uniqueness of parent table. Index is automatically generated for column.

CHECK : Applies a condition to an input column value

DISABLE: Suffix DISABLE to any other constraint to make Oracle ignore the constraint, the constraint will still be available to applications / tools and you can enable the constraint later if required

Example:

```
CREATE TABLE DAILY_PIPELINE_LOANS_FACT
(LOANNO     NUMBER(4) CONSTTRAINT DLY_PRIM PRIMARY KEY,
LOAN_STATUS VARCHAR2(10)      CONSTRAINT LOAN_STATUS_CONS
                  CHECK(LOAN_STATUS=UPPER(LOAN_STATUS)),
 LOAN_AMOUNT      NUMBER(10) AMOUNT_CONS NOT NULL,
 UPDATE_DATE      DATE DEFAULT SYSDATE,
 PRODUCT_KEY      VARCHAR2(5) CONSTRAINT DLY_PRODUCT
                              FOREIGN  KEY(PRODUCT_KEY)
                  REFERENCES PRODUCT_DIM(PRODUCT_KEY))
```

ALTER TABLE: Alter table statement can be used to perform the followings task.

- Add a new column.
- Modify an existing Column.
- Define a default value for the new column.
- Drop a column

Example:

Add a new column lien_code to existing table.

```
ALTER TABLE DAILY_PIPELINE_LOAN_FACT  ADD (lien_code varchar2(5));
```

Change the existing columns data type and size.

```
ALTER TABLE DAILY_PIPELINE_LOAN_FACT MODIFY (lien_code Varchar2 (10));
```

Dropping a column:

```
ALTER TABLE DAILY_PIPELINE_LOAN_FACT  DROP COLUMN lien_code
```

SET UNUSED OPTION: The set unused option marks one or more column as unused so that they can be dropped when required.

Example: `ALTER TABLE DAILY_PIPELINE_LOAN_FACT SET UNUSED (lien_code)`

Dropping a table: While we dropped a table

- All data and structure in the table deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- You can not rollback the drop table statement.
- Only the creator of the table or the user with drop any table privilege can drop a table

Example:

```
DROP TABLE DAILY_PIPELINE_LOAN_FACT
```

Rename table: You can use `RENAME TABLE` keyword to change the name of the table. To change the name of the table, you must be owner of the table.

Example:

```
RENAME TABLE   Daily_pipeline_loan_fact to Dly_pipeline_loan_fct
```

Truncating a table

- Remove all rows from the table.
- Release the storage space used by the table.
- You can not rollback rows removal when using TRUNCATE.

Example:

```
TRUNCATE TABLE daily_pipeline_loan_fact
```

Data Manipulation Language (DML)

A DML statement is executed When You

- Add new rows to a table.
- Modifying existing rows in a table.
- Removing existing rows from a table.

Adding a new row to a table: You can insert new rows in a table by using `INSERT` statement.

```
Syntax:
INSERT INTO table_name [(Column),(column)….]
VALUES          (value [, value]…);
```

In the syntax:

- Table_name is the name of the table
- Column is the name of the column in the table to populate
- Value is the corresponding value for the column.

Example:

```
INSERT INTO DAILY_PIPELINE_LOAN_FACT
(LOANNO,
LOAN_STATUS,
LOAN_AMOUNT)
VALUES (1001,
 ‘FUNDED’,
1000)
```

If you are inserting a new row that contains values for each column, the column list is not required in the INSERT clause .However, if you do not use the column list, the value must be listed according to the default orders of the columns in the table, and a value must be provided for each column.

Use and substitution in a SQL statement to prompt for values

```
INSERT INTO DAILY_PIPELINE_LOAN_FACT
                    (LOANNO,
                      LOAN_STATUS,
                      LOAN_AMOUNT)
           VALUES (&LOANNO,
                        '& LOAN_STATUS',
                         & LOAN_AMOUNT)
```

Copying rows from another table

```
INSERT INTO COPY_DLY_PIPELINE_LOAN_FACT
SELECT * FROM DAILY_PIPELINE_LOAN_FACT
```

DELETE Statement: You can remove existing rows from a table by using the delete statement.

```
DELETE [FROM] Table
[WHERE condition]
```

Table is the name of the table.

Condition identifies the rows to be deleted.

Example:

```
DELETE FROM daily_pipeline_loan_fact WHERE loan_no='1001' ;
```

MERGE Statement:

Using the MERGE statements you can update or insert a row conditionally into a table, thus avoiding multiple Update statements. The decision whether to insert or update into the target table is based on a condition in the ON clause.

Because the MERGE command include both INSERT and UPDATE commands , you need to have both INSERT and UPDATE privilege on the target table and SELECT privilege on the source table .

Syntax:

```
MERGE INTO Table_name AS table_alias
USING (table|view|subqueries) AS alias
ON (Join condition)
WHEN MATCHED THEN
UPDATE SET
col1=val1
col2=val2
WHEN NOT MATCHED THEN
INSERT (column_list)
```

```
VALUES (Column_values);
```

In the Syntax:

`INTO` clause specifies the target table you are updating or inserting.

`USING` clause identifies the source of the data to be updated or inserted; can be a table, view, or subquery.

`ON` clause the condition upon which the `MERGE` operation either updates or inserts

`WHEN MATCHED`

`WHEN NOT MATCHED` Instruct the server how to respond to the result of the join condition.

Example:

```
MERGE INTO copy_dly_pipeline_loan_fact  AS  c
USING daily_pipeline_loan_fact d
ON (d.loanno=c.loanno)
WHEN MATCHED THEN
UPDATE SET
         c.laon_status=d.loan_status
         c.loan_amount=d.loan_amount
         c.update_date=d.update_date
         c.product_key=d.product_key

WHEN NOT MATCHED THEN
INSERT VALUES(d. laon_status,
                        d.loan_amount,
                         d.update_date,
                         d.product_key)
```

TCL Statements

Transaction control language (TCL) manages the change made by DML statements.

With commit or rollback statements, you can do the following:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logical related transaction.
- An automatic `COMMIT` occurs when:
  - DDL statement is issued
  - DCL statement is issued
  - Normal exit from SQL*PLUS without issuing explicit `COMMIT` or `ROLLBACK`
- An automatic `ROLLBACK` occurs under an abnormal termination of SQL *PLUS or due to system failure

State of data before commit or Rollback

- The previous state of data can be recovered.
- The current user can review the results of the DML operations by using the SELECT statement.
- Other users cannot view the results of the DML statements by the current user.
- The affected rows are locked; other users cannot change the data within the affected row.

State of data after commit

- Data changes are made permanent in the database.
- The previous state of data is permanently lost.
- All users can view the results.
- All save points are erased.

State of data after Rollback

- Discard all pending changes by suing the ROLLBACK statement.
- Data changes are undone
- Previous state of the data is released

DCL Statements

You can use GRANT or REVOKE to Grant or remove access to both database and its objects. Collectively known as data control language (DCL)

Granting system privileges

GRANT create table, create view, Create sequence TO Scott;

Role: A role is a named group of related privileges that can de grant to a user.

Create a Role

```
CREATE ROLE manager;
```

Grant privileges to a role

```
GRANT create table, create view, create sequence TO manager;
```

Grant a role to users

```
GRANT manager to Scott, scott1;
```

Granting object privileges

```
GRANT  SELECT ON daily_pipeline_loan_fact TO scott1;
```

Revoke object privileges: Use the revoke statement to revoke all the privileges granted to other user.

```
REVOKE SELECT ON       daily_pipeline_loan_fact TO        scott1;
```

SELECT Statements

The SELECT statement is an SQL statement that specifies which rows and columns to fetch from one or more tables or views. The SELECT statement is very helpful in finding filtered records from a database. If the database happens to be very large, the SELECT statement, used with the right parameters, can work wonders in finding the right information.

The following are the features of a SELECT statement:

- It is used for retrieving data from the database either selectively or collectively.
- Column names can be specified or * may be specified for displaying all columns.
- Calculated columns can be displayed.
- Duplication can be removed.
- Sorting of rows is possible.
- Summary information can be displayed
- SQL statement includes a:
  - SELECT clause, which lists the columns to be displayed
  - FROM clause, which specifies the table involved
  - [WHERE] clause, which limits the number of rows returned
  - [GROUP BY] clause, which groups the rows based on specified columns
  - [HAVING] clause, which specifies the table involved
  - [ORDER BY] clause, which sorts the rows based on specified columns

Examples:

To select all the columns:

```
SELECT * FROM  daily_pipeline_loan_fact;
```

To select specific columns:

```
SELECT loanno, loan_amount FROM    daily_pipeline_loan_fact;
```

The DISTINCT clause:

The default output of a SELECT query is all rows including duplicate rows. The DISTINCT keyword is used to eliminate duplicate rows from the output.

Example:

```
SELECT DISTINCT loan_status FROM  daily_pipeline_loan_fact;
```

The Where clause:

- Applies conditions to filter rows
- Appears immediately after the SELECT and FROM clause
- Does not allow alias names
- Works on row levels

Syntax:

```
SELECT <COLUMN NAME(S)>
FROM    <Table name>
WHERE <CONDITION>
```

Example:

```
SELECT  loanno
FROM daily_pipeline_loan_fact
WHERE loan amount >10000
```

Summary

- SELECT statement is used to retrieve data from the database either selectively or collectively.
- The WHERE clause is used to apply conditions to filter rows.
- Oracle provides DDL statements for creating, altering, dropping, and truncating tables.
- Oracle provides DML statements to insert, update, and delete rows.
- Constraint is a rule that restricts the values for a column on which it is defined.
- Oracle provides TCL statements to ensure data consistency.
- Grant and revoke privileges and role using DCL statement

Test Your Understanding

1. What are the different types of SQL statement?
2. What is a constraint?
3. What is the difference between unique key and primary key?
4. What is the use of MERGE statement?
5. What is save point?
6. What is a role?
7. Which statements are auto commit?
8. When there will be an automatic rollback?

Session 06: SQL Operators

Learning Objectives

After completing this session, you will be able to:

- Write query using arithmetic and concatenation operators
- Write query using comparison and logical operators
- Write query using set operators and SQL operators

SQL Operators

The two general classes of SQL operators are:

- **Unary:** A unary operator operates on only one operand. A unary operator typically appears With its operand in this format:
  - Operator operand
- **Binary:** A binary operator operates on two operands. A binary operator appears With its operands in this format:
  - Operand1 operator operand2

Arithmetic Operators

Oracle allows arithmetic operations to be performed on the data stored in tables.
The following are the arithmetic operators (in their order of precedence):

- Multiplication (*)
- Division (/)
- Addition (+)
- Subtraction (-)

Parenthesis () are used to enforce prioritized valuation.

Example:

```
SELECT  Loan_amount+300
FROM    daily_pipeline_loan_fact
```

Operators Precedence:

- Multiplication and division takes priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- You can use parentheses to force the expression within parentheses to be evaluated first.

Concatenation Operator

The concatenation operator manipulates character strings and CLOB data.

Operator ||

Example:

```
SELECT   area_name||region_name
                FROM     operation_dim
```

Comparison Operators

Comparison operators: Relational Operators will test the following conditions:

Operator	Meaning
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
! =	Not equal to

Logical Operators

Logical operators will test the following conditions:

Operator	Meaning
AND	Returns TRUE if both conditions are TRUE
OR	Returns TRUE if either condition is TRUE
NOT	Returns TRUE if the condition is FALSE

Parenthesis can be used to override precedence.

Example:

```
SELECT laonno,loan_amount
FROM   daily_pipeline_loan_fact
WHERE (loan_status = 'FUNDED' OR loan_status = 'SHIPPED')
AND    loan_amount >= 1000;
```

SQL Operators

There are four SQL operators as follows:

Operator	Meaning
between..and..	Between two values
IN(list)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

The BETWEEN operator:

- Searches for a specific range.
- Includes both lower and higher limits
- Eliminates multiple AND operator in a WHERE statement.

Example:

```
SELECT      laonno,loan_amount,loan_staus
FROM              daily_pipeline_loan_fact
WHERE       loan_amount between 10000 and 20000
```

The IN operator:

- Searches in a specified list of values
- Reduces the number of OR operators in a SELECT statement

Example:

```
SELECT      laonno,loan_amount,loan_staus
FROM              daily_pipeline_loan_fact
WHERE       loan_status IN ('FUNDED','SHIPPED');
```

Wild Card Matching:

- The LIKE operator is used for wild card matching.
- _ is used for a single character and % for multiple characters or no characters.

Example:

```
SELECT       laonno,loan_amount,loan_staus
FROM               daily_pipeline_loan_fact
WHERE       loan_status LIKE ('F%');
```

Checking for NULL Values

- A NULL value cannot be checked with any comparison operators
- IS NULL is used to check for a NULL value.

- To obtain the reverse effect, the IS NOT NULL operator is used.

Example:

```
SELECT       laonno,loan_amount,loan_staus
FROM               daily_pipeline_loan_fact
WHERE       loan_status IS NOT NULL;
```

SET Operators

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries.

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows selected by the first query but not the second

The UNION SET operator: The UNION operator returns all rows selected by the either query.

- The number of columns and data type of the columns being selected must be identical in all the SELECT statements used in the query.
- UNION operates over all of the columns being selected.
- Null values are not ignored during the duplicate checking.
- By default the output is stored in the ascending order of the first column of the select clause.

Example:

```
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact
UNION
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact_hist
```

The UNION ALL operators: use the UNION ALL operator to returns all rows from the multiple queries. Unlike UNION, duplicate rows are not eliminated and the output is not stored by default. The DISTINCT keyword cannot be used.

Example:

```
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact
UNION ALL
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact_hist
```

The INTERSECT Operator : Use the INTERSECT Operator to return all rows common to multiple queries .The number of columns and data type of the columns being selected must be identical in all the SELECT statements used in the query .Reversing the order of intersected table does not alter the result .It does not ignore null values .

Example:

```
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact
INTERSECT
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact_hist
```

The MINUS Operator: Use the MINUS Operator rows return by the first query that are not present in the second query. It gives the result of first select statement MINUS The result of second select statement .The number of columns and data type of the columns being selected must be identical in all the SELECT statements used in the query .All of the columns in the where clause must be in the select clause for the minus operator to work.

Example:

```
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact
MINUS
SELECT loanno,loan_amount,loan_status
FROM    daily_pipeline_loan_fact_hist
```

Operator Precedence

Rule of precedence: Override rules of precedence by using parentheses.

Order evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison operators
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT logical condition
7	AND logical condition
8	OR logical condition

Sorting Rows

The following are the features of the ORDER BY clause:

- ORDER BY clause is used to arrange the output
- Default order is ascending (ASC)
- For arranging in descending order, DESC is used
- ORDER BY should be the last clause in a SELECT statement
- Can be arranged in multiple columns

Example:

```
SELECT loanno,loan_amount,loan_status
FROM   daily_pipeline_loan_fact
ORDER BY loanno DESC;
DESC specifies a descending order
```

Default order is ascending (ASC) in the following manner:

- Numeric values lowest first
- Date values earliest first
- Character values alphabetically

Ordering by many columns:

- In the ORDER BY clause, specify the columns to order by, separated by commas. If any or all are to be reversed, specify DESC after any or each column
- Null values are displayed last for ascending sequences, and are reported first when rows are sorted in descending order.

Example:

```
SELECT loanno,loan_amount,loan_status
FROM   daily_pipeline_loan_fact
ORDER BY loanno,loan_amount DESC;
```

Summary

- Oracle allows the use of Arithmetic, Comparison and Logical Operators in SQL queries
- The concatenation operator manipulates character strings and CLOB data
- Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries.
- Override rules of precedence by using parentheses
- Rows retrieved from the database can be sorted using Order by clause
- Order by clause should be the last clause in the Select statement

Test Your Understanding

1. What are the different logical operators?
2. What are the different comparison operators?
3. What are the different SET operators?
4. What is difference between UNION and UNION ALL?
5. What are the different SQL operators?

Session 08: SQL Function

Learning Objectives

After completing this session, you will be able to:

- Write single row functions
- Write aggregate functions
- Write Analytical functions

Understand Function

Functions are similar to operators in that they manipulate data items and return a result. Functions differ from operators in the format of their arguments. This format enables them to operate on zero, one, two, or more arguments:

```
function(argument, argument, ...)
```

Functions can:

- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

The following are the types of functions:

- Single-row functions: Operate on single-rows only, and return one result per row
- Multiple-row functions: Operate on groups of rows, and produce one result per group of rows

Single – Row Functions

Single row functions Operate on single-rows only, and return one result per row. They can be used in SELECT, WHERE and ORDER BY clause. Single row functions can manipulate data item and may change the data type. They can be nested.

Single row functions are of the following types:

- Character
- Number
- Date
- Conversion
- General functions

Character Function

Single row character function accepts character data as input and can return both character and numeric value.

Character function can be divided into following:

- Case manipulation function
- Character manipulation function

Case manipulation function

- `LOWER (column|expression):` Converts alpha character values to lowercase.
- `UPPER (column|expression):` Converts alpha character values to uppercase.
- `INITCAP (column|expression):` Converts alpha character values to upper case for the first letter of each word and lower case for all other letters.

Example:

```
SELECT LOWER('Hello') FROM DUAL
Result : hello
SELECT  UPPER('Hello') FROM DUAL
Result : HELLO
SELECT  INITCAP ('hello') FROM DUAL
Result : Hello
```

Character manipulation function:

Function	Purpose
`CONCAT(column1|expression1, Column2|expression2)`	Concatenates the first character value to the second character value.
`SUBSTR(column|expression, m, [n])`	Returns the specified characters from the character starting at position m and retrieving the next n characters. (If n not specified retrieves to end of field, if m negative counts backward from end.
`LENGTH(column|expression)`	Returns the number of characters in a value.
`INSTR(column|expression, m, [n])`	Returns the numeric position of a named string
`RPAD(column|expression, n,'string')` `LPAD(column|expression, n,'string')`	Pads the character value right justified to a total width of n character position Pads the character value left justified to a total width of n character position
`TRIM(leading|trailing|both, Trim_character FROM trim_source)`	Enable you to trim heading or trailing (or both) from a character string

Function	Purpose
REPLACE(text,search_string,replacement_string)	Search a text expression for a character string, and if found replace it with a specified replacement string.
CHR(number)	Returns the character based on a ASCII number.
ASCII(character)	Returns the ASCII number based on a character.

Example:

```
Function                                Result
CONCAT ('Hello','World')        HelloWorld
SUBSTR('HelloWorld',1,5)        Hello
LENGTH('HelloWorld')                  10
INSTR('HelloWorld','W')               6
LPAD(salary,10,*)               *****10000
RPAD(salary,10,*)                        10000*****
TRIM('H' FROM 'HelloWorld')     elloWorld
```

Numeric Functions

Number function accepts numeric input and returns numeric value.

Function	Purpose
ROUND(column|expression, n)	Rounds the column, expression, or value to decimal places. If n is omitted, no decimal places. If n is negative, numbers to left of the decimal point are rounded.
TRUNC(column|expression, n)	Truncates the column, expression or value to n decimal places. If n is omitted to no decimal places. If n is negative numbers to the left of the decimal point are truncated to 0
MOD(m, n)	Returns the remainder of m/n where m and n can be columns, expressions or values.

Example

```
SELECT ROUND(45.926,2) FROM DUAL ;
Result : 45.93
SELECT  TRUNC(45.926,2) FROM DUAL ;
Result : 45.92
SELECT  MOD(1600,300) FROM DUAL ;
Result : 100
```

Date Functions

Oracle stores dates in an internal numeric format, century, year, month, day, hours, minutes, seconds. The default date display is DD-MON-YY

Example:

```
to display the current date using the DUAL table
SQL> SELECT SYSDATE FROM SYS.DUAL
```

Function	Purpose
NEXT_DAY(date, 'day of week')	Finds the next date that is the specified day of the week. For example NEXT_DAY('15-OCT-98', 'SUNDAY') fids the first Sunday following October 15, 1998.
LAST_DAY(date)	Finds the date of the last day of the month that contains the date specified. For example LAST_DAY(SYSDATE) returns the date of the last day of the current month.
Add_Months(date, n)	Adds the specified number of calender months to the date, n can be negative (months are subtracted), but must bre an integer. For example ADD_MONTHS('15_-OCT-98', 2) returns 15_DEC-98.
MONTHS_BETWEEN(date1, date2)	Finds number of months between 2 dates by subtracting date 2 form date 1. Result can be negative (If date later than date 1) and includes fractional portions of months. For example MONTHS_BETWEEN('30-NOV-98', '15-NOV-98') returns .5.
ROUND(date, 'fmt')	Rounds the date to the unit listed in the format model. For example ROUND('20-OCT-98', 'MON') returns 01-NOV-98. If no format is specified, rouning is to the nearest day. ROUND(SYSDATE) returns the current date till noon and the next day after noon.
TRUNC(date, 'fmt')	Like round except truncates to the format unit specified. TRUNC('20-OCT-98) returns 01-OCT-98. TRUNC(SYSDATE) returns a value of midnight on the current date, the fractional time element is truncated.

Example :

```
SELECT  MONTHS_BETWEEN('01-SEP-07','11-JAN-07')
FROM    DUAL;
Result : 19.6774194
SELECT  ADD_MONTHS('11-JAN-07',6)
FROM    DUAL;
Result : 11-JUL-07
SELECT  NEXT_DAY('01-SEP-07', 'MONDAY')
FROM    DUAL;
Result : 08-SEP-07
SELECT  LAST_DAY('01-FEB-07')
FROM    DUAL
Result :28-FEB-07
```

```
Assume sysdate = '17-FEB-08'
```

```
SELECT ROUND(SYSDATE,'MONTH') FROM DUAL ;
Result : 01-FEB-08

SELECT ROUND(SYSDATE,'YEAR') FROM DUAL ;
Result : 01-JAN-08

SELECT  TRUNC(SYSDATE,'MONTH') FROM DUAL ;
Result : 01-FEB-08
SELECT  TRUNC(SYSDATE,'YEAR') FROM DUAL ;
Result : 01-JAN-08
```

Conversion functions:

Conversion can either be implicitly or explicitly. Oracle can implicitly convert from varchar2 or char to number, varchar2 or char to Date, Number to varchar2, and date to varchar2. Explicit conversion is required primarily for conversion and display of date/time data to character form and vice-versa.

Function	Purpose
`TO_CHAR(number|date,'fmt')`	Converts a number or date value (could be a column or expression) to a VARCHAR2 string with the format model fmt.
`TO_DATE(char,['fmt'])`	Converts a character string representing a data to a date value according to the fmt specified. If fmt omitted, format is DD-MON-YY.
`TO_NUMBER(char)`	Converts a character string containing digits to a number.

The following is an example of the TO_CHAR () function:

```
SQL> SELECT TO_CHAR (SYSDATE, 'dd-mon-yyyy hh:mi:ss'),
  2         TO_CHAR (SYSDATE, 'day-monthdd, yyyy'),
  3         TO_CHAR (SYSDATE, 'dd/mm/yy') from sys.dual;

TO_CHAR(SYSDATE,'DD-MON-YYYYHH:MI:SS')
--------------------------------------
TO_CHAR(SYSDATE,'DAY-MONTHDD,YYYY')
--------------------------------------
TO_CHAR(SYSDATE,'DD/MM/YY')
--------------------------------------
04-jun-1998 06:54:52
thursday -june     04, 1998
04/06/98
```

General Function

These functions work with any data type and pertain to the use of NULL values in the expression list.

NVL(expr1, expr2): Converts a null value to an actual value

NVL2(expr1, expr2, expr3): If expr1 is not null ,NVL2 returns expr2 . If expr1 is null ,NVL2 returns expr3.The argument expr1 can have any data type

NULLIF(expr1, expr2): Compares two expressions and returns null if they are equal , or the first expression if they are not equal.

COALESCE(expr1, expr2,…expr n): Returns the first not null expressions in the expression list.

Example:

```
SELECT  NVL(intrest_rate,1)
FROM     daily_pipeline_loan_fact

SELECT NVL2(total_amount,'loan_amount*intrest_rate',
                     loan_amount)
FROM     daily_pipeline_loan_fact

SELECT  laonno,
             COALESCE(loan_amount,intrest_rate,10)  amt
FROM daily_pipeline_loan_fact

SELECT  laonno,
              NULLIF(intrest_rate,10) result
FROM daily_pipeline_loan_fact
```

DECODE function

The DECODE function applies logic similar to CASE or IF-THEN-ELSE statements in programming languages.

Syntax:

```
DECODE (col/expression, search1, result1,
        [, search2, result2, . . .]
                  [, default])
```

Example of DECODE function:

Suppose, the ABC wants to give a interest rate of 15 percent on all personal loans 'P', a interest rate of 12 percent on home loan 'H", and a interest rate of 16 percent on all other loans. The code to retrieve the data is:

```
SELECT  loan_no,loan_amount, DECODE(loan_type,'P', 15, 'H', 12, 16) as
interestrate
      FROM daily_pipeline_loan_fact;
The above DECODE statement is equivalent to the following
 IF-THEN-ELSE statement:
IF  loan_type = 'P' THEN
Result :=15;
ELSEIF loan_type = 'H' THEN
Result := 12;
ELSE Result := 16;
```

Nesting of functions: Single-row functions can be nested to any level.

Example:

```
SELECT ENAME (NVL(TO_CHAR(MGR), 'NO MANAGER') FROM EMP;
```

Aggregate Functions

Aggregate function operates on set of rows to give one result per group.

The following are the aggregate functions:

- SUM
- AVG
- MAX
- MIN
- COUNT(*)
- COUNT (column name)

All group functions except COUNT(*) ignore NULL. MAX and MIN functions can be used for any datatype.

Group By clause:

Query results can be summarized using the GROUP BY clause. Multiple columns can be used in GROUP BY clause.

Following can be used with GROUP BY clause:

- Constant
- Function without parameters (SYSDATE, USER)
- Group functions

Example:

```
SELECT loan_status, SUM(loan_amount)
FROM daily_pipeline_loan_fact
GROUP BY loan_status;
```

The `HAVING` clause: Used for restricting groups

Sequence

- Rows are first grouped
- The group function is applied
- Groups matching the HAVING clause is then displayed

Example:

```
SELECT loan_status, SUM(loan_amount)
FROM daily_pipeline_loan_fact
GROUP BY loan_status;
HAVING SUM(loan_amount)> 10000;
```

Summary

- Functions are used to perform calculations on data, modify individual data items, manipulate output for groups of rows, format dates and numbers for display, convert column data types.
- The types of functions are single-row functions and multiple-row functions.
- The types single row functions are character, number, date, conversion, list, and general functions and can be used with SELECT, WHERE, and ORDER BY.
- Oracle can implicitly convert from varchar2 or char to number, varchar2 or char to Date, Number to varchar2, and date to varchar2.
- Explicit conversion is required primarily for conversion and display of date/time data to character form and vice-versa.

Test Your Understanding

1. What are the different types of functions?
2. What are the different types of single row functions?
3. What are the different types of date function?
4. What are the different types of character function?
5. What are the different types of number function?
6. What is the use of COALESCE?
7. What is the use of NULLIF?
8. What is the use of DECODE?
9. What are the different aggregated functions?
10. What is the difference between WHERE clause and HAVING clause?

Session 10: SQL Expressions

Learning Objectives

After completing this session, you will be able to:

- Define simple and complex expressions
- Define Case expressions
- Define date time expressions
- Define function expressions

Understand the SQL Expressions

An **expression** is a combination of one or more values, operators, and SQL functions that evaluates to a value. An expression generally assumes the datatype of its components.

You can use expressions in:

- The select list of the SELECT statement
- A condition of the WHERE clause and HAVING clause
- The CONNECT BY, START WITH, and ORDER BY clauses
- The VALUES clause of the INSERT statement
- The SET clause of the UPDATE statement

SQL Expressions can be classified as

- Simple Expressions
- Compound Expressions
- CASE Expressions
- Date time Expressions
- Function Expressions

Simple Expressions

A simple expression specifies a column, pseudo column, constant, sequence number, or null.
Some valid simple expressions are as follows:

- `Daily_pipeline_loan_fact.loanno`
- 'this is a text string'
- 10

Compound Expressions

A compound expression specifies a combination of other expressions. You can use any built-in function as an expression

In a compound expression, some combinations of functions are inappropriate and are rejected. For example, the LENGTH function is inappropriate within an aggregate function.

Some valid compound expressions are:

```
('CLARK' || 'SMITH')
LENGTH('MOOSE') * 57
SQRT(144) + 72
my_fun(TO_CHAR(sysdate,'DD-MMM-YY'))
```

CASE Expressions

CASE expression let you use IF-THEN-ELSE logic in SQL statements without having to invoke procedures.

```
CASE expr WHEN comparision_expr1 THEN return_expr1
                [WHEN comparision_expr2 THEN return_expr2
                WHEN comparision_exprn THEN return_exprn
                 ELSE else_expr]
END
```

In a simple expression the Oracle server searches for the first WHEN…..THEN pair for which expr is equal to comparision_expr1 and returns return_expr1 .If none of the WHEN…..THEN pairs meet this condition and an else clause exists then oracle returns else_expr. Otherwise the Oracle server returns null.

You cannot specify the literal null for all the return_exprs and else_expr.

Example : Suppose, the ABC wants to give a interest rate of 15 percent on all personal loans ‘P’, a interest rate of 12 percent on home loan ‘H”, and a interest rate of 16 percent on all other loans. The code to retrieve the data is as follows:

```
SELECT  loan_no,loan_amount,
             CASE loan_type WHEN ‘P’ THEN 15
                                      WHEN ‘H’ THEN 12
                                       ELSE  16
              END            as interestrate
      FROM daily_pipeline_loan_fact;
```

Date and Time Expressions

A datetime_value_expr can be a datetime column or a compound expression that yields a datetime value.

Arithmetic With date: As the database stores dates as numbers, you can perform calculation using arithmetic operators such as addition and subtraction.

Operation	Result	Description
Date + number	date	Adds a number of days to a date
Date _ number	date	Subtract a number of days to a date
Date - Date	Number of days	Subtract one date from another
Date + number/24	date	Adds a number of hours to a date.

Example:

```
SELECT loanno,(SYSDATE – update_date)/7 AS weeks FROM
daily_pipeline_loan_fact
```

The earlier query will display the loanno and the number of week since when the loan is not updated.

Interval Expression: An interval expression yields a value of `INTERVAL YEAR TO MONTH` or `INTERVAL DAY TO SECOND`.

INTERVAL YEAR TO MONTH: Stored as an interval of years and months.

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

year_precision is the number of digit in the year date time field. The default value of year_precision is 2.

Example

```
INTERVAL  ‘312-2’ YEAR(3) TO MONTH
Indicates an interval of 312 years and 2 months.
INTERVAL  ‘312’  YEAR(3)
Indicates an interval of 312 years and 0 months.
INTERVAL  ‘300’  MONTH(3)
Indicates an interval of 300 months.
```

INTERVAL DAY TO SECOND: The INTERVAL DAY TO SECOND data type stores a period of time represented as days, hours, minutes and seconds with a fractional part .

```
INTERVAL DAY [(year_precision)] TO SECOND [(sec_precision)]
```

Year_precision is the number of digits in the DAY datetime field. The range of year_precision is 0 to 9. The default value is 2.

sec_precision is the number of digits in the fractional part of the SECOND datetime field. The range of sec_precision is 0 to 9. The default value is 6.

Example:

```
SELECT (SYSTIMESTAMP - update_date) DAY(9) TO   SECOND
FROM   daily_pipeline_loan_fact
WHERE loanno = ‘10001’;
```

Function Expressions

You can use any built-in SQL function or user-defined function as an expression. Some valid built-in function expressions are:

- `LENGTH('BLAKE')`
- `ROUND(1234.567*43)`
- `SYSDATE`

Summary

- A simple expression specifies a column, pseudo column, constant, sequence number, or null
- A compound expression specifies a combination of other expressions
- CASE expression let you use IF-THEN-ELSE logic in SQL statements without having to invoke procedures

Test Your Understanding

1. What are different types of expression?
2. Give an example of compound expression?
3. Give an example of date time expression?
4. Give an example of function expression?
5. What is a pseudo column?

Session 12: Joins and sub queries

Learning Objectives

After completing this session, you will be able to:

- Explain joins
- Describe different types of joins
- Write sub-query
- Write co-related sub-query

Understand the Join

A **join** is a query that combines rows from two or more tables, views, or materialized views ("snapshots"). Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

The following are the requirements for using joins:

- Normalization splits a complex table into multiple simple tables. Tables are joined temporarily using a table join strategy to produce a single resultant table for query purposes.
- As it is only for query purposes, data redundancy is completely avoided.
- Tables in a database can be related to each other with common keys (primary key and foreign keys). The purpose is to bind data together, across tables, without repeating all of the data in every table.

Types of joins:

- Equi Join
- NON Equi Join
- Outer Join
- Self Join

Equi Join

What is an Equi-join?

- Is defined as a join in which more than one tables are joined together with the help of a common column that exists in both the tables
- In this type of join the relationship between the tables are specified in the where clause, by using an equal (=) symbol
- EQUI Joins are also called simple joins and inner joins

Example:

```
SELECT loan_no,loan_amount,product_name
FROM   daily_pipeline_loan_fact loan,
           Product_dim product
WHERE product.product_key= loan.product_key
```

Removing Ambiguity

When more than one column joined, and if one column exists in both the tables and that column needs to be displayed in the output, it must be preceded by the table name followed by a period (.) .Otherwise an ambiguity will occur because SQL will not understand from which table the said column to display

Example:

```
SELECT   loan.product_id,loan_no,
              loan_amount,product_name
FROM     daily_pipeline_loan_fact loan,
              Product_dim product
WHERE  product.product_id=loan.product_id
```

Table Aliases

- Should not be more than 30 characters long, but the shorter it is the better.
- The table alias should be substituted for the table name throughout the SELECT statement.
- Valid only for the current SELECT statement.
- Should be meaningful.

Non Equi Join

A non equi join is a join condition containing something other than a equal operator .Others condition such as <=,>=, <>, BETWEEN can be used.

Example:

```
SELECT   loan.product_key,loan_no,
              loan_amount,product_name
FROM     daily_pipeline_loan_fact loan,
              Product_dim product
WHERE  product.product_key >=110
```

Outer Join

An outer join extends the result of a simple join. An **outer join** returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. A simple join does not return such rows. To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle returns NULL for any select list expressions containing columns of B.

Outer join queries are subject to the following rules and restrictions:

- The (+) operator can appear only in the WHERE clause or, in the context of left correlation (that is, when specifying the TABLE clause) in the FROM clause, and can be applied only to a column of a table or view.
- If A and B are joined by multiple join conditions, you must use the (+) operator in all of these conditions. If you do not, Oracle will return only the rows resulting from a simple join, but without a warning or error to advise you that you do not have the results of an outer join.
- The (+) operator can be applied only to a column, not to an arbitrary expression. However, an arbitrary expression can contain a column marked with the (+) operator.

The following are the types of outer join:

- Left outer, means all rows of the table left to the Join condition will appear
- Right outer, means all rows of the table right to the Join condition will appear
- Full outer, means missing rows from both the tables being joined

Example of left outer join:

```
SELECT   loan.product_key,loan_no,
               loan_amount,product_name
FROM   Product_dim product,
           daily_pipeline_loan_fact loan
WHERE  product.product_key=loan.product_key(+)
SELECT   loan.product_key,loan_no,
               loan_amount,product_name
FROM   Product_dim product
LEFT OUTER JOIN
             daily_pipeline_loan_fact loan
ON product.product_key=loan.product_key
```

Example of right outer join:

```
SELECT   loan.product_key,loan_no,
               loan_amount,product_name
FROM   Product_dim product,
           daily_pipeline_loan_fact loan
WHERE  product.product_key(+)=loan.product_key

SELECT   loan.product_key,loan_no,
               loan_amount,product_name
FROM   Product_dim product
RIGHT OUTER JOIN
             daily_pipeline_loan_fact loan
ON product.product_key=loan.product_key
```

Example of full outer join:

```
SELECT   loan.product_id,loan_no,
               loan_amount,product_name
FROM   Product_dim product
FULL OUTER JOIN
            daily_pipeline_loan_fact loan
ON product.product_key=loan.product_key
```

Self Join

When one row of one table is to be compared with another row of the same table, then self join is used. Logical tables need to be created from the same table temporarily. The table, on which the self join will be used, appears twice in the FROM clause, and is followed by table aliases that qualify column names in the join condition.

Example:

The following query uses a self join to return the name of each employee along with

The name of the employee's manager:

```
SELECT e1.ename||' works for '||e2.ename
"Employees and their Managers"
FROM emp e1, emp e2   WHERE e1.mgr = e2.empno;
```

Natural Join

The Natural Join clause is based on all columns in the two tables that have the same name. It select rows from the two tables that have equal values in all matched column. If the columns having the same names have different data types , then an error is returned .

Example:

```
SELECT   loan.product_id,loan_no,
               loan_amount,product_name
FROM   Product_dim product,
           daily_pipeline_loan_fact loan
WHERE  NATURAL JOIN product_dim;
```

Understand Subquery

A subquery is a SELECT statement that is nested within another SELECT statement .In subquery the result of the inner one is passed as an argument to the outer one. You can place the subquery in a number of SQL clauses, including:

- The WHERE clause
- The HAVING clause
- The FROM clause

Subqueries may be:

- Single row subquery (Single row comparison operators can be used for example =, <, >, <=, >=, and so on)
- Multiple row subquery (Multi row comparison operator for example IN, SOME/ANY, or ALL operators)

Guidelines for using subqueries:

- A subquery must be enclosed in parentheses
- Place the subquery on the right side of the comparison condition for readability.

Example:

```
SELECT  loanno,loan_status,loan_type
FROM     daily_pipeline_loan_fact
WHERE  SAL=(SELECT MAX(loan_amount)
                            FROM   daily_pipeline_loan_fact);
```

EXISTS operator: Exists returns a value TRUE if the subquery that follows it returns at least one row.

Example:

```
SELECT  loanno, loan_amount
FROM   daily_pipeline_loan_fact
WHERE  EXISTS  (SELECT COUNT(*)
                                  FROM daily_pipeline_loan_fact
                                  WHERE loan_status='FUNDED'
                                   HAVING COUNT(*) > 5)
ORDER BY loanno;
```

ANY operator: The ANY operator compares the main query with any of the values returned by the inner query.

Example:

```
SELECT loanno, loan_amount,loan_status
FROM   daily_pipeline_loan_fact
WHERE loan_amount > ANY (SELECT loan_amount
                                              FROM daily_pipeline_loan_fact
                                               WHERE loan_status ='FUNDED');
```

ALL operator: The ALL operator compares a value to every value return by the sub query.

Example:

```
SELECT loanno, loan_amount,loan_status
FROM   daily_pipeline_loan_fact
WHERE loan_amount > ALL (SELECT loan_amount
                                          FROM daily_pipeline_loan_fact
                                           WHERE loan_status ='FUNDED');
```

Correlated Subquery

Correlated sub queries are used for row by row processing. Each sub query executed once for every row of outer query.

Steps of execution of Correlated Sub queries:

- Get candidate row (fetched by outer query)
- Execute Inner query using candidate row's value.
- Use Value(s) resulting from inner query to qualify or disqualify candidate.
- Repeat until no candidate row remains

Syntax:

```
SELECT column1,column2
FROM   table1 outer
WHERE column1 operator (SELECT column1,column2
                                      FROM   table2
                                      WHERE expr1=outer.expr2
```

The sub query references a column from a table in the parent query.

Example: Find all the loans having loan amount more then the average loan amount in it's product category.

```
SELECT loanno,loan_amount,product_key
FROM    daily_pipeline_loan_fact outer
WHERE loan_amount >(SELECT avg(loan_amount)
                                    FROM   daily_pipeline_loan_fact
                                   WHERE product_key=outer.product_key)
```

Correlated UPDATE: Use a correlated sub query to update rows in one table based on rows from another table.

```
UPDATE table1 alias1
SET       column=(SELECT expression
                            FROM   table2 alias2
                            WHERE alias1.column=alias2.column)
```

Example:

```
UPDATE daily_pipeline_loan_fact outerloan
SET       loan_status=(SELECT loan_status
                                FROM    loan_status_dim status
                                WHERE  outerloan.loan_key=
                                            status.loan_key)
```

Correlated DELETE: Use a correlated sub query to delete rows in one table based on rows from another table.

```
DELETE table1 alias1
SET       column=(SELECT expression
                           FROM   table2 alias2
                           WHERE alias1.column=alias2.column)
```

Example:

```
DELETE daily_pipeline_loan_fact outerloan
SET       loan_status=(SELECT loan_status
                                FROM    loan_status_dim status
                                WHERE  outerloan.loan_key=
                                            status.loan_key)
```

Hierarchical Subquery

Using hierarchical queries, you can retrieve data based on a natural hierarchical relationship between rows in a table .A relational database does not store records in a hierarchical way. However where a hierarchical relationship exists between the rows of a single table, a process called tree walking enables the hierarchy to be constructed. A hierarchical query is a method of reporting, in order, the branches of a tree.

```
SELECT [LEVEL],column,expr,…
FROM    table
[WHERE  condition(s)]
[START WITH condition(s)]
[CONNECTED BY PRIOR condition(s)]
```

Hierarchical queries can be identified by the presence of the CONNECT BY and START WITH Clauses.

LEVEL: For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root row and 2 for a child of a root, and so on.

START WITH: Specifies the root rows of the hierarchy (where to start). This clause is required for a true hierarchical query.

CONNECT BY: Specifies the columns in which the relationships PRIOR between parent and child rows exist. This clause is required for a hierarchical query.

Example:

```
SELECT  employee_id,last_name,job_id,
            manager_id
FROM     employees
START WITH employee_id=101
CONNECT BY PRIOR manager_id=employee_id
```

Summary

- A join is a query that combines rows from two or more tables, views or materialized views (snapshots). Oracle performs a join whenever multiple tables appear in the query's FROM clause.
- In Oracle three types of joins are permitted: Equi Join, Outer Join, and Self Join.
- A sub-query is a select statement that is nested within another select statement, where the result of the inner one is passed as an argument to the outer one.
- In correlated sub-query, each execution of the outer query will ensure the inner query to be executed for all of its values
- A hierarchical query is a method of reporting ,in order, the branches of a tree

Test Your Understanding

1. What are the different types of join?
2. What is a Non equijoin?
3. What are the different types of outer join?
4. What is natural join?
5. What is a subquery?
6. What is a correlated subquery?
7. What is a Hierarchical subquery?

Session 15: Schema Objects

Learning Objectives

After completing this session, you will be able to:

- Describe and create views
- Work with indexes
- Create public or private synonym
- Write sequence
- Work with materialized view
- Work with cluster
- Work with database link

Schema Objects

A schema is a collection of objects .Schema objects are the logical structures that directly refer to the data in a database. Schema objects include

- Tables
- Views
- Synonyms
- Sequences
- Stored procedures
- Indexes
- Clusters
- Database links

Views

A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed .The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the in the data dictionary.

Advantages of Views:

- Views restrict access to the data because the view can display selective columns from the table.
- Views can be used to make simple queries to retrieve the result of complex queries.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide group of users access to data according to their particular criteria.

There are two classification of View simple and complex.

- A simple view is one that
  - Derives data from only one table.
  - Contains no function or group of data.
  - Can perform DML operations through the View.
- A Complex view is one that
  - Derives data from many tables.
  - Contains function or group of data.
  - Does not always allow operations through the View.

Syntax:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
            [(alias)[,alias]…)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

OR REPLACE	Recreate the view if it already exist
FORCE	Create the view regardless of whether the base table exist or not
NO FORCE	Create the views only if the database table exists. This is the default.
View	Is the name of the view.
alias	Specifies names for the expression selected by the view's query.
Subquery	Is a complete SELECT statement.
WITH CHECK OPTION	Specifies that only rows accessible to the view can be inserted or updated.
Constraint	Is the name assigned to the CHECK OPTION constraint
WITH READ ONLY	Ensures that no DML operations can be performed on this view.

Example:

```
CREATE VIEW funded_loan
AS SELECT  loanno,loan_amount,loan_status
     FROM     daily_pipeline_loan_fact
     WHERE   loan_status='FUNDED'
```

Guidelines for creating a View:

- The subquery that defines a view can contain complex SELECT syntax, including joins, groups and sub queries.
- The subquery that defines a view can not contain an ORDER BY clause
- If you do not specify a constrain name for a view created with the WITH CHECK OPTON, the system assigns a default name in the format SYS_Cn.
- You can use the OR REPLACE option to change the definition of the view without dropping and recreating it or regranting object privileges previously granted on it.

Retrieving data from a View

SELECT * FROM funded_loan;

- When you access data using a view the Oracle server performs the following operations.
- It retrieves the view definition from the data dictionary table USER_VIEWS.
- It checks access privileges for the view base table.
- It converts the view query into an equivalent operation on the underlying base table or tables.

Modifying a view: A view can be modified by using CREATE OR REPLACE.

Example:

```
CREATE OR REPLACE VIEW funded_loan
              (loanno,loanamount,loanstatus)
AS SELECT  loanno,loan_amount,loan_status
     FROM     daily_pipeline_loan_fact
     WHERE    loan_status='FUNDED'
```

Example of Complex VIEW:

```
CREATE OR REPLACE VIEW funded_loan
              (product_name,minloanamount,maxloanamount)
AS SELECT  product_name,min(loan_amount),
                  max(loan_amount)
                  loan_status
     FROM     daily_pipeline_loan_fact d,
                   product_dim p
     Where    d.product_id=p.product_id
GROUP BY   p. product_name
```

DML Operations on view:

You cannot remove a row if the row contains the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keywords
- The pseudocolumn ROWNUM keyword.

You cannot modify data in a view if it contains the following:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keywords
- The pseudocolumn `ROWNUM` keyword.

Columns defined by expressions.

You cannot modify data through a view if it contains the following:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keywords
- The pseudocolumn `ROWNUM` keyword.
- Columns defined by expressions
- `NOT NULL` columns in the base table that are not selected by view.

Use of CHECK Option in View Definition: The check option restricts updation of the column which forms the view definition:

```
CREATE VIEW funded_loan
AS SELECT  loanno,loan_amount,loan_status
      FROM     daily_pipeline_loan_fact
     WHERE    loan_status='FUNDED'
WITH CHECK OPTION;
```

The following insert and update command on the view would not work:

```
INSERT INTO funded_loan (loanno,loan_status) VALUES (1001,'SHIPPED');
UPDATE funded_loan SET loan_status = 'SHIPPED';
```

Read-only Views: The Read-only view allows the user only to read from the view:

```
CREATE VIEW funded_loan
AS SELECT  loanno,loan_amount,loan_status
FROM     daily_pipeline_loan_fact
WHERE    loan_status='FUNDED'
WITH READ ONLY;
```

`WITH READ ONLY` clause specifies that you will only be able to select records from the view. Thus modifications to the base table through the view are prevented.

Removing a View: You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW funded_loan;
```

Inline Views:

An inline view is a subquery with an alias (or correlation name) that you can use within a SQL statement.

A named subquery in the FROM clause of the main query is an example of inline view.

An inline view is not a schema object.

Example:

```
SELECT  loan.loanno,loan.loan_amount,loan.product_id,
maxloan.maxloanamount
FROM    daily_pipeline_loan_fact  loan,
            (SELECT max(loan_amount) maxloanamount,
                          product_id
             FROM daily_pipeline_loan_fact
             GROUP BY product_id) maxloan
WHERE   loan.product_id=maxloan.product_id
AND        loan.loanamount< maxloan. Maxloanamount
```

Indexes

Usually, the application developer is responsible for designing the elements of an application, including the tables. DB administrators are responsible for setting storage parameters and defining clusters for tables, based on information from the application developer about how the application works and the types of data expected.

While working with your application developer, carefully plan each table so that the following occurs:

- Tables are normalized.
- Each column is of the proper data type.
- Columns that allow nulls are defined last, to conserve storage space

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

When processing a request, Oracle 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 dollars) or a specific row.

Indexes are created on one or more columns of a table. After it is created, an index is automatically maintained and used by Oracle. Changes to table data (such as adding new rows, updating rows, or deleting rows) are automatically incorporated into all relevant indexes with complete transparency to the users.

An index can be created automatically or manually.

- Automatically: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
- Manually: Users can create non-unique indexes on columns to speed up the access to the rows

Syntax:

```
CREATE INDEX index_name
ON  table (column[, column]…);
```

Example:

```
CREATE INDEX  daily_loan_status_idx
ON                    daily_pipeline_loan_fact(loan_status)
```

You should create an index if:

- A column contains a wide range of values.
- A column contains large number of null values.
- One or more columns are frequently used together in a where clause or a join condition.
- The table is large and most queries are expected to retrieve less than 2 to 4 % of the rows.

It is usually not worth creating an index if:

- The table is small.
- The columns are not often used as a condition in the query.
- Most queries are expected to retrieve more than 2 to 4 % of the rows in the table.
- The table is updated frequently.
- The indexed columns are referenced as part of an expression

B-tree index

B-tree indexes are used to avoid large sorting operations. For example, a SQL query requiring 10,000 rows to be presented in sorted order will often use a b-tree index to avoid the very large sort required to deliver the data to the end user. Oracle offers several options when creating an index using the default b-tree structure. It allows you to index on multiple columns (concatenated indexes) to improve access speeds. Also, it allows for individual columns to be sorted in different orders. For example, we could create a b-tree index on a column called last_name in ascending order and have a second column within the index that displays the salary column in descending order.

```
Create index
    name_salary_idx
on
```

```
   person
(
   last_name asc,
   salary desc);
```

While b-tree indexes are great for simple queries, they are not very good for the following situations:

- Low-cardinality columns—columns with less than 200 distinct values do not have the selectivity required in order to benefit from standard b-tree index structures.
- No support for SQL functions—B-tree indexes are not able to support SQL queries using Oracle's built-in functions. Oracle9i provides a variety of built-in functions that allow SQL statements to query on a piece of an indexed column or on any one of a number of transformations against the indexed column.

Bitmapped indexes

Oracle bitmap indexes are very different from standard b-tree indexes. In bitmap structures, a two-dimensional array is created with one column for every row in the table being indexed. Each column represents a distinct value within the bitmapped index. This two-dimensional array represents each value within the index multiplied by the number of rows in the table. At row retrieval time, Oracle decompresses the bitmap into the RAM data buffers so it can be rapidly scanned for matching values. These matching values are delivered to Oracle in the form of a Row-ID list, and these Row-ID values may directly access the required information.

Function – Based Index

- A function – based index is an index based on expressions.
- The index expression is built from table columns, constants, SQL functions, and user defined functions.

Example:

```
CREATE INDEX    upper_loan_status_idx
ON                    daily_pipeline_loan_fact(UPPER(loan_status))
```

Removing an index: Remove an index by using DROP index command

Example:

```
DROP INDEX upper_loan_status_idx;
To drop an index, you must be the owner of the index or have the DROP
ANY INDEX privilege
```

Synonym

Simplify access to objects by creating a synonym (another name for an object) with synonym you can do the following:

- Easy referring to a table owned by another user
- Shorten lengthy objects name

```
CREATE [PUBLIC] SYNONYM synonym_name
FOR       object;
```

`[PUBLIC]` Create a synonym accessible to all users.

Example:

```
CREATE SYNONYM daily_loan FOR       daily_pipeline_loan_fact;
```

Removing a synonym

```
DROP SYNONYM daily_loan;
```

Sequence

A sequence:

- Automatically generates unique numbers.
- Is a sharable object.
- Is typically used to create a primary key value.

Syntax:

```
CREATE SEQUENCE sequence_name
             [INCREMENT BY n]
             [START WITH n]
             [{MAXVALUE n | NOMAXVALUE}]
             [{MINVALUE n | NOMINVALUE}]
             [{CYCLE | NOCYCLE }]
             [{CACHE n | NOCACHE];
```

Sequence_name	Is the name of the sequence generator
INCREMENT BY n	Specifies the interval between sequence numbers where n is an integer. If this clause omitted, the sequence increments by 1.
START WITH n	Specifies the first sequence number to be generated. Integer. If this clause omitted, the sequence increments by 1.
MAXVALUE n	Specifies the maximum value the sequence can generate.
NOMAXVALUE	Specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence.
MINVALUE n	Specifies the minimum sequence value.

NOMINVALUE	Specifies a minimum value of 1 for an ascending sequence and -10^27 for a descending sequence
CYCLE | NOCYCLE	Specifies whether the sequence continues to generate values after reaching its minimum or maximum value. NOCYCLE is the default option.
CACHE n| NOCACHE	Specifies how many values the Oracle server pre-allocates and keeps in memory. By default, the oracle server caches 20 values.

Example:

```
CREATE SEQUENCE  proddim_product_id_seq
                              INCREMENT BY 10
                              START WITH     120
                              MAXVALUE        9999
                               NO CACHE
                               NO CYCLE;
```

Verify the sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT sequence_name,min_value,max_value ,
           increment_by,last_number
FROM   user_sequences;
```

The last_number column specifies the next available sequence number if nocache is specified.

NEXT VAL and CURRVAL pseudocolumns:

- NEXTVAL returns the next available sequence value.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

You can use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a SELECT statement that is not part of a subquery.
- The SELECT list of a subquery in an INSERT statement.
- The values clause of an INSERT statement
- The SET clause of an UPDATE statement

You cannot use NEXTVAL and CURRVAL in the following contexts

- The select list of a view.
- A select statement with the distinct keyword.
- A select statement with the GROUP BY, HAVING, or ORDER_BY clause.
- A subquery in a SELECT, DELETE, or UPDATE statement.
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement.

Modifying a sequence: A sequence can be modified by using ALTER SEQUENCE statement.

You must be the owner or have the ALTER privilege for the sequence in order to modify it.

Only future sequence numbers are affected by the ALTER SEQUENCE statement.

The START WITH option cannot be changed using ALTER SEQUENCE. The sequence must be dropped and re-created in order to restart the sequence at a different number.

Some validation is performed. For example, a new MAXVALUE that is less than the current sequence number cannot be imposed.

Example:

```
ALTER SEQUENCE  proddim_product_id_seq
                            INCREMENT BY 20
                            MAXVALUE     99999
                            NO CACHE
                            NO CYCLE
```

Removing a sequence: Remove a sequence from the data dictionary by using the DROP SEQUENCE statement. Once removed, the sequence can no longer be referenced.

```
DROP SEQUENCE proddim_product_id_seq;
```

Materialized View

Materialized view is a database object that stores the result of a query. Materialized view automates the data replication and refreshes process. When Materialized view are created a refresh interval is established to schedule refreshes of replicated data .

Features/Capabilities

- Can be partitioned and indexed
- Can be queried directly
- Can have DML applied against it
- Several refresh options are available
- Best in read-intensive environments

Advantages

- Useful for summarizing, pre-computing, replicating and distributing data
- Faster access for expensive and complex joins
- Transparent to end-users
- MVs can be added/dropped without invalidating coded SQL

Disadvantages

- Performance costs of maintaining the views
- Storage costs of maintaining the views

Syntax:

```
CREATE MATERIALIZED VIEW <name>
 {<storage parameters>}
  <build option>
  REFRESH <refresh option> <refresh mode>
  [FOR UPDATE]
 [ENABLE|DISABLE] QUERY REWRITE
  AS  SELECT <select clause>;
```

The create MV's command has four major section.

- `CREATE MATERIALIZED VIEW <name>`
- The second section of the create MV's is as follows:

```
{<storage parameters>}
        <build option>
```

The <build option> determines when MV is built

`BUILD IMMEDIATE`: view is built at creation time
`BUILD DEFFERED`: view is built at a later time
`ON PREBUILT TABLE`: use an existing table as view source

- The third section of the create MV's is

```
REFRESH <refresh option> <refresh mode>
```

Refresh Options are as follows:
COMPLETE:

- Totally refreshes the view
- Can be done at any time; can be time consuming

FAST:

- incrementally applies data changes
- A materialized view log is required on each detail table
- Data changes are recorded in MV logs or direct loader logs
- Many other requirements must be met for fast refreshes

FORCE:

- Does FAST refresh in favour of a `COMPLETE`
- The default refresh option

Refresh Modes:

ON COMMIT: Refreshes occur whenever a commit is performed on one of the view's underlying detail table(s)

- Available only with single table aggregate or join based views
- Keeps view data transactionally accurate
- Need to check alert log for view creation errors

ON DEMAND: Refreshes are initiated manually using one of the procedures in the DBMS_MVIEW package

- Can be used with all types of materialized views
- Manual Refresh Procedures

```
DBMS_MVIEW.REFRESH(<mv_name>, <refresh_option>)
DBMS_MVIEW.REFRESH_ALL_MVIEWS()
```

START WITH [NEXT] <date>: Refreshes start at a specified date/time and continue at regular intervals

- The fourth section of the create MV's command is the query that the MV will use.

```
[FOR UPDATE]
 [ENABLE|DISABLE] QUERY REWRITE
  AS SELECT <select clause>
```

If you specify `FOR UPDATE`, the MV's will be updatable; Otherwise,
It will be read only.

Example:

```
CREATE MATERIALIZED VIEW  loan_info
REFRESH FORCE
AS
SELECT  product_id,count(loan_no),sum(loan_amount)
FROM     daily_pipeline_loan_fact
GROUP BY product_id;
```

Cluster

A cluster is a schema object that contains data from one or more tables, all of which have one or more columns in common. Oracle Database stores together all the rows from all the tables that share the same cluster key.

- To create a cluster in your own schema, you must have `CREATE CLUSTER` system privilege.
- To create a cluster in another user's schema, you must have `CREATE ANYCLUSTER` system privilege. Also, the owner of the schema to contain the cluster must have either space quota on the tablespace containing the cluster or the `UNLIMITED TABLESPACE` system privilege.

- Oracle Database does not automatically create an index for a cluster when the cluster is initially created. Data manipulation language (DML) statements cannot be issued against cluster tables in an indexed cluster until you create a cluster index with a `CREATE INDEX` statement.

Syntax:

```
CREATE CLUSTER cluster_name
 (column datatype[, column datatype]…)
[other options]
```

Creating a Cluster: Example The following statement creates a cluster named personnel with the cluster key column department, a cluster size of 512 bytes, and storage parameter values:

```
CREATE CLUSTER personnel
                          (department NUMBER(4))
SIZE 512
STORAGE (initial 100K next 50K);
```

Cluster Keys: Example The following statement creates the cluster index on the cluster key of personnel:

```
CREATE INDEX idx_personnel ON CLUSTER personnel;
```

After creating the cluster index, you can add tables to the index and perform DML operations on those tables.

Departmental tables from the sample `hr.employees` table and add them to the personnel cluster created in the earlier example:

```
CREATE TABLE dept_10
CLUSTER personnel (department_id)
AS SELECT * FROM employees WHERE department_id = 10;
```

Database Link

Objects in the remote database can be access from a local database through a database link.

Syntax:

```
CREATE [SHARED][PUBLIC] DATABASE LINK <link_name>
CONNECT TO {current user|user_name
IDENTIFIED BY password [authentication clause]}
USING ‘connect string’
```

To create a database link you must have the `CREATE DATABASE LINK` system privileges.
To create a public database link you must have the `CREATE PUBLIC DATABASE LINK` system privileges

Example:

```
CREATE PUBLIC DATABASE LINK remote_connect
CONNECT TO user_name IDENTIFIED BY password
USING ‘connect string’
```

A public database link is available to all users in a database. A private database link is only available to the user, who created it.

Access remote table using database link.

```
SELECT * FROM user_info@remote_connect.
```

Summary

- A view is a logical table based on a table or another view
- Views restrict access to the data because the view can display selective columns from the table
- The check option restricts updation of the column which forms the view definition
- The Read-only view allows the user only to read from the view
- Index is used by the oracle server to speed up the retrieval of rows by using a pointer
- use synonym to provide alternative names for objects
- Sequence Is a sharable object which automatically generates unique numbers.
- Materialized view is a database object that stores the result of a query .
- Materialized view automate the data replication and refresh process
- A cluster is a schema object that contains data from one or more tables, all of which have one or more columns in common
- Objects in the remote database can be access from a local database through a database link

Test Your Understanding

1. What are the different types of view?
2. What is the significance of `FORCE` key word in the View definition?
3. What is the use of `WITH CHECK OPTION` and `WITH READ ONLY` in the View definition?
4. What are the advantages of Index?
5. What is a function based index?
6. What is a synonym?
7. What is the difference between public synonym and private synonym?
8. What is the maximum value and minimum value for a sequence?
9. What is the use of `CACHE` and `CYCLE` option?
10. What are the different refresh mode and refresh option available for MVs?
11. What is the difference between Views and MVs?
12. What is a cluster?
13. Why database links are required?

Session 19: Flashback Queries

Objectives

After completing this session, you will be able to:

- Describe a Flashback Query
- Describe a Flashback using SCN or Timestamp
- Describe a Flashback Transaction
- Describe a Flashback Table
- Describe a Flashback Row History
- Describe The Recycle bin

Introduction

As part of its read consistency model, Oracle displays data that has been committed to the database. A flashback query is used to see the data as it existed prior to the commit.

Flashback Query

As part of its read consistency model, Oracle displays data that has been committed to the database. A flashback query is used to see the data as it existed prior to the commit. You can use the result of the flashback query to restore the result of the data .To support flashback query the database should use system managed undo. System managed undo is a feature introduced in Oracle 9i.To automate management of rollback segments The DBA must create an undo tablespace , enable Automatic undo management , and establish an undo retention time window. To use some features of flashback queries, you must have the EXECUTE privilege on the DBMS_FLASHBACK package

Time – Based Flashback

Time - Based Flash back example: Suppose there are 10 rows in product_dim table and while refreshing the table is deleted, but somehow not all the product is included, so the delete is inappropriate.

Before delete

```
SELECT COUNT(*) FROM product_dim
COUNT(*)
-----------
10
DELETE FROM product_dim;
COMMIT;
```

After delete

```
SELECT COUNT(*) FROM product_dim
COUNT(*)
-----------
0
```

There are two way to restore the data

- Data pump import can be used to restore the table.
- Perform a physical database recovery to recover the database to a point in time prior to delete.

However, with flashback queries, you can avoid the need to perform these recovery options.

```
SELECT COUNT(*) FROM product_dim
AS OF TIMESTAMP ( SYSDATE – 5/1440) ;
COUNT(*)
-----------
10
```

Create a back up table to save the data.

Example:

```
CREATE TABLE  product_dim_old
AS SELECT * FROM product_dim
AS OF TIMESTAMP ( SYSDATE – 5/1440) ;
```

SCN – Based Flashback

To begin with SCN – based flashback, you must first know the SCN of your transaction. To get the latest change number, issue a commit and then use the AS OF SCN clause of the `SELECT COMMAND` .You can find the current SCN by executing the `GET_SYSTEM_CHANGE_NUMBER` function of the `DBMS_FLASHBACK` package prior to execute your transaction.

Before delete

```
SELECT COUNT(*) FROM product_dim
COUNT(*)
-----------
10
DELETE FROM product_dim;
COMMIT;
```

After delete

```
SELECT COUNT(*) FROM product_dim
COUNT(*)
-----------
0
SELECT  COUNT(*) FROM product_dim
AS OF SCN(:SCN_FLASH);
```

```
COUNT(*)
-----------
10
```

```
CREATE TABLE  product_dim_old
AS SELECT * FROM product_dim
AS OF SCN(:SCN_FLASH);
```

SCN_TO_TIMESTAMP can be used to find out the latest timing on which a change is made to a particular table.

Example:

```
To see the SCN associated with each row use ORA_ROWSCN, which is a new
feature introduced in Oracle 10g.
INSERT  INTO product_dim (product_id,
                                        product_name)
                            Values(1009,
                                        'FIXED – 48  Month')
COMMIT;
SELECT  product_name, ORA_ROWSCN
FROM     product_dim ;

PRODUCT_NAME                    ORA_ROWSCN
-------------------                  ----------------------
ARM                                       553531
FIXED                                     553531
FIXED – 48  Month                  553853
SELECT SCN_TO_TIMESTAMP(553853)
FROM    dual;

SCN_TO_TIMESTAMP(553853)
---------------------------------
20-FEB-04 03.11.28.00000000 PM
```

Flashback Table

The flashback table command restores an earlier state of a table in the event of human or application error. Oracle cannot restore a table to an earlier state across any DDL operations that change the structure of the table .The database should be using Automatic Undo Management (AUM) for flashback to work. The ability to flashback the old data is limited by the amount of undo retained in the undo tablespace and the UNDO_RETENTION initialization parameter setting.

- You cannot roll back a flashback table statement.
- Record the current SCN before issuing a flashback table command.

- You must have either the FLASHBACK object privilege on the table or the FLASHBACK ANY TABLE system privilege.

Recovering Dropped Tables :

```
DROP TABLE product_dim;
```

As of Oracle 10g, a dropped table does not fully disappear.
Its blocks are still maintained in its tablespace.
The dropped objects can be seen by querying RECYCLEBIN data dictionary view.

```
SELECT * FROM RECYCLEBIN;
```

RECYCLEBIN is a public synonym for the USER_ RECYCLEBIN data dictionary view. DBAs can see all dropped objects by querying DBA_ RECYCLEBIN data dictionary view.

The FLASHBACK TABLE TO BEFORE command can be use to recover the table from the RECYCLEBIN.

```
FLASHBACK TABLE product_dim TO BEFORE DROP;
```

Flashback table using timestamp or SCN:

Before flashback the table to the time just prior to a wrong update, first we must enable the row movement for the table.

```
ALTER TABLE product_dim enable row movement;
```

We can then flashback the table.

```
FLASHBACK TABLE product_dim TO timestamp (systimestamp – 5/1440)
```

You can use the TO SCN clause if you wish to specify an SCN instead of a timestamp.

```
FLASHBACK TABLE product_dim TO SCN (720880)
```

Flashback Database

The flashback database command returns the database to a past time or SCN providing a first alternative to performing incomplete database recovery.

Following a flashback database operation, in order to have write access to the flashed back database you must reopen it with an ALTER DATABASE OPEN RESETLOGS command.

You must have the SYSDBA system privilege in order to use the Flashback Database command.

Syntax:

```
FLASHBACK TABLE DATABASE TO timestamp (systimestamp – 5/1440)
```

Summary

- As part of its read consistency model, Oracle displays data that has been committed to the database.
- A flashback query is used to see the data as it existed prior to the commit.
- To begin with SCN – based flashback, you must first know the SCN of your transaction. To get the latest change number, issue a commit and then use the AS OF SCN clause of the `SELECT COMMAND`.
- `SCN_TO_TIMESTAMP` can be used to find out the latest timing on which a change is made to a particular table.
- The flashback table command restores an earlier state of a table in the event of human or application error.
- The flashback database command returns the database to a past time or SCN providing a first alternative to performing incomplete database recovery.

Test Your Understanding

1. What is time base flashback?
2. What is SCN base flashback?
3. What is the use of `ORA_ROWSCN`?
4. What is the use of `SCN_TO_TIMESTAMP`?
5. What is flashback table?
6. What is flashback database?

References

Websites

- http://otn.oracle.com
- http://www.docnmail.com/learn/Oracle.htm

Books

- **Oracle 10g complete reference:** Kevin Loney, George Koch, Oracle Press

STUDENT NOTES: