ORACLE 11g

COURSE MATERIAL

TM

NIT NARESH technologies

Opp. Satyam Theatre,
Ameerpet, Hyderabad - 500 016.

SOFTWARE TRAINING & DEVELOPMENT

E-mail:info@nareshit.com www.nareshit.com

Ph:23746666, 23734842 Cell: 9000994007, 9000994008

What is Database Actually?

Database is a platform where we can place the data fro the future references. Database is a collection of interrelated data, i.e. database always stores data along with it's relationships.

What is Database Management System?

A database Management System is essentially a collection interrelated data and a set of programs to access this data. This collection of data is called the Database. The primary objective of a DBMS is to provide a convenient environment to retrieve and store database information. Database System support single user and multi-user environment. While on one hand DBMS permits only one person to access the database at a given time, on the other RDBMS allows many users simultaneous access to the database.

A Database System consists of two parts namely, Database Management System and Database Application. Database Management System is the program that organizes and maintains the information whereas the Database Applications is the program that lets us view, retrieve and update information stored in the DBMS.

- Database is a collection of data in one or more files for the future reference.
- Database is a collection of interrelated data, i.e. database always stores data along with it's relationships.

Database Models:

The main object of the database is store the inter related data and maintain the data. The very basic elementary piece of data is called as 'data item'.we assume that item cannot be subdivided into smaller data types and at the same time retain any meaning to the users of the data. The relationship among the data items, which shows how they are related, is called as Data model'.

The database models are

1. File Management System(FMS)
2. Hierarchicla database System(HDS)
3. Network database System(NDS)
4. Relational database Management System (RDBMS).

File Management Systems:

The File Management System was the first method used to store data in a computerized database. Each data item is stored on disk sequentially in one large file. In order to locate one particular item the search starts at the beginning and each item is checked subsequently till the match is found.A particular relationship cannot be drawn between the items other than the sequence in which it is stored.

Drawbacks:

- A particular record can not be located quickly. If the data has be stored, the whole file has to be read and rewritten in the new order.
- It will not support data types.
- It will not support to store or allocate memory dynamically.
- Data con not be shared with concurrent users.
- Data duplication.
- **Very poor security**: Operating system provide only a password mechanism for security. This is not sufficiently flexible to enforce security polices in which different users have permission to access different subsets of the data.
- **Retrieving data from a file is not faster:** We have to write special programs to answer each question a user may want to ask about the data. These programs are likely to be complex because of the large volume of data to be searched.

Hierarchical Database System(HDS):

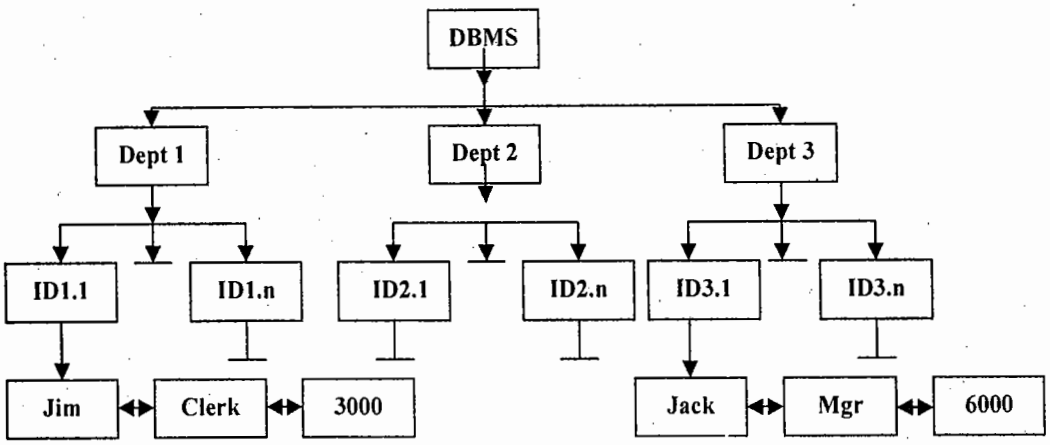


Fig 1.0

Data storage is in form of a parent-child relationship.The origin of a data tree is the root.Data located at different levels along a particular branch from the root is called the node.The last node in the series is called the leaf.This model supports One-to-Many relationship.From the figure 1.0 it can be seen that the nodes in the third level are interrelated.Each child has pointer to numerous and there is just one pointer to the parent thus resulting in a One-to-Many relationship.

Suppose an information is required ,say ID 3.1, it is not necessary for the DBMS to search the entire file to locate the data.Insted, it first follows the Dept3 branch and fetches the data.

Disadvantages:

It is not possible to enter a new level into the system. As and when such a need arises the entire structure has to be revamped. Another disadvantage is that this model does not support Many-to-Many relationship. In case this sort of relation is required then multiple copies of the same data have to made which result in redundancy. To overcome this drawback, the Network Database Model was introduced.

Note: The terminal points shows, for example below ID.1.n.indicate that similar data are present at that point.

Network Database Systems(NDS):

The main idea behind the NDS model is to bring about Many-to_Many relationship. The relationship between the different data items is called as sets. This system also uses a pointer to locate a particular record(i.c called as Physical Links)

Disadvantages:

The use of pointers leads to complexity in the structure. As a result of the increased complexity mapping of related data become very difficult.

Relational Mode(RDBMS):-

- The Model was first outlined by E.F Codd 1970.
- The Components of Relation Model are:
  - Collection of objects or relations that stores the data.
  - A set of operations that can act on the relations to produce other relations.
- Data integrity for accuracy and consistency.
- It has only logical representations.(i.e in the form of table(rows and columns.)
- Intersection of rows and columns gives single value.

- No data redundancy .
- No physical link relations are maintain logically.
- Supports NULL values,integrity constraints.
- Provides high security.
- Supports unlimited size.
- supports to store any data types (number,characters,date, images,audio,text,files.)
- Data can be stored among several users at time.
- Data can also be shared across several plot forms.
- Oracle 10g is proven to be the fastest database for Transaction processing Datawarehousing, and third party applications on servers of all sizes.
- Use of Flashback Queries

1)It queries the database by TIME or user specified SCN (System Change Number).

2)It uses Oracle's multi version read-consistency capabilities to restore data by applying

UNDO as needed .

ENTITY RELATIONSHIP Model:

- In an effective system data is divided into discrete categories or entities.
- An ER-Model is an illustration of various entities in a business and the relationships between them.
- It is built during the analysis phase of the System Developing Life Cycle.
- ER-Model separates the information required & the business from the activities performed.

ER-MODEL Benefits:

- It document information for the organization in a clear, precise format.
- Provides a clear picture of the scope of the information requirement.
- Provides an easily understood pictorial map for the database design.
- It offers an effective framework for integrating multiple application.

Key Components in ER-MODEL:

- **ENTITY:** It is a thing of significance about which the information need to be know.
- **ATTRIBUTES:** It is something that describes or qualifies an entity.

** Each attribute may be mandatory or optional but one attribute Mandatory.

Relation among data:

- A relationship is defined as "an association among entities.
- A relationship type is an association of entity types.
- Several relationships may exist between the same entity.

The three different types of relationships recognized among various data stored in the database are:

- One-to One
- One-to Many(or Many-to-One)
- Many-to-Many

One-to-One:

- Consider for example a set of students in a class. Each student can only one roll number. Similarly, each roll number can be associated only with one student.

This is the case of One-to-One relationship Fig 1.2 illustrates this relationship.

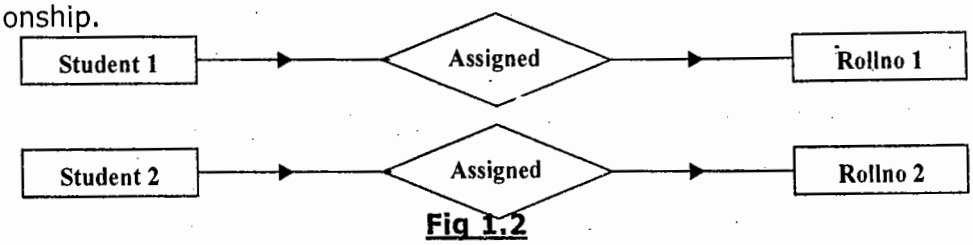


Fig 1.2

Many-to-One:

One student can register for only one particular course at a time, where a number of students could register for the same course.

This is illustrated with Fig1.3

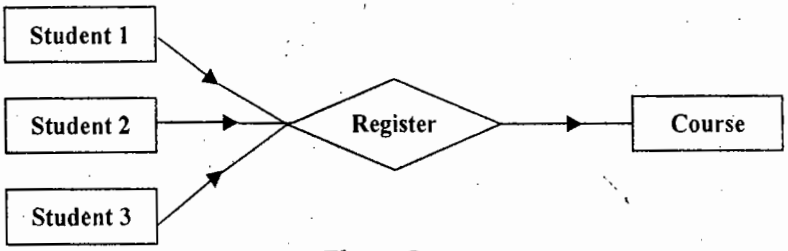


Fig 1.3

Many-to Many:

A vendor can sell a number of items and many vendors can sell particular item. This can be understand from Fig 1.4 given below.

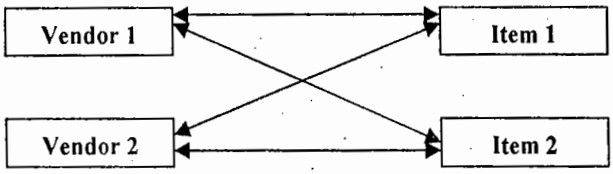


Fig 1.4

Relation Database Terminology:

Row or Tuple :

- It represents all data required for a particular instance in entity.
- Each row is an entity is uniquely identified by declaring it has PRIMARY KEY or UNIQUE.
- The order of the rows is not significant, while retrieving the data.

Column Or Attribute:

- It represent one kind of data in a table.
- The column order is not significant when storing the data.

A Field:

- It can be found at the Intersection of row and a column.
- A field can have only one value, or may not have a value at all, the absence of value in Oracle is represented as NULL.

Relating Multiple Tables:

- Each table contains data that describes exactly only one entity.
- Data about different entities is stored in different tables.
- RDBMS enables the data in one table to be related to another table by using the Foreign keys.
- A Foreign Key is a column or a set of Column that refer to a Primary key in the same table or another table.

Rational Database Properties:

- Should not specify the access route to the tables, and should not reveal the physical arrange.

- The Database is accessed using Structured Query Language(SQL)
- The language is a collection of set operators.

Communicating With RDBMS:

- The Structured Query Language is used to Communicate with RDBMS.

Structure Query Language:

- Structure Query Language and commonly pronounced, as "SEQUEL" (Structured English Query Language).
- Dr E.F Codd published the paper on relational database model in June 1970. IBM Corporation , Inc.
- SEQUEL later become SQL.
- It allows the user to communicate as the server.
- It is easy to learn and use.
- It is functionally complete, by allowing the use to define, retrieve and manipulate the data.

Components of SQL:

- Oracle SQL Complies with industry accepted standards.
- The SQL Contains 5 Sub Language.

☼**Data Retrieval/ Query Language(DRL/DQL)**

SELECT

- It used to retrieve the information from database objects for read only purpose.

☼ **Data Manipulation Language(DML).**

- It used to manipulate the data in database objects

INSERT(new content) UPDATE(modify) DELETE(remove)

☼**Data Definition Language(DDL)**

- Used to define database objects
  creation
  modification
  removing

CREATE ALTER DROP TRUNCATE RENAME

☼ **Data Control Language(DCL)**

- It used to share the information between users

GRANT(give) REVOKE(cancel)

☼ **Transaction Control Language(TCL)**

- It used to save or cancel DML operations

COMMIT ROLLBACK SAVEPOINT

Oracle Architecture:

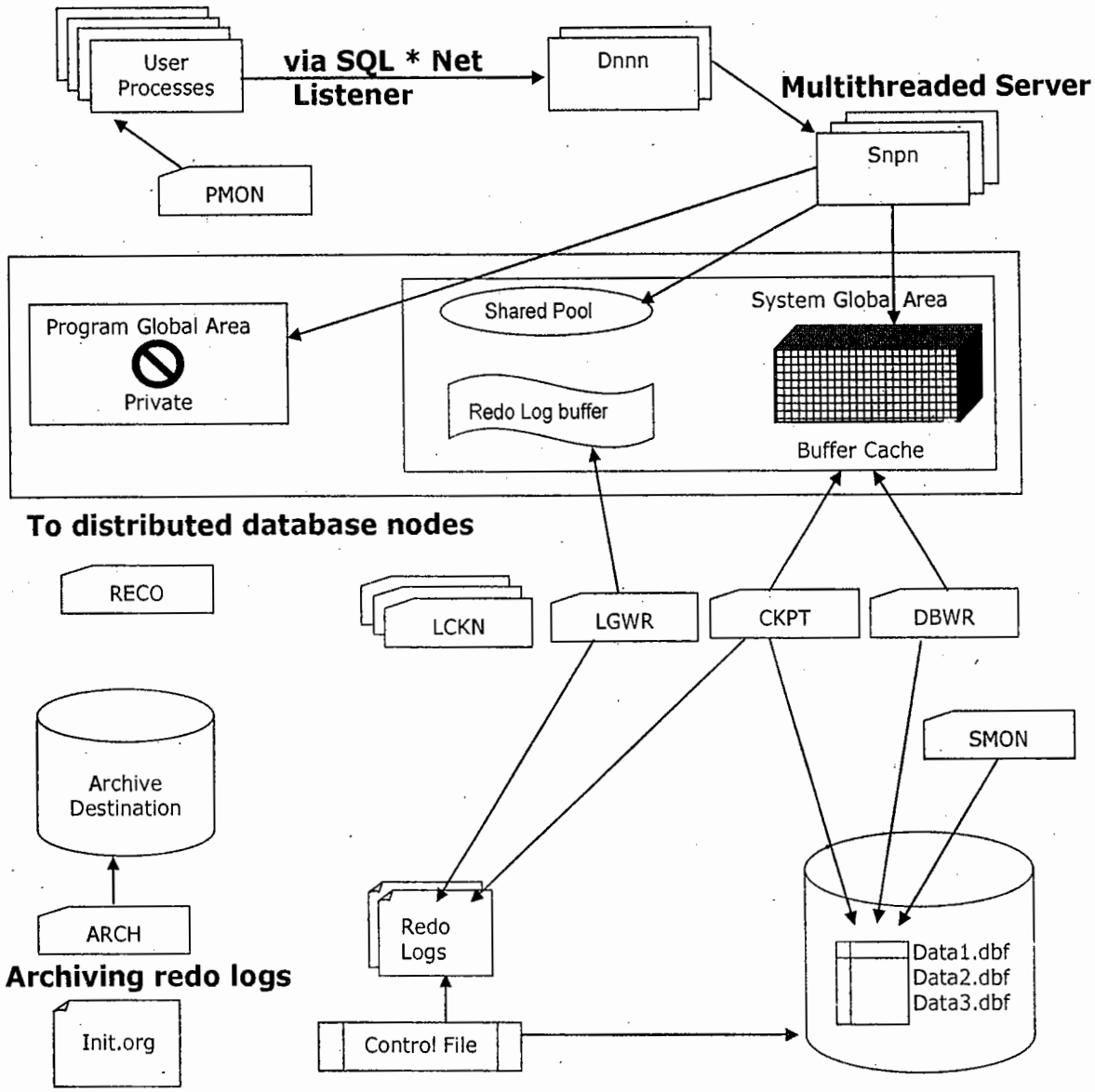


Fig 1.1

User Processes

- When a user runs an application program, such as a Pro*C program, or an Oracle tool, such as Server Manager, Oracle creates a user process to run the user's application.

-->user process will create in client side.
-->SQL * Net Listener will run on server for particular port.
--> Listener will listen for incoming request and service the request.(it will create a server process on server?)

Dispatcher Processes(Dnnn)

- The Dispatcher processes allow user processes to share a limited number of server processes.
- Without a dispatcher, each user process requires one dedicated server process.

Multithreaded Server(Shared server)

→ In dedicated server architecture each user process will create a server process on the server where as in shared server architecture each server process will be shared by multiple user process.

Oracle SGA(System Global Area):

- Oracle allocates a memory area called the System Global Area(SGA) and starts one or more Oracle processes.
- A System Global Area(SGA) is group of shared memory structures that contain data and control information for one Oracle database instance.
- **Note:** The combination of SGA and the Oracle processes is called an Oracle database instance.
- The SGA, consists of several elements.

Buffer Cache:

- The buffer cache stores Oracle data in memory for users to view or change .In this way, user never make changes directly to disk files.

Redo Log Buffer:

- The log buffer stores special information called redo, which helps oracle reconstruct data changes in the event of system failure.

Shared Pool

- Components of the shared pool include the library cache, for storing parsed SQL statements for reuse by other users.

Library Cache

- The library cache includes shared SQL areas, private SQL areas, PL/SQL procedures and packages, and control structures such as locks and library cache handles.

Dictionary Cache

- The data Dictionary is a collection of database tables and views ,and its users.

  Data Dictionary stores

  →Names of all tables and views in the database

  →Names and data types of columns in database tables

  → Privileges of all oracles users.

Server Processes

- Server process created on behalf of each user's application may perform one or more of the following:
  - Parse and execute SQL statements issued via the application.
  - Read necessary data blocks form disk(datafiles) into the shared database buffers of the SGA, if the blocks are not already presents in the SGA.

Background Processes

The background processes in an Oracle instance include the following

Lock(LCKn)

- With the parallel Server option, up to ten Lock processes(LCK0,....LCK9) provide inter-instance locking. However, a single LCK process(LCK0) is sufficient for most Parallel Server Systems.

Database Writer(DBWR)

- Database Writer process writes buffers to datafiles.
- The primary job of the DBWR process is to keep the buffer cache "clean" by writing dirty buffers to disk.

Log Writer Process(LGWR)

- The Log Writer Process(LGWR) writer the redo log buffer to a redo log file on disk.
- LGWR writes one contiguous portion of the buffer to disk, LGWR writes a commit record when a user process commits a transaction.

Recoverer Process(RECO)

- The Recoverer process(RECO) is a process used with the distributed option that automatically resolves failures involving distributed transactions.
- The remote server is not available or the network connection has not been re-established ,RECO automatically tries to connect again a timed interval.

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 CKPT.

Process Monitor(PMON):

- The process Monitor (PMON) performs process recovery when a user process fails.
- PMON is responsible for cleaning up the cache and freeing resources that the process was using.

  Ex: It resets the status of the active transaction table, release locks.

Snapshot Refresh Processes(Snpn)

- With the distributed option, up to ten Snapshot Refresh processes(SNP0,...SNP9) can automatically refresh table snapshots.
- These processes wake up periodically and refresh any snapshots that are scheduled to be automatically refreshed.

System Monitor(SMON)

- The System Monitor process(SMON) performs instance recovery at instance start up.
- SMON is also responsible for cleaning up temporary segments that are no longer in use.

Datafiles

- This mandatory disk components is used for storing oracle dictionary and application database object.
- Datafiles stores Oracle data.
- Each datafile can be associated with only one database.

Redo Logs

- This mandatory disk components is used for storing redo information on disk. (ic rollback segment data).

Control file

- The Physical locations of both datafiles and redo logs in the server's files system are stored in your control files.

Parameter Files(init.ora)

- This mandatory disk components is used for configuring how oracle operates while it is running .

Sql * Plus Buffer:

- SQL * Plus ORACLE has developed it's environment tool.
- We can use directly SQL & PL/SQL statements in this environment tool.
- All commands of SQL are typed at the SQL prompt.
- Only one SQL Statement is managed in the SQL Buffer.
- The Current SQL statement replaces the previous SQL statement in the Buffer.
- SQL statement can be divided into different lines within the SQL Buffer.
- Only one line i.e., the current line can be active at a time in the SQL Buffer.
- At SQL prompt, editing is possible only in the current SQL Buffer line.
- Every statement of SQL should be terminated Using(;).
- To run the previous or current SQL statement in the Buffer type '/' at SQL prompt.
- To open the SQL Editor type ed at SQL prompt.
- It provides.....
  - SQL commands
  - PL/SQL block
  - It has own commands to set environment as requirement.
  - To generate report.
  - To save SQL command as file(.sql)
  - To save PL/SQL command as file(.sql)
  - To save output as file(.lst)

Creating and Managing Tables:

Database Objects:

- A Oracle database can contain multiple data structures.
- The different Database objects in Oracle are:

TABLE: Used to store data, Basic Unit
VIEW: Logically represent subsets of data from one or more tables.
SEQUENCE: Used to Generate Primary Key values.
INDEX: It is used to improve the performance of some queries.
SYNONYM: Used to give alternate names to objects.

Table in Oracle:

- Table can be crated at any time, even when the users are using the database.
- Size of the table need not be specified.

For create table:

1)Table Name 2)Column Name 3)Data Type

Table Restriction:

- The user should have permission or CREATE TABLE command, and storage area.
- Table name must be unique in schema
- The Table name should begin with a letter and can be 1-30 characters long.
- Maximum 1000 columns(8.0), 256 in 7.x
- Names can contain:
  **A -- Z **a -- z **0—9 **_,$,#
- Names can not be duplicated for another object in the same ORACLE Server.
- Name cannot be oracle server reserved words.
- Names are not case sensitive.

COLUMN NAME:

- The column name should begin with a letter and can be 1-30 characters long.
- Column name is Unique in a table.
- Column name can be renamed.
- Maximum columns in table is 1000.

Data Types in Oracle:

- Each value in ORACLE is manipulated by a data type.
- The values of one data type are different from another data type.
- The data type defines the domain of values that each column can contain.
- The Built-in-data types of ORACLE are Categorized as
  - CHARACTER Data Types
  - NUMBER Data Types
  - LONG and RAW Data Types
  - DATETIME Data Types
  - ROWID Data Types

Character Data Types:

They store character data which can Alphanumeric data.

- The Information can be
  - Words
  - Database Character set
  - National Character set

- They are less restrictive than other data types an have very few properties.
- They data is stored in strings with byte values
- The different character data types are:

CHAR NCHARVARCHAR VARCHAR2 NVARCHAR2

CHAR Data Type:

- It specifies fixed length character string.
- The size should be specified.
- If the data is less than the original size, blank pads are applied.
- The default length is 1 Byte and the Maximum is 2000 Bytes.

NCHAR Data Type:

- It is first defined in ORACLE 9i, and contains Unicode data only.
- The column's maximum length is determined by the National Character set definition.
- The Maximum size allowed is 2000 Bytes and size has to be specified.
- If the data short than the actual size then the blank pads are applied.

Varchar2 Data Type:

- A data type used for storing text data.
- The Minimum size is 1 Byte and the Maximum size is 4000 Bytes.
- It occupies only that space for which the data is supplied.
- Any text character (including special characters, number, dashes, and so on) can be stored.

Nvarchar2 Data Type:

- It is first defined in ORACLE 9i, and contains Unicode data only.
- The Minimum size is 1 Byte and the Maximum size is 4000 Bytes.

Number Data Type:

Number(Precision, Scale)

- Number data type stores the numeric data, the precision is the total nc of digits required and scale stands for the rounding of decimal place.
- Range of precision is from 1 to 38.
- Range of Scale is -84 to 127.

Integer Data Type:

- Integer data type will converted as Number Data type with the maxim size 38 digit as precision .

LONG:

- This data type is used to store characters or numbers.
- Maximum size limit is 2 GB.
- Only one Long column is valid per table .

Date & Time Data Type:

- It is used to store dates and time information.
- The information revealed by date is:

*Century *Year *Month

***Date** ***Hour** ***Minute** ***Second**

- The default date format in ORACLE is DD-MON-YY.
- The default time accepted by ORACLE date is 12:00:00 AM(Midnight).
- The default date accept by ORACLE data is the First day of the Current month.
- The Date range provide by Oracle is

JANUARY 1,4712 BC to DECEMBER 31,9999 AD.

Timestamp Date Type:

- It is an extension of the DATE data type.
- It stores

***Day** ***Month** ***Year** ***Hour** ***Minute** . ***Seco nd**

Syntax: TIMESTAMP(Fractional-Seconds-Precision)

- Fractional –Seconds-Precision optional specifies the number of digits in the fractional part of the SECOND datetime field.
- It can be a number in the range of 0-9, with default as 6.

RAW:

- It stores binary information like Photos, Signature, Thumb impressions ect.
- Maximum length is 2000 Bytes.
- The Oracle converts the RAW and LONG RAW data into Hexadecimal form.
- Each Hexadecimal character represent four bites of RAW data.

LONG RAW:

- It stores the binary data similar to RAW but can store more bytes than RAW.
- Maximum length is 2GB.
- No size is required.

Large Object(LOB) Data Types:

- The Built in LOB data types are
  ***BLOB *CLOB *NCLOB**
- These data types are stored inerally.
- The Bfile is an LOB which is stored externally.
- The LOB data type can store large and unstructured data like Text, Image, Video and Spatial data.
- The maximum size is upto 4GB.
- LOB columns contains LOB locators, Which can refer to out-of-line to or in-line LOB values.
- LOB's selection actually return the LOB's locator.

BFILE Data Type:

- It enables access to binary file LOB's which are stored in the systems outside ORACLE.
- A BFILE column or the attributes stores the BFILE locator.
- The BFILE locator maintains the directory alias and the filename.
- The Binary File LOB's do not participate in transaction and are not recoverable.
- The maximum size is 4 GB.

BLOB Data Type:

- It stored unstructured Binary Large Objects.
- They are Bit streams with no character set semantics.
- They are provide with full transactional support.

CLOB Data Type:

- It Dynamic data type.
- They are provide with full transactional support.
- The maximum size is 4 GB.

NCLOB Data Type:

- It stores Unicode data using the National Character set.

ROWID Data Type:

- Each row in the database has as address.
- The rows address can be queried using the pseudo column ROWID.
- ROWID's efficient support partitioned table and Indexes.

Codd's Rules:-

Dr E.F.CODD's listed 12 rules for relational databases In 1985.

1. The Relational data base management rule:

- A relational data base management system must use Only its relational capabilities to manage the information stored in the database.

2. The Information rule:

- All information in relational database must be represented as values within columns in rows.
- Pointers are variables that contain address of the memory location where the data is stored.

3. The Guaranteed Access rule:
- Every item of data must be logically addressable with the help of a table name, primary key value and column name.

4. The Systematic Treatment of NULL value rule:
- A relational database must support the NULL value for representing missing or inapplicable data.

5. The Dynamic On-line catalog Based on the Relational Model Rule:
- The structure of the data base should be represented at Logical level in a tabular form.
- The representation of database structure as a collection of tables is known as catalog or data dictionary.

6. The Comprehensive Data sublanguage rule:
- An RDBMS support at least one clearly defined language that can be used interactively as well as can be embedded in programs.(SQL).

7. The view updating rule:
- All theoretically updatable views must be updatable in table.

8. The High-Level insert Update and delete rule:
- The date retrieval from a relational database must be possible in sets that may comprise data from multiple rows or multiple tables.

9. The physical Data Independence rule:
- The database user need not known about the physical Implementation of data storage and retrieval.

10. The Logical Data Independence rule:
- The data base user's view of a data should not be effected by any change in the database tables.(Materialized view).

11. The Integrity Independence rule:
- Data integrity rules must be defendable in the relational database and should be stored in data dictionary.

12. The Distribution Independence rule:
- The system must be able to access or manipulate the data that is distributed in other systems.

13. Non-subversion Rule:
- If a database management system has a low level language, then it cannot be used to violate the integrity rules or constrains that have been expressed using a higher-level language.

To Create Table:

Syntax:.

```
Sql>Create Table <Table Name>
        (<Column1><Type>,<Column1><Type>,.....);
```

Note:

The character ';' is the terminator for SQL statement.

Example:

```
Sql>Create Table Emp_det
   (    Empno Number(4),  Ename Varchar2(15), Job Varchar(15),  Mgr
Number(4),
          Hiredate Timestamp,  Sal Number(6,2),  Comm Number(6,2),  Ephoto
BLOB,
          Deptno Number(2)   );

Sql>Create Table Student_Det
        (StudNo Number(4), Fname Varchar2(15), Lname Varchar2(15), DOB
Date,
                DOJ Date, Fees Number(6,2),Gender Char );
```

Inserting Data in Table:

- A row is inserted into a table by using INSERT command.
- Rows can insert into a
  - Table
  - Views Base Table.
  - A partition of a Table.
  - A Sub Partition of a Composite Partition Table.
  - An Object Table.
  - An Object View's Base Table.

Inserting of data into a table can be executed in two ways.

- Conventional INSERT.
- Direct-Path INSERT.

In conventional Insert statement, Oracle reuses free space in the table into which the data in being Insert and Maintains Referential Integrity Constraints.

In Direct-Path Insert, Oracle append the insert data after existing data in the Table, the free space is not reused.

Syntax:

Sql>INSERT INTO <table name> [list of columns] VALUES(list of values);

```
Sql> INSERT Into Student_Det Values  ( 1001,   'THOMAS',  'SIEBEL',  '30-DEC-81' ,
                                       '02-JAN-91', 30000, 'M'  );
```

- In this case the values should be provided to all the columns that exists inside the table.
- The order of values declared in the Values clause should follow the original order of the columns in the table.
- The Character and Date type data should be declared in single Quotes.
- Number information can be applied normally.

Inserting Data into Required Columns:

```
Sql>  Insert Into Student_Det(  StudNo, Fname, Lname , DOJ  ) Values
                              (  1002, 'JAN', 'SMITH', '12-JAN-09' );
```

- In this case the Order of columns declared in insert need not be the same as that of the original table order.
- The data values in the values caluse should match with that of INSERT list.
- The columns not supplied with data are filled with NULL values, Until the NOT NULL constraint is declared.

Inserting NULL values into Table:

- Null represent information that is not available.
- Null Value is
  - Unknown value
  - Undefined value
  - Not equal to 0 or blank space represented with '**NULL**' keyword.
  - Not allowed arithmetic, relational operation on Null, If performed it return Null only.
  - RDBMS must support NULL Values.
- NULL values can be inserted in two ways.
  - Implicit→Omit the columns from list oracle supplies NULL values.
  - Explicit→Specify the NULL keyword.

```
Sql>Insert Into Student_Det (      StudNo,  Fname ,  Lname ,  DOB,  DOJ,  Fees,
Gender  )
          Values ( 1003, 'RAMA',  NULL,  '12-JAN-81',  NULL,  30000,  'M' );
```

Inserting special Values into Table:

SYSDATE FUNCTION

- SYSDATE is Pseudo-column.
- This function return the current date& time.

USER FUNCTION:

- This function returns the user name of the user who has logged in.

Example:

```
SQL>CREATE TABLE Student (Stuid Number(4),Name  Clob,DOB  Date,DOJ Date,
                Fees Number(8,2),Gender char);
```

Example:

```
Sql>INSERT INTO student VALUES(1001,user,'01-JAN-85',sysdate,2000,'M');
```

Substitution Variables:

- These variables are used to stored values temporaily
- The values can be stored temporarily through
  - →Single Ampersand(&)
  - →Double Ampersand(&&)
  - →DEFINE and ACCEPT Commands
- The Single Ampersand substitution variable applies for each instance when the SQL statement is create or execute.
- The Double Ampersand substitution variable applies for all instances until that SQL statement is existing.

USING SINGLE AMPERSAND SUBSTITUTION VARIABLE:

```
Sql> INSERT INTO dept VALUES(&Dno,'&Dname','&Loc');
Sql> INSERT INTO emp(Empno,Ename,Hiredate,Deptno)
                    VALUES(&Eno,'&Ename','&Hiredate',&dno);
```

Note: To run command is used for executing a previous command...

```
>run
 or
>/
```

USING DOUBLE AMPERSAND SUBSTITUTION VARIABLE:

```
Sql>INSERT INTO student VALUES(&Sid,'&Name','&Dob',&Doj,&&fee,'&Gender');
```

DEFINING CUSTOMIED PROMPTS:

```
Sql>ACCEPT Deptno prompt 'Please Enter the deptnumber:'
Sql>ACCEPT Dname prompt 'Please Enter the Deptname:'
Sql>ACCEPT Location prompt 'Please Enter the Location:'
Sql> INSERT  INTO dept VALUES(&Deptno,&Dname,&Location);
```

CREATING AN .LST FILE:

- The List file is used to save all information which performed in SQL *Plus .

1)In SQL * Plus

Navigation: File→Spool→Spool File

2) Give the file name

3) Performed all tasks.

4) Now Spool Off.

CREATING AN SQL SCRIPT FILE:

The SAVE Command is used to store the current contents of the SQL Buffer.

Steps:

1)At SQL prompt type the full name of the path where the file has to be created.

2)Give the name of the file with .sql extension.

3)If the has to be replaced with the same existing name then use REPLACE ALL clause.

Example:

Sql>SAVE E:\nitdir\ins

Sql> SAVE E:\nitdir\ins REPLACE

To Get Script:

Get used to display the script file .

Sql>Get E:\nitdir\ins

To Run Script:

Run is used to display the script and run the file.

Sql>RUN E:\nitdir\ins To simple the script file.

Sql>@ E:\nitdir\ins

To Open the Script file in editor.

Sql>ed E:\nitdir\ins

Data Retrieval Languages(DQL/DRL):

QUERY: It is an operation that retrieves data from one or more table or view.

SELECT Statement:

- Select used to retrieve data from one or more than table ,view, object tables.
- It is for read only purpose.
- The select is the most frequently used command as access to information is needed all the time.

PREREQUISTIES:

- The user must have the SELECT privileges on the specified object.

CAPABILITES OF SQL SELECT STATEMENT:

The different types SELECT criteria.

- SELECT:
  - It chooses the rows in a table that are expected to return by a query.
- PROJECTION:
  - It chooses the columns in a table that are expected to return by a query.
- JOIN :
  - It chooses the data in from one or more numbers of tables.

Data Retrieving Language:

Syntax:

Sql>SELECT *|{[DISTINCT/UNIQUE] column | expression [Alias],.....} FROM table.

- SELECT →identifies what columns, specifies a list of column(one/more)
- FROM →identifies which table
- DISTINCE →Suppress Duplicates.
- * →Select all Columns
- Columns/ Expression →Select the Named Columns or the expression.
- Alias →Gives select columns different headings

WRITING SQL STATEMENTS TO SELECT DATA FROM TABLES.

The sample table.

1)DEPT

Column Name	Data	Type
DEPTNO	NUMBER	(2)
DNAME	VARCHAR2	(14)
LOC	VARCHAR2	(13)

2)EMP

Column Name	Data Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
JOB	VARCHAR2(9)
MGR	NUMBER(4)
HIREDATE	DATE
SAL	NUMBER(7,2)
COMM	NUMBER(7,2)
DEPTNO	NUMBER(2)

3)SALGRADE

Column Name	Data Type
GRADE	NUMBER
LOSAL	NUMBER
HISAL	NUMBER

Dept:

```
SQL>  Select *From Dept;

  DEPTNO          DNAME          LOC
---------         -------------  -------------
      10          ACCOUNTING     NEW YORK
      20          RESEARCH       DALLAS
      30          SALES          CHICAGO
      40          OPERATIONS     BOSTON

SQL> SELECT * FROM MY_EMP;

EMPNO  ENAME   JOB       MGR       HIREDATE      SAL      COMM    DEPTNO
-----  ------- --------  ------    --------- --------- --------- ------
 7369  SMITH     CLERK     7902     17-DEC-80      800                20
 7499  ALLEN     SALESMAN  7698     20-FEB-81     1600      300      30
 7521  WARD      SALESMAN  7698     22-FEB-81     1250      500      30
 7566  JONES     MANAGER   7839     02-APR-81     2975               20
 7654  MARTIN    SALESMAN  7698     28-SEP-81     1250     1400      30
 7698  BLAKE     MANAGER   7839     01-MAY-81     2850               30
 7782  CLARK     MANAGER   7839     09-JUN-81     2450               10
 7788  SCOTT     ANALYST   7566     09-DEC-82     3000               20
 7839  KING      PRESIDENT          17-NOV-81     5000               10
 7844  TURNER    SALESMAN  7698     08-SEP-81     1500         0     30
 7876  ADAMS     CLERK     7788     12-JAN-83     1100               20
 7900  JAMES     CLERK     7698     03-DEC-81      950               30
 7902  FORD      ANALYST   7566     03-DEC-81     3000               20
 7934  MILLER    CLERK     7782     23-JAN-82     1300               10

14 rows selected.

SQL> Select *From Salgrade;
      GRADE      LOSAL      HISAL
      ---------  ---------  ---------
      1          700        1200
      2          1201       1400
      3          1401       2000
      4          2001       3000
      5          3001       9999
```

Retrieving Data from All Columns:

Sql>SELECT *FROM Emp;
Sql>SELECT *FROM Dept;
Sql>SELECT *FROM Salgrade;

- In this the '*' is a projection operator.
- It project data from all the columns existing in the table with all records.
- The data is displayed in a table format.

Retrieving Data from Specific Columns:

Sql> SELECT empno, ename, job, sal FROM Emp;
Sql>SELECT loc, dname, deptno FROM Dept;
Sql> SELECT Hisal,Losal,Grade FROM Salgrade;

- The columns names need not be in the same order as table.
- The columns should be separated using comma.
- The column names can be separated into different lines within the SQL Buffer.
- The casing of column names is not important.

Column Heading Default:

- The default justification of the data after it is retrieved from the table is...
  LEFT →Date and Character
  RIGHT→ Number Data
- The default Columns Heading display in UPPER case.

Applying Arithmetic Operations in Select Statements:

- Arithmetic Expressions can be implemented through SELECT statement.
- Arithmetic Expressions can be implemented to
  - Modify the way the data is displayed.
  - Perform calculations.

An Arithmetic Expression can contain:

- Simple columns names
- Constant numeric values
- Arithmetic operators

ARITHMETIC OPERATORS:

- The Arithmetic operations can be used to create expressions on NUMBER and DATE data.
- The Arithmetic operators supported are....
  - + →Addition
  - - → Subtraction
  - *→ Multiplication
  - / → division.

- The Arithmetic operators can be used in any clause of a SQL statement. except the FROM clause.
- SQL * Plus ignores Blank Spaces before and after the Arithmetic operator.

Sql> Select empno,ename,sal,sal+500 FROM Emp;
Sql> Select empno,ename,sal,sal-1000 FROM Emp;

Operator Precedence:

- Multiplication and Division take priority over addition and subtractions(*/+-)
- Operators of the same priority are evaluated from left to right.
- To prioritize evaluation and to increase clarity parenthesis can be implemented.

Sql> Select empno,ename,sal,12*sal+100 from Emp;
Sql>Select empno,ename,sal,(12*sal)+100 from Emp;
Sql>Select empno,ename,sal,12*(sal+500) from Emp;

Handling Null Values:

NULL:

- Unknown value
- Undefined value
- Not equal to 0 or blank space.
- If a row lacks the data for a particular column, than that value is said to be NULL or to contain NULL.
- It represented with NULL keyword
- No operations allowed on Null,(If performed any arithmetic operations it return null only).
- RDBMS must support NULL Values.

```
Sql>Select ename,job,sal,comm from emp;
Sql> Select ename,job,sal,comm,12*sal+comm from emp;
```

General Function:

This functions work with any data type and pertain to using null value.

NVL Function:

- The NVL function is used to convert a NULL value to an actual value.

Syntax:

NVL(Expr1,Expr2)

Expr1: is the source value or expression that may contain NULL.
Expr2: is the target value for converting NULL.

NVL Conversions for Various Data Types:

- NVL(Number_column, 0)
- NVL(date_column, '01-JAN-09')
- NVL(Character_column, 'Unavailable')

Note : If Expr1 is Character data then Expr2 may any Data type.

```
Sql>Select NVL(100,200) from dual;
Sql> Select NVL(null,200) from dual;
Sql> Select ename,sal,comm,sal+NVL(comm,0) from emp;
Sql> Select ename,sal,comm,(sal*12)+NVL(comm,0) from emp;
Sql> Select ename,sal,comm,(sal+500)+NVL(comm,0) from emp;
```

NVL2 Function:

Syntax:

NVL2(expr1,expr2,expr3)

- If expr1 is not null,NVL2 returns expr2.if expr1 is null, NVL2 returns expr3.
- Expr1 may any data type.
- The data type of the return value is always the same as the data type of expr2 , unless Expr2 is character data .

Example:-

```
1) select nvl2(comm,0,1000) from emp;
2) select sal,comm, sal+nvl2(comm,100,1000)  from emp;
SQL>select ename,sal,comm, nvl2(comm,'sal+comm','sal') income  from emp where
deptno in(10,30);
```

NULLIF Function:

Syntax:

NULLIF(expr1,expr2)

- Compares two expressions and returns null if they are equal,or the first if there are not equal.

Example:

```
sql>SELECT NULLIF(100,200) from dual;
sql>SELECT ENAME,LENGTH(ENAME) "expr1", JOB,LENGTH(JOB) "expr2",
        NULLIF(LENGTH(ENAME),LENGTH(JOB)) result  from emp;
```

Note:

- The data type of the expr1 is same as the data type of expr2.
- The **NULLIF** function is logically equivalent to CASE expression.

COALESCE:

- It return first non-null expression in the expression list.

```
Sql> SELECT coalesce(100,600,200) FROM dual;
Sql> SELECT coalesce(null,600,200) FROM dual;
Sql>SELECT Ename,Deptno,COALESCE(COMM,SAL,10) COMM  FROM EMP
```

OCP Questions :

1) The EMPLOYEE tables has these columns:
LAST_NAME VARCHAR2(35)
SALARY NUMBER(8,2)
COMMISSION_PCT NUMBER(5,2)

You want to display the name and annual salary multiplied by the commission_pct for all employees. For records that have a NULL commission_pct, a zero must be displayed against the calculated column.

Which SQL statement displays the desired results?
A. SELECT last_name, (salary * 12) * commission_pct FROM EMPLOYEES;
B. SELECT last_name, (salary * 12) * IFNULL(commission_pct, 0) FROM EMPLOYEES;
C. SELECT last_name, (salary * 12) * NVL2(commission_pct, 0) FROM EMPLOYEES;
D. SELECT last_name, (salary * 12) * NVL(commission_pct, 0) FROM EMPLOYEES;

Defining a Column Alias:

- An Alias is an alternate name given for any Oracle Object.
- Aliases in Oracle are of two types.
  *Column Alias *Table Alias
- Column alias rename a Column Heading.
- Alias Headings appear in uppercase By default.
- Specify the alias after the column in the SELECT list using a space as a separator.
- AS keyword between the column name and alias is optional.
- The Alias contains spaces or special characters (such as # or $), or is case sensitive, enclose the alias in double quotation marks (" ").
- An alias cannot be used, any where in the SELECT list for operational purpose.
- An alias effectively renames the SELECT list item for the duration of the Query.

```
Sql> SELECT Empno EmpNumber,  Ename EmpName,  Sal     "EmpSalary"
                                                Job  Designation  FROM Emp;
Sql> SELECT Grade AS   "SalGrade",      Hisal       "High Salary Range",
                          Losal  "Lower Salary Range"   From Salgrade;

Sql>    Select  empno  "EmpNumber",  Sal  "Basic",Sal*0.25  HRA,  Sal*0.20
DA,Sal*0.15 "Pf",
                    Sal+Sal*0.25+Sal*0.20-Sal*0.15 "Gross" FROM emp;
```

Concatenation Operator:

- The Concatenates operator concatenate columns or character strings or expressions or constant to other values or columns.
- Is represented by two vertical bars (||).
- The resultant column that is a character .

```
Sql> SELECT empno||ename FROM Emp;
Sql> Select 'The Basic Salary of '||Ename|| 'is Rs '||Sal Employee From Emp;
Sql>Select Empno||Ename||Ename||' ,Designation is '||job  "Employees
Information " From Emp;
```

LITERALS IN ORACLE:

- **A Literal** and **Constant** value are synonyms to one another and refer to a fixed data value.
- The types of Literals recognized by Oracle are
  - Text Literals
  - Number Literals
  - Interval Literals

Text Literals:

- It specifies a text or character literal.
- It is used to specify values whenever 'text' or CHAR appear in
  *Expression *Condition *SQL Function * SQL Statements.
- It should be enclosed in single quotes.
- A text literal can have a maximum length of 4000 Bytes.

Example:

'Employee Information'
'Manager's Specification'

Using Literal Character Strings:

- A literal value is a character, a number, or a date that is included in the SELECT list.
- A literal value not a column name or a column alias.
- A literal is printed for each row returned, that is retrieved by the SELECT statement.
- Literal strings of free-format text can be included in the query.
- A free-format text treated the same as a column in the SELECT list.
- Date and character literal must be enclosed within the single quotation marks ' '.
- Literal increase the readability of the output.

Sql>Select Ename ||':'|| 'Month Salary='|| Sal As Salaries From Emp;

Sql>Select 'The Designation of '|| Ename ||' is '|| job As Designation From Emp;

Sql> Select 'The employee name is: '||Ename||' and '||'Designation is :'||job from Emp;

Sql> Select 'The Annual Salary of '||Ename||' is '||Sal*12 As Annual_Salary From Emp;

Sql> Select Dname||'Department is Located at '||Loc From Dept ;

Sql> Select Ename||' Joined the Organization on '|| Hiredate From Emp;

Sql> Select Ename||' Works in Department Number '||Deptno||' as ' ||job From Emp;

Sql> Select 'The Employee Name is: '||Ename||', Designation is '||job FROM Emp;

Sql> SELECT 'The Gross salary of '||Ename||' is '||(Sal+Sal*.25+Sal*.20-Sal*.15) FROM Emp;

Duplicate Rows:

- Unless we indicate otherwise, iSQL*Plus displays the results of a query without eliminating duplicate rows.
- To eliminate duplicate rows in the result, the DISTINCT keyword is used.
- We can specify multiple columns after the DISTINCT qualifier.
- The DISTINCT qualifier affects all the selected columns, and the result is every distinct combination of the columns.

Sql>Select DISTINCT Job From Emp;

Sql> Select DISTINCT Job,Deptno From Emp;

Sql>Select DISTINCT Deptno,Job From Emp;

Filter Of Record:

The number of rows returned by a query can be limited using the WHER clause.

- The method of restriction is the basis of the WHERE clause in SQL.
- A WHRE Clause contains a condition that must be met and should directly follow the From Clause.

Sql>SELECT *|{[DISTINCT] column|expression [alias],...} FROM table [WHERE condition(s)];

- The WHERE clause can compares
  - Values in Columns
  - Literal Values
  - Arithmetic Expressions
  - Function
- The components of WHER clause are
  - Column Name
  - Comparison Operator
  - Column Name, constant or list of values.
- The Character strings and dates should be enclosed in single quotation marks.
- Character values are case sensate and Date values are format sensitive (DD-MON-YY)
- The Comparison operator are used in conditions that compare one expression to another.

Relational Or Comparison operators:

< >,^=,!=>**>=**<**<=**=

Example:

Sql>SELECT empno, ename, job, sal from emp;

Sql>SELECT loc, dname, deptno from dept;

Sql>SELECT * from emp;

Sql>SELECT * from students;

Sql>SELECT * from emp WHERE deptno=10;

Logical Operators:

- A logical condition combines the result of two component conditions to produce a single result.
- Three logical operators are available in Oracle.

AND **OR** **NOT**

AND Operator:

- The AND operator allows creating an SQL statement based on two or more conditions being met.
- It Returns FALSE if either is FALSE, else returns unknown.

Truth Table:

AND

TRUE	TRUE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE

Example:

Sql> Select Ename,Sal,Job Emp Where (Sal>=1500 AND Sal<=5000) AND Job='MANAGER';

OR Operator:

- It return TRUE if either component conditions is TRUE.
- It returns FALSE if both are FALSE, else return unknown.

Truth Table:

OR

TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

```
Sql>Select Empno,Ename,Sal,Deptno From Emp Where Sal>=2000 OR
Deptno=20;

Sql>Select Empno,Ename,Job,Hiredate From Emp Where Job='MANAGER' OR
Deptno=30;

Sql>Select  Empno,Ename,Job,Deptno From Emp Where (Deptno=10 OR
Deptno=20)
                                                    OR JOB='MANAGER';

Sql> Select Ename,Job From Emp Where (Job='CLERK' or Job='SALESMAN'
                                                          or
ob='ANALYST');

Sql>  Select Ename,Hiredate,Deptno From Emp Where Job='MANAGER' OR
Deptno=30;

Sql>Select Ename,Sal,Job From EmpWhere (Sal<=2500 OR Sal>=5000) OR
Job='MANAGER';
```

NOT Operator:

- It returns TRUE if the following condition is FALSE.
- It returns FALSE if the following condition is TRUE.
- If the condition is Unknown, it returns Unknown.

Combination of AND OR Operator:

```
Sql> Select Ename,Sal From Emp
        Where (Job='CLERK' or Job='PRESIDENT' or Job='ANALYST')
         And Sal>3000;

Sql> Select Empno,Ename,Job,Sal From Emp
           Where (Sal>1500 OR Job='MANAGER') AND Deptno=10;

Sql> Select Empno,Ename,Job,Sal From Emp
        Where (Deptno=20 OR Job='MANAGER')  AND Sal>=3000;
```

Not Operator Example:

```
Sql> Select Empno,Ename,Job,Sal From Emp Where NOT
       Ename='SMITH';

Sql>Select Empno,Ename,Job,Sal From Emp Where NOT
       Sal>=5000;

Sql> Select Empno,Ename,Job,Sal From Emp Where NOT
       Job='CLERK';

Sql> Select Empno,Ename,Job,Sal From Emp Where NOT
       Sal<=5000;
```

```
Sql> Select Empno,Ename,Job,Hiredate From Emp
Where NOT Hiredate='17-DEC-80';

Sql> Select Empno,Ename,Job,Hiredate
      From Emp
     Where NOT JOb='CLERK' AND Deptno=20;
```

Some Things To Note:

```
Sql> Select Ename,Sal,Job From Emp Where Job>'CLERK';

Sql> Select Ename,Sal,Job From Emp Where Job<'CLERK';

Sql> Select Ename,Deptno,Hiredate From Emp Where Hiredate>'20-
     DEC-81';

Sql> Select Ename,Deptno,Hiredate From Emp Where Hiredate<'20-
     DEC-81';

Sql> Select Ename,Job,Hiredate From Emp Where Job^='CLERK';

Sql> Select Ename,Job,Hiredate From Emp Where NOT Job^=
     'CLERK';

Sql> Select Ename,Job,Hiredate From Emp Where NOT
     Job='CLERK';

Sql> Select Ename,Job,Hiredate From Emp
     Where NOT Hiredate='17-DEC-1980';

Sql> Select Ename,Job,Hiredate From Emp
     Where NOT Hiredate>'18-DEC-1981';
```

Rules of Precedence:

- The default Precedence order is
  - All comparison operators
  - NOT logical condition
  - AND logical condition
  - OR logical condition

Note:

- Override rules of precedence by using parentheses.

Example:

```
Sql> SELECT Ename,Job, SalFROM   Emp
     WHERE  Job = 'CLERK' OR  Job = 'MANAGER' AND Sal> 1500;

Sql> SELECT Ename,Job, Sal FROM   Emp
     WHERE  (Job = 'CLERK' OR  Job = 'MANAGER') AND Sal> 1500;
```

SQL *Plus Operator:

- BETWEEN .... AND.....; NOT BEWEEN ....AND....
- BETWEEN is used to o display rows based on a range of va...
- The declared range is inclusive.
- The lower limit should be declared first.

```
Sql>Select Empno,Ename,comm. From Emp Where Comm
    Between 500 AND 1000;
```

```
Sql>Select Ename,Sal,Job From Where Sal NOT Between 1000 AND 1500;

Sql>Select Ename,Sal,Job From Emp Where Job Between 'MANAGER'
    AND 'SALESMAN';

Sql>Select Ename,Sal,Job From Emp Where Job NOT Between
    'MANAGER' AND 'SALESMAN';

Sql>Select Ename,Sal,Job,Hiredate From Emp Where Hiredate
    Between '17-FEB-1981' AND '20-JUN-1983';

Sql>Select Ename,Sal,Job,Hiredate From Emp Where Hiredate
    NOT Between '17-FEB-1981' AND '20-JUN-1983';
```

IN Operator: NOT IN Operator:

- The Operator is used to test for values in a specified list.
- The Operator can be used upon any datatype.

```
Sql>Select Ename,Sal,Job From Emp Where Ename
      IN('FORD','SMITH');

Sql>Select Empno,Job,Sal From Emp Where Ename NOT
      IN('FORD','SMITH');

Sql>Select Ename,Sal,Deptno From Emp Where Deptno IN(20,30);

Sql>Select Ename,Sal,Deptno From Emp Where Deptno NOT IN(20,30);

Sql>Select Ename,Sal,Deptno From Emp Where
                  Hiredate  IN('20-FEB-1981','09-JUN-1981');

Sql>Select Ename,Sal,Deptno From Emp Where
                  Hiredate  NOT IN('20-FEB-1981','09-JUN-1981');
```

LIKE Operator : **NOT LIKE Operator:**

- Use the LIKE condition to perform wildcard .
- The LIKE Operator searches of valid search string values.
- Search conditions can contain either literal characters or numbers.
- The available wild cards are
  % → It is represent any sequence of Zero or more character.
  _ →Represent any single character, only at that position only.

- The Wild Card symbols can be used in any combination with literal character.
- For finding exact match for '%' and '-' the ESCAPE option has to be used, which is '/' symbol with ESCAPE option.

```
Sql>SELECT Empno,Ename From Emp  Where Ename LIKE 'M%';

Sql> SELECT Empno,Ename From Emp WHERE Ename NOT
       LIKE'M%';

Sql>SELECT Empno,Ename From Emp WHERE Ename LIKE '_O%';

Sql> SELECT Empno,Ename From Emp WHERE Ename
     NOT LIKE '_O%';
```

```
Sql> SELECT Empno,Ename From Emp WHERE Ename  LIKE 'SM%';

Sql> SELECT Empno,Ename,Job From Emp WHERE Job  LIKE '_____';

Sql> SELECT Ename,Hiredate  From Emp WHERE Hiredate
          LIKE '%-FEB-1981';

Sql> SELECT Ename,Hiredate From Emp WHERE Hiredate
         LIKE '%JAN%';

Sql> Select *From Dept Where Dname LIKE '__\_%' ESCAPE '\';
(update dept set dname='SO_FT_WARE' where deptno=50;)
```

IS NULL Operator : IS NOT NULL Operator:

- The Operator tests for NULL values.
- It is the only operator that can be used to test for NULL's.
- NULL value means the value is unav ailable, unassigned, unknown, or inapplicable.

```
Sql>SELECT Ename,Deptno,Comm  From Emp WHERE Comm IS NULL;

Sql>SELECT Ename,Deptno,Mgr,Job  From Emp WHERE Mgr IS NULL;

Sql> SELECT Ename,Deptno,Comm  From Emp WHERE Comm>=0;

Sql>SELECT Ename,Deptno,Comm  From Emp
          WHERE Comm IS NOT  NULL;

Sql>SELECT Ename,Deptno,Comm  From Emp WHERE Mgr
     IS NOT NULL;
```

ORDER BY Clause:

- The Order of rows returned in a query result is undefined.
- The ORDER BY clause can be used to sort the rows.
- The ORDER BY clause must be the last clause of the SQL statement.
- An expression, or an alias, or column position as the sort condition.
- Default ordering of Data is Ascending.
  - **Number 1-999
  - **Dates Earlist-Latest
  - **String A-Z;NULLS→Last.

Syntax:

```
SELECT expr FROM  table [WHERE  condition(s) ]
        [ORDER BY {column, expr} [ASC|DESC]];
```

- The default order upon column is Ascending, to change the default ordering DESC should be used after the column name.
- Sorting can be implemented on column aliases ,and can also be implemented upon multiple columns.

```
Sql>Select Ename,Job,Sal,Deptno From Emp ORDER BY Sal;
Sql>Select Ename,Job,Sal,Deptno From Emp ORDER BY Sal DESC;
Sql>Select Ename,Job,Sal,Deptno From Emp Where Job='CLERK'  ORDER BY Sal;
Sql>Select Ename,Job,Sal,Deptno From Emp Where Sal>=2000 ORDER
                                                BY Deptno,Ename DESC;
```

```
Sql>Select Ename,Job,Sal,Sal*12 Annsal From Emp ORDER BY AnnSal;
Sql>Select Ename,Job,Sal,Deptno From Emp ORDER BY Deptno,Job,Sal ;
Sql>Select Ename,Job,Sal,Deptno From Emp ORDER BY 2 DESC;
Sql>Select * From Emp  ORDER BY 5 DESC;
```

The single row functions can appear in:

**SELECT List
**WHERE List
**Start With Clause
**CONNECT BY Clause

The types of single row functions are

**CHARACTER **NUMBER **DATE **CONVERSION

Multiple Row Functions:

- These function manipulate group of rows to give one result per group of rows.

Single Rows Function:

They are used to manipulate data items.

- They accept one or more arguments and return one value for each row returned by the query.
- An argument can be :
  - User-supplied constant
  - Variable value
  - Column name
  - Expression

Syntax:

Func_Name(Column/Expr,[Arg1,Arg2,.....])

Features of single-row functions include:

- Acting on each row returned in the query.
- Returning one result per row.
- Returning a data value of a different type than that referenced.
- Expecting one or more arguments.
- Can be used in SELECT, WHERE, and ORDER BY clauses.
- Can be nested.

Specification Behavior of Function:

- **Character functions:** Accept character input and can return both character and
- number values.
- **Number functions:** Accept numeric input and return numeric values.
- **Date functions:** Operate on values of the DATE data type (All date functions return a value of DATE data type except the MONTHS_BETWEEN function, which returns a number.)
- **Conversion functions:** Convert a value from one data type to another.
- **General functions:**
  - NVL
  - NVL2
  - NULLIF
  - COALSECE
  - CASE
  - DECODE

Character Functions:

- They return the data type VARCHAR2,limited to a length of 4000 Bytes.
- If the return value length exceeds, then the return value is truncated, without an error.
- Functions can be divided into
  - Case-manipulation functions.
  - Character-manipulation functions.

Character Function Purpose:

LOWER(column|expression)

- It converts alpha character values to lowercase .
- The return value has the same data type as argument char type(CHAR or VARCHAR2)

Syntax: LOWER(column | xpression)

```
Sql>Select LOWER('NARESH I TECHNOLOGIES ') From Dual;
Sql>Select Ename,Job,LOWER('MY DATA') From Emp;
Sql>Select  Ename,LOWER(Ename) From Emp Where Deptno=10;
Sql> Select 'The '||Ename || '"s Designation is '|| Job From EmpWhere
LOWER(Job)='manager'
```

Upper Function:

- It Converts the Alpha character values to Upper Case.
- The return value has the same data type as the argument char.

Syntax:

UPPER(Column | Expression)

```
Sql> Select Upper('Naresh  i Technologies ') From Dual;
Sql>Select Ename,Job, Upper ('My Data') From Emp;
Sql>Select Ename,Job, Upper (Ename), Upper (Job) From Emp Where
Deptno=20;
Sql> Select Ename,Job From Emp Where Job=Upper('Manager');
Sql> Select Upper('E.F Codd') "Capitalised" From Dual;
Sql>Select 'The '||Ename || '"s Designation is '|| Lower(Job) From Emp
                                    Where Job=Upper('manager') Order by Sal;
Sql> Select Upper('The '||Ename|| ' Basic Salary is '||Sal) "Emp  Salaries" ,Job
From Emp Where Job In(Upper('Manager'),Upper('clerk')) Order by Sal Desc;
```

INITCAP Function:

- It returns a string with the first letter of each word in upper case, keeping all othered
- letters in Lower case.

```
Sql> Select initcap(Ename) From emp;
Sql> Select Initcap('naresh  i technologies ') From Dual;
Sql>Select 'The Job Title for'||Inticap(Ename)||' is'|| Lower(Job) Details
From Emp;
Sql>Select Ename,Upper(Ename),Lower(Ename),Initcap(Ename) From Emp;
Sql>Select Empno,Initcap(Ename),Deptno From Emp     Where
Ename=Upper('ford');
```

CONCAT Function:

- It Concatenates the first characters value to the second character value. Only two parameters accept.
- It return the character data type.

Syntax: CONCAT (Column1/Exp1, Column2/Exp2)

```
Sql>Select Concat('Oracle','Naresh Technologies') From Emp;
Sql>Select Ename,Job,Concat(Ename,Job) From Emp Where Deptno=20;
Sql> Select Concat(Concat(Ename,Job),Sal) From Emp;
Sql> Select Concat('The Employee Name is',Initcap(Ename)) As "Employee Names " From Emp Where Deptno in(10,20);
Sql>Select Concat(Concat(Initcap(Ename),'is a'),Job) Job From Emp Where Deptno in(10,20);
Sql>Select Concat('&Fname','&Sname') "Full Name" From Dual;
```

SUB STRING Function:

- Returns specified characters from character value, string from a specified position 'm' to 'n' characters long.
- To extract the portion of the string it is mainly used.

Points to Remember...

- If m is 0, it is treated as 1.
- If m is positive, Oracle counts from the beginning of char to find the first character.
- If n is Omitted, Oracle returns all characters to the end of char.
- If n is less than 1 or 0, A null is returned.Floating points numbers passed as arguments to substr are automatically converted to Integers.

Syntax: SUBSTR(Col/Exp,m,[n])

C→char to be searched.
M→ Starting position .
N→ Occurrence Number.

```
Sql> Select Substr('SIVA RAMA KRISHNA',1,4) From Dual;
Sql> Select Substr('SIVA RAMA KRISHNA',6,4) From Dual;
Sql>Select Substr('SIVA RAMA KRISHNA',11) From Dual;
Sql>Select Substr('SIVA RAMA KRISHNA',-7) From Dual;
Sql> Select Substr('SIVA RAMA KRISHNA',-12,4) From Dual;
Sql> Select Substr('SIVA RAMA KRISHNA',-12,4) From Dual;

Sql> Select Ename,Job From Emp Where Substr(Job,6)=Upper('man');
Sql>SelectConcat(Initcap(Ename),Concat('is a ',Concat(Initcap(Substr(Job,1,3)),' Eater.'))) From Emp Where Substr(Job,4,3)=Upper('Age');
```

Real Time Scenario:

LENGTH Function:

- Returns the number of characters in a value.
- If the char has data type CHAR, the length includes all trailing blanks.
- If the char is NULL, if retrun NULL.

Syntax:

LENGTH(Column|Expression)

```
Sql> Select Length('E.F CODD') From Dual;
Sql> Select Length(Ename)||' Characters exists in '||Initcap(Ename)||' "s Name.'As "Names and Lengths " From Emp;
Sql>Select Initcap(Ename),Job From Where Length(Ename)=5;
Sal> Select Initcap(Ename),Job From Emp Where
Substr(Job,4,Length(Substr(Job,4,3)))='AGE';
```

Real Time Scenario:

INSTRING Function:

- It returns the numeric position of a named character.
  Syntax: INSTR(Column | Expression, 'C', [,m], [n])
- Searches for Column / Expression beginning with its 'n'th character for the 'm' th occurrences of character 'C', and return the position of the character .
- 'm' can be positive or negative, if negative searches backward from the end of Column / Expression.
- The value of 'n' should be positive.
- The default value of both 'm' and 'n' are 1.
- If search is unsuccessful, the return value is zero.

```
Sql> Select Instr('SIVA RAMA KRISHNA','A',1,1) From Dual;
Sql> Select Instr('SIVA RAMA KRISHNA','A',17,2) From Dual;
Sql> Select Instr('SIVA RAMA KRISHNA','MA',7,1) From Dual;
Sql> Select Instr('SIVA RAMA KRISHNA','A',-1,1) From Dual;
Sql> Select Instr(Job,'A',1,2) From Emp Where Job='MANAGER';
Sql> Select Instr(Job,'A',2) From Emp Where Job='MANAGER';
Sql> Select Instr(Job,'A') From Emp Where Job='MANAGER';
```

LPAD Function:

- Pads the character value right-justified to a total width of n character positions.
- The default padding character is space.
  Syntax:
  LPAD(Column | Expression, n, 'C')
- Fill extra spaces with char 'C' up to 'n' position on left side.

```
Sql> Select Lpad('Page 1',20,'*') From Dual;
Sql> Select Lpad('Page 1',20) From Dual;
Sql> Select Lpad(Ename,20,'@') From Emp Where Deptno=10;
```

RPAD Function:

- Pads the character value left-justified to a total width of n character positions.
- The default padding character is space.

Syntax: RPAD(Column | Expression, n, 'C')

```
Sql> Sql> Select Rpad('Page 1',20,'*') From Dual;
Sql> Select Rpad('Page 1',20) From Dual;
Sql> Select Rpad(Ename,20,'@') From Emp Where Deptno=10
Sql>Select Ename,Lpad(Ename,10),Rpad(Ename,10,'-')FromEmpWhere
Deptno=10;
Sql> Select Ename,Lpad(Rpad(Ename,10,'-'),15,'-') From Emp
```

Real Time Scenario:

LTRIM Function:

- It enables to trim heading character from a character string.
- All the left most characters that appear in the set are removed.

Syntax: LTRIM(Char,Set)

```
Sql> Select Ltrim('xyxyORACLE 10g','xy') From Dual;
Sql> Select Ltrim('MM KRISHNA','M') From Dual;
```

RTRIM Function:

- It enables to trim heading character from a character string.
- All the right most characters that appear in the set are removed.

Syntax: RTRIM(Char,Set)

```
Sal> Select Rtrim('ORACLE 10gxyxy','xy') From Dual;
Sql> Select Rtrim('KRISHNA AAA','A') From Dual;
Sql> Select Rtrim(Job,'ER'),Job From Emp Where Ltrim(Job,'MAN') Like 'GER';
```

TRIM Function:

- Trims heading or trailing characters (or both) from a character string.
- If trim_character or trim_source is a character literal, you must enclose it in single quotes.
- If **Leading** is specified concentrates on leading characters.
- If **Trailing** is specified concentrates on trailing characters.
- If **Both** or none is specified concentrates both on leading and trailing.
- Returns the varchar2 type.

```
Sql> Select Trim( 'S' From 'MITHSS') From Dual;
Sql> Select Trim( 'S' From 'SSMITH') From Dual;
Sql> Select Trim( 'S' From 'SSMITHSS') From Dual;
Sql> Select Trim(Leading  'S' From 'SSMITHSS') From Dual
Sql> Select Trim(Trailing 'S' From 'SSMITHSS') From Dual;
Sql> Select Trim(Both  'S' From 'SSMITHSS') From Dual;
```

REPLACE Function:

- It return the every Occurrence of search string replace by the replacement string.
- If the replacement string is omitted or null, all occurrences of serch string are removed.
- If substitutes one string for another as well as to remove character strings.

Syntax: REPLACE(text,Search_string, [Replacement_string])

```
Sql> Select Replace('Led','L','R') From Dual;
Sql> Select Replace('Led','L','Ra') From Dual;
Sql> Select Replace('Led','Le','R') From Dual;
Sql> Select Ename,Replace(Job,'MAN','DAM') From EmpWhere Job='MANAGER';
Sql> Select Job,Replace(Job,'P') From Emp Where Job='PRESIDENT';
Sql> Select Job,Replace(Job,'MAN','EXECUTIVE') From EmpWhere
Job='SALESMAN';
```

Real Time Scenario:

TRANSLATE Function:

- Used to Translate Character by character in a String.
  Syntax: TRANSLATE(char,From ,To)
- It returns a char with all occurrences of each character in 'From' replaced by corresponding character in 'To'.
- Characters in char that are not in From are not replaced.
- The argument From can contain more characters than To.
- If the extra characters appear in Char, they are removed from the return value.

```
Sql> Select Translate(Job,'P',' ') From Emp Where Job='PRESIDENT';
Sql> Select Translate(Job,'MN','DM') From Emp Where Job='MANAGER';
Sql> Select Job,Translate(Job,'A','O') From Emp Where Job='SALESMAN';
Sql> Select Translate('Led','Le','R') From Dual;
```

Real Time Scenario:

CHR Function:

It returns a character having the binary equivalent to 'n'.

- It returns the equivalent for 'n' in database character set or national character set.

Syntax: CHR(n)

Sql>Select Chr(75)||Chr(82)||Chr(73)||Chr(83)||Chr(72)||Chr(78)||Chr(65) Name From Dual;

ASCII Function:

- It returns the decimal representation in the character database set of the first characters of the Char.

Syntax: ASCII(Char)

Sql> Select Ascii('A') From Dual;

Sql> Select Ename,Ascii(Ename) From Emp;

Sql> Select Ascii('&name') From Dual;

NUMBER Function:

- These function accept number input and return numeric values.
- Many functions return values that are accurate to 38 decimal digits.

ROUND Function:

Systax: ROUND(m,n)

- It returns 'm' round to 'n' places right of the decimal point.
- If 'n' omitted , n is rounded to 0, places
- 'n' can be negative , and rounds off the digits to the left of the decimal point.
- 'n' must be an integer.

Sql>Select 19.637 Num1ROUND(19.637,1) Rounded From Dual;

Sql>Select 19.637 Num-,ROUND(19.637,-1) Rounded From Dual;

Sql>Select 7843.637 Num1_,ROUND(7843.637,2) Rounded, ROUND(7843.637,-1) Rounded,ROUND(7843.637,-2)Rounded,ROUND(7843.637,-3)Rounded, ROUND(7843.637,-4)Rounded From Dual;

TRUNCATE Function:

Systax:TRUNC(m,n)

- It returns 'm' Truncated to 'n' decimal places.
- If 'n' omitted , n is truncated to 0 decimal places.
- 'n' can be negative to truncate 'm' digits left of the decimal point.

Sql>Select 19.637 Num1,TRUNC(19.637,1) Truncated From Dual;

Sql> Select 19.637 Num1-, TRUNC(19.637,-1) Truncated From Dual;

Sql> Select 7843.637 Num1, TRUNC(7843.637, 2) Truncated, TRUNC(7843.637,-1) Truncated,TRUNC(7843.637,-2)Truncated,TRUNC(7843.637,-3)Truncated, TRUNC(7843.637,-4) Truncated From Dual;

CEIL Function:

Syntax:CEIL(n)

- Returns the Largest integer greater than or equal to 'n'.
- The adjustment is done to the highest Nearest decimal value.

Sql>Select 19.001 Num1,CEIL(19.001) Ceiled From Dual;

Sql> Select 19.34 Num1,CEIL(19.32) Ceiled ,CEIL(19.2) Ceiled ,CEIL(19) Ceiled From Dual;

FLOOR Function:

Syntax: FLOOR(n)

- Returns the smallest integer less than or equal than 'n'.
- The adjustment is done to the lowest Nearest decimal value.

Sql>Select 19.001 Num1, FLOOR(19.999) Floor From Dual;

Sql> Select 18.34 Num1,FLOOR(18.34) Floor,FLOOR(18.9) Floor,FLOOR(18) Floor From Dual;

MODULUS Function:

Syntax: MOD(m,n)

- It Returns remainder of 'm' divided by 'n'.
- It returns 'm' if 'n' is 0.

Sql> Select MOD(100,10) Modulus, MOD(17,4) Modulus From Dual;

POWER Function:

Syntax: POWER(m,n)

- It Returns 'm' Raised to the 'n'th power.
- The base 'm' and the exponent 'n' can be any number.

Sql> Select POWER(5,2) Power, POWER(-5,2)Power From Dual;

Sql> Select POWER(5,-2) Power, POWER(-5,-2)Power From Dual;

SQUARE Function:

Syntax: SQRT(n)

- It Returns Square Root of 'n' as Real Value.
- The Value of 'n' cannot be negative.

Sql> Select SQRT(25) From Dual;

ABSOLUTE Function:

Syntax: ABS(n)

- It Returns the Absolute value of 'n'

Sql> Select ABS(-100) From Dual;

Sql>Select Sal, Comm, Sal-Comm, ABS(Sal-Comm) FROM Emp;

SIGN Function:

Syntax: SIGN(n)

- It Returns the SIGN, Specification of a number.
  - If n<0,returns -1
  - If n=0,returns 0
  - If n>0,returns 1

Sql> Select SIGN(-10), SIGN(10), SIGN(0) From Dual;

Sql> Select Sal, Comm, SIGN(Sal-Comm), ABS(Sal-Comm) FROM Emp
Where SIGN(Sal-Comm)=-1

Working With Dates:

- Oracle stores dates in an internal numeric format.
- The dates in Oracle range from January 1,4712 BC to December 31,9999 AD.
- The default display and input format for any date is DD-MON-YY.
- The numeric format represents

**Century **Year **Month **Day **Hours
**Minutes **Seconds

SYSDATE:

- It is a date function that returns current date and time.
- SYSDATE is generally selected upon a DUAL Table.

Sql>Select SYSDATE From Dual;

Date Arithmetic:

- As database stores dates as numbers, it allows to perform calculations using arithmetic operators such as addition and subtraction.
- We can perform the following operations.....
  - Date + Number Date Adds a number of days to a date
  - Date - Number Date Subtracts a number of days from a date .
  - Date - Date Number of days Subtracts one date from another.
  - Date + Number/24 Date Adds a number of hours to a date.

Sql>Select Sysdate From Dual;

Sql>Select Sysdate,Sysdate+10 From Dual;

Sql>Select Sysdate,Sysdate+48/24 From Dual;

Sql>Select Ename,Hiredate,Hiredate+10 From Emp;

Sql>Select Ename,Hiredate,Hiredate-5 From Emp;

Sql>Select Ename,Hiredate,Sysdate-Hiredate "ExofEmps" From Emp;
Sql> Select Ename,Round((Sysdate-Hiredate)/7) Weeks From Emp;
Sql>Select Empno,Hiredate,Round((Sysdate-Hiredate)/365) From Emp;

DATE Function:

Add_ months Function:

Syntax: ADD_MONTHS(D, +(or)-N)

- Adds 'N' number of calendar months to date.
- The value of 'N' must be an integer and can be negative.

Sql> Select Sysdate, Add_months(Sysdate,1) From Dual;
Sql> Select Ename, Sal, Hiredate, Add_months(Hiredate,1) From Emp Where Deptno=30;

Months_Between Function:

Syntax: Months_between(D1,D2)

- It gives the different between dates D1 and D2 In months.
- If D1 is later than D2, the result is Positive, else Negative.
- If D1 and D2 are either the same days of the months or both last days of the months, the result is always an integer.

Sql>Select Ename,Hiredate, Round(Months_Between(Sysdate,Hiredate)/12) " Experience In Years" From Emp;
Sql> Select Ename,Hiredate,Months_Between(Sysdate,Hiredate) From Emp Where Months_Between(Sysdate,Hiredate)<320;

Next_Day Function:

Syntax: Next_day (D, Char)

- It returns the date of the first week day named by char, that is later than the data D.
- The CHAR must be a day of the week in the sessions data language.
- The day of the week can be full name or the abbreviation.

Sql>Select Sysdate, Next_day(Sysdate,'WED') From Dual;
Sql>Select Sal,Hiredate,Next_day(Hiredate,'MONDAY') From Emp;

Last_Day Function:

Syntax: Last_day(D)

- It returns the date of the last day of the month that contains D.
- Mostly used to determine how many days are left in the current month.

Sql> Select Sysdate, Last_day(Sysdate) Last, Last_day(Sysdate)-Sysdate Daysleft From Dual ;

Real Time Scenario:

Sql> Select Add_months(Last_day(Sysdate),-1)+1 From Dual;

Rounding of Dates:

Syntax: Round(Date,'Format')

- Returns Date rounded to the Unit specified by the format.
- If format is omitted, Date is rounded to the nearest day.

Sql> Select Round(Sysdate,'DAY') From Dual;
Sql> Select Round(Sysdate,'MONTH') From Dual;
Sql> Select Round(Sysdate,'YEAR') From Dual;

Truncating Dates:

Synatx:Trunc(Date,'Format')

- Return Date with the time portion of the day truncated to the specified unit.
- If format is omitted, data is truncated to the nearest day.

Sql> Select Round(Sysdate,'DAY'), Trunc(Sysdate,'DAY') From Dual;

Sql> Select Round(Sysdate,'MONTH'), Trunc(Sysdate,'MONTH') From Dual;
Sql> Select Round(Sysdate,'YEAR'), Trunc(Sysdate,'YEAR') From Dual;

Conversion Function:

- The Conversation functions convert a value from one data type to another.
- The Data type conversion in Oracle is two types.
  **Implicit Data type Conversion
  **Explicit Data type Conversion

Implicit Data type Conversion:

- Implicit Date type conversion work according to the convention specified by oracle.
- The Assignment succeeds if the Oracle server can convert the date type of value.
- CHAR to NUMBER conversion succeed only if the character string represent a valid NUMBER.
- CHAR to DATES conversion succeed only if the character string represent the default format of DD-MON-YY.

In Assignment Operator:

- Varchar2/Char →Number
- Varchar2/Char →Date
- Number →Varchar2
- Date →Varchar2

Explicit Data type Conversion:

- SQL provided three function to convert a value from one data type to another.
- The explicit conversation function are
  - TO_CHAR → To Character Conversion.
  - TO_DATE → To Date Conversion.
  - To_Number → To Number Conversion

TO_CHAR Conversion:

- This function can be used in two ways.
  - TO_CHAR(Number Conversion)
  - TO_CHAR(Date Conversion)

TO_CHAR(Number Conversion)

Syntax: TO_CHAR(NUMBER,fmt)

- Converts Number of Number data type to a value of VARCHAR2 data type.
- 'fmt' is the optional number format, that can be used.

TO_CHAR(Date Conversion)

Syntax: TO_CHAR(DATE,fmt)

- Converts Date of Date data type to a value of VARCHAR2 data type in the format specified.
- 'fmt' is the optional Date format, that can be used.

Decimal Indicator:D→99D99

- It returns the specified position of the decimal character .
- The default decimal delimiter is period '.'
- Only one decimal indicator can be specified in a number format model.

Sql> Select 1234,TO_CHAR(1234,'9999D99') From Dual;
Sql> Select 1234,TO_CHAR(1234,'999D99') From Dual;

Scientific Notation Indicator:EEEE→9.9EEEE

- Returns a Numeric value using scientific notation.
  Sql> Select TO_CHAR(5634,'9.9EEEE') From Dual;

Group Separator: G→9G999

- Returns the specified position of the Group separator
- Multiple Group separators can be specified.

Sql> Select TO_CHAR(1234567,'99G99G9999') From Dual;
Sql> Select Sal,TO_CHAR(Sal,'9G999') From Emp;

Local Currency Indicator: L→L999 OR 999L

- Returns the specified position of the local currency symbol.

```
Sql>Select 1234,TO_CHAR(1234,'L9999')  From Dual;
Sql>Select Sal,TO_CHAR(Sal,'L999999') Currency From Emp  Where Deptno=10;
Sql> Select Sal,TO_CHAR(Sal,'L99G999D99','NLS_CURRENCY=IndRupees') Sal
From Emp Where Deptno=20;
```

Trailing Minus Indicator: MI→9999MI

- Return negative value with a trailing minus sign'-'.
- Returns positive value with a trailing Blank.
- 'MI' Format should be declared as Trailing argument only.

```
Sql>select -10000,TO_CHAR(-10000,'L99G999D99MI') From Dual ;
Sql> select Sal,Comm,Comm-Sal,  TO_CHAR(Comm-Sal,'L99G999D99MI') From
Emp;
```

Negative Number Indicator:PR→9999PR

- Returns negative number in '<>'.
- It Can appear only as trailing declaration.

```
Sql> select TO_CHAR(-1000,'L99G999D99PR') From Dual;
Sql> select Sal,Comm,Comm-Sal, TO_CHAR(Comm-Sal,'L9999PR') From Emp;
```

Roman Number Indicator:

- RN→Returns Upper Roman Number.
- rn→ Returns Lower Roman Number.
- The value can be an integer between 1 and 3999.

```
Sql> select 12,TO_CHAR(12,'RN'),TO_CHAR(12,'rn') From Dual;
```

Sign Indicator:S→S99999 OR 99999S

- Returns Negative value with a leading Minus Sign.
- Returns Positive value with a leading Plus Sign.
- Returns Negative value with Trailing Minus Sign.
- Returns Positive value with a Trailing Plus Sign.
- 'S' can appear as First or Last value.

```
Sql> select 1000,TO_CHAR(1000,'S9999'),TO_CHAR(-1000,'S9999') From Dual;
Sql>select TO_CHAR(1000,'9999S'),TO_CHAR(-1000,'9999S')  From Dual;
Sql>select Sal,TO_CHAR(Sal,'S99999'),TO_CHAR(Sal,'9999S') From Emp;
Sql>select Sal,Comm,TO_CHAR(Comm-Sal,'S99999'), TO_CHAR(Comm-Sal,'9999S')
                                                              From Emp;
```

Hexadecimal Indicator: X→XXXX

- Returns the Hexadecimal value of the specified number of Digits.
- If the Number is not an integer, oracle rounds it to an integer.
- Accepts only positive values or 0.

```
Sql>select 2000,TO_CHAR(2000,'XXXX') From Dual;
Sql> Select Ename,Sal,TO_CHAR(Sal,'XXXX') Hexsal   From Emp;
```

Group Separator:→9,999

- Returns a comma in the specified position.
- Multiple commas can be specified.

```
Sql> select 20000,TO_CHAR(20000,'99,999.99') From Dual;
Sql> Select Ename,Sal,TO_CHAR(Sal,'99,999.99') From Emp;
```

Decimal Indicator:→99.99

- Returns a decimal point, at the specified position.
- Only one period can be specified in a number format model.

```
Sql> Select 20000,TO_CHAR(20000,'L99,999.99') From Dual;
Sql> Select Ename,Sal,TO_CHAR(Sal,'L99,999.99') From Emp;
```

Dollar Indicator: $ →$9999

- Return value with a leading dollar sign.

```
Sql> Select 20000,TO_CHAR(20000,'$99,999.99') From Dual;
Sql> Select Ename,Sal,TO_CHAR(Sal,'$99,999.99') From Emp;
```

Zero Indicator:0→0999 OR 9990

- Returns Lading OR Trailing Zeros.

```
Sql>select 1000,TO_CHAR(1000,'0999999'),TO_CHAR(1000,'09999990') From Dual;

Sql> select Ename,Sal,TO_CHAR(Sal,'$099,999.99') From Emp;
```

Digit Place Marker: 9→9999

- Returns value with a specified number of digits with a Leading space when positive or Leading minus when Negative.

```
Sql> select 1000,600,TO_CHAR(1000-600,'99999'), TO_CHAR(600-1000,'99999')
From Dual;
 Sql>select 20.25,20,TO_CHAR(20.55-20,'99999'), TO_CHAR(20.25-20,'99999')
From Dual;
```

ISO Currency Indicator: C→C9999

- Return specified position of the ISO Currency Symbol.

```
Sql>select 1000,TO_CHAR(1000,'C9999.99')  From Dual;
Sql>select Ename,Sal, TO_CHAR(Sal,'C9999.99') From Emp;
```

Date Format Models:

- The date format models can be used in the TO_CHAR function to translate a DATE value from original format to user format.

Data Format Elements:

- A Date format model is composed of one or more date format elements.
- For input format models, format items cannot appear twice, and format items that represents similar information cannot be combined.
- Capitalization in a spelled word, abbreviation, or roman numeral follows capitalization in the corresponding format element.
- Punctuation such as Hyphens, Slashes, Commas, periods and Colons.

AD or A.D./BC or B.C. Indicator:

- Indicator AD/BC with or without periods.

```
Sql>Select Sysdate,TO_CHAR(Sysdate,'AD') From Dual;
Sql>Select TO_CHAR(Sysdate,'B.C.'), TO_CHAR(Sysdate,'A.D.')  From Dual;
Sql>Select Ename,Sal,Hiredate,TO_CHAR(Hiredate,'A.D.')  From Emp;
```

Meridian Indicator:

- It indicates meridian indicator with or without periods.

```
Sql>Select Sydate,TO_CHAR(Sysdate,'A.M.'),TO_CHAR(Sysdate,'PM')  From Dual;
Sql>Select Ename,Sal,Hiredate,TO_CHAR(Hiredate,'AM')  From Emp;
```

Century Indicator: CC

- Indicates the century.

```
Sql> Select Sydate,TO_CHAR(Sysdate,'CC-AD')  From Dual;
```

Numeric Week day indicator:D→(1-7)

- Returns the week day number.

```
Sql>Select Sysdate,TO_CHAR(Sysdate,'D') From Dual;
Sql>Select Ename,Sal,Hiredate,TO_CHAR(Hiredate,'D') From Emp;
```

Week day spelling indicator:→Day.

- Pads to a length of 9 characters.

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'DAY')  From Dual;

Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'DAY') From Emp Where
                                        TO_CHAR(Hiredate,'DY')='MON';
```

Month day Indicator:DD

- It indicates the day of the Month(1-31)

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'DD-DAY')  From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'DD_DAY') From Emp;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'DY') From Emp
                                    Where TO_CHAR(Hiredate,'DD-DY')='28-MON';
```

Year day indicator:DDD

- It indicates the day of the Yaer(1-366)

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'DDD')  From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'DDD')  From Emp;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'DDD')  From Emp
                                Where TO_CHAR(Hiredate,'DY')='MON';
```

Abbreviated week day: DY

- It indicates the day of the Yaer(1-366)

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'D-DY-DAY') From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'D-DY-DAY')  From Emp
                                        Where Deptno in(10,20);
```

ISO standard year week indicator: IW

- Specifies the week of the yaer(1-52 or 1-53) based on the ISO standard.

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'IW') From Dual;
Sql>Select Ename,Hiredate,TO_CHAR(Hiredate,'IW') From Emp;
```

ISO standard 4 digit year indicator: IYYY

- Specifies 4 digit year based on the ISO standard.
- It can even be used in combination of IYY,IY,I.

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'IYYY') From Dual;
Sql>Select Ename,Hiredate,TO_CHAR(Hiredate,'IYYY') From Emp
                                Where TO_CHAR(Hiredate,'DY')='MON';
```

Four Digit Year Indicator:YYYY or SYYY

- Return four digit year,'S' prefixes BC dates with '-'.
- It can even be used in combination of YYY or YY or Y.
- Y,YYY returns year with comma in that position.

```
Sql>Select to_char(sysdate,'syyyy') from dual
Sql>Select Hiredate,TO_CHAR(Hiredate,'YYYY'),  TO_CHAR(Hiredate,'YYY') From
Emp;
 Sql>Select TO_CHAR(Sysdate,'YYYY'),TO_CHAR(Sysdate,'YYY'),
TO_CHAR(Sysdate,'YY') From Emp;
```

Spelled Year Indicator: YEAR or SYEAR

- Returns the numerical year in spelling.

```
Sql>   Select SYSDATE, TO_CHAR(Sysdate,'Year'),TO_CHAR(Sysdate,'YEAR')  From
Dual;
Sql>   Select Empno,Ename,Hiredate,TO_CHAR(Hiredate,'Year') From Emp;
```

Week of the month indicator: W

- Specifies the week of the month(1-5).
- Week starts on the first day of the month and ends on the seventh day.

```
Sql>  Select Sysdate,TO_CHAR(Sysdate,'W') From Dual;
Sql>  Select Empno,Ename,Hiredate,TO_CHAR(Hiredate,'W') From Emp;
```

Year week Indicator:WW

- Specifies the week of the year(1-53)
- Week 1 starts on the first day of the year and continues to the seventh day in that year.

```
Sql>  Select Sysdate,TO_CHAR(Sysdate,'WW') From Dual;
Sql>  Select Empno,Ename,Hiredate,TO_CHAR(Hiredate,'WW') From Emp;
```

Quarter of the Year Indicator:Q

- Returns the Quarter if the year.
- Quarter starting with the month of January and ending with every three months.

```
Sql>  Select Sysdate,TO_CHAR(Sysdate,'Q') From Dual;
Sql>  Select Empno,Ename,Hiredate,TO_CHAR(Hiredate,'Q') From Emp Where
                                                    O_CHAR(Hiredate,'Q')=3;
```

Julian Day Indicator: J

- Returns the JULIAN Day of the given date.
- It is the number of day since January 1,4712 BC.
- Number specified with 'J' must be integers.

```
Sql>  Select Sysdate,TO_CHAR(Sysdate,'J')   From Dual;
Sql>  Select Empno,Ename,Hiredate,TO_CHAR(Hiredate,'J-DDD-DD-D') From Emp;
```

Number Month Indicator:MM

- Returns the numeric abbreviation of the month.

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'MM-YYYY') From Dual;
Sql> Select Ename,TO_CHAR(Hiredate,'DD-MM-YYYY') From Emp Where
                                                TO_CHAR(Hiredate,'MM')=12;
```

Abbreviated Month Indicator:MON

- Returns the abbreviated name of the Month.

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'MON')   From Dual;
Sql> Select Hiredate,TO_CHAR(Sysdate,'MONTH') From Emp;
```

Twelve Hour Clock Mode: HH or HH12

- It is default clock mode.
- Returns the hour of the day in twelve hour clock mode.

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'HH'), TO_CHAR(Sysdate,'HH12,PM')
From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'HH12:PM')  From Emp;
```

Twenty Hour Clock Mode:HH24

- Returns the hour of the day in twenty four hour clock mode.(0-23)

```
Sql> Select Sysdate,TO_CHAR(Sysdate,'HH24') From Dual;
```

Real Time Scenario:

Minutes Indicator: MI

- Returns the minutes from the given date(0-53).

Sql> Select Sysdate,TO_CHAR(Sysdate,'MM') From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'HH:MM') From Emp Where Deptno=20;

Roman Month Indicator: RM

- Return the roman numeral month(I-XII).

Sql> Select Sysdate,TO_CHAR(Sysdate,'RM'), TO_CHAR(Sysdate,'DD-RM-YY') From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'DD-RM-YY') From EmpWhere Deptno=20;

Seconds Indicator: SS

- Returns seconds from the given the dates(0-59)

Sql> Select Sysdate,TO_CHAR(Sysdate,'SS'), TO_CHAR(Sysdate,'HH:MI-SS') From Dual;
Sql> Select Sysdate,TO_CHAR(Sysdate,'RM'), TO_CHAR(Sysdate,'Day-Month-Year HH24:MI:SS PM.') From Dual;
Sql> Select Ename,Hiredate,TO_CHAR(Hiredate,'HH24-MI-SS') From EmpWhere Deptno=30;

Date Format Punctuators:

- The punctuation marks that can be used in date formats are...
  '-', '/', '!', '.', ';', ':', 'text'

Date Format element suffices: TH or SP

- TH→Suffixes the ordinal number with 'st' or 'nd' or 'rd' or 'th'.

Example:

Sql>Select Sysdate,TO_CHAR(Sysdate,'DDth,MONTH,YYYY') From Dual;
Sql>Select Ename,Hiredate, TO_CHAR(Hiredate,'DDth,MONTH,YYYY') From Emp;

SP→Spells ordinal numbers.

Sql>Select Sysdate,TO_CHAR(Sysdate,'DDsp,MONTH,YYYY') From Dual;
Sql>Select Ename,Hiredate,TO_CHAR(Hiredate,'DDsp,MONTH,YYYY') From Emp;
Sql>Select Ename,Hiredate, TO_CHAR(Hiredate,'DDspth,MONTH,YYYY') From Emp;
Sql>Select Sysdate,TO_CHAR(Sysdate,'DDspth Month YYYYSP') From Dual;

Date Format Elements restrictions:

- The suffixes when added to date return values always in English.
- Data suffixes are valid only on output, hence cannot be used to insert a data into the database.

Format Model Modifiers:

Fill Mode Indicator :FM

- It suppresses blank padding in the return value of the TO_CHAR function.

Sql> Select Sysdate,TO_CHAR(Sysdate,'DDSPTH MONTH YYYYSP'), TO_CHAR(Sysdate,'FMDDSPTH MONTH YYYYSP') From Dual;

TO_NUMBER Function:

Syntax:TO_NUMBER(Char,fmt)

- It convert a char ,value of CHAR or VARCHAR2 data type containing a NUMBER in the format specified by the optional format model 'fmt', to a value of NUMBER data type.

Sql> Select '$10,000.00',TO_NUMBER('$10,000.00','L99,999.99') From Dual;
Sql> Select '$10,000.00', TO_NUMBER('$10,000.00','L99,999.99')+500 From Dual;

TO_DATE function:

Syntax:TO_DATE(Char,'fmt')

- Converts given char of Char or Varchar2 data type to a value of DATE data type.
- The 'fmt' is optional Date format specified the format of CHAR.

```
Sql> Select TO_CHAR(TO_DATE('12-JAN-1980'),'DDSP')  From Dual;
Sql> Select Ename,Hiredate, ADD_MONTHS(TO_DATE ('1980-DECEMBER-17','YYYY-
                    MONTH-DD'),3)  FROM Emp WHERE Hiredate='17-DEC-1980';
Sql>Select Ename,Hiredate, ADD_MONTHS(TO_DATE('1980-DECEMBER-17','YYYY-
MONTH-DD'),3) FROM Emp WHERE  TO_CHAR(Hiredate,'FMYYYY-MONTH-DD')=
                                                '1980-DECEMBER-17';
```

INSERT Statement With Invalid Number:

```
Sql>Create Table Temp      ( Empno Number(8),  Doj Date );
Sql>Insert into Temp Values(    TO_NUMBER('1,23,456','9G99G999'), SYSDATE
);
Sql> Insert into Temp Values( TO_NUMBER('1,23,456-','9G99G999MI'), SYSDATE
);
```

Invalid Dates:

```
Sql>Insert Into Temp Values(1001,'12-JAN-09 11:30:15');
Sql> Insert Into Temp Values(1001,TO_DATE('12-JAN-2009 11:30:15 P.M.','DD-
MON-YYYY HH:MI:SS P.M' );
Sql> Insert Into Temp Values(1001,TO_DATE('12-JAN-3712 B.C.','DD-MON-YYYY
B.C. '));
Sql> Insert Into TempValues(1001,TO_DATE(100,'J'));
Sql>Select TO_DATE(10,'DD') From Dual;
Sql> Select TO_DATE(10,'DDD') From Dual;
Sqi>Select TO_DATE(1,'W') From Dual;
Sql> Select TO_DATE(1,'WW') From Dual;
```

RR Date Format Elements:

- The RR date format element is similar to the YY date format element.
- Te RR format elements provides additional flexibility for storing data values in other centuries.
- The RR date format element allows to store the date to the previous as well as the next centuries.
- The RR format elements should be used in association with TO_DATE conversion function only, else the system treats it as YY format element.
- The format can be used as either 'RR' or 'RRRR' format element.

Century Identification:

Last Two Digits of the year managed by clock	0 To 49	50 To 99
	Returns the date in Current century	Returns the date in the previews Century.

Some Combinations

```
Sql>Select TO_CHAR( ADD_MONTHS(Hiredate,1), 'DD-MON-YYYY' ) "Next Month"
                                        From Emp Where Empno=7788;
Sql> Select Concat( Concat(Ename, 'is a '), Job )Designation From Emp Where
Deptno=20;
Sql> Select Trunc( TO_DATE('27-OCT-92','DD-MON-YY'), 'YEAR' ) "New Year"
From Dual;
Sql>Select TO_CHAR(  ADD_MONTHS(  LAST_DAY(Hiredate),5),'DD-MON-YYYY' )
                    "Five Months" From Emp  Where Empno=7788;
Sql>Select MONTHS_BETWEEN(TO_DATE( '02-02-1995','MM-DD-YY' ),
                        TO_DATE( '02-02-1995','MM-DD-YYYY' ))Months  FROM
Dual;
Sql>Select NEXT_DAY('15-MAR-98','TUESDAY') From Dual;
```

Sql>Select Ename,NVL(TO_CHAR(Comm),'Not Applicable') "Commission" From Emp WHERE Deptno=30;

Sql>Select Round(TO_DATE('27-OCT-92','DD-MON-YY'),'YEAR')"New Year"From Dual;

Sql>Select TO_CHAR(Hiredate,'MONTH DD,YYYY') From Emp Where Ename='FORD';

Sql>Select Ename, TO_CHAR(Hiredate,'FMMonth,DD YYYY') Hiredate From Emp Where Deptno=20;

Sql> Select TO_CHAR(TO_DATE('27-OCT-98','DD-MON-RR'),'YYYY') Year From Dual;

- Assumption: Queries are issued Between Year 1950-1999.

Sql> Select TO_CHAR(Sysdate,'FMDDTH')||' of '|| TO_CHAR(Sysdate,'Month')||' , '|| TO_CHAR(Sysdate,'YYYY')Idea From Dual;

Sql>Select Ename,Job,NVL(TO_CHAR(MGR),'Supreme Authority') "Managers" From Emp Order by Sal Desc;

Spelling a Number:

Sql> Select TO_CHAR(TO_DATE('&GiveNumber','J'), 'JSP') "Spelled Number" From Dual;

Selecting a Date Specific to its Century:

Sql>Select TO_CHAR(TO_DATE(Hiredate,'DD-MON-RRRR'), 'DD-MON-YYYY') Hiredate From Emp;

Text Encryption:

Sql>Select 'KRISHNA REDDY' OrgName,Translate('KRISHNA REDDY', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',1234567890!@#$%^&*()-=_+;,.')Encryptedname From Dual;

Text Decryption:

Sql> Select '!*9(8$1 *544;' Encryptedname, Translate('!*9(8$1 *544;', '1234567890!@#$%^&*()-=_+;,', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')Encryptedname From Dual

Aggregating or Group Functions:

- Group functions operate on sets of rows to give one result per group.
- Group functions can appear in select lists and in ORDER BY and HAVING clauses.
- The Oracle Server applies the group functions to each group of rows and returns a single result row for each group.
- The group functions to return summary information for each group.

Syntax:

group_function(DISTINCT/ALL Column)

Guidelines for Using Group Functions:

- The data types for the arguments can be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions except COUNT(*) ignore null values. To substitute a value for null values, use he NVL function. COUNT returns either a number or zero.
- The Oracle Server implicitly sorts the result set in ascending order of the grouping columns specified, when you use a GROUP BY clause. To override this default ordering, you can use DESCin an ORDER BY clause.
- When a group function is declared in a SELECT list, on single row columns should be declared.

Average Function:

Syntax: AVG(DISTINCT/ALL Column)

- It returns the Average value of column.
- It ignores NULL values.

Sql>Select Avg(Sal),Avg(DISTINCT Sal) From Emp;
Sql> Select Avg(Comm),Avg(DISTINCT Comm) From Emp;

Sum Function:
Syntax: SUM(DISTINCT/ALL Column)

- It returns the SUM value of column.
- It ignores NULL values.

Sql>Select SUM(Sal),SUM(DISTINCT Sal) From Emp;
Sql> Select SUM (Comm), SUM (DISTINCT Comm) From Emp;

Maximum Function:
Syntax: MAX(DISTINCT/ALL Column)

- It returns the Maximum value of column.
- It ignores NULL values.

Sql>Select MAX(Sal),MAX(DISTINCT Sal) From Emp;
Sql> Select MAX (Comm), MAX (DISTINCT Comm) From Emp;
Sql> Select MAX(Ename) From Emp;

Minimum Function:
Syntax: MIN(DISTINCT/ALL Column)

- It returns the Minimum value of column.
- It ignores NULL values.

Sql>Select MIN(Sal),MIN(DISTINCT Sal) From Emp;
Sql> Select MIN (Comm), MIN (DISTINCT Comm) From Emp;
Sql> Select MIN(Ename) From Emp;
Sql> Select MIN(Hiredate),Max(Hirdate) From Emp;

Standard Deviation Function:
Syntax: STDDEV(DISTINCT/ALL Column)

- It returns the Standard Deviation of column.
- It ignores NULL values.

Sql>Select STDDEV(Sal),STDDEV(DISTINCT Sal) From Emp;
Sql> Select STDDEV (Comm), STDDEV (DISTINCT Comm) From Emp;

Variance Function:
Syntax: Variance(DISTINCT/ALL Column)

- It returns the Variance of N.
- It ignores NULL values.

Sql>Select VARIANCE(Sal), VARIANCE (DISTINCT Sal) From Emp;
Sal> Select VARIANCE (Comm), VARIANCE (DISTINCT Comm) From Emp;

Count Function:
Syntax: COUNT(*/DISTINCT/ALL Column)

- It Gives No of Rows in the Query.
- If '*' used to returns all Rows, Including duplicated and NULLs.
- It can used to specify the count of all rows or only distinct values of column.

Sql>Select COUNT(Empno) From Emp;
Sql> Select COUNT(Job),COUNT(DISTINCT Job) From Emp;
Sql> Select COUNT(Sal),COUNT(DISTINCT Sal) From Emp;
Sql>Select Count(*) From Emp;
Sql> Select COUNT(Empno),COUNT(DISTINCT MGR) From Emp;
Sql>Select Select COUNT(Job),COUNT(DISTINCT MGR) From Emp Where Deptno=20;

Creating Groups of Data:

- The Group by clause is used to decide the rows in table into groups.

Syntax1:

```
Sql>SELECT columnname1, Columnname2,.....
      FROM table
      [WHERE condition(s)]
      [GROUP BY Column name(s)]
      [ORDER BY column(s)];
```

Syntax2:

```
Sql>SELECT columnname1, GRP_FUN(Column)
      FROM table
      [WHERE condition(s)]
      [GROUP BY Columnname(s)]
      [ORDER BY column(s)];
```

Guidelines to use Group By Clause:

- All GROUP BY CLAUSE columns list may or may not used in SELECT clause.
- The Extra non Group functional column should be declared in the GROUP BY Clause.
- If the GROUP Function is included in a SELECT clause, we should not use individual result columns.
- Using WHERE clause, Rows can be pre excluded before dividing then into groups.
- Column aliases cannot be used used in GROUP BY CLAUSE.
  By default, Rows are sorted by ascending order of the columns included in the GROUP BY LIST.

Example:

```
Sql> Select Job  From Emp  GROUP BY Job;
Sql> Select Deptno  From Emp GROUP BY Deptno;
Sql> Select Mgr  From Emp GROUP BY Mgr;
Sql> Select TO_CHAR(Hiredate,'YYYY') YearGroup  From Emp
                                  GROUP BY TO_CHAR(Hiredate,'YYYY');
Sql> Select TO_CHAR(Hiredate,'Month') MonthGroup
          From Emp GROUP BY TO_CHAR(Hiredate,'Month');
Sql> Select TO_CHAR(Hiredate,'DD') DayGroup
            From Emp  GROUP BY TO_CHAR(Hiredate,'DD');
Sql> Select TO_CHAR(Hiredate,'Month') MonthGroup
       From Emp Where TO_CHAR(Hiredate,'Mon')<>'Dec'
                                  GROUP BY TO_CHAR(Hiredate,'Month');
```

Creating Group wise summaries:

```
Sql>Select Owner,COUNT(Object_Name) From dba_objects  Group By Owner;
Sql> Select Deptno,COUNT(*)  From emp  Group By Deptno;
Sql> Select Deptno, AVG(Sal)  From Emp  Group By Deptno Oredr By AVG(Sal);
Sql> Select Deptno,Min(Sal),Max(Sal)  From Emp Group by Deptno;
Sql> Select Ename, Max(Sal) From Emp Group By Deptno;
```

Note: It will give error .

```
Sql>Select Deptno,SUM(Sal) From Emp Group By Deptno;
Sql>Select Deptno,Job,SUM(Sal)  From Emp GROUP BY Deptno,Job;
Sql> Select Deptno,Job,SUM(Sal)  From Emp  GROUP BY Deptno,Job;
Sql> Select Job,Min(Sal),Max(Sal)  From Emp Where Deptno=30  GROUP BY Job;
Sql>Select Deptno,Sum(Sal),Max(Sal)  From Emp Where Job='CLERK'  GROUP BY
Deptno;
```

OCP Questions:

1)Examine the description of the STUDENTS table:

STD_ID NUMBER(4)
COURSE_ID VARCHARD2(10)

START_DATE DATE
END_DATE DATE
Which two aggregate functions are valid on the START_DATE column? (Choose two)
A. SUM(start_date) B. AVG(start_date)
C. COUNT(start_date) D. AVG(start_date, end_date)
E. MIN(start_date) F. MAXIMUM(start_date)

Having Clause:

You may not want all the summary rows returned by a GROUP BY query. You know that you can use WHERE to eliminate detail rows returned by a regular query. With summary queries, you can use the HAVING clause to eliminate summary rows.

- It used generally with GROUP BY Clause the having is useful for specifying a condition for the group.
- The Clause is used to filter data that is associated with group function.

Syntax:

```
SELECT [column,] group_function(column). . .
FROM Table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING having_expression];
[ORDER BY Column/Alias];
```

The HAVING Clause:

- Groups are formed and group functions are calculated before the HAVING clause is applied to the groups.
- The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical.
- **Steps when you use the HAVING clause:**
  1. Groups rows
  2. Applies the group functions to the groups and displays the groups that match the criteria in the HAVING clause.
- GROUP BY Clause can used, Without a Group function in the SELECT list.
- HAVING Clause can used, Without a GROUP BY Clause in the SELECT Statement.
- If rows are restricted based on the result of a group function, we must have a GROUP BY clause as well as the HAVING Clause.
- Existence of GROUP BY clause dose not guarantees the Existence of HAVING clause.

```
Sql>Select Deptno,COUNT(Deptno) From Emp GROUP BY Deptno HAVING COUNT(Deptno)>3;
Sql> Select Deptno,COUNT(Deptno) From EmpHAVING COUNT(Deptno)>3 GROUP BY Deptno;
Sql> Select Deptno,AVG(Sal) From EmpGROUP BY Deptno HAVING AVG(Sal)>2500;
Sql> Select Job,SUM(Sal) Payroll From Emp Where Job<>'SALESMAN' GROUP BY Job HAVING SUM(Sal)>=5000 ORDER BY SUM(Sal) Desc;
Sql> Select Deptno,MIN(Sal),MAX(Sal),SUM(Sal) From Emp Where Job='CLERK'
                GROUP BY Deptno HAVING MIN(Sal)<1000;
Sql> Select Deptno,AVG(Sal),SUM(Sal),MAX(Sal),MIN(Sal) From Emp GROUP BY Deptno HAVING COUNT(Deptno)>3;
Sql>Select Deptno,AVG(Sal),SUM(Sal) From Emp GROUP BY Deptno HAVING AVG(Sal)>2500;
Sql>Select Deptno,Job,AVG(Sal),SUM(Sal) From Emp GROUP BY Deptno,Job
                                        HAVING AVG(Sal)>2500;
```

Nesting of Group Function:

- Group Functions can be Nested to Depth of two levels.

```
Sql> Select MAX(AVG(Sal)) From Emp GROUP BY Deptno;
```

Sql> Select MAX(SUM(Sal)),MIN(SUM(Sal)) From Emp GROUP BY Deptno;

Sal> Select MAX(SUM(Sal)),MIN(AVG(Sal)) From Emp GROUP BY Deptno;

Miscellaneous Function:

GREATEST Function:

Syntax: GREATEST(Expr1,Expr2,.....)

- It used to pick highest values from list of values.
- All exprs after the first are implicitly converted to the data type of the first expr,before the comparison.
- Character comparison is based on the value of the character in the data base character list.

Sql>SELECT GREATEST ('SIBEL','CODD','SCOTT') FROM DUAL;

Sql>SELECT GREATEST(1000,2000,3000,4000) FROM DUAL;

Sql> SELECT GREATEST('10-JUL-05','20-JUL-05') FROM DUAL;

LEAST Function:

Syntax: LEAST(Expr1, Expr2,..)

- It used to pick lowest value from list of values
- All exprs after the first are implicitly converted to the data type of the first expr,before the comparison.

Sql>SELECT LEAST('M','N','O','X','Y','Z') FROM DUAL;

Sql>SELECT LEAST (1000,2000,3000,4000) FROM DUAL;

Sql> SELECT LEAST ('10-JUL-05','20-JUL-05') FROM DUAL;

USER Function:

Syntax: USER

- It returns the current oracle users names within the VARCHAR2 data type.
- The function cannot be used in the condition of the CHECk constraint.

Sql>Select USER from Dual;

UID Function:

Syntax: UID

- It gives a number that identified the user since oracle is used in a multi-user environment.
- UID is the way identifying a user.

Sql>Select USER,uID From Dual;

USERENV Function:

Syntax: USERENV(Option)

- Returns information of varchar2 data type above the current session.

The values in options are

- 'ISDBA'→Returns 'TRUE' if DBA is enabled.
- 'LANGUAGE'→Returns the language and territory used in current Session.
- 'TERMINAL'→Returns the OS identifier for the current sessions terminal.
- 'SESSIONID'→ Returns the auditing session identifier.
- 'ENTRYID'→Returns the available auditing entry identifier.
- 'LANG'→Returns the ISD abbreviation for the language name.
- 'INSTANCE'→Returns the instance identification number of the current instance.

Sql>Select USERENV('ISDBA') From Dual;

Sql> Select USERENV('LANGUAGE') From Dual;

Sql> Select USERENV('TERMINAL') From Dual;

Sql> Select USERENV('SESSIONID') From Dual;

Sql> Select USERENV('LANG') From Dual;

Sql> Select USERENV('INSTANCE') From Dual;

VSIZE Function:

Syntax: VSIZE

- It returns the NUMBER of bytes in the internal representation of Expr.
- If Expr is NULL, Function returns NULL.

Sql>Select Ename,VSIZE(Ename) From Emp;

Sql>Select Deptno,VSIZE(Deptno) From Emp;

Sql> Select Ename,Hiredate,VSIZE(Hiredate) From Emp;

SOUNDEX Function:

Syntax :SOUNDEX(CHAR)

- It returns a character string containing the representation of CHAR.
- It allows comparison of words that are spelled differently, but sound alike in English.

Sql>Select Ename From emp Where SOUNDEX(Ename)= SOUNDEX ('SMYTHE');

Sql> Select Ename,Job From EmpWhere SOUNDEX(Job)=SOUNDEX('manger');

Sql> Select Job From emp Where SOUNDEX(Job)=SOUNDEX('CLRK');

INTEGRITY CONSTRAINTS

Objectives:

→ Introduction to Integrity Constraints

→ Implementation of Constraints

Data Constraints

All business of the world run on business data being gathered, stored and analyzed . Business managers determine a set of business rules that must be applied to their data prior to it being stored in the database/table of ensure its integrity. For instance, no employee in the sales department can have a salary of less than Rs.1000/-.Such rules have to be enforced on data stored. Only data, which satisfies the conditions set , should be stored for future analysis. The data, which is not obeying integrity constraints, is called inconsistent data. Every software must enforce integrity constraints, otherwise inconsistent data is generated.

Remember: →It is used to impose business rules to DBs.

→It allows to enter only valid data.

Various types of Integrity Constraints:

Domain Integrity:

- A Domain means a set of permitted values.
- These constraints set a range, enforcement of not null constraint in a table.
- There are basically three types of Domain Integrity Constraints .
  - Not Null Constraint
  - Check Constraint
  - Default Constraint

Entity Integrity:

Entity integrity constraints are two types .

- Unique Constraints
- Primary key Constraints

→It defines a entity or column as a UNIQUE for a particular table.

→Primary key constraint avoids duplicate and null values.

Referential Integrity:

→The referential Integrity constraint enforces relationship between tables.

→It designates a column or combination of columns as a foreign key.

→The foreign key establish a relationship with a specified primary or unique key in another table called the referenced key.

→When referential integrity is enforced, it prevents from ....

- Adding records to a related table if there is no associated record in the Primary Table.
- Changing values in a Primary Table that result in Orphaned records in a related table.
- Deleting records from a Primary Table if there are matching related records.

Note:-The table containing the foreign key is called the child table and the table containing the referenced key is called the Parent key.

As per Oracle these are Six Types of Constrains:

1) NOT NULL Constraint
2) UNIQUE Constraint
3) PRIMARY KEY Constraint
4) FOREIGN KEY(Self Reference)
5) CHECK Constraint

Level of constraint:-

1)Column level :-

→ Used to define constraints next in column name

→Define with each column.

→Composite key can not be defined.

2)Table Level:-

→Defining constraints after defining all columns.

→Not Null can not be defined.

The Constraint clause can appear in

- CREATE Table
- ALTER Table
- CREATE View

Oracle does not support Constraint on Column or attributes whose types is

- USER_DEFINED OBJECT
- NESTED TABLE
- VARRAY
- REF
- LOB

Exceptions:

- **NOT NULL** Constraint are supported for a column or attributes whose type is USER_DEFINED object VARRAY, REF, LOB.

NOT NULL Constraint:

- Used to suppress null values into columns.
- Data must be entered.
- Duplication values allowed.
- NOT NULL should be defined only at COLUMN Level.
- The default if specified is NULL.

Example:

Sql> CREATE TABLE branch_mstr (Branch_No Varchar2(10) Constraint Branchno_NN NOT NULL,Name Varchar2(20) Constraint Name_NN NOT NULL);

UNIQUE Constraints:

- Used to suppress duplicate values into columns.
- Accepts NULL values.
- A table can have more than one UINQUE key which is not possible in Primary Key.
- Unique key can defined on more than one column.(ic composite unique key).
- A composite key UNIQUE key is always defined at the table level only.
- One table can have more than one Composite UNIQUE Key.
- Oracle creates an indexed automatically.

Restrictions:

- Unique key can not be implemented on columns having ....
  .LOB .LONG .LONG RAW .VARRAY .NESTED TABLE .OBJECT .BFILE .REF .TIMESTAMP .
  A composite UNIQUE key cannot have more than 32 columns.

UNIQUE KEY Constraint Defined At Column Level

Syntax:

<Column Name> <Data Type>(<Size>) UNIQUE

Example:

CREATE TABLE branch_mstr (Branch_No Varchar2(10) Constraint Branchno_UNQ UNIQUE, Name Varchar2(20) Constraint Name_NN NOT NULL);

UINQUE Constraint Defined At the Table Level

Syntax:

CREATE TABLE tableName(<ColumnName1> <Datatype>(<Size>),<ColumnName2> <Datatype>(<Size>),UNIQUE(<ColumnName1>,[ColumnName2,.....]));

```
Sql >CREATE TABLE Product_Master ( Product_no Varchar2(6) , Des Varchar2(25),
     Qty_on_hand Number(8), Constraint prono_UNQ UNIQUE(Product_no) );
Sql>CREATE TABLE FZ_SPDNOTE ( Fz_No Varchar2(10) , Sp_No Varchar2(10),
          Empno Number, Constraint FzSpNo_CUNQ UNIQUE(Fz_No,Sp_No) );
```

PRIMARY KEY Constraint:

- Used to define key column of table.
- It will not accepts Null Values and Duplicate values.
- It is provided with an automatic index.
- A Primary Key Constraint combines a NOT NULL and UNIQUE behavior in one declaration.

Restrictions:

- Only one Primary key Or Composite Primary Key is allowed per table.
- PRIMARY KEY can not be implemented on columns having ....
  .LOB .LONG .LONG RAW .VARRAY .NESTED TABLE .OBJECT .BFILE .REF .TIMESTAMP .
  A composite PRIMARY KEY cannot have more than 32 columns.
- PRIMARY KEY can not support in Nested Object.

PRIMARY KEY Constraint Defined At Column Level:

Syntax:

<Column Name> <Data Type>(<Size>) PRIMARY KEY

Example:

```
Sql>Create Table Incr(IncrId Number(4) Constraint Incr_Id_PK PRIMARY KEY,
        IncrDate Date NOT NULL, incrAmt Number(8,2) NOT NULL);
```

PRIMARY KEY Constraint Defined At the Table Level

Syntax:

```
CREATE TABLE tableName(<ColumnName1>
<Datatype>(<Size>),<ColumnName2>
        <Datatype>(<Size>), PRIMARY
    KEY(<ColumnName1>,[ColumnName2,....]));
```

Example:

```
Sql>CREATE TABLE Product_Master (Product_no Varchar2(6) ,Des Varchar2(25),
        Qty_on_hand Number(8), Constraint prono_PK  PRIMARY KEY(Product_no)
);
```

Sql> Create table SalesDet(Prodno **Number(8)**, Custid Number(8), SalesDate Date NOT NULL,
SaleDesc Varchra2(20) NOT NULL,Constraint Prod_Cust_PK(ProdId,CustId));

FOREIGN KEY (Or) REFERENTAIL INTEGRITY Constraint:

- Foreign Key represent relationships between tables.
- A Foreign Key is a column or group of columns whose values are derived from the Primary Key or Unique Key.
- Foreign Key is column(s) that references a column(s) of a table and it can be the same table also.
- The Table or View constraining the Foreign key is called the Child object.
- Child may have duplicates and nulls but unless it is specified.
- A Foreign Key constraint can be defined on a single key column either column level (In line) or Table level(Out of line) column.
- A composite Foreign Key on attributes should be declared at table level or out of line.
- A Composite Foreign Key constraint, must refer to a composite unique key or a Composite Primary Key in the Parent Table.

Restriction:

- Master table cannot be update if child record exists.
- The Foreign Key column cannot be applied on.....
  .LOB .LONG .LONG RAW .VARRAY .NESTED TABLE
  .OBJECT .BFILE .REF .TIMESTAMP.
- A composite Foreign Key cannot have more than 32 columns.
- A Child and parent table must be on same database.
- To enable Referential Integrity across nodes of a distributed database triggers are used.

Note:

1)FOREIGN KEY identifies the column or combination of columns in the child table that makes up of the Foreign key.

2)**REFERENCES** identifies the parent table and the column or combination of columns that make up the referenced key.

ON DELETE Clause:

- CASCADE option used to remove the child table record automatically , when parent record is removed.
- Specify SET NULL if we want Oracle to convert dependent FOREIGN KEY values to NULL.

Steps to be Followed for Creating References Constraint:

Step1: CREATE TABLE Dept
(Deptno Number(2) Constraint Deptno_PK PRIMARY KEY,
Dname Varchar2(20) Constraint Dname_NN NOT NULL,
Loc Varchar2(20) Constraint Loc_NN NOT NULL);

Step2: Create Detail/Child/Sub/Dependent Table

- These are Tables which can contain Primary Key of their own as well as Foreign key's referring to other Primary Master's or to themselves.

```
Sql>CREATE TABLE Emp( Empno Number(4) Constraint Empno_PK PRIMARY KEY,
                      Ename Varchar2(20) Constraint Ename_NN NOT NULL,
                      Job Varchar2(15) Constraint Job_NN NOT NULL,
                      Mgr Number(4) Constraint Mgr_FK_Self REFERENCES
                            Emp(Empno) ON DELETE SET NULL,
                      Hiredate Date Constraint Hiredate_NN NOT NULL,
                      Sal Number(8,2) Constraint Sal_NN NOT NULL,
                      Comm Number(8,2),
                      Deptno Number Constraint Deptno_FK REFERENCES
                            Dept(Deptno) ON DELETE CASCADE );
```

Working With Composite PRIMARY KEY:

```
Sql> CREATE TABLE FZ_SRV_Item( Fzno Varchar2(20) Constraint Fzno_PK PRIMARY
KEY,Fztype Varchar2(20) Constraint Fztype_NN NOT NULL, Rdate Date Constraint
                            Rdate_NN NOT NULL );
Sql> CREATE TABLE FZ_SP_Item( Fzspno Varchar2(20) Constraint Fzspno_PK
PRIMARY KEY,Fzstype Varchar2(20) Constraint fzstype_NN NOT NULL, Rspdate Date
Constraint rspdate_NN NOT NULL );
Sql> CREATE TABLE FZ_DNote_Item( FznoRef Varchar2(20) Constraint Fznoref_FK
        REFERENCES FZ_SRV_Item(Fzno), FzspnoRef Varchar2(20) Constraint
            Fzspnoref_FK REFERENCES FZ_SP_Item(Fzspno), Dndate Date,
                Constraint fzref_Comp_PK PRIMARY
            KEY(FznoRef,FzspnoRef));
```

CHECK Constraint:

- Used to impose a conditional rule on a table column.
- It defines a condition that each row must satisfy.
- A single columns can have multiple CHECK constraints that can reference the column in the definitions.
- There is no limit to the number of CHECK constraints that can be defined on a column.
- The CHECK constrains can be defined at the column level or Table level.

Restrictions:

- The constructs that cannot be include are
  - Queries to refer to values in other rows
  - References to the CURRVAL,NESTVAL,LEVEL or ROWNUM.
  - Calls to functions SYSDATE,UID,USER,USERENV.
  - Date constant that are not fully specified.

Example:

```
CREATE TABLE Dept( Deptno number(2) constraint dno_pk PRIMARY KEY
        constraint Deptno_Chk CHECK(Deptno BETWEEN 10 and 99),
        Dname varchar2(15) constraint dname_nn NOT NULL constraint
        Dname_Chk CHECK(Dname=UPPER(Dname)), Loc varchar2(15) default
        'NEW YORK' constraint Loc_Chk CHECK (Loc IN('NEW YORK', 'DALLAS',
                'BOSTON','CHICAGO')) );
```

```
CREATE TABLE Emp( Empno Number(4) Constraint Empno_Pk PRIMARY KEY,
        Enamie Varchar2(20) Constraint ename_NN NOT NULL
                CHECK( SUBSTR(Ename,1,1) BETWEEN 'A' AND 'Z') AND
                        Ename=UPPER(Ename)), Job Varchar2(15) Constraint
                Job_Chk
            CHECK(Job IN('ANALYST','CLERK','MANAGER','PRESIDENT','SALESMAN')),
            Hiredate date DEFAULT SYSDATE, Sal Number(8,2)  Constraint Sal_NN
    NOT NULL Constraint CHK_Sal CHECK(Sal BETWEEN 1000 and 10000),
                Comm number(8,2), Deptno Number(2), Constraint Tot_Sal_Chk
                                        CHECK(Sal+Comm<=100000) ) ;
```

Default Option:

- If values is not provided for table column default will be considered.
- The options prevents NULL Values from entering the Columns, If a row is inserted without a value for a column.
- The DEFAULT value can be a literal, an expression or a SQL Function.
- The DEFAULT Expression must match the data type of the Column.

Adding Constraint to a table:

- A constraint can be added to a table at any time after the table was created by using by ALTER TABLE Statement, using **ADD** Clause.

Syntax:

```
Sql>ALTER TABLE <Table Name> ADD [Constraint <Constraint Name>]
            Cons_Type(Column_Name,[ Column_Name,..]);
```

Guidelines:

- The Constraint Name syntax is optional, but recommended.
- Table Constraints are applied to table if data previously placed in the table violated such constraints.
- We can ADD,DROP,ENABLE, or DISABLE a Constraint, but modify the structure.
- NOT NULL,DEFAULT can be added to existing column by using the MODIFY Clause of the ALTER TABLE statement.

Example:

```
Sql>ALTER TABLE Emp ADD Constraint Empno_PK PRIMARY KEY(empno);
Sql>ALTER TABLE Emp ADD Constraint Emp_Mgr_FK  FOREIGN KEY(Mgr)
        REFERENCES  Emp(empno);
Sql>ALTER TABLE Emp ADD CONSTRAINT Dept_Dno_FK FOREIGN KEY(Deptno)
                    REFERENCES Dept MODIFY(Deptno NOT NULL);
Sql> ALTER TABLE Emp MODIFY Empno number(3) Constraint   empno_NN NOT
NULL;
Sql> ALTER TABLE Emp MODIFY Hiredate date default sysdate;
```

DROPPING Constraints:

- To drop a constraint identify the constraint name the USER_CONSTRAINTS and USER_CONS_COLUMNS Data dictionary views.
- The ALTER TABLE Statement is used with the DROP Clause.
- The CASCADE Option of the DROP Clause causes any dependent constraints also to be dropped.
- When a constraint is dropped, the constraint is no longer enforced and is no longer available in the data dictionary.

Syntax:

```
ALTER TABLE <Table Name>  DROP PRIMARY KEY/UNIQUE(Column)/
        CONSTRAINT Constraint_Name[CASCADE];
```

Note: When drop the PRIMARY KEY/UNIQUE Constraints the related INDEX will drop automatically.

```
Sql>ALTER  TABLE  Emp DROP PRIMARY KEY;
Sql>ALTER  TABLE  Dept DROP UNIQUE(Dname);
Sql>SELECT index_name from user_indexes WHERE table_name='EMP';
```

DISABLING Constraint:

- The constraint can be disabled without dropping it or recreating it.
- The ALTER TABLE statement is used with the DISABLE Clause.

Syntax: ALTER TABLE <TableName> DISABLE CONSTRAINT<ConstraintName>[CASCADE];

Guidelines:

- The CASCADE clause disable dependent integrity constraints.

Example:

```
Sql> ALTER TABLE Emp DISABLE CONSTRAINT Empno_PK CASCADE;
```

ENABLE Constraint:

- The constraint can be enabled without dropping it or recreating it.
- The ALTER TABLE statement is used with the ENABLE Clause.

Syntax: ALTER TABLE <Table Name> ENABLE CONSTRAINT <Constraint Name>;

- Enabling a Constraint applied to all the data in the table.
- When an UNIQUE or PRIMARY KEY Constraint is ENABLED , the UNIQUE or PRIMARY KEY Index is automatically created.

```
Sql>ALTER TABLE Emp ENABLE CONSTRAINT Mgr_FK;
```

VIEWING Constraints:

- To View all Constraints on table by Query the USER_CONSTRAINTS table.
- The Codes that are revealed are....
  - P-Primary
  - U-Unique
  - R-References
  - C-Check & Not Null

```
Sql> SELECT owner,constraint_name,constraint_type     FROM user_constraints
                                          WHERE table_name='FZ_SRV_ITEM';
```

VIEWING The Columns Associated With Constraints:

- The Names of the columns that are involed in constraints can be know by querying the USER_CONS_COLUMNS Date Dictionary View.

```
Sql>SELECT constraint_name,column_name
      FROM USER_CONS_COLUMNS  WHERE table_name='FZ_SRV_ITEM';
```

Note: When drop the table all corresponding Integrity Constrains will droped automatically.

OCP:

1)Examine the structure of the EMPLOYEES table:

```
EMPLOYEE_ID NUMBER Primary Key
FIRST_NAME VARCHAR2(25)
LAST_NAME VARCHAR2(25)
```

Which three statements inserts a row into the table? (Choose three)

```
A. INSERT INTO employees VALUES (NULL, 'JOHN','SMITH');
B. INSERT INTO employees( first_name, last_name) VALUES ('JOHN','SMITH');
C. INSERT INTO employees VALUES ('1000','JOHN','NULL');
D. INSERT INTO employees(first_name,last_name, employee_id)  VALUES
('1000',
                                'john','Smith');
E. INSERT INTO employees (employee_id) VALUES (1000);
F. INSERT INTO employees (employee_id, first_name, last_name)
                                                VALUES ( 1000,'john',' ');
```

2) Which syntax turns an existing constraint on?

A. ALTER TABLE table_name ENABLE constraint_name;

B. ALTER TABLE table_name STATUS = ENABLE CONSTRAINT constraint _ name;

C.ALTER TABLE table_name ENABLE CONSTRAINT constraint _ name;

D.ALTER TABLE table_name STATUS = ENABLE CONSTRAINT constraint _ name;

E.ALTER TABLE table_name TURN ON CONSTRAINT constraint _ name;

F. ALTER TABLE table_name TURN ON CONSTRAINT constraint _ name;

3) You need to modify the STUDENTS table to add a primary key on the STUDENT_ID column. The table is currently empty.
Which statement accomplishes this task?

A. ALTER TABLE students ADD PRIMARY KEY student_id;
B. ALTER TABLE students ADD CONSTRAINT PRIMARY KEY (student _ id);
C. ALTER TABLE students ADD CONSTRAINT stud_id_pk PRIMARY KEY (student _ id);
D.ALTER TABLE students ADD CONSTRAINT stud _ id _pk PRIMARY KEY student _ id;
E. ALTER TABLE students MODIFY CONSTRAINT stud _ id _pk
PRIMARY KEY (student _ id);

4)Which SQL statement defines the FOREIGN KEY constraint on the DEPTNO column of the EMP table?

A.) CREATE TABLE EMP (empno NUMBER(4), ename VARCNAR2(35),
deptno NUMBER(7,2) NOT NULL
CONSTRAINT emp_deptno_fk FOREIGN KEY deptno REFERENCES dept deptno);

B) CREATE TABLE EMP (empno NUMBER(4), ename VARCNAR2(35),
deptno NUMBER(7,2) CONSTRAINT emp_deptno_fk REFERENCES dept (deptno));

C) CREATE TABLE EMP (empno NUMBER(4) ,ename VARCHAR2(35),
deptno NUMBER(7,2) NOT NULL, CONSTRAINT emp_deptno_fk REFERENCES dept (deptno) FOREIGN KEY (deptno));

D. CREATE TABLE EMP (empno NUMBER(4),
ename VARCNAR2(35), deptno NUMBER(7,2) FOREIGN KEY
CONSTRAINT emp_deptno_fk REFERENCES dept (deptno));

5)Updating a Primary key value will be allowed by Oracle when it has dependent records.

A)True B)False C)None

6)The _____ allows for automatic make it null value of child record when a parent record is deleted.

A) Add constraint
B) Drop Constraint
C) On delete cascade
D) None

DISPLAYING DATA FROM MULTIPLE TABLE

JOINS:

When data from more than one table in the database is required, a join condition is used. Rows in one table can be joined to rows in another table according to common values existing in corresponding columns, that is, usually primary and foreign key columns. To display data from two or more related tables, write a simple join condition in the WHERE clause.

- Use a join to query data from more than one table.
- A Join is Query that combines rows from two or more tables, Views, or Materialized Views.
- A Join is performed whenever multiple table appear in the Queries FROM clause.

Sql>Select Empno,Ename,Dname,Loc From Emp,Dept ;

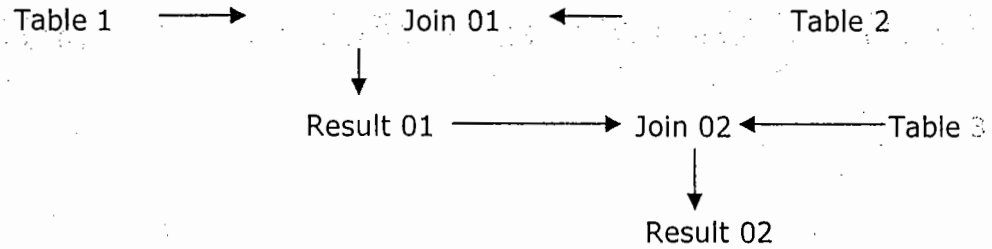
Sql>Select Empno,Ename,Sal,Grade From Emp,Salgrade;

Sql>Select Empno,Ename,Dname,Loc,Grade From Emp,Dept,Salgrade

Sql> Select Empno,Ename,Emp.Deptno,Dname,Loc From Emp,Dept

JOIN Condition:

- The Applied condition is called a JOIN CONDITION.
- To Execute a join.....
- Oracle combines pairs of rows, each containing one row from each table, for which the JOIN condition evaluates to TRUE.
- The column in the join condition need not be part of the SELECT list.
- The WHERE clause of join Query can also contain other conditions that refer to columns of only one table.
- If a join involves over two tables then oracle joins the first two based on the condition and then compares the result with the next table and so on.



- The oracle optimizer determines the order in which ORACLE should join thetables based on.....
- Given JOIN condition(s)
- INDEXES upon the tables.
- The LOB columns cannot be specified in the WHERE clause, when the WHERE clause contains any JOINS.
- The LOB columns cannot be specified in the WHERE clause, when the WHERE clause contains any JOINS.

Syntax:

WHERE table1.column1 = table2.column2;

Guidelines:

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join n tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated
primary key, in which case more than one column is required to uniquely identify each row.For more information.

Equi Join or Simple Joins or Inner Joins:

- Based on equality condition tables are joined.
- Only matching records are displayed.
- Joining tables must have at least one common column with same data type and same values. (not column name same.)

Qualifying Ambiguous column names:

- The name of the columns names should be qualified in the WHERE caluse with the table name to voied ambiguity.
- If there are no common column names between the two tables,the qulifiction is not necessary but it is better.

Syntax:

```
Sql> Select col1,col2,col3....
  From <table1>,<table2>.....
 Where <table1>.<common col name>=<table2>.<common col name>
       And .....
     (This is join condition)
Sql>Select  Emp.Empno Empno, Emp .Ename Ename, Emp.Deptno Deptno,
     Dept.Deptno Deptno, Dept.Dname Dname, Dept.Loc Loc From Emp,Dept
          Where Emp.Deptno=Dept.deptno;
Sql> Select Empno ,Ename,Emp.Deptno Deptno,Dname From Emp,Dept
          Where Emp.Deptno=Dept.deptno Empno=&eno;
Sql> Select  Empno ,Ename,Emp.Deptno Deptno,Dname From Emp,Dept
          Where Emp.Deptno=Dept.deptno  Job=UPPER('clerk');
Sql> Select Empno ,Ename,Sal*12 AnnuSal,Emp.Deptno,Loc From Emp,Dept
          Where Emp.Deptno=Dept.deptno And Dname='SALES'
```

Using Table Aliases:

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. You can use table aliases instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, therefore using less memory. The table name is specified in full, followed by a space and then the table alias. The EMP table has been given an alias of 'e', and the DEPT table has an alias of 'd'.

Guidelines:

- Table aliases can be up to 30 characters in length, but the shorter they are the better.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid only for the current SELECT statement.

```
Sql> Select E.Empno ,E.Ename,D.Deptno D.Dname From Emp E,Dept D
                    Where E.Deptno=D.Deptno;
Sql> Select E.Empno ,E.Ename,E.Job,D.Deptno D.Dname From Emp E,Dept D
          Where E.Deptno=D.Deptno  E.Job IN('MANAGER','SALESMAN');

Sql> Select E.Empno ,E.Ename,E.Job,D.Deptno D.Dname From Emp E,Dept D
                  Where E.Deptno=D.Deptno AND D.Dname<>'SALES';
```

Non Equi Join:

- A non equi-Join specifies the relationship between columns belonging to different tables by making use of the relational operations other than =.
- It is used to join table if value of one column a table falls the range of two column values of other table.
- It used between operator, it called as Between join.

Syntax:- Select col1,col2,.....
From<table A>,<table B>
Where <tableA>.<col1>Between <table B>.<col1>and <table B>.<col2>;

Sql> Select E.empno,E.Ename,E.Sal,S.Grade,S.Losal,S.Hisal From Emp E,Salgrade S Where E.Sal Between S.Losal and S.Hisal;
Sql> Select E.empno,E.Ename,E.Sal,S.Grade,S.Losal,S.Hisal From Emp E,Salgrade S Where E.Empno=&Eno And E.Sal Between S.Losal and S.Hisal;
Sql> Select E.empno,E.Ename,E.Sal,S.Grade From Emp E,Salgrade S Where (E.Sal>=S.Losal And E.Sal<=S.Hisal) And S.Grade=1;

Self Joins:

Self Join:-

- It indicates joining the table to itself.
- The same table appears twice in the FROM clause and is followed by table aliases.
- The table aliases must qualify the column names in the join condition.
- It is used on same table the table must have at least 2 column with same.
  - Data type.
  - Value
- To perform a self join, oracle combines and returns rows of the table that satisfy the join condition.

Syntax:

Sql>Select Columns
From Table1 T1,Table1 T2
Where T1.Column1=T2.Column2;

Example:

Sql>Select E.Ename "Emp Name",M.Ename "Manager Name" From Emp E,Emp M Where E.Mgr=M.Empno;
Sql> Select E.Ename "Emp Name",M.Ename "Manager" From Emp E,Emp M Where E.Mgr=M.Empno And E.Hiredate<M.Hiredate;
Sql> Select E.Ename||'Emp Name',M.Ename|| 'Manager Name' From Emp E, Emp M Where E.Mgr=M.Empno;
Sql> Select E.Ename ||'Works for'|| M.Ename "Employees and Managers" From Emp E,Emp M Where (E.Mgr=M.Empno);
Sql> Select E.Ename ||"'s Works for'|| M.Ename "Employees and Managers" From Emp E,Emp M Where (E.Mgr=M.Empno) And E.Job='CLERK'

Cartesian Products:

- A Cartesian product results in all combinations of rows displayed. This is done by either omitting the WHERE clause or specifying the CROSS JOIN clause.
- During Cartesian product oracle combines each row of one table with each row of the other.
- It tends to generate a large number of rows and the result is rarely useful.
- A Cartesian product is formed when:
  - A join condition is omitted.
  - A join condition is invalid.
- All rows in the first table are joined to all rows in the second table.
- To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

Sql>Select Ename,Job,Dname From Emp,Dept;
Sql>Select Ename,Job,Dname From Emp,Dept Where Job='MANAGER';

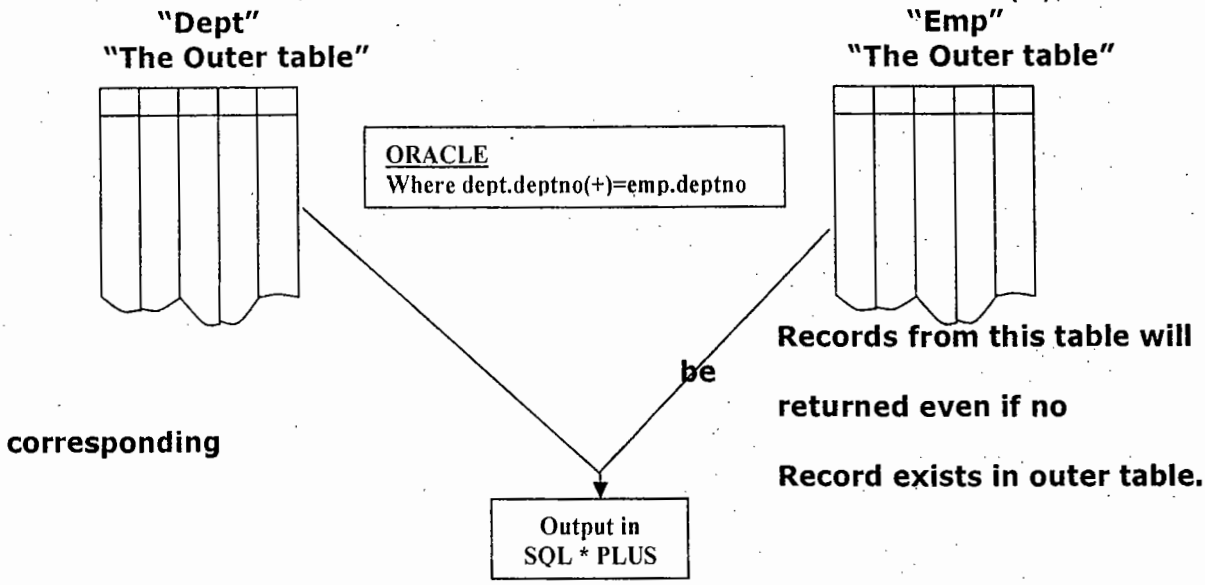
Outer Joins:

- Used to retrieve all the rows from one table but matching rows from other table.
- An Outer join extends the result of a simple or inner join.
- An OUTER join returns all rows that satisfy the join condition and also

those rows from one table for which no rows from the other satisfy the join condition.

- It is use an operator **(+)**,It called join operator.
- (+) used with the table which missing the data.
- To perform an outer join of table 'A' and 'B' and returns all rows from 'A', apply the outer join operator '(+)'to all columns of table 'B'.
- For all rows in 'A' that have no matching rows in 'B' oacle returns null for any select list expressions containing columns of 'B'.

```
Sql>Select      Table1.Column,Table2.Column
                From Table1,Table2
                        Where Table1.Column(+)=Table2.Column;
Sql>Select      Table1.Column,Table2.Column
                From Table1,Table2
                        Where Table1.Column=Table2.Column(+);
```



Rules And Restrictions:

- The (+) operator can appear only in the WHERE clause.
- If 'A' and 'B' are joined by multiple join conditions, we must use the (+) operator in all of these conditions.
- (+) used only with one table.
- A condition cannot use the IN comparison operator to compare a column marked with the(+) operator with an expression.
- A condition cannot compare any column marked with the (+) operator with a sub Query.

```
Sql> Select E.Ename,D.Deptno,D.Dname From Emp E,Dept D
                Where E.deptno(+)=D.deptno ORDER BY E.Deptno;
Sql> Select E.Ename,D.Deptno,D.Dname From Emp E,Dept D
                Where E.deptno=D.Deptno(+)  E.deptno(+)=10 ORDER BY
E.Deptno;
        Sql> Select E.Ename "Employe", NVL(M.Ename,' Supreme Authority')
"Manager"From Emp E,Emp M Where E.MGR=M.Empno(+);
```

Joining Data From More than Two Table:

- JOINS can be establish on more than two tables.
- The Join is First executed upon the two most relevant tables and then the result is applied upon the Third Table.

```
Sql>Select  E.Ename Ename,E.Deptno,M.Ename Manager,M.deptno From Emp E,Dept
D,Emp M Where E.Mgr=M.Empno And E.deptno=D.Deptno;
```

```
Sql> Select E.Ename Ename, M.Ename Manager, D.Dname Dname, E.Sal Esal,
SE.Grade  Egrade,M.Sal Msal, SM.Grade Mgrade From Emp E,Dept D,Emp M,salgrade
SE, Salgrade SM  Where (E.Deptno=D.Deptno) And (E.Mgr=M.Empno) And
                (E.Sal BETWEEN SE.Losal And SE.Hisal) And (M.Sal BETWEEN
                                                SM.Losal And SM.Hisal);
```

Categories Of Joins:

Oracle Proprietary Joins(8i And Prior)

- Equi Join
- Non-Equi Join
- Outer Join
- Self Join
- Cartesian Join

ANSI SQL :1999 Compliant Joins:

- Inner Join
- Cross joins
- Natural Join
- Using Clause
- Full Or Two Sided Outer Join
- Arbitrary Join conditions for outer joins

ISO or ANSI Joins:

Cross join:

- CROSS JOIN Returns a Cartesian product from the two tables.

```
Sql>Select Ename,Dept.Deptno,Dname,Loc From Emp CROSS JOIN Dept
                                Where Emp.Deptno=Dept.Deptno;
```

Natural Join:

```
Sql>Select Empno,Ename,Sal,Deptno,Dname,loc  From Emp Natural Join Dept;
```

Note:-Natural not accept a alias names.

Creating Joins with the USING Clause :

- If several columns have the same names but the data types do not match, the NATURAL JOIN.
- Clause can be modified with the USING clause to specify the columns that should be used for an equi join.
- Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

```
Sql> SELECT e.ename, d.dname  FROM   Emp e JOIN dept d USING (deptno)
                                WHERE  deptno =10
```

Inner Join:

Sql>Select e.Empno,e.Ename,e.Sal,e.Deptno, d.Dname,d.Loc From Emp e INNER JOIN Dept d ON(e.Deptno=d.Deptno);

Sql>Select e.Ename "Employee Name",m.Ename "Manager" From Emp e INNER JOIN Emp m ON(e.mgr=m.empno);

Join on more than two tables:

```
Sql> Select Ename,e.Sal,Grade,D.Deptno,Dname From Emp E JOIN Dept D
        ON E.Deptno=D.Deptno JOIN Salgrade ON E.Sal BETWEEN Losal And
Hisal;
 Sql> Select E.Ename,M.Ename,Sal,Grade,D.Deptno,Dname  From Emp E INNER
JOIN Dept D ON E.Deptno=D.Deptno INNER JOIN Emp M ON E.MGR=M.Empno
                        INNER JOIN Salgrade S ON E.Sal BETWEEN Losal And Hisal;
```

Left Outer Join:

```
Sql>select a.empno,a.ename,a.sal,b.deptno,   b.dname,b.loc   from emp a left
outer join dept b on(a.deptno=b.deptno);
```

Rigth Ourter Join:

Sql>select a.empno,a.ename,a.sal,b.deptno, b.dname,b.loc from emp a right outer join dept b on(a.deptno=b.deptno);

Full outer Join:

Sql>select a.empno,a.ename,a.sal,b.deptno, b.dname,b.loc from emp a full outer join dept b on(a.deptno=b.deptno);

Note:-

Full outer join=left outer join+Right outer join.

Some Special Example:

Sql>Select e.Ename "Employee Name",m.Ename "Manager" From Emp e LEFT OUTER JOIN Emp m ON(e.mgr=m.empno) Order by 2;

OCP:

1)What is the out put of the following Query.

```
>Select e.ename,e.deptno,e.sal,d.dname,d.loc From emp e,dept d
                                        Where e.deptno(+)=d.deptno(+);
```

2) Display all the employees who earn more than their manager's salary

```
>SELECT A.EMPNO FROM EMP A,EMP B WHERE B.EMPNO=A.MGR
                                    AND A.SAL>B.SAL;
```

3)Examine the data in the EMPLOYEES and DEPARTMENTS tables.

EMPLOYEES :

LAST_NAME	DEPARTMENT_ID	SALARY
Getz	10	3000
Davis	20	1500
Bill	20	2200
Davis	30	5000
Kochhar		5000

DEPARTMENTS :

DEPARTMENT_ID	DEPARTMENT_NAME
10	Sales
20	Marketing
30	Accounts
40	Administration

You want to retrieve all employees, whether or not they have matching departments in the departments table.

Which query would you use?

A. SELECT last_name, department_name FROM employees , departments(+);

B.SELECT last_name,department_nameFROM employees JOIN departments(+);

C. SELECT last_name, department_name ON(e. department_ id = d. departments_id); FROM employees(+) e JOIN departments d

D. SELECT last_name, department_name FROM employees e RIGHT OUTER JOIN departments d ON (e.department_id = d.department_id);

E. SELECT last_name, department_name FROM employees(+) , departments ON(e.department_id=d.department_id);

F. SELECT last_name, department_name FROM employees e LEFT OUTER JOIN departments d ON(e. department _ id = d. departme.t _id);

4)Joining of a table to itself is called_____Join

A)Outer Join

B)Full Outer Join

C)Codd Join

D) Self Join

5)Table aliases are used to make multiple table queries shorter and readable.
A)True B)False C)None

6)To retrieve rows which do not satisfy a query, the outer join concept can be used.
A)True B)False C)None

7)Examine the description of the EMPLOYEES table:
EMP_ID NUMBER(4) NOT NULL
LAST_NAME VARCHAR2(30) NOT NULL
FIRST_NAME VARCHAR2(30)
DEPT_ID NUMBER(2)
JOB_CAT VARCHAR2(30)
SALARY NUMBER(8,2)

Which statement shows the department ID, minimum salary, and maximum salary paid in that department, only of the minimum salary is less then 5000 and the maximum salary is more than 15000?
A. SELECT dept_id, MIN(salary(, MAX(salary) FROM employees
WHERE MIN(salary) <5000 AND MAX (salary) > 15000;
B. SELECT dept_id, MIN(salary), MAX(salary) FROM employees
WHERE MIN(salary) < 5000 AND MAX(salary) > 15000
GROUP BY dept_id;
C. SELECT dept_id, MIN(salary), MAX(salary) FROM employees
HAVING MIN(salary) <5000 AND MAX (salary) > 15000;
D. SELECT dept_id, MIN(salary), MAX(salary) FROM employees
GROUP BY dept_id HAVING MIN (salary) < 5000 AND MAX(salary)
E. SELECT dept_id, MIN(salary), MAX(salary) FROM employees
GROUP BY dept_id, salary HAVING MIN (salary) <5000
AND MAX (salary) > 15000;

8)Examine the structure of the EMPLOYES, DEPARTMENTS, and LOCATIONS tables.

EMPLOYEES
EMPLOYEE_ID NUMBER NOT NULL,Primary Key
EMP_NAME VAR CHAR2 (30)
JOB_ID VAR CHAR2 (20)
SALARY NUMBER
MGR_ID NUMBER References EMPLOYEE_ID column
DEPARTMENT_ID NUMBER Foreign key to
DEPARTMENT_ID column of the DEPARTMENTS table

DEPARTMENTS
DEPARTMENT_ID NUMBER NOT NULL, Primary Key
DEPARTMENT_NAME VARCHAR2 (30)
MGR_ID NUMBER
References NGR_ID column of the EMPLOYEES table
LOCATION_ID NUMBER Foreign key to
LOCATION_ID column of the LOCATIONS table

LOCATIONS
LOCATION_ID NUMBER NOT NULL, Primary Key
CITY VARCHAR2 (30)
Which two SQL statements produce the name, department name and the city of all the employees who earn more then 10000? (Choose two)

A. SELECT emp_name, department_name, city FROM employees e
JOIN departments d USING (department_id) JOIN locations 1 USING (location_id) WHERE salary>10000;

B. SELECT emp_name, department_name, city FROM employees e, departments d, locations 1 JOIN ON (e.department_id = d.department id) AND (d.location_id =1.location_id) AND salary > 10000;

C. SELECT emp_name, department_name, city FROM employees e, departments d, locations 1 WHERE salary > 10000;

D. SELECT emp_name, department_name, city FROM employees e, departments d, locations 1 WHERE e.department_id = d.department_id AND d.location_id = 1.location_id AND salary > 10000;

E. SELECT emp_name, department_name, city FROM employees e NATURAL JOIN departments, locations WHERE salary > 10000;

9)Examine the following output from SQL*Plus:

PRODUCT.ID	PRODUCT.NAME	BOX.LOCATION
57-X	WIDGET	IDAHO
456-Y	WIDGET	TENNESSSEE

Which of the following choice identify the type of query that likely produced this result.

A).Full outer join B)Left outer join C) Rigth outer join D)Equijoin

TYPES OF QUERY

Types of Query:

- Root Query
- Parent Query(or)Outer Query (or)Main Query
- Child Query(or)Inner Query(or) sub Query

Root Query:-

- The query which is not depend on any other query for its conditions value.
  (or)
  Independent query.

Example:-

Sql>Select ename,sal from emp Where Deptno=10;

Parent Query:-

- The query which depend on any other query for its condition value.

Sub Query :-

- The query which provides, conditional values to its parent query.
- A sub Query in the WHERE clause of a SELECT statement is called as NESTED SUBQUERY.
- A sub Query in the FROM clause of a SELECT statement is called as INLINE VIEW.
- A sub Query can be part of a column, in the SELECT list
- Sub Query can contain another sub query .
- Oracle imposes no limit on the number of sub query levels in the FROM clause of the Top-Level Query.
- Within the WHERE clause up to 255 sub queries can be nested.
- To make the statement easier for readability ,qualify the columns in a Sub Query with the table name or table alias.

Note: parent query and sub query both are combined.

Purpose of a Sub Query:

- To define the set of rows to be inserted into the target table of an INSERT or CREATE TABLE statement.
- To define the set of rows to be included in a View or a Materialized INSERT or CREATE TABLE statement.
- To define one or mozre values to be assigned to existing rows in UPDATE statement.
- To provide values for condition in a WHERE clause, HAVING clause, START WITH clause of SELECT,UPDATE, and DELETE statements.

SUB QUERY Principle:

- Solve a problem by combining the two queries, placing one query inside the other Query.
- The inner query or the sub query returns a value that is used by the outer query upon the main query.

Sub Query usage:

- A Sub Query is a SELECT statement that is embedded in a clause of another SELECT statement.
- We can build powerful statements out of simple ones by using Sub Queries.
- They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

Syntax:

```
Sql>SELECT select_list
          FROM table
    WHERE Column name/expr operator(SELECT select_list FROM table);
```

Comparison conditions fall into two classes:

- Single -row operators (>, =, >=, <, <>, <=)
- Multiple-row operators (IN, ANY, ALL).

Single row sub Query:

- These queries returns only one row from the Inner SELECT statement.

Multiple row sub query:

- These queries returns more than one column from the Inner SELECT statement.

Guidelines for Using Subqueries:

- Enclose Sub Queries in parentheses.
- Place Sub Queries on the right side of the comparison condition.
- The ORDER BY clause in the Sub Query is not needed unless you are performing top-nanalysis.
- Only one ORDER BYclause can be used for a SELECT statement.
- Two classes of comparison conditions are used in Sub Queries:
  - Single-row operators
  - Multiple-row operators.

Simple Sub Query:

- Sub Query processed first and processed completely

Co-related Sub Query:

- In the Co-Related Sub Query a parent query will be executed first and based on the out put of outer query the inner query execute.
- If Parent Query returns N rows inner query executed for N times.

Simple Sub Query with Single Row:

```
Sql>Select Empno,Ename,Sal,Deptno From Emp  Where Sal<(Select Sal
                        From Emp  Where Empno=7566 );
Sql> Select Empno,Ename,Job From Emp Where Job=(Select Job From Emp
                              Where Ename=UPPER('ford') ) and Ename='FORD';
Sql> Select Empno,Ename,Hiredate,Sal,Job From Emp Where Hiredate>(Select
Hiredate From Emp Where Ename='MILLER' ) Order by Sal;
Sql> Select Empno,Ename,Sal,Job  From Emp Where Deptno=(Select Deptno
From Dept Where Dname='SALES' );
Sql> Select Empno,Ename,Sal,Job From Emp  Where Deptno=(Select Deptno
From Dept Where Loc='NEW YORK' ) And Job='CLERK';
Sql> Select Empno,Ename,Sal,Job From Emp Where Deptno=(Select Deptno
From Dept Where Dname='SALES' );
Sql> Select Empno,Ename,Sal,Comm,Sal+NVL(Comm,0)  From Emp Where
Deptno=(Select Deptno From Dept Where Loc='CHICAGO'  );
Sql>Select *From Emp Where Deptno=(Select Deptno From Emp Where
Ename='FORD')AND Job IN(Select Job  From Emp Where Deptno=(Select Deptno From
Dept Where  Dname='SALES'));
```

Appling GROUP functions in SUB QUERIES:

- The data from the main query can be displayed by using a Group Function is a Sub Query.
- As a Group Function returns a single row, the query passes the success state.The inner sub query should not have a GROUP BY Clause in this scenario.

Examples:

```
Sql>SELECT ename,job,sal FROM emp WHERE sal=(SELECT Max(sal)  FROM
emp  );
Sql> SELECT ename,job,sal FROM emp WHERE sal=(SELECT Min(sal)  FROM emp
);
Sql>SELECT ename,job,sal FROM emp WHERE sal>(SELECT Avg(sal) FROM emp
);
Sql>SELECT ename,job,sal FROM emp WHERE sal<(SELECT Variance(sal) FROM
emp);
```

Appling HAVING Clause With SUB QUERIES:

- A Sub Query can be also applied in HAVING Clause.
- The Oracle server executes the sub query, and the result are returned into the HAVING Clause of the Main Query.

Example:-

```
Sql>SELECT deptno,Min(sal) FROM emp GROUP BY deptno HAVING Min(sal)>
        (SELECT Min(sal) FROM emp WHERE deptno=20 );
Sql>SELECT job,Avg(sal) FROM emp GROUP BY job HAVING Avg(sal)=
        (SELECT Min(Avg(sal)) FROM emp GROUP BY job );
Sql> SELECT job,Avg(sal) FROM emp GROUP BY job HAVING Avg(sal)<
        (SELECT Max(Avg(sal)) FROM emp GROUP BY job );
```

Sub Query Returns more than one row:

- Subqueries that return more than one row are called **Multiple-row Sub Queries.**
- We use a multiple-row operator, instead of a Single-Row operator, with a Multiple -row Sub Query.
- The multiple-row operator expects one or more values.
- Use multiple-row comparison operators
  - IN → Equal to any member in the list
  - ANY/SOME →Compare value to each value returned by the Sub Query.
  - ALL → Compare value to every value returned by the Sub Query.

```
Sql>SELECT ename,sal,deptno FROM emp WHERE sal IN(SELECT MAX(sal) FROM
emp GROUP BY deptno );
Sql>SELECT ename,sal,job,deptno FROM emp WHERE sal IN(SELECT MAX(sal)
FROM emp GROUP BY job );
Sql>SELECT ename,sal,deptno FROM emp WHERE sal IN(SELECT MIN(sal) FROM
emp GROUP BY deptno);
```

ANY Operator

```
Sql> SELECT ename,sal,deptno FROM emp WHERE sal <SOME(1250,1500,1600);
Sql>SELECT ename,sal,deptno FROM emp WHERE sal <SOME(SELECT sal FROM
emp Where Job='SALESMAN');
Sql> SELECT ename,sal,deptno,Job FROM emp WHERE sal <ANY(SELECT sal FROM
emp Where Deptno in(20,30) And Job<>'MANAGER');
Sql>Select Empno,Ename,Sal,Deptno From emp Where Sal>ANY(Select Sal From
Emp where deptno=10 );
```

Note:<ANY/SOME means less than the Maximum value in the list.

```
Sql> SELECT ename,sal,deptno FROM emp WHERE sal >SOME(1250,1500,1600);
Sql>SELECT ename,sal,deptno FROM emp WHERE sal >ANY(SELECT sal FROM emp
        Where Job='SALESMAN');
```

Note:>ANY/SOME means More than the Minimum value in the list.

```
Sql> SELECT ename,Job,Deptno FROM emp WHERE sal =ANY(SELECT sal
                FROM emp Where Job='MANAGER' );
        Note =ANY it is equivalent to In operator.
```

All Operator:

```
Sql> Select Ename,Job,Sal,Deptno From Emp Where Sal>ALL(Select Sal From Emp
    Where Deptno=30);
Sql> Select Ename,Job,Sal,Deptno From Emp Where Sal>ALL(Select AVG(Sal) From
Emp Group by Deptno);
```

Note:>ALL →it means more than the maximum in the list.

```
Sql> Select Ename,Job,Sal,Deptno From Emp Where Sal<ALL(Select AVG(Sal) From
Emp Group by Deptno);
```

Note:<ALL →it means less than the minimum value in the list.

Sub Query Returning multiple Columns:

- In sub Queries multiple columns can be compared in the WHERE clause, by writing a compound WHERE clause using logical operator.

Syntax:

```
Sql>Select Column1,Column2,....
     From Table Name Where
          (Column a, Column b,.....) IN(Select Column a, Column B,.....
               From TableName Where Condition);
```

- The column comparisons in a multiple column sub Query can be.
  - Pair wise comparison.
  - Non pair wise comparison.
- Pair wise comparisons Each candidate row in the SELECT statement must have both the same values associated with each column in the group.
- The Non pair wise comparison, the candidate row must match the multiple conditions in the WHERE clause but the values are compared individually.

Pair wise Comparison or compound WHERE clause based SubQuery

```
Sql>Select Ename,Sal,Deptno From Emp Where (Deptno,Sal) IN(Select
Deptno,Max(Sal) From Emp Group by Deptno) And Deptno<>10;
```

Non Pair wise Comparison or compound WHERE clause based Sub Query

```
Sql>Select Ename,Sal,Deptno From Emp  Where Deptno IN(Select Deptno From
Emp Group by Deptno) And Sal IN(Select Max(Sal) From Emp Group by Deptno)
                                        And Deptno<>10;
```

Null Values in a Sub Query:

- One of the values returned by the inner query is a null value, and hence the entire query returns no rows. The reason is that all conditions that compare a null value result in a null.
- So whenever NULL values are likely to be part of the results set of a Sub Query, do not use the NOT IN operator. The NOT IN operator is equivalent to <> ALL operator.

```
Sql> Select E.Ename From Emp E  Where E.Empno IN(Select M.Mgr From Emp M );
```

Appling Sub Query in From Clause:

- A Sub Query in the from clause is equivalent to a view.
- The Sub Query in the from clause defines a Data source for that particular SELECT statement.

```
Sql>Select  d.Deptno,d.Dname,v.cnt  From Dept d, (Select Deptno,Count(*) cnt
          From  Emp Group by Deptno ) v  Where d.Deptno=v.Deptno And
          cnt>3;
Sql>Select    E.Ename,E.Sal,E.deptno,S.Salavg        From    Emp    E,(Select
Deptno,AVG(Sal) Salavg
 From Emp Group By Deptno) S Where E.Deptno=S.Deptno AND E.Sal>S.Salavg;
Sql> Select Deptno,SUM(Sal), SUM(Sal)/Tol_Sal*100"Salary%" From Emp,
          (Select SUM(Sal) Tol_Sal From Emp ) Group by Deptno,Tol_Sal;
Sql> Select E.EmpCount,D.DeptCount From(Select COUNT(*) EmpCount
          From Emp) E,  (Select COUNT(*) DeptCount  From Dept )D;

Sql>Select                                E.EmpCount,D.DeptCount,S.GradeCnt,
E.Empcount+D.DeptCount+S.GradeCnt
     TotalRecCnt From(Select COUNT(*) EmpCount From Emp) E,
       (Select  COUNT(*)  DeptCount  From  Dept  )D,  (Select  COUNT(*)
  GradeCnt From Salgrade)S;

Sql>Select A.Deptno "department Number", (A.NumEmp/B.TotalCount)*100
"%Employees",
       (A.SalSum/B.TotalSal)*100 "%Salary"  From(Select Deptno,Count(*)
NumEmp,
```

```
SUM(Sal) SalSum From Emp Group By Deptno) A,
        (Select COUNT(*) TotalCount,      SUM(Sal) TotalSalFrom
Emp)B;
```

Sub Select Statements:

- These are SELECT statements declared as part of the SELECT list.

```
Sql>Select Ename,Sal, (Select SUM(Sal) From Emp) "organizaton TolSal" From
Emp;

Sql> Select Ename,Sal,(Select MAX(Sal) From Emp) "organizaton Maximum
Salary", (Select MIN(Sal) From Emp ) "organizaton Lowest Salary" From Emp;
```

Co-related Sub Queries:

- It is another way of performing Queries upon the data with a simulation of Joins.

Syntax

```
Sql> SELECT column1, column2, ...
       FROM   table1 T_Alias1
       WHERE  column1 operator
       (SELECT  colum1, column2
        FROM    table2 t_Alias2
        WHERE   T_Alias1.Column Operator T_Alias2.Column
        );
```

Steps Performed

- Parent Query processed first.
- Passes the Qualified column value to the sub query WHERE clause.
- Get a candidate row (fetched by the outer query).
- Execute the inner query using the value of the candidate row.
- Use the values resulting from the inner query to qualify or disqualify the candidate.
- Repeat until no candidate row remains.

Exists:

- Returns true if inner query is success other wise False.

```
Sql> Select Empno,Ename,Sal,Deptno From Emp Where Deptno=10 AND
        EXISTS (Select COUNT(*)  From Emp  Where Deptno=10  AND
   Job='ANALYST' Group By Job  Having COUNT(*)>5);
```

Note: In the case Group by clause become as dummy clause so without using group by we can use having.

```
Sql> Select Empno,Ename,Sal,Deptno From Emp Where Deptno=30 AND
        EXISTS(Select COUNT(*) From Emp Where Deptno=10 AND
   Job='SALESMAN' Having COUNT(*)>3);

Sql>Select Deptno,Dname From Dept D  Where  EXISTS (Select * From Emp E
Where D.Deptno=E.Deptno);
Sql>Select Deptno,Dname From Dept D  Where NOT EXISTS(Select *  From Emp
E Where D.Deptno=E.Deptno );

Sql> Select P.Ename From Emp P Where EXISTS(Select *From Emp C
                    Where C.Empno=P.MGR);

Sql> Select P.Ename From Emp P Where NOT EXISTS(Select * From Emp C
                        Where C.Empno=P.MGR );

Sql> Select P.Ename From Emp P  Where  EXISTS(Select * From Emp C
                    Where C.MGR=P.Empno );
```

Sql> Select P.Ename From Emp P Where NOT EXISTS(Select * From Emp C Where C.MGR=P.Empno);

Sql>Select Empno,Ename,E.Deptno,Sal,MGR From Emp E Where E.Sal>ANY (Select M.Sal FROM Emp M Where M.Empno=E.MGR);

OCP:

Examine the data in the EMPLOYEES table:

LAST_NAME	DEPARTMENT_ID	SALARY
Getz	10	3000
Davis	20	1500
Bill	20	2200
Davis	30	5000

...

Which three subqueries work? (Choose three)

A. SELECT * FROM employees where salary > (SELECT MIN(salary) FROM employees GROUP BY department _ id);
B. SELECT * FROM employees WHERE salary = (SELECT AVG(salary) FROM employees GROUP BY department _ id);
C. SELECT distinct department_id FROM employees Where salary > ANY (SELECT AVG(salary) FROM employees GROUP BY department _ id);
D. SELECT department_id FROM employees WHERE SALARY > ALL (SELECT AVG(salary) FROM employees GROUP BY department _ id);
E. SELECT last_name FROM employees Where salary > ANY (SELECT MAX(salary) FROM employees GROUP BY department _ id);
F. SELECT department_id FROM employees WHERE salary > ALL (SELECT AVG(salary) FROM employees GROUP BY ANG (SALARY));

2)Examine the description of the EMPLOYEES table:

EMP_ID NUMBER(4) NOT NULL
LAST_NAME VARCHAR2(30) NOT NULL
FIRST_NAME VARCHAR2(30)
DEPT_ID NUMBER(2)
JOB_CAT VARCHARD2(30)
SALARY NUMBER(8,2)

Which statement shows the maximum salary paid in each job category of each department?

A. SELECT dept_id, job_cat, MAX(salary) FROM employees WHERE salary > MAX (salary);
B. SELECT dept_id, job_cat, MAX(salary) FROM employees GROUP BY dept_id,job_cat;
C. SELECT dept_id, job_cat, MAX(salary) FROM employees;
D. SELECT dept_id, job_cat, MAX(salary) FROM employees GROUP BY dept_id;
E. SELECT dept_id, job_cat, MAX(salary) FROM employees GROUP BY dept _ id job _ cat salary;

3)Examine the structure of the EMPLOYEES and DEPARTMENTS tables:

EMPLOYEESColumn name Data type Remarks
EMPLOYEE_ID NUMBER NOT NULL, Primary Key
EMP_NAME VARCHAR2 (30)
JOB_ID VARCHAR2 (20)
SALARY NUMBER
MGR_ID NUMBER References EMPLOYEE_ID COLUMN
DEPARTMENT ID NUMBER Foreign key to DEPARTMENT ID column of the DEPARTMENTS table
DEPARTMENTSColumn name Data type Remarks
DEPARTMENT_ID NUMBER NOT NULL, Primary Key
DEPARTMENT_NAME VARCHAR2(30)
MGR_ID NUMBER References MGR_ID column of the EMPLOYEES table

Evaluate this SQL statement:

SELECT employee_id, e.department_id, department_name, salary
FROM employees e, departments d WHERE e. department_id = d.department_id;

Which SQL statement is equivalent to the above SQL statement?

A. SELECT employee_id, department_id, department_name, salary FROM employees WHERE department_id IN (SELECT department_id FROM departments);
B. SELECT employee_id, department_id, department_name, salary FROM employees NATURAL JOIN departments;
C. SELECT employee_id, d.department_id, department_name, salary FROM employees e JOIN departments d ON e.department _ id = d. department_id;
D. SELECT employee_id, department_id, department_name, Salary FROM employees JOIN departments USING (e.department_id, d.department_id);

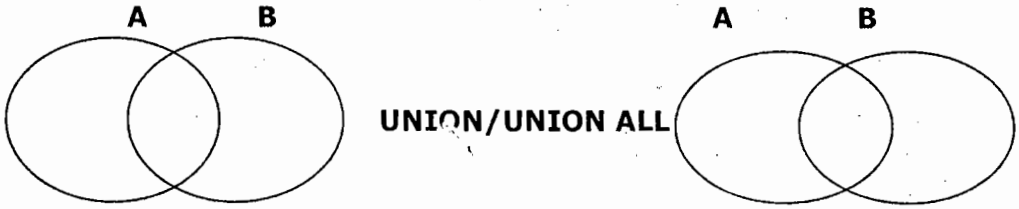
Using SET Operators

SET Operators

- The SET operators combine the results of two or more component queries into one result. Queries containing SET operators are called compound queries.
- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- All SET operators have equal precedence. If a SQL statement contains multiple SET operators, the Oracle server evaluates them from left (top) to right (bottom) if no parentheses explicitly specify another order.
- Different types of SET operators are
  - UNION Operator
  - UNION ALL Operator
  - INTERSECT Operator
  - MINUS Operator

Note: whenever these operators are used select statement must have

- Equal no of columns.
- Similar data type columns.



The Generic Syntax:

<Component query>
{UNION| UINON ALL |MINUS| INTERSECT}
<Component query>

UNION:

- The UNION operator returns all rows selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.

Guidelines:

- NULL values are not ignored during duplicate checking.
- By default, the output is sorted in ascending order of the first column of the SELECT clause.
- The IN operator has a higher precedence than the UNION operator.

UNION ALL:

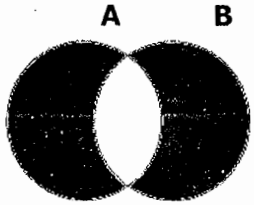
- Combines the results of Two **SELECT** statement into one result set including the duplicates.

Guidelines:

- Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.

INTERSECT:

- Use the INTERSECT operator to return all rows common to multiple queries.



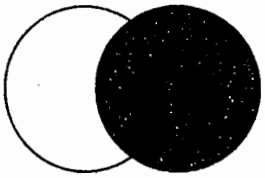
INTERSECT

Guidelines:

- Reversing the order of the intersected tables does not alter the result.
- INTERSECT does not ignore NULL values.

MINUS:

- The MINUS operator returns rows from the first query that are not present in the second query.



MINUS

Notes:

- The Queries are all executed independently but their output is merged.
- Only final query ends with a Semicolon.

Example:

```
Sql>Select Job  From Emp  Where Deptno=10
                UNION
      Select Job  From Emp Where Deptno=20;

Sql> Select Deptno,Job  From Emp Where Deptno=10
                    UNION
      Select Deptno,Job From Emp Where Deptno=20;

Sql> Select Empno,Ename From Emp Where Deptno=10
                    UNION
      Select Empno,Ename From Emp Where Deptno=20 Order by 2;

Sql>Select Empno,Ename,Job From Emp Where Deptno=(Select Deptno From Dept
Where Dname='SALES')
                              UNION
      Select Empno,Ename,Job From Emp Where Deptno=(Select Deptno From Dept
Where Dname='ACCOUNTING');
Sql>Select Deptno,Job From Emp Where Deptno=10
                    UNION ALL
      Select Deptno,Job From Emp Where Deptno=20;

Sql> Select Job  From Emp Where Deptno=10
                    INTERSECT
      Select Job  From Emp Where Deptno=20;

Sql>Select Job  From Emp  Where Deptno=10
                    MINUS
      Select Job From Emp Where Deptno=20

Sql>Select Deptno,Job From Emp Where Deptno=10
                    UNION
      Select Deptno,Job From Emp Where Deptno=20;

Sql> Select Deptno,Job From Emp Where Deptno=10
                    INTERSECT
      Select Deptno,Job From Emp Where Deptno=30

Sql>Select Deptno,Job From Emp Where Deptno=10
                    INTERSECT
      Select Deptno,Job From Emp Where Deptno=20
```

CONTROLLING USER ACCESS

And Data Base Security:

- Used to share the data, data base objects with other users.
- Users can also cancel the permission.
- Use the **GRANT** and **REVOKE** statements to grant and revoke object privileges

Privileges:

Privileges are the right to execute particular SQL statements. The database administrator (DBA) is a high-level user with the ability to grant users access to the database and its objects. The users require system privileges to gain access to the database and object privileges to manipulate the content of the objects in the database. Users can also be given the privilege to grant additional privileges to other users or to roles, which are named groups of related privileges.

Schemas:

A schema is a collection of objects, such as tables, views, and sequences. The schema is owned by a database user and has the same name as that user.

Grant:

- The GRANT command is used to allow another schema access to a privilege.
- GRANT command can be issued not only on TABLE OBJECT, but also on VIEWS,SYNONYMS,SEQUENCES Etc.
- The users are GRANTED at time all four(i.e. INSERT,UPDATE,DELETE,SELECT) by using ALL.

Syntax:

Sql>GRANT <privilege1 [, privilege2...] ON <ObjectName>TO <user1> [, user| role, PUBLIC...];

Creating Users:

Login as
User--system
Password--manager

Syntax:

Sql>Create User <User Name> Identified By <Password>;
Sql>Grant Connect,Resource To <User Nname>;

Illustrate:

Sql> Create User UserA Identified By userA;
Sql>Grant Connect To userA;

Login as Scott

Sql>Grant ALL ON Emp TO userA;
Sql>Grant INSERT,SELECT,DELETE ON Dept TO userA;

Grant columns:

Sql>Grant INSERT(Empno,Ename,Sal,Deptno) ON Emp TO userC;

Note:

- Only insert and update permission are allowed.

Sql>Grant Select ON Student TO userA,userB;

Sql>Grant Delete ON Supp TO Public;

Public

- Public Keyword represents all users in database.
- All users can Delete on Supp table.

Note:

- We cannot give permission at time on more than one data base object

Steps To Be Performed:

- Connect to the required user using the USER Name and Password.
- Execute the required SQL statement using the object hierarchy.

Login as userA

Sql>Select * From Scott.emp;
Sql>Insert Into Scott.Dept values(50,'SW','Hyd');
Sql>Select * From scott.Student;

Note:

- At time DML,DRL operation we have to mention owners name.

REVOKE Command:

- It used to remove the access allowed by GRANT.
- REVOKE privileges is assigned not only on TABLE OBJECT, but also on VIEWS,SYNONYMS,SEQUENCES Etc.

```
Sql>REVOKE <privilege1 [, privilege2...]
     ON <ObjectName>
      FROM <user1> [, user| role, PUBLIC...];

Sql>Revoke  Select ON Student From userA,userB;
 Sql> REVOKE ALL   ON Emp   From userA;
```

Types of Privileges:

System Privileges:

- They allow a user to perform certain action within the Database.

Object Privileges:

- An object privilege allow a user to perform certain actions on Database Objects.

Checking the object privileges granted:

- The schema object that stores that stores the information about the privileges granted is USER_TAB_PRIVS_MADE.
- The columns of USER_TAB_PRIVS_MADE.
- GRANTEE
- TABLE_NAME
- GRANTOR
- PRIVILEGE
- GRANTABLE
- HIERARCHY

```
Sql> Select GRANTEE,TABLE_NAME,GRANTOR,PRIVILEGE  From
                    USER_TAB_PRIVS_MADE  Where TABLE_NAME='EMP';
```

USER_COL_PRIVS

- This table query to view the object privileges granted to the user on specific columns

```
Sql>Select Grantee,Grantor,Column_Name  From USER_COL_PRIVS Where
Table_Name='EMP';
Sql>Select  GRANTEE  From All_Tab_Privs  Where Grantor='SCOTT' AND
Table_Name='EMP';
```

Checking Object Privileges Received:

- The schema object that stores that stores the information about the privileges that are received is USER_TAB_PRIVS_RECD.
- The columns of USER_TAB_PRIVS_RECD.
- OWNER
- GRANTEE
- TABLE_NAME
- GRANTOR
- PRIVILEGE
- GRANTABLE
- HIERARCHY

```
Sql> Select OWNER,TABLE_NAME,GRANTOR,PRIVILEGE From
USER_TAB_PRIVS_RECD;
```

Roles In Oracle:

- A role is a named group of related privileges that can be granted to the user.

Advantages:

- Rather than assigning Privileges one at a time directly to s USER, we can CREATE a ROLE, assign PRIVILEGES to that ROLE, and then GRANT that ROLE to Multiple USERS and ROLES.
- When you add or delete a privilege from a role, all users and roles assigned that ROLE automatically receive or lose those privileges.
- We can assign multiple roles to a single USER or ROLL to another ROLE.
- A role can be assigned with a psssword.

ROLE'S Creation:

- To creation a role we should have the CREATE ROLE SYSTEM privilege.
- The steps in implementing the roles
- Role creation
- Granting privileges to roles
- Granting ROLES TO USERS OR OBJECTS

Syntax:

```
Sql>CREATE ROLE<Role Name>
     [IDENTIFIED BY <Password>];

Sql> CREATE ROLE Sales_manager Identified By SalesAudit;
```

Note: We can alter the ROLE for password and new password.

```
Sql> Alter ROLE Sales_manager identified by salesAudit;
```

Grant a ROLE to a USER:

```
Sql>Grant Sales_manager TO  Scott;
```

Granting Multiple ROLES to another ROLE

```
Sql>Grant ROLE1,ROLE2,...
          TO <Target_role_name>;
```

Checking ROLES Granted To A user:

- The schema object USER_ROLE_PRIVS specifies the ROLES granted to a USER.
- The column of USER_ROLE_PRIVS
  - USERNAME
  - GRANTED_ROLE
  - ADMIN_OPTION
  - DEFAULT_ROLE
  - OS_GRANTED

```
Sql> Select USERNAME,GRANTED_ROLE From USER_ROLE_PRIVS;
```

System privileges Granted to a ROLL:

- The schema object ROLE_SYS_PRIVS specifies the SYSTEM PRIVILEGES granted to a ROLE.
- The columns of ROLE_SYS_PRIVS
  - ROLE
  - PRIVILEGE
  - ADMIN_OPTION
  -

Object privileges Granted to a ROLL:

- The schema object ROLE_TAB_PRIVS specifies the SYSTEM PRIVILEGES granted to a ROLE.
- The columns of ROLE_TAB_PRIVS

ROLE	COLUMN_NAME
OWNER	PRIVILEGE
TABLE_NAME	GRANTABLE

```
Sql>Select ROLE,PRIVILEGE From ROLE_TAB_PRIVS;
```

Revoking a ROLE:

Sql>Revoke Sales_Manager From Scott;
Sql> Revoke ALL ON Emp From Sales_Manager;

Dropping A ROLE:

Sql>Drop ROLE<Role name>;
Sql>DROP ROLE Sales_manager;

To change the password of user
(WE MUST BE UNDER your Owen login.)

Sql>Alter user krishna identified by wisdom;
Sql>Revoke connect,resource from krishna;
Sql>Drop user krishna;

Ocp:

1) You need to give the MANAGER role the ability to select from, insert into, and modify existing rows in
the STUDENT_GRADES table. Anyone given this MANAGER role should be able to pass those
privileges on to others.
Which statement accomplishes this?
A. GRANT select, insert, update ON student_grades TO manager
B. GRANT select, insert, update ON student_grades TO ROLE manager
C. GRANT select, insert, modify ON student_grades TO manager WITH GRANT OPTION;
D. GRANT select, insert, update ON student_grades TO manager WITH GRANT OPTION;
E. GRANT select, insert, update ON student_grades TO ROLE manager WITH GRANT OPTION;
F.GRANT select, insert, modify ON student_grades TO ROLE manager
WITH GRANT OPTION;

2) Which data dictionary table should you query to view the object privileges granted to the user on specific columns?

A. USER_TAB_PRIVS_MADE
B. USER_TAB_PRIVS
C. USER_COL_PRIVS_MADE
D. USER_COL_PRIVS

3) Which two statements accurately describe a role? (Choose two.)

A. A role can be given to a maximum of 1000 users.
B. A user can have access to a maximum of 10 roles.
C. A role can have a maximum of 100 privileges contained in it.
D. Privileges are given to a role by using the CREATE ROLE statement.
E. A role is a named group of related privileges that can be granted to the user.
F. A user can have access to several roles, and several users can be assigned the same role.

4) When should you create a role? (Choose two)

A. To simplify the process of creating new users using the CREATE USER xxx IDENTIFIED by
statement.
B. To grant a group of related privileges to a user.
C. When the number of people using the database is very high.
D. To simplify the process of granting and revoking privileges.
E. To simplify profile maintenance for a user who is constantly traveling.

5) Scott issues the SQL statements:

CREATE TABLE dept
(deptno NUMBER(2),
dname VARCHAR2(14),
loc VARCHAR2(13)};
GRANT SELECT

ON DEPT
TO SUE;

If Sue needs to select from Scott's DEPT table, which command should she use?
A. SELECT * FROM DEPT;
B. SELECT * FROM SCOTT. DEPT;
C. SELECT * FROM DBA.SCOTT DEPT;
D. SELECT * FROM ALL_USERS WHERE USER_NAME = 'SCOTT'
AND TABLE NAME = 'DEPT';

6) What is true about the WITH GRANT OPTION clause?
A. It allows a grantee DBA privileges.
B. It is required syntax for object privileges.
C. It allows privileges on specified columns of tables.
D. It is used to grant an object privilege on a foreign key column.
E. It allows the grantee to grant object privileges to other users and roles.

7) The DBA issues this SQL command:

CREATE USER scott
IDENTIFIED by tiger;
What privileges does the user Scott have at this point?

A. No privileges.
B. Only the SELECT privilege.
C. Only the CONNECT privilege.
D. All the privileges of a default user.

8) You are the DBA for an academic database. You need to create a role that allows a group of users to
modify existing rows in the STUDENT_GRADES table.
Which set of statements accomplishes this?

A. CREATE ROLL registrar; GRANT MODIFY ON student_grant TO registrar; GRANT registrar to user 1, user2, user3
B. CREATE NEW ROLE registrar; GRANT ALL ON student_grant TO registrar; GRANT registrar to user 1, user2, user3
C. CREATE ROLL registrar; GRANT UPDATE ON student_grant TO registrar; GRANT ROLE to user1, user2, user3
D. CREATE ROLL registrar; GRANT UPDATE ON student_grant TO registrar; GRANT registrar to user 1, user2, user3;
E. CREATE registrar; GRANT CHANGE ON student_grant TO registrar; GRANT registrar;

9) Which describes the default behavior when you create a table?
A. The table is accessible to all users.
B. Tables are created in the public schema.
C. Tables are created in your schema.
D. Tables are created in the DBA schema.
E. You must specify the schema when the table is created.

10) The user Sue issues this SQL statement:
GRANT SELECT ON sue. EMP TO alice WITH GRANT OPTION;
The user Alice issues this SQL statement:
GRANT SELECT ON sue. EMP TO reena WITH GRANT OPTION;

The user Reena issues this SQL statement:
GRANT SELECT ON sue. EMP TO timber;
The user Sue issues this SQL statement:
REVOKE select on sue. EMP FROM alice;
For which users does the revoke command revoke SELECT privileg on the SUE.EMP table?

A. Alice only B. Alice and Reena C. Alice, Reena, and Timber

D. Sue, Alice, Reena, and Timber

11) Examine the statement:

GRANT select, insert, update
ON student_grades
TO manager
WITH GRANT OPTION;
Which two are true? (Choose two.)

A. MANAGER must be a role.
B. It allows the MANAGER to pass the specified privileges on to other users.
C. It allows the MANAGER to create tables that refer to the STUDENT_GRADES table.
D. It allows the MANAGER to apply all DML statements on the STUDENT_GRADES table.
E. It allows the MANAGER the ability to select from, insert into, and update the STUDENT_GRADES table.
F. It allows the MANAGER the ability to select from, delete from, and update the STUDENT_ GRADES table.

12) The user Alice wants to grant all users query privileges on her DEPT table. Which SQL statement accomplishes this?

A. GRANT select ON dept TO ALL_ USER;

B. GRANT select ON dept TO ALL;

C. GRANT QUERY ON dept TO ALL_USERS

D. GRANT select ON dept TO PUBLIC;

13) Examine these statements:

CREATE ROLE registrar
GRANT UPDATE ON dtudent_grades TO registrar;
GRANT registrar to user1, user2, user3;
What does this set of SQL statements do?

A. The set of statements contains an error and does not work.

B. It creates a role called REGISTRAR, adds the MODIFY privilege on the STUDENT_GRADES object to the role, and gives the REGISTRAR role to three users.

C. It creates a role called REGISTRAR, adds the UPDATE privilege on the STUDENT_GRADES object to the role, and gives the REGISTRAR role to three users.

D. It creates a role called REGISTRAR, adds the UPDATE privilege on the STUDENT_GRADES object to the role, and creates three users with the role.

E. It creates a role called REGISTRAR, adds the UPDATE privilege on three users, and gives the REGISTRAR role to the STUDENT_GRADES object.

F. It creates a role called STUDENT_GRADES, adds the UPDATE privilege on three users, and gives the UPDATE role to the registrar.

VIEWS IN ORACLE

After a table is created and populated with data, it may become necessary to prevent all users from accessing all column of a table, for data security reasons. This would mean creating several tables having the appropriate number of columns and assigning specific users to each table, as required. This will answer data security requirements very well but will give rise to a great deal of redundant data being resident in tables, in the database.

To reduce **redundant data** to the minimum possible, Oracle allows the creation of an object called a **View.**

VIEW:

- t can be defined as an stored select statement.
- It will not hold data or store data by itself.
- It is a logical table based on one or more tables or views.
- View can be created based on a table called a BASE TABLE.
- View can be created as Object Views or Relational Views.
- The Object Views support
  *LOBs *Object Types *REFs
  *Nested Tables *Varray
- DML,DESC,SELECT allowed on views.

PREREQUISITES:

- Should have **CREATE VIEW** System privilege.
- To create a subview, we need **UNDER ANY VIEW** system privilege
- The OWNER of the Schema should have **SELECT,INSERT,UPDATE** or **DELETE** Rows from all tables or Views on which the **VIEW** is based.
- The above privileges must be granted the privileges directly, rather than a role.

Advantages of Views:

- Provides high security while sharing data between users
- **USER-VIEWS** hold the details of views.(date dictionary table).
- View can be used to make simple queries to retrieve the result of complicated quires.
- Views provide groups of users access to according to their particular criteria.

Syntax:

```
CREATE [OR REPLACE] [FORCE | NOFORCE ] VIEW view name
      [(AliasName[,AliasName.....])] As SubQuery
            [WITH {CHECK OPTION/READ ONLY}
                  CONSTRAINT ConstraintName];
```

***OR REPLACE:**

- Re-creates the view if it already exists.

***FORCE:**

- Specifies the View Should not be create if the base table does not Exist, which is default.

NOFORCE:

- Creates the view only if the base tables exist, Which is the default.

Alias Name:

- Specifies names for the expressions selected by the view query.

WITH CHECK OPTION:

- Specifies that only rows accessible to the view can be INSERTED or UPDATED or DELETED.

CONSTRAINT:

- Constraint is the name assigned to the WITH CHECK OPTION or READ ONLY constraint.

WITH READ ONLY:

- Ensures that no DML operations can be performed on this view.

TYPES OF VIEWS:

Simple Views:

- The view which is created without following clauses.
  - Join condition.
  - Group by
  - Having clause
  - Set operators
  - Distinct

Complex View

- The view which is created with any restriction clause is called complex view.
- Complex Views can contain Sub Queries .
- The SubQuery can contain, Queries
  - That Retrieve from multiple Base Tables
  - Group Rows using a GROUP BY or DISTINCT Clause.
  - Contain a Functional call.

```
Sql>Create VIEW Employees As  Select Empno "Emp Id", Ename Name, Sal "Basic
Salary",
                Job Designation  From Emp;
```

Selecting Data From A View:

```
        Sql> Select Name,Job From Employees;
        Sql> Select "Emp Id",name,"Basic Salary"*12 From Employees;
        Sql>select "Emp Id" ,Name,To_char("Basic Salary",'99,99,999.99') Monthly,
"Basic Salary"*12 Annual   From Employees  Where "Basic Salary">2500;
```

Complex View:

Example:

```
Sql>Create VIEW EmpInfo As        Select E.Empno EmployeeNo,E.Ename Name,
                D.Deptno DepartmentId, D.Dname DeparatmentName From Emp E,Dept D
                        Where D.deptno=E.Deptno ORDER BY D.Deptno;

Sql>Create VIEW EmpGrades As Select E.Ename Name, E.Sal   Basic, S.Grade Garde
                From Emp E,Salgrade S Where E.Sal BETWEEN S.Losal AND S.Hisal
                                                ORDER BY S.Grade;

Sql>Create VIEW EmpManagers As  Select Rownum SerialNo, Initcap(E.Ename)||'Works
Under'|| M.Ename "Employee And Managers"   From Emp E,Emp M    Where
E.Mgr=M.Empno;
Sql> Create or Replace VIEW EmpAccount As   Select Ename,Deptno,Sal Monthly,
                Sal*12 Annual  From Emp Where Deptno=(Select Deptno From Dept
                                Where Dname='ACCOUNTING') ORDER BY Annual;

Sql>Create VIEW PayInfo As Select Empno Ecode,Sal Basic,
                        Sal*0.25 Da, Sal * 0.35 HRA ,Sal * 0.12 PF, Sal + Sal *0.25 + Sal
                *0.35-sal *0.12 GROSS From Emp;

Sql> Create VIEW Dept_Analysis As SELECT Deptno,count(*) NoEmp,MIN (Sal)
Low_Pay,MAX(Sal) High_Pay, SUM(Sal) Tot_Pay,AVG(Sal) Avg_Pay FROM Emp Group By
Deptno;

Sql> Create or Replace VIEW CumSum As Select B.Sal,SUM(A.Sal) As Cum_Sal From
Emp A, Emp B          Where A.RowId<=B>RowId GROUP BY B.RowId,B.Sal;
Sql>Create or Replace VIEW OrgDesignations As Select Job From Emp Where
Deptno=10
                                UNION
        Select Job From Emp Where Deptno IN(20,30);
```

VIEW in Data Dictionary:

- Once the View has been created, we can query upon the DATA DICTIONARY table called USER_VIEWS to see the Name and definition of the View.
- The TEXT of the SELECT statement that constitutes the VIEW is stored in a LONG Column.

Retrieving Data from a View:

- Retrieves the VIEW definition from the Data Dictionary table USER_VIEWS.
- Check the Access privileges for the view.
- We can retrieve data from a view as you would from any table.
- We can display either the contents of the entire view or just specific rows and columns.

```
Sql> Select  GRANTEE  From All_Tab_Privs Where Grantor='SCOTT' AND
                        Table_Name=' OrgDesignations';
```

Modify A View :

- OR REPLACE Option is used to Modify an existing VIEW with a new definition.
- A VIEW can be altered without Dropping, recreating, and regranting object privileges.
- The assigned column Aliases in the CREATE VIEW Clause, are listed in the same order as the column in the SubQuery.

Creating Views With Columns Declarations:

- When a VIEW is being created, we can specify the Names of the Columns, that it can project, along with the VIEW's definition.

```
Sql>CREATE OR REPLACE VIEW EmpV (id_number, name, sal, department_id)
      AS SELECT  empno, ename, sal, deptno FROM    emp WHERE   deptno =30;

Sql>Select *From EmpV;

Sql>Create VIEW DeptSalSummary (DepartmentName,  MinimumSalary, MaxSalary,
AverageSalary,
              SalarySum) As  Select
      D.Danem,Min(E.Sal),Max(E.Sal),Avg(E.Sal),Sum(E.Sal)
                  From Emp E , Dept D Where E.Deptno=D.Deptno Group By
      D.deptno;
 Sql>Create View VDept20 As       Select *From Emp Where Deptno=20;
```

Inserting The Total Data of Employees From Department 20 Using View:

```
 Sql>Create Table Dept20 As Select *from  VDept20;
 Sql> Create Table EmpGrades(Employee, Designation, BasicSalary, Garde )As
       Select Ename,Job,Sal,Grade From Emp E, Salgrade S Where E.Sal BETWEEN
                  S.Losal AND S.Hisal;
```

DROPING A VIEW:

- A VIEW can remove without losing data ,because a view is based on underlying tables in the database.
- The DROP VIEW statement is used to remove a View permanently.
- Dropping a View has no affect on the tables upon which the View is created.
- VIEWS or APPLICATIONS based on deleted Views become invalid.
- We need DROP ANY VIEW privileges to remove the VIEWS.

Syntax:

>DROP VIEW view;

Example:

Sql >DROP VIEW EmpGrades;

INLINE Views:

- An INLINE view is a subquery with an alias (or correlation name) that you can use

within a SQL statement.

- An inline view is not a schema object.
- An inline view is placing a subquery in the FROM clause and giving that subquery an alias.

Example:

Sql>Select E.Ename, E.Sal, E.Deptno, I. maxsal FROM Emp E,(SELECT Deptno, max(Sal) maxsal FROM Emp GROUP BY Deptno) I
WHERE E.Deptno = I.deptno AND E.Sal < I.maxsal;

Rules for Performing DML Operations On A View:

- DML operations can be performed upon a table through VIEW.
- A row can be removed from a VIEW unless it contains.
  - JOIN Condition.
  - GROUP By Clause
  - HAVING Clause
  - SET Operators
  - DISTINCT/UNIQUE Key Word
  - The Column defined by Expressions
- Data can be added through a view, unless it contains any of the above rules and there are NOT NULL Columns.

Example:

Sql>Create View Vsalval As Select Empno,Ename,Sal,Job From Emp Where Sal>7000;
Sql>Desc Vsalval;

- Byexecuting this statement empno,ename, sal,job will be displayed because the Vsalval view contains four columns only.

Using the WITH CHECK OPTION Clause:

- To ensure that DML operations on the view stays within the domain of the VIEW by using the WITH CHECK OPTION clause.
- It is possible to perform referential integrity checks through views.
- Using VIEWS we can enforce constraints at the database level.
- The view can be used to protect DATA INTEGRITY, but the use is very limited.
- The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the VIEW cannot create rows which the VIEW cannot SELECT.
- VIEWS allows integrity constraints and data validation checks to be enforced on data being Inserted or Updated.

Example:

Sql>Create Or Replace VIEW Empvu20 As Select * From Emp Where Deptno=20 WITH CHECK OPTION CONSTRAINT Empvu20;

Sql> Create Or Replace VIEW EJobsman As Select *From Emp Where Job='SALESMAN' WITH CHECK OPTION CONSTRAINT EJobsman;

Adding the WITH READ ONLY option:

- We can ensure that no DML operations occur on your view by creating it with the WITH READ ONLYoption.
- Any attempt to perform a DML on any row in the view results in an Oracle server error.

Example:

Sql>Create Or Replace View empvu10 (employee_number, employee_name, job_title)AS Select employee_id, last_name, job_id from employeesWhere
department_id = 10 WITH READ ONLY;

View On View:

- We can create view on there view.

```
Sql>Create View V1 As  Select *from emp;
Sql> Create View v2 As  Select Empno,Ename,Job,Sal*0.22 Bonus From v1;
Sql> Drop view Emp;
```

Note:

- V1,V2 Views will become invalid.
- View will get the component and not to maintain.
- When we create an view with out a base table.
  (message:-view created with compilation errors.)
- As soon as create the base table Invalid become as Valid View.

VIEW CONSTRAINT:

- In practically Oracle does not enforce View constraint, but operations on VIEW are subject to the INTEGRITY CONSTRAINT defined on the underlying base tables.
- CONSTRAINTS on VIEWS can be enforces through CONSTRAINTS on base table.

Restriction On View Constraints:

- The VIEW Constraints are subset of TABLE CONSTRAINTS.
- Only UNIQUE,PRIMARY KEY And FOREIGN KEY CONSTRAINTS can be specified on VIEWS.
- The CHECK CONSTRAINT is imposed using WITH CHECK OPTION.
- As VIEW CONSTRAINTS are not enforced directly, we cannot specify INITIALLY DEFERRED or DEFREEABLE.
- VIEW CONSTRAINTS are supported only in DISABLE NOVALIDATE mode.

INITIALLY DEFERRED:

- It indicated that Oracle should check this CONSTARINT at the end of
- SUBSEQUENT transactions.

Note:

- INITIALLY DEFERRED is not valid if the CONSTRAINT is declared to be NOT DEFERRED.
- NOT DEFREEABLE CONSTRAINT is AUTOMATICALLY INITIALLY IMMEDIATE and can never be INITIALLY DEFERRED.

VALIDATE/NOVALIDATE:

- The Behavior of **VALIDATE** and **NOVALIDATE** always depends on whether the **CONSTRAINT** is **ENABLED/DISABLED** ,either **EXPLICTLY** or by **DEFAULT**.

DEFERRABLE Clause:

- The **DEFERRABLE** and **NOT DEFERRABLE** parameters indicate whether or not, in the subsequent transaction, constraint checking can be deferred until the end of the transaction using the SET CONSTRAINT statement.
- The default is NOT DEFERRABLE.

NOT DEFERRABLE:

- The checking of a **NOT DEFERRABLE CONSTRAINT** can Never be **DEFERRED** to the end of the transaction.

DEFERRABLE:

- It indicate that in subsequent transactions we can use the **SET CONSTRAINT(s)** Clause to defer checking the **CONSTRAINT** until after the transaction is committed.
- The setting is effect lets the user to disable the **CONSTRAINT** temporarily while making changes to the database that might violate the CONSTRAINT until all the changes are complete.
- To alter the CONSTRAINT Deferability, we must DROP the CONSTRAINT and recreate it.

Initially Immediate:

- It indicated that Oracle should check this CONSTRAINT at the end of each SQL statement.

- VIEW CONSTRAINT cannot be defined on attribute of an object column.
- Constraint State specifies how and when ORACLE should enforce the CONSTRAINT.
- The constraint state clauses can be specified in any order, but each state can be
  Specified only once.

```
Sql>Create  View EmpSalary (EmpId,Ename,Email  UNIQUE RELY DISABLE
NOVALIDATE,Constraint ID_PK PRIMARY KEY(EmpId) RELY disable NOVALIDATE)
                                    As  Select Empno,Ename,Email From Emp;
```

Example:

```
Sql>Create table FZ_SRV_ITEMS(Status number  Constraint chk_sta Check
                        (Status in(0,1,2,3,4)) ;
```

Note:

In the above case the default constraint is NOT DEFERRABLE INTITALLY IMMEDIATE
Constraint CHECK.

```
Sql>Create table FZ_SRV_ITEMS(
        Status number  Constraint chk_sta check (Status in(0,1,2,3,4))
DEFERRABLE  );
Sql>Alter Table FZ_SRV_ITEMS Modify Constraint chk_sta INITIALLY DEFERRED
```

SNAPSHOT:

- SNAPSHOT is Data Base object.
- It is a static picture of data..
- SNAPSHOT has
  - Structure
  - Data
- No DML possible on snapshot.
- Only select is possible on it.
- Performance is fast.

Syntax:

```
CREATE SANPSHOT<snapshot name>
AS
SELECT *FORM
<USER NAME>.<object name>@<database link>;
```

Note:

- When create SANPSHOT On oracle 8i we get message is
  'Materialized view created '.
- When delete the date in table, but the SNAPSHOT data not will be delete

Drawback:

- It static, we do any changes on base table there is no effected in SANPSHOT.

MATERIALIZED VIEWS:

- It is introduced in Oracle (8i) .
- It holds data.
- No DML allowed.
- Equivalent to snap-shop(DBA object)
- Materialized Views are used in DATA WAREHOUSES.
- They are used to increase the speed of queries on Very large databases.

Query Rewrite:

- Materialized Views improve query performance by PRECALCUTING Expensive JOIN and AGGREGATION operations on the DATABASE PRIOR to Execution time and stores the resuits in the DATABASE.
- The Query OPTIMIZER can make use of MATERIALIZED VIEWS by automatically recognizing when an Existing MATERIALIZED VIEW can and should be used to satisfy a Request.

- After above process is completed then QUERY OPTIMIZER transparently rewrites the request to use the MATERIALIZED VIEW.
- QUERIES are then directed to the MATERIALIZED VIEWS and not to the underling DETAIL TABLES or VIEWS.
- REWRITING QUERIES to use MATERIALIZED VIEWS rather than detail relations, results in a significant performance gain.

PREREQUISITES FOR MATERIALIZED VIEWS PRIVILEGES:

- Needs create materialized view privilege.
- Grant Query rewrite to scott;
- Alter Session Set QUERY_REWRITE_ENABLED=TRUE;

Set the InitSid.ORA File:

- OPTIMIZER_MODE=CHOOSE
- JOB_QUEUE_INTERVAL=3600
- JOB_QUEUE_PROCESSES=1
- QUERY_REWRITE_ENABLED=TRUE
- QUERY_REWRITE_INTEGRITY=ENFORCED

Syntax:

```
Sql> Create MATERIALIZED VIEW <View name>
        [ENABLED QUERY REWRITE]
        [Refresh On Commit]
        As
        <Select statements>;
```

Example:

```
Sql>Create MATERIALIZED VIEW Emp_Dno  ENABLED QUERY REWRITE
      As  Select Deptno,Sum(Sal),Count(Empno) From Emp Group By Deptno;
```

Creating Optimizer Statistics and Refreshing Materialized Views:

```
Sql>EXECUTE DBMS_UTILITY.ANALYZE_SCHEMA('SCOTT','ESTIMATE');
Sql> EXECUTE DBMS_MVIEW.REFRESH('Emp_Dno');
```

Testing Materialized View:

```
Sql>SET AUTOTRACE ON EXPLAIN;

Sql> Select Deptno,Sum(Sal),Count(Empno) From Emp Group By Deptno;
```

Example of Materialized View with Join/Aggregation:

```
Sql> Create MATERIALIZED VIEW Emp_Dept_Sum  ENABLE QUERY REWRITE
      AS  Select Dname,Job,Sum(Sal) From Emp E,Dept D Where
E.Deptno=D.Deptno Group By Dname,Job;
```

Creating Optimizer Statistics and Refreshing Materialized Views:

```
Sql>EXECUTE DBMS_UTILITY.ANALYZE_SCHEMA('SCOTT','ESTIMATE');
Sql> EXECUTE DBMS_MVIEW.REFRESH('Emp_Dept_Sum');
```

Testing Materialized View:

```
Sql>SET AUTOTRACE ON EXPLAIN;
Sql>  Select  Dname,Job,Sum(Sal)  From  Emp  E,Dept  D  Where
E.Deptno=D.Deptno Group By Dname,Job;
```

Putting the Things With RollUp:

```
Sql> Create Materialized View Emp_Dept_Agg  ENABLED QUERY REWRITE
      AS  Select Deptno,Job,Count(*),Sum(Sal)  From Emp
                  Group By Rollup(Deptno,Job);
```

OCP Eaxmples:

1)Which two statements about views are true?

A. They hide data complexit.

B. DML operation On based table of view can not retrieve by Simple View .

C. You can Create View with out a Base Table .

D. DML operation can perform through Materialized View when it create REFRESH ON
COMMIT option.

2) Which two statements about views are true? (Choose two.)
A. A view can be created as read only.
B. A view can be created as a join on two or more tables.
C. A view cannot have an ORDER BY clause in the SELECT statement.
D. A view cannot be created with a GROUP BY clause in the SELECT statement.
E. A view must have aliases defined for the column names in the SELECT statement.

3) Which View statements give the out in ascending order by Ename.
A. CREATE or replace VIEW my_view AS SELECT*FROM emp ORDER BY ename;
B. CREATE or replace VIEW my_view AS (SELECT*FROM emp ORDER BY ename);
C. A And B
D.None

4) You created a view called EMP_DEPT_VU that contains three columns from the EMPLOYEES and DEPARTMENTS tables:
EMPLOYEE_ID, EMPLOYEE_NAME AND DEPARTMENT_NAME.
The DEPARTMENT_ID column of the EMPLOYEES table is the foreign key to the primary key
DEPARTMENT_ID column of the DEPARTMENTS table.
You want to modify the view by adding a fourth column, MANAGER_ID of NUMBER data type from the EMPLOYEES tables.
How can you accomplish this task?
A. ALTER VIEW EMP_dept_vu (ADD manger_id NUMBER);
B. MODIFY VIEW EMP_dept_vu (ADD manger_id NUMBER);
C. ALTER VIEW emp_dept_vu AS SELECT employee_id, employee_name, department_name, manager_id FROM employee e, departments d
WHERE e.department _ id = d.department_id;
D. MODIFY VIEW emp_dept_vu AS SELECT employee_id, employee_name, department_name, manager_id FROM employees e, departments d
WHERE e.department _ id = d.department_id;
E. CREATE OR REPLACE VIEW emp_dept_vu AS SELECT employee_id, employee_name, department_name, manager_id FROM employee e, departments d WHERE e.department _ id = d.department_id;
F.You must remove the existing view first ,and then run the CREATE VIEW Command with a new column list to modify a view.

5)Which two statements about views are true?
A. They hide data complexit.
B. DML operation On based table of view can not retrieve by Simple View .
C. You can Create View with out a Base Table .
D. DML operation can perform through Materialized View when it create REFRESH ON COMMIT option.

6)What is necessary for your query on an existing view to execute successfully?
A. The underlying tables must have data.
B. You need SELECT privileges on the view.
C. The underlying tables must be in the same schema.
D. You need SELECT privileges only on the underlying tables.

7) Which view statements is true?
A. Operation performed on view dose not affect the base table.
B. view cannot be created without a table.
C. Only Key Preserved table of Join view can be updated
D. View can be thought of as a Stored Query or a Virtual Table.

8) To impose constraint on view _____option is used.
A. With read only B. With Check Option
C. Join View D. None

9) You need to create a view EMP_VU. The view should allow the users to manipulate the records of only the employees that are working for departments 10 or 20.

Which SQL statement would you use to create the view EMP_VU?
A. CREATE VIEW emp_vu AS SELECT * FROM employees WHERE department_id IN (10,20);
B. CREATE VIEW emp_vu AS SELECT * FROM employees WHERE department_id IN (10,20) WITH READ ONLY;
C. CREATE VIEW emp_vu AS SELECT * FROM employees WHERE department_id IN (10,20) WITH CHECK OPTION;
D. CREATE FORCE VIEW emp_vu AS SELECT * FROM employees WHERE department_id IN (10,20);
E. CREATE FORCE VIEW emp_vu AS SELECT * FROM employees WHERE department_id IN (10,20) NO UPDATE;
10)Examine the structure if the EMPLOYEES table:
Column name Data Type Remarks
EMPLOYEE_ID NUMBER NOT NULL, Primary Key
EMP_NAME VARCHAR2(30)
JOB_ID VARCHAR2(20) NOT NULL
SAL NUMBER
MGR_ID NUMBER References EMPLOYEE_ID column
DEPARTMENT_ID NUMBER Foreign key to DEPARTMENT_ID
column of the DEPARTMENTS table
You need to create a view called EMP_VU that allows the user to insert rows through the view. Which SQL
statement, when used to create the EMP_VU view, allows the user to insert rows?

A. CREATE VIEW emp_Vu AS SELECT employee_id, emp_name, department_id FROM employees WHERE mgr_id IN (102, 120);
B. CREATE VIEW emp_Vu AS SELECT employee_id, emp_name, job_id department_id FROM employees WHERE mgr_id IN (102, 120);
C. CREATE VIEW emp_Vu AS SELECT department_id, SUM(sal) TOTALSAL FROM employees WHERE mgr_id IN (102, 120) GROUP BY department_ id;
D. CREATE VIEW emp_Vu AS SELECT employee_id, emp_name, job_id, DISTINCT
department_id FROM employees;

WORKING WITH INDEXES:

INDEX:

- INDEX is a schema object ,Which a pointer locates the physical address of data.
- INDEX used by the Oracle Server to speed up the retrieval, manipulate of rows.

Specification Of An INDEX:

- INDEX can be created explicitly or automatically.
- INDEX is activated when indexed column is used in where clause.
- INDEX Necessity of disk I/O by using an indexed path to locate data quickly.
- INDEX is activated when indexed column is used in where clause.
- INDEX is used and maintained automatically by the Oracle Server.
- When you drop a table, corresponding indexes are also dropped.

INDEX Creation is of Two Types:

1)Automatic:

- A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.

2) Manual:

- Users can create nonunique indexes on columns to speed up access to the rows.
- One Table more than one Index can be created, But this does not mean that , more the Indexes lead to more faster performance.
- Each DML operations that is committed on table on a table with Indexes means that Indexes must be updated.

- USER _INDEXES holds the details of Indexes.

Process Of INDEXES:

- Indexing involves forming a two dimensional matrix completely independent of the table on which the index is being created.
- The two dimensional matrix will have a
  - Single Column
  - Address Field
- **Single Column** will hold sorted data, extracted from the table column(s) on which the index is created.
- **Address Filed** identifies the location of the record in the Oracle database.(**ROWID**)

Note:

- The records in the index are stored in the ascending order of the INDEX column(s).

Types Of INDEXES:

NORMAL INDEXES:

- They are default Indexes.
- They are created with

BITMAP INDEXES:

- They stores ROWIDs associated with a key value as a Bitmap.

COMPOSITE INDEX:

- If we define a index on more than one column, It is called Composite Index.

PARTITIONED INDEX:

- They contain partitions containing an entry for each value that appears in the Indexed column of the table.

FUNCTION BASED INDEXES:

- When we create Index on column with function it is called function based index.
- Enable query to evaluate values returned by expression.

DOMAIN INDEXES:

- **They are INDEXES which are instances of an application specific index of type Index type.**

PRE REQUISITES:

- The table or CLUSTER to be INDEXED must be in the Own Schema.
- INDEX object privilege should be available on the table to INDEXED.
- Create any Index SYSTEM privilege must be available.
- UNLIMITED TABLESPACE system privilege or SPACE QUOTA on TABLE SPACES must be available.

Restriction On INDEX Columns:

- An INDEX cannot be created on Columns or ATTRIBUTES whose type is...
  - LONG
  - LONG RAW
  - LOB
  - REF
- An INDEX on REF type column Or attributes have to be declared with a SCOPE clause.

RESTRICTION:

- If INDEX is locally partitioned then the TABLE must be portioned.
- If Global Temporary Table ,then INDEX is also Temporary with the same scope,as that of the table.

Syntax:

```
Create [UNIQUE] INDEX IndexName
        OR
[BITMAP] ON
        TableName(Column_name[,ColumnName....])
        TABLESPACE TableSpaceName;
```

Unique:

- Specify UNIQUE to indicate that the value of the column or columns upon which the Index is based must be UNIQUE.

Restrictions:

- We cannot specify both UNIQUE and BITMAP.
- UNIQUE Cannot be specified for a domain Index

Restrictions:

- BITMAP cannot be specified when creating a global portioned Index.
- We cannot specify both UNIQUE and BITMAP.
- BITMAP cannot be specified for a DOMAIN INDEX.

CLUSTER-INDEX-CLAUSE:

- It identified the CLUSTER for which a CLUSTER INDEX has to be created.

Simple Index Example:

```
Sql>Create INDEX SalIdx On Emp(Sal);

Sql>Create INDEX DnoIdx ON Dept(Dname);
```

Creating Unique INDEXES:

```
Sql>Create Unique Index Eno_Unq_Idx On Emp(Empno);
```

Creating Composite Unique Indexes:

- Composite Index is an Index on multiple Columns.

  Sql>Create Unique Index Eno_Ename_CINX On Emp(Empno,Ename);

Creating Function Based INDEXES:

- 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.
- Function-based indexes defined with the UPPER(Column_Name) or LOWER(Column_Name) allow case-insensitive searches.
- To ensure that the Oracle Server uses the index rather than performing a full table scan, be sure that the value of the function is Not Null in subsequent queries.
- The Oracle Server treats indexes with columns marked DESC as function-based indexes.

Example:

```
Sql>Create INDEX upper_dept_name_idx ON  Dept(UPPER(Dname));

Sql>Select *From  Dept Where  UPPER(Dname) = 'SALES';
```

Note: The Function Based Indexes are used only when the Query Statement is executed through the specified function.

Bitmap Indexing:

- Specify BITMAP to indicate that INDEX has to be create with a BITMAP for each DISTINCT KEY.
- BITMAP Indexes store the ROWID's associated with a key value as a BITMAP.
- BITMAP Indexes should be used only when the data is infrequently updated.
- BITMAP Indexes add to the cost of all data manipulation transaction against the tables they INDEX.
- The ORACLE OPTIMIZER can dynamically convert Bitmap Indexes to ROWID's during the query processing.

Example:

```
Sql>Create Bitmap Index EmpBitMapJob On Emp(Job);
```

Limitation:

- **BITMAP INDEXES** should not be used for tables involved in **Online Transaction Processing applications** due to the Internet Mechanisms Oracle user to maintain them.

When to Create An Index:

- A Column contains a wide range of values.
- A Column contains a 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
- The Column is used frequently in the **WHERE** clause or **Join Condition**.

When Not to Create an Index:

- 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.**

Removing an Index:

- Remove an index from the data dictionary by using the DROP INDEX command.

DROP INDEX <Index_Name>;

Ocp:

1) _____a pointer locates the physical address of data.

A. View B. Table C. Index D. None

2) Creation of _____ Index imposes a constraint on the column.

A. Primary Key B. Unique C. A and B D. None

3) _____Index has less storage when compared to normal index.

A. Unique B. Function Based C. Bitmap D. None

4) Examine the SQL statements that creates ORDERS table

```
CREATE TABLE orders
(SER_NO NUMBER UNIQUE,
ORDER_ ID NUMBER,
ORDER_DATE DATE NOT NULL
STATUS VARCHARD2(10)
CHECK (status IN ('CREDIT','CASH')),
PROD_ID_NUMBER
REFERENCES PRODUCTS(PRODUCT_ID),
ORD_TOTAL NUMBER,
PRIMARY KEY (order id, order date));
```

For which columns would an index be automatically created when you execute the aboveSQL statement?

(Choose two)

A. SER_NO B. ORDER_ID C. STATUS D. PROD_ID E. ORD_TOTAL

F. Composite index on ORDER_ID and ORDER_DATE

5)For which two constraints does the Oracle Server implicitly create a unique index? (Choose two.)

A. NOT NULL B. PRIMARY KEY C. FOREIGN KEY D. CHECK

E. UNIQUE

6) In which scenario would index be most useful?

A. The indexed column is declared as NOT NULL.

B. The indexed columns are used in the FROM clause.

C. The indexed columns are part of an expression.

D. The indexed column contains a wide range of values.

7) Examine the structure of the EMPLOYEES table:

Column name Data type Remarks
EMPLOYEE_ID NUMBER NOT NULL, Primary Key
LAST_NAME VARCNAR2(30)
FIRST_NAME VARCNAR2(30)
JOB_ID NUMBER
SAL NUMBER
MGR_ID NUMBER References EMPLOYEE_ID column
DEPARTMENT_ID NUMBER

You need to create an index called NAME_IDX on the first name and last name fields of the EMPLOYEES table. Which SQL statement would you use to perform this task?

A. CREATE INDEX NAME _IDX (first_name, last_name);
B. CREATE INDEX NAME _IDX (first_name, AND last_name)
C. CREATE INDEX NAME_IDX ON (First_name, last_name);
D. CREATE INDEX NAME_IDX ON employees (First_name, AND last_name);
E. CREATE INDEX NAME_IDX ON employees (First_name, last_name);
F. CREATE INDEX NAME_IDX FOR employees (First_name, last_name);

HIERARCHICAL Queries:

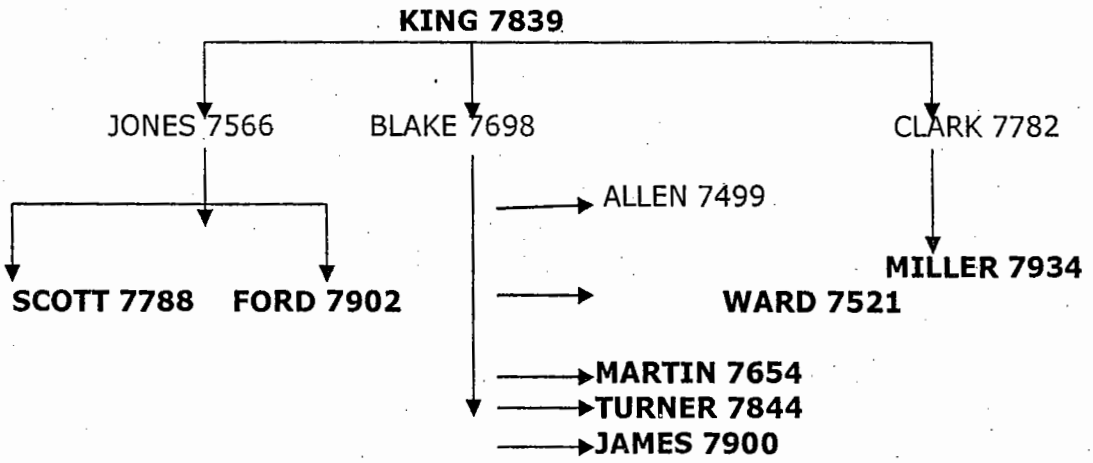
- These are queries that are executed upon tables that contain hierarchical data.
- To Execute the hierarchical queries, we need the following queries.

- START WITH : It specifies the root rows of the hierarchy.
- CONNECT BY : It is used to specify the relationship between parent rows and child rows of the hierarchy.
- WHERE : It is used to restrict the rows returned by the Query without affecting other row of the hierarchy.

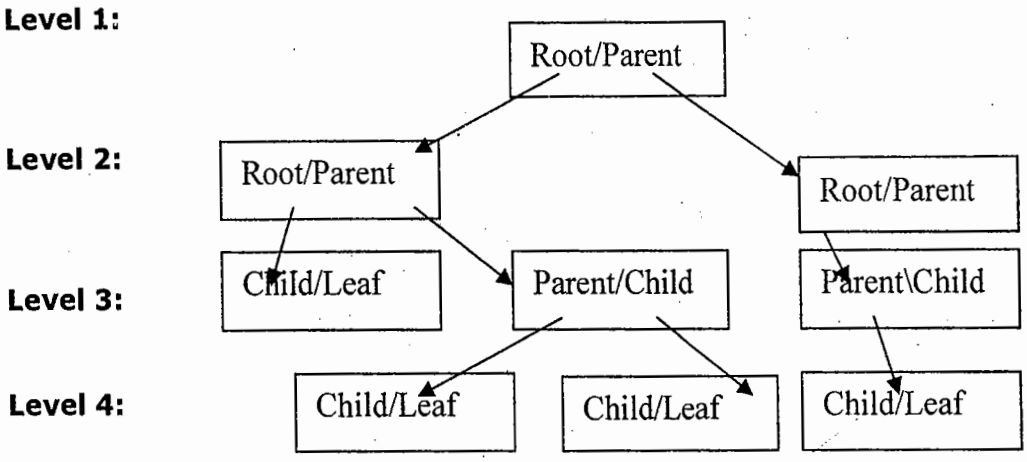
Restriction:

- They cannot be used to perform joins.
- They cannot select data from a View, Whose Query performs a Join.
- If ORDER BY Clause is used, then the rows are returned as per the specification in the ORDER BY Clause.

The General Hierarchy of Emp Data:



The General Structure Hierarchical Tree:



Note: To define Hierarchical Queries properly we must use the following clauses.

- START WITH
- CONNECT BY

START WITH Clause:

- It identifies the row(s) to be used as the Root(s) of a hierarchical Query.
- It specifies a condition that the roots must specify.
- If START WITH is omitted, Oracle uses all rows in the table as ROOT rows.
- A START WITH Condition can constraint a Sub Query.

CONNECT By Clause:

- This clause specifies the Relationship between Parent and Child Rows, in a hierarchical query.
- This clause contains a condition that defines a relationship.
- This condition can be any condition as defined by the syntax description.
- Within the condition, some part of the condition must use the PRIOR operator, which refers to the Parent row.
- The forms of PRIOR operators is
  - PRIOR Expr Comparison –operator Expr.
  - Expr Comparison – operaor Expr.
- The Clause can contain other conditions to further filter the rows selected by the query.
- It cannot contain a Sub Query.

Example:

```
Sql>Select Ename,Empno,Mgr,Job From Emp CONNECT BY PRIOR Empno=Mgr;

Sql>Select Ename,Empno,Mgr,Job From Emp  START WITH Job='PRESIDENT'
                CONNECT BY PRIOR Empno=Mgr;

Sql>Select Ename,Empno,Mgr,Job From Emp  START WITH Ename='KING'
          CONNECT BY PRIOR Empno=Mgr;

Sql>Select Ename,Empno,Mgr,Job From Emp START WITH Sal=5000 CONNECT
BY PRIOR Empno=Mgr;
```

SYS_CONNECT_BY_PATH Function:\

- The function returns the path of a column value from Root to Node, with column values separated by 'Char' for each row returned by CONNECT BY condition.
- Can work on any datatype CHAR,VARCHAR2,NCHAR, or NVARCHAR2.

```
Sql>select Ename,SYS_CONNECT_BY_PATH(Ename,'/') "Path"From Emp
                Start With Ename='KING'  Connect By Prior Empno=MGR;
```

PSEUDO Columns

- Pseudo column behave like a table column, but is not actual stored in a table.
- Upon Pseudo columns only SELECT's can be implemented, but INSERT,UPDATE or DELET cannot be implemented.
- The available Pseudo Columns are....
  - CURRVAL
  - NEXTVAL
  - LEVEL
  - ROWID
  - ROWNUM

CURRVAL And NEXTVAL Pseudo Columns:

- These Pseudo columns are applied upon the SEQUENCE Schema Object.
- CURRVAL returns the Current Value of Sequence.
- NEXTVAL Increments the sequence and returns the Next Value.
- The CURRVAL and NEXTVAL can be used only in....
  - The SELECT list of a SELECT statement
  - The VALUES clause of an INSERT Statement
  - The SET Clause of an UPDATE Statement.

Restrictions:

- The CURRVAL and NEXTVAL cannot be used in.....
  - A Sub Query
  - A View's Query or SNAPSHOT's Query.
  - S SELECT Statement with a GROUP BY or ORDER BY Clause.
  - A SELECT Statement with the DISTINCT operator.
  - A SELECT Statement that is combined with another SELECT statement with UNION,INERSECT,MINUS SET Operator.
  - The WHERE Clause of a SELECT Statement.
  - The DEFAULT Value of Column in a CREATE TABLE or ALTER TABLE Statement.
  - The Condition of a CHECK Constraint.
  - In a single SELECT Statement, all referenced sequences, LONG Columns, Updated tables, and locked tables, must be located on the same database.
  - The First reference to the NEXTVAL return the SEQUENCES Initial Value .
  - Before the CURRVAL can be used for a SEQUENCE in a session, first the SEQUENCE should be incremented with NEXTVAL.
  - A SEQUENCE can be incremented only in a single SQL Statement.
  - A SEQUENCE can be accessed by many users concurrently with no WAITING or LOCKING.
  - CURRVAL and NEXTVAL should be qualified with the name of the Sequence.

Syntax:

SEQUENCENAME.CURRVAL→ Returns the Current Value of the Sequence.

SEQUENCENAME.NEXTVAL →Increments the Sequence Value by the Declare Specification .

SEQUENCE:

☼A SEQUENCE is a schema object that can generate unique sequential values.

☼The SEQUENCE Values are often user for PRIMARY KEY'S and UNIQUE KEY's.

☼To refer to the Current or Next Value of a SEQUENCE in the Schema of another used. The following privileges be available.

- SELECT OBJECT PRIVILAGE
- SELECT ANY SEQUENCE

☼For SEQUENCES in other Schema the Qualifying Syntax is

- SCHEMANAME.SEQUENCENAME.CURRVAL
- SCHEMANAME.SEQUENCENAME.NEXTVAL

☼To refer to the value of a SEQUENCE on a Remote Database, The SEQUENCE should be qualified with a complete or partial name of the Database Link.

- SCHEMANAME.SEQUENCENAME.CURRVAL@DBLINK
- SCHEMANAME.SEQUENCENAME.NEXTVAL@DBLINK

Creating SEQUENCES:

Purpose:

- An object from which multiple uses may generate Unique Inter.
- Can be used to generate PRIMARY KEY values automatically.

Syntax:

```
CREATE SEQUENCE sequenceName
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE n | NOCACHE]
ORDER/NOORDER;
```

INCREMENT BY:

- Specifies the interval between the Sequence Numbers.
- Value can be Positive or Negative, but can not be 0.
- If the value is positive it is Incremental Sequence else it Decremental Sequence.

MINVALE:

- **Specifies the Sequence's Minimum Value.**

NOMINVALUE:

- Specifies a minimum value of 1 for an ascending sequence and -(10)^26) for a descending sequence .

MAXVALUE:

- Specifies the Maximum value that sequence can generate.

NOMAXVAULE:

- Specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence.

START WITH:

- Specifies the first sequence number to be generated.
- For Ascending sequences the default value is SEQUENCES's MINIMUM value.

CYCLE:

- Specifies whether the sequence continues to generate values after reaching its maximum or minimum value .

NOCYCLE:

- Specifies the SEQUENCE cannot general more values after the targeted limit

CACHE:

- Specifies how many values the Oracle Server preallocates and keep in memory.

NOCACHE:

- Specifies the values of a SEQUENCE are not Preallocated.
  - If the above parameters are not specified by default 20 values are cached.

ORDER:

- Guarantee the sequence numbers to be generated in the order of request.

NOORDER:

- Does not guarantee the sequence Number with Order.

Note:

- If the Above parameters are not specified by default
  - START WITH Will be 1.
  - INCREMENT BY Will be positive 1.
  - SEQUENCE is NOCYCLE.
  - The CACHE Value Will be 20
  - SEQUENCE is ORDER

Test Table:

```
Sql>CREATE TABLE TDEPT   (
        DEPTNO  NUMBER(4)Constraint Tdeptno_PK Primary Key,
        DNAME   VARCHAR2(14 BYTE), LOC     VARCHAR2(13 BYTE)  );
```

Creation of Incremental Sequence:

```
Sql> CREATE SEQUENCE TDept_DeptNo_Seq
        INCREMENT BY 10
        START WITH 10
        MINVALUE 0
        MAXVALUE 9999
        NOCACHE
        NOCYCLE;
```

- In this sequence named Dept_**DeptNo_Seq** to be used for the Deptno column of the Dept table.
- The sequence starts at 10, does not allow caching, and does not cycle.

```
Sql> Insert into TDept Values(TDept_DeptNo_Seq.NEXTVAL,'SOFTWARE','HYD');
```

Creating A Sequence with CYCLE:

```
Sql>CREATE SEQUENCE TDept_DeptNo_Seq
        INCREMENT BY 10
        START WITH 10
        MINVALUE 0
        MAXVALUE 9999
        NOCACHE
        CYCLE;
```

Creation Of Decremental Sequence:

```
Sql>CREATE SEQUENCE TDept_DeptNo_Seq
        INCREMENT BY -1
        START WITH 10
        MINVALUE 0
        MAXVALUE 10
        NOCACHE
        NOCYCLE;
```

Modifying a Sequence:

- The ALTER command can be used to change the present status of a SEQUENCE.
- The ALTER SEQUENCE command can be used to change....
  - Increment Value
  - Maximum Value
  - Minimum Value
  - Cycle Option
  - Cache Option

Syntax:

```
>ALTER  SEQUENCE sequence
        [INCREMENT BY n]
        [{MAXVALUE n | NOMAXVALUE}]
        [{MINVALUE n | NOMINVALUE}]
```

[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];

Example:

```
Sql>Alter SEQUENCE TDept_DeptNo_Seq
        INCREMENT BY 10
        MAXVALUE 500
        NQCACHE
        NOCYCLE;
```

Guidelines for Modifying a Sequence:

- You must be the owner or have the ALTER privilege for the sequence 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.

Viewing the Current Value of a Sequence

```
Sql> Select TDept_DeptNo_Seq.Currval from Dual;
```

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.

```
Sql>DROP SEQUENCE TDept_DeptNo_Seq;
```

Remember:

- The owner of the sequence or have the DROP ANY SEQUENCE privilege to remove it.
- Once removed, the sequence can no longer be referenced.
- The Data Dictionary in which the information of SEQUENCES are stored is USER_OBJECTS.

```
Sql>Select Object_Name From User_Objects  where Object_Name
                                        Like'TDEPT_DEPTNO_SEQ';
```

- The Setting of the SEQUENCE can be confirmed by Selecting on USER_SEQUENCES catalog.

```
Sql>Select Sequence_Name,Min_Value,Max_Value,Increment_By,
                    Last_Number  From User_Sequences
                        Where
                    Sequence_Name='TDEPT_DEPTNO_SEQ';
```

OCP:

1) Which two statements about sequences are true? (Choose two)

A. You use a NEXTVAL pseudo column to look at the next possible value that would be generated from a sequence, without actually retrieving the value.

B. You use a CURRVAL pseudo column to look at the current value just generated from a sequence, without affecting the further values to be generated from the sequence.

C. You use a NEXTVAL pseudo column to obtain the next possible value from a sequence by actually retrieving the value from the sequence.

D.You use a CURRVAL pseudo column to generate a value from a sequence that would be used

for a specified database column.

E. If a sequence starting from a value 100 and incremented by 1 is used by more then one application, then all of these applications could have a value of 105 assigned to their column whose value is being generated by the sequence.

F. You use REUSE clause when creating a sequence to restart the sequence once it generates the maximum value defined for the sequence.

2) What is true about sequences?

A. Once created, a sequence belongs to a specific schema.
B. Once created, a sequence is linked to a specific table.
C. Once created, a sequence is automatically available to all users.
D. Only the DBA can control which sequence is used by a certain table.
E. Once created, a sequence is automatically used in all INSERT and UPDATE statements.

3) What is true about sequences?

A. The start value of the sequence is always 1.
B. A sequence always increments by 1.
C. The minimum value of an ascending sequence defaults to 1.
D. The maximum value of descending sequence defaults to 1.

4) This sequence pseudo column contains the most recently generated value the sequence has derived: _____

A.CURRVAL B.NEXTVAL C.NOCYCLE D.MAXVALUE

5) The alter sequence command will affect only the future sequences numbers.

A. True B. False C. A&BD. None

6) Before using currval for a sequence we must first increment the sequence with nextval.

A. True B. False C. A&B D. None

Level Pseudo Column:

The LEVEL pseudo column returns 1 for a root row, 2 for a child of a root, and so on.

- Child→Any non root node.
- Root→Highest Node within an Inverted tree.
- Parent→Any Node/Row That has Children.
- Leaf→any node without children.
- To establish the Hierarchical relationship with LEVEL we need.
  - START WITH Clause.
  - CONNECT BY Clause.

```
Sql>Select Ename,Job,MGR,Level From Emp;

Sql> Select level,Ename,Empno,Mgr,Job From Emp Start with Job='PRESIDENT'
        Connect By Prior Empno=Mgr order by level;

Sql> Select Lpad(' ',2*(level-1)) ||Ename Org_Lvel,Empno,Mgr,Job From Emp
        Start with Job='PRESIDENT' Connect By Prior Empno=Mgr;

Sql> Select Lpad(' ',2*(level-1)) ||Ename Org_Lvel,Empno,Mgr,Job From Emp
        Where Job<>'ANALYST' Start with Job='PRESIDENT' Connect By Prior
    Empno=Mgr;
```

Sql>Select Lpad(' ',2*(level-1)) ||Ename Org_Lvel,Empno,Mgr,Job From Emp Start with
Job='PRESIDENT' Connect By Prior Empno=Mgr and Level<=2;

Select Nth Highest Value from table:
Syntax

Sql>Select Level,Max(ColName)
From TableName
Where Level=&Levelno
Connect By Prior Colname>Colname
Group by Level;

Example:

Sql>Select Level,Max(Sal) From Emp Where Level=&Levelno Connect By Prior Sal>Sal
Group By Level;

Select Nth Lowest Value From Table:
Syntax

Sql>Select Level,Min(ColName) From TableName Where Level=&Levelno
Connect By Prior Colname<Colname Group by Level;

Example:

Sql> Select Level,Min(Sal) From Emp Where Level=&Levelno Connect By Prior Sal<Sal
Group By Level;

ROWNUM Pseudo Column:

- The Oracle Engine assign a ROWNUM value to each row is it is retrieved.
- The first row select has a ROWNUM of 1,The second has 2.and so on...
- When ORDER BY clause follows a ROWNUM, The rows will be Re-Ordered by ORDER BY clause.
- If ORDER BY clause is embedded in s Sub Query and ROWNUM condition is placed in the TOP_LEVEL Query, Then the ROWNUM condition can be forced to get applied after the ordering of the rows.
- Conditions testing for ROWNUM values greater than a positive integer are always False.
- It is temporary.
- It can be use at
  - Select
  - Where
  - Group function.
  - Having

Sql> select rownum,empno,ename,deptno from emp;

Sql> select rownum,empno,ename,deptno from emp where deptno=10;

Sql> Select Rownum,Empno,Ename,Sal From Emp Where Rownum<=5 ;

Sql> Select Lpad(' ',Rownum,'*') From Emp;

Sql> Select RPAD('&Str',ROWNUM) From Emp Where ROWNUM<=Length('&Str')
Sql>Select ename from empWhere rownum=1;

Sql> Select RPAD('KRISHNA',ROWNUM) From Emp Where ROWNUM<=7
ORDER BY ROWNUM DESC;
Sql> select Rownum,Ename From Emp Group By Rownum,EnameHaving Mod(Rownum,2)=0;

Sql>Select Rownum,Ename From Emp Group by Rownum,Ename Having Mod(Rownum,2)!=0

Querying for Top 'N' Records

- We can ask for Nth largest or smallest values of a column.
- Never use ROWNUM and ORDER BY clause together as oracle first fetches the rows according to ROWNUM and then sorts the found rows.
- From Orcale 8i,ORDER BY clause can be used in INLINE VIEWS.

Sql> Select Ename,Sal,Rownum From emp Where Rownum<=5 Order By Sal Desc;

Sql>Select Ename,Sal,Rownum From(select Rownum,Ename,Sal From Emp Order By Sal Desc)
Where Rownum<=5;

Sql> Select *from(select * from emp order by sal) Where rownum<=5;

Sql>Select *From Emp Where ROWNUM<=15
MINUS
Select *From Emp Where ROWNUM<=10;

Sql> select rownum,empno,ename,sal from(select * from emp order by sal desc)
group by rownum,empno,ename,sal having rownum='&n';

ROWID Pseudo Column:

- **ROWID** is an exact physical address of row.
- **ROWID** is used by oracle to locate any row.
- The **ROWID** value is assign by oracle itself.
- ROWID's are unique identifies for a row in a table.
- When a Row is DELETED,ORACLE may reassign its ROWID to a new row that is inserted.
- The ROWID can never be INSERTED,UPDATED and DELETED manually.
- The ROWID pseudo column can be used in SELECT and WHERE clauses.

Sql>Select Rowid,Empno,Ename From Emp;

Sql>select max(rowid) from emp;

Sql>Select *from emp Where rowid=' AABN8cAAhAAAVpcAAD';

Sql> select max(rowid) from emp;

Sql> Select *from emp Where rowed<' AABN8cAAhAAAVpcAAD';

Sql>SELECT *FROM EMP P WHERE ROWID<(SELECT MAX(ROWID)
FROM EMP S WHERE P.ENAME=S.ENAME);

Sql> DELETE FROM EMP P WHERE ROWID<(SELECT MAX(ROWID) FROM EMP S
WHERE P.ENAME=S.ENAME);

Sql>select *from stu where rowid not in(select min(rowid) from stu group by s_id);

Sql>Select empno,ename From emp Where(rowid,1) in (select rowid,mod(rownum,2) from emp);

Sql> select rownum,ename from emp group by rownum,ename having mod(rownum,2)=0;

Sql> Select B.sal,Sum(A.Sal) From Emp A,Emp B Where A.Rowid<=B.Rowid Group By B.Rowid,B.Sal;

Sql> Select B.Ename,B.Job,B.sal,Sum(A.Sal) "Cum Sal" From Emp A,Emp B Where
A.Rowid<=B.Rowid Group By B.Rowid,B.Sal,B.Ename,B.Job;

OLAP FEATURES IN ORACLE

OLAP Features in ORACLE:

- Some features for query processing in ORACLE include the use of ONLINE ANALYTICAL PROCESSING(OLAP) upon the Data Base.
- OLAP features are useful for Data Warehousing and Data Mart Applications.
- The OLAP Operations are performance enhancements.
  - TOP-N QUERIES.
  - GROUP BY
  - CUBE
  - ROLLUP

ROLLUP Option:

- It is used with group by clause to display the summarized data.
- ROLLUP grouping produces a results set containing the regular grouped rows and the subtotal values.
- ROLLUP grouping produces subtotal and Gant Total.
- The Totaling is based on a one dimensional data hierarchy of grouped information.

Syntax:

GROUP BY ROLLUP(Column1,Column2,....)

Illustrations:

```
Sql>Select Deptno,Sum(Sal)  From Emp  Group By Rollup(Deptno);

Sql>Select Job,Sum(Sal)  From Emp Group By Rollup(Job);

Sql> Select Deptno,Avg(Sal)  From Emp  Group By Rollup(Deptno);
```

Passing Multiple Columns to ROLLUP

- When Multiple Column are passed to ROLLUP, The ROLLUP, Groups the rows into blocks with the same column values.

```
Sql>SELECT Deptno,Job,SUM(SAL)  FROM EMP   GROUP BY ROLLUP(DEPTNO,JOB);

Sql>SELECT Job,Deptno,SUM(SAL) Tolsal  FROM Emp  GROUP BY
ROLLUP(JOB,DEPTNO);

Sql>SELECT Job,Deptno,Avg(SAL) Tolsal  FROM Emp  GROUP BY
ROLLUP(JOB,DEPTNO);
```

- Null values in the output of ROLLUP operations typically mean that the row contains subtotal or Grant total information.
- Use NVL()function for proper meaning.

CUBE Operators:

- CUBE grouping produces a results set containing the rows from ROLLUP and cross-tabulation rows.
- It is extension similar to ROLLUP.
- The Result of cube is A summary that shows subtotal for every combination of columns or expressions in the Group By Clause.

```
Sql> SELECT Deptno,Job,SUM(SAL)  FROM EMP  GROUP BY CUBE(DEPTNO, JOB)

Sql>SELECT Job, Deptno,SUM(SAL)  FROM EMP GROUP BY CUBE(Job, Deptno);
```

Grouping() Function:

- The Grouping() Function accepts a column and returns 0 or 1.

- Grouping() Function returns 1 when the column value is NULL, and returns 0 when the column value is NOT NULL.
- Grouping() function is useful when we want to display a value when a NULL would otherwise be returned in OLAP Queries.
- Grouping() function is used only upon Queries that use ROLLUP or CUBE.

```
Sql> Select Grouping(Deptno),Deptno,SUM(SAL)  FROM Emp  GROUP BY
Rollup(Deptno);

Sql> Select Grouping(Job),Job,SUM(SAL) FROM Emp  GROUP BY Rollup(Job);

Sql> Select Grouping(Job),Job,Deptno,SUM(SAL) FROM Emp GROUP BY
Cube(Job,Deptno)
```

Decode Function:

- The DECODE function can be used to expand any abbreviation used in the table.
- The Function works on the Same principle as the IF-THEN-ELSE.
- We can pass a variable number of value into the call of the DECODE() function.
- The first item is may be column name or value that need to decoded.
- The function has no restriction on the INPUT and OUTPUT data type.
- The function can work for only an analysis that considers an equality operator in the logical comparision.

Syntax:

```
Sql>SELECT DECODE(<column name/value>,
                                <coded value>,<decoded val>,
                                 <coded value>, <decoded val>,..)
          From <table name>;
Sql> Select  Decode(Deptno,10,'ACCOUNTING',20,'RESEARCH',30,'SALES',
          40,'OPERATIONS','OTHER') Departments From Emp;
```

Grouping()With DECODE():

- The DECODE() function can be used to convert 1 and 0 returned through Grouping() into a meaningful output.

```
Sql> Select Decode(Grouping(Deptno),  1,'All Departments', Deptno)Departments,
          Sum(Sal) From Emp Group By ROLLUP(Deptno);

Sql>Select Decode(Grouping(Job),  1,'All Designations', Job)Designations, Sum(Sal)
          From Emp Group By ROLLUP(Job);
```

Decode() and Grouping() to converting multiple column values:

```
Sql> Select  Decode(Grouping(Deptno), 1,'All Departments', Deptno)Departments,
       Decode(Grouping(Job), 1,'All Designations', Job)Designations, Sum(Sal)
                From Emp Group By ROLLUP(Deptno,Job);
```

Grouping() With Decode() and CUBE:

```
Sql> Select  Decode(Grouping(Deptno), 1,'All Departments', Deptno)Departments,
            Decode(Grouping(Job), 1,'All Designations', Job)Designations, Sum(Sal)
                From Emp Group By Cube(Deptno,Job);
```

Appling Grouping SETS Clause:

- The GROUPING SETS clause is used to get the SUBTOTAL rows.

```
Sql> Select Deptno,Job,Sum(Sal) From Emp Group By Grouping
Sets(Deptno,Job);
```

Working With CASE Expressions:

- The CASE expression can be used to perform IF-THEN-ELSE logic in SQL.
- It can be used even for Executing conditions on range based comparison.
- Case expressions are of two types
  - SIMPLE CASE Expressions
  - SEARCHED CASE Expressions.

Simple CASE Expressions

- It accepts more than one column expression .
- These Expression are used to determine the returned value.
- They work with equality comparison only, almost all similar to DECODE.
- It has a select which associates to the compared value either from the column or constraint.
- The value in the select is used for comparison with the expressions used in the WHEN clause.

Syntax:

```
Sql>CASE Search_expr
        WHEN Expr1 THEN Result 1
        WHEN Expr2 THEN Result 2
        ELSE Default_Result
        END;
```

Example:

```
Sql>Select Empno,Ename,Job,Sal,
      Case  Job
        When 'MANAGER' THEN 'Man'
        When 'CLERK' THEN 'Clk'
              When 'SALESMAN' THEN 'Sman'
       Else 'Other Job'
       End
         Cjob  From Emp;

Sql> Select Empno,Ename,Deptno,
      Case  Deptno  When 10 THEN 'ACCOUNTS'
                    When 20  THEN 'RESEARCH'
                          When 30  THEN 'SALES'
                          When 40  THEN 'OPERATIONS'
                   Else 'Not Found'
                   End   From Emp;
```

Searched CASE Expressions:

- The statement uses conditions to determine the returned value.
- It helps in writing multiple conditions for evaluation
- Helps in range analysis of values also.

Syntax:

```
Sql>CASE
        WHEN Condition 1 THEN Result 1
        WHEN Condition 2 THEN Result 2
        WHEN Condition n THEN Result n
        ELSE Default_Result
        END;
```

Example:

```
Sql> Sclect empno,ename,job,sal,
        Case
        when job='MANAGER' then 'Mag'
         when sal=3000 then 'Mpay'
         when job='SALESMAN' then 'Sman'
        Else 'Not Specified' end
        Cjob From emp

Sql> Select Empno,Ename,Deptno,
        Case
          When Deptno=10 THEN 'ACCOUNTS'
          When Deptno=20  THEN 'RESEARCH'
                When Deptno=30  THEN 'SALES'
                When Deptno=40  THEN 'OPERATIONS'
         Else 'Not Specified '
         End  From Emp;

 Sql> Select Ename,Sal,
      Case
            When Sal>= 800 AND Sal<=2000 THEN 'Lowest Pay'
            When Sal>=2001 AND Sal<=4000 THEN 'Moderate Pay'
             Else 'High pay '
             End From Emp;
```

GROUPING_ID() Function:

- The function is used to Filter Rows using A HAVING Clause to exclude rows that do not contain a subtotal OR Grand Total.
- The function accepts one or more columns and returns the decimal equivalent of the GROUPING BIT VECTOR.
- The GROUPING BIT VECTOR is computed by combining the results of a call to the GROUPING() function for each column in order.

Computing the GROUPING BIT VECTOR:

- GROUPING() Function returns 1 when the column val returns 0,Based on this concept.
- GROUPING_ID() Returns 0,When Deptno and Job are is NOT NULL.
- GROUPING_ID() Returns 1,If Deptno is NOT NULL and Job is NULL.
- GROUPING_ID() Returns 2,If Deptno is NULL and Job is NOT NULL.
- GROUPING_ID() Returns 3,If Deptno is NULL and Job is NULL.

```
Sql> Select Deptno,Job,Grouping(deptno) GDPT,Grouping(Job) GJOB,
                GROUPING_ID(Deptno,Job) GRPID,  SUM(Sal)  From Emp
                                Group By ROLLUP(Deptno,Job);
Sql> Select Deptno,Job,Grouping(deptno) GDPT,Grouping(Job) GJOB,
                    GROUPING_ID(Deptno,Job) GRPID, SUM(Sal) From Emp
                                                        Group By
                                                CUBE(Deptno,Job);
```

GROUPING_ID() and HAVING Clause:

```
Sql> Select Deptno,Job, GROUPING_ID(Deptno,Job) GRPID, SUM(Sal) From Emp
            Group By CUBE(Deptno,Job) Having
     GROUPING_ID(Deptno,Job)>0;
```

Representing Column multiple Times in a GROUP BY Clause:

```
Sql>Select Deptno,Job,SUM(Sal) From Emp Group By
Deptno,ROLLUP(Deptno,Job);

Sql> Select Deptno,Job,SUM(Sal) From Emp Group By
Deptno,CUBE(Deptno,Job);
```

Appling GROUP_ID Function:

- The GROUP_ID Function is used to remove the duplicate rows returned by GROUP BY Clause.
- The GROUP_ID() Does not accept any parameters.
- If 'N' duplication exits for a particular Grouping,GROUP_ID() returns numbers in the range 0 To N-1.

```
Sql> Select Deptno,Job,GROUP_ID(),SUM(Sal)  From Emp  Group By
Deptno,CUBE(Deptno,Job) having group_id()=0;

Sql>Select Deptno,Job,GROUP_ID(),SUM(Sal) From Emp Group By
Deptno,CUBE(Deptno,Job) Having GROUP_ID=0

Sql> Select Deptno,Job,GROUP_ID(),SUM(Sal) From Emp Group By
Deptno,CUBE(Deptno,Job) Having GROUP_ID()=0;
```

Enhancing the Power of OLAP using Analytic Functions:

- Analytic function are designed to address problems such as
- Calculate a running total.
- Find percentages with in a Group.
- Top "N" Queries
- Compute "Moving Averages".
- Analytic functions add greater performance to the standard query processing.

How analytic Functions work?

- Analytic functions compute an aggregate value based on a group of rows.
- The group of rows are called as "WINDOW" and is defined by the Analytic clause.
- The Group of rows are called as "WINDOW" and is defined by the Analytic Clause.
- The window define the range of rows used to perform the calculation for the current row.
- Window sizes can be based upon either a physical number of rows or a logical
- Analytical function can appear only in the SELECT list OR ORDER by clause.

Syntax:

```
Analytic Function(Arg1,Arg2,Arg3)
Over(Partition Clause
        ORDER BY Clause
        Windowing Clause)
```

- Analytic function takes o or 3 arguments.
- The PARTITION by clause logical breaks a Single result set into 'N' groups.
- The Analytic function are applied for each group independently ,and they are reset for each group.
- The ORDER BY Clause specifies how data is stored within each GROUP(partition).
- The Output of Analytical function is affected by ORDER BY clause.
- The windowing clause gives us a way to define a Sliding (OR) Analytical function will operate, within a Group.

Analytic Function Categories:

Ranking Function:

- They enable us to calculate Ranks,percentiles and N-Tiles.

Inverse percentile Functions

Enable to calucate the value corresponding to a per

Window Functions:

- Enable to calculate Cumulative and moving aggregates.

Normal Ranking:

```
Sql>Select Ename,Deptno,Sal,      RANK() OVER(ORDER BY Sal)EmpRank
            From Emp Group By Deptno,Ename,Sal ORDER BY Empr  ;
```

Sql> Select Ename,Deptno,Sal, DENSE_RANK() OVER(ORDER BY Sal Desc)EmpRank From Emp Group By Deptno,Ename,Sal ORDER BY Emprank;

Ranking With Partion:

Sql> Select Ename,Deptno,Sal, RANK() OVER(PARTITION BY Deptno ORDER BY Sal Desc)"TOP Sal" From Emp ORDER BY Deptno,Sal DESC;

Ranking with Partition and Filters:

Sql>Select * From(Select Ename,Deptno,Sal,RANK() OVER(PARTITION BY Deptno ORDER BY Sal Desc) TOPSal From Emp) WHERE TOPSAL<=3 ORDER BY Deptno,Sal DESC;

Appling Windows:

- The Windowing clause give a way to define a SLIDING or ANCHORED WINDOW of data, upon which the ANALYTIC FUNCTION will operate, with A GROUP.
- The default Window is an ANCHORED WINDOW that simple starts at the FIRST ROW of a GROUP and continues to the CURRENT ROW.
- Window can be based on RANGES of data values.
- Existences of an ORDER BY in an analytic function will add a DEFAULT WINDOW clause of "RANGE UNBOUNDED PRECEDING", which gets all rows in the partition that came before the ORDER BY clause.

Row WINDOW:

Sql>Select Deptno,Ename,Sal, SUM(SAL) OVER(PARTITION BY Deptno ORDER BY Ename ROWS 1 PRECEDING) "Sliding Total" From Emp ORDER BY Deptno,Ename;

Sql>Select Deptno,Ename,Sal, SUM(SAL) OVER(PARTITION BY Deptno ORDER BY Ename ROWS 2 PRECEDING) "Sliding Total" From Emp ORDER BY Deptno,Ename;

Range Window:

- RANGE WINDWO collect ROWS together based on a WHERE clause.
- The RANGE UNITS can either be numeric comparisons OR date comparisons.
- Range units are not valid if data type is other than number or dates.

Sql> Select Ename, Sal,Count(*) OVER(ORDER BY Sal ASC RANGE 5000 PRECEDING) Salcnt From Emp;

Sql> Select Ename, Sal,sum(Sal) OVER(ORDER BY Sal ASC RANGE 5000 PRECEDING) SalTol From Emp;

NULLS FIRST NULLS LAST

Sql>Select Empno,Sum(Sal),RANK() OVER(ORDER BY SUM(Sal) DESC NULLS LAST)Rank,DENSE_RANK() OVER(ORDER BY SUM(Sal) DESC NULLS LAST)Dense_Rank From Emp Group By Empno Order by Empno

Sql>Select Empno,comm,RANK() OVER(ORDER BY comm DESC NULLS LAST)Rank, DENSE_RANK() OVER(ORDER BY comm DESC NULLS LAST)Dense_Rank From Emp Group By Empno,comm.

Sql>Select Empno,comm,RANK() OVER(ORDER BY comm DESC NULLS FIRST)Rank, DENSE_RANK() OVER(ORDER BY comm DESC NULLS FIRST)Dense_Rank From Emp Group By Empno,comm.;

The default is NULLS FIRST.

LOCK TABLE STATEMENT

Locks:

- Locks are the mechanisms used to prevent destructive interaction between users accessing the same resources simultaneously.
- A resource can either table or a specific row in a table.
- Thus locks provide a high degree of data concurrency.

Concurrent user:-

- Multiple users using same resource at same time.
- Locking system of oracle prevent concurrent users to do destructive transactions.
- Locks can be acquired at two different levels.
  - Row level lock (for specific rows)→
  - Table level lock (for entire table)

Row level locks:

- In the row lock, a row is locked exclusively so that others users cannot modify the row until the transaction holding the lock is committed or rollback.

Example:

```
Sql>select *from emp  where empno=7788 for update of hiredate;
Sql>Update emp set  hiredate='12-Jan-06' Where empno=7788;
```

Table Level lock:

- A table level lock will protect table data thereby guaranteeing data integrity when data is being accessed concurrently by multiple users.
- A table lock can be in several modes.
- Share lock
- Share update lock
- Exclusive lock

Syntax:

- Lock table <table_name> in <share or share update or exclusive mode>;

Share lock:

- A share lock locks the table allowing other users to only query but not insert, update or delete rows in a table.

Example:

```
Sql>lock table emp in share mode;
```

Share update lock:

- It locks rows that are to be updated in a table.
- It permits other users to concurrently query,insert, update or even lock other rows in the same table.
- It prevents the other users from updating the row that has been locked.

Example:

```
Sql>Lock table emp in share update mode;(scott)
```

Scott:

```
Sql>Update emp set sal=3000 Where deptno=10;
        (As scott1 cannot update 10 but can be update 20,30..)
```

Exclusive Lock:-

- When issued by one user, it allows the other user to only Query but not insert, delete or update rows in a table.→
- It is almost similar to a share lock but only one user can place an exclusive lock on a table at a time, where as many users can place a share lock on the same table at the same time.

Example:

```
Sql>lock table emp in exclusive mode;
```

Note:

- Locks can be released by issuing either2 rollback or commit;

Dead lock:

- A deadlock occurs when two users have a lock, each On a separate object , and, they want to acquire a lock on the each object. When this happens, the first user has to wait for the second user to release the lock, but the second user will not release it until the lock on the first user's object is freed .
- At this point ,both the users are at an impasse and cannot proceed with their business. In such a case, Oracle detects the deadlock automatically and solves the problem by aborting one of the two transactions.

Roles In Oracle:

- A role is a named group of related privileges that can be granted to the user.

Advantages:

Rather than assigning Privileges one at a time directly to s USER, we can CREATE a ROLE, assign PRIVILEGES to that ROLE, and then GRANT that ROLE

Working With SYNONYMS:

- It is Data Base Object , which acts as an alternate name for an existing object, Next to View.
- The CREATE Synonym privileges is necessary to execute the creation of a Synonym.
- Simplify access to objects by creating a synonym(another name for an object). With synonyms, you can:
  - Ease referring to a table owned by another user.
  - Shorten lengthy object names.
- DML, Description, Select allowed on SYNONYM provides security.
- USER_SYNONYMS tables holds details of synonyms.

Synonyms are two types:

Private synonym:

- Created by user.
- Used by specific users which have permission.

Syntax:

```
CREATE [PUBLIC] SYNONYM <syn_name>  FOR
                [user name].
                <DBobject name>[@database link];
```

Example:

```
Sql>CREATE SYNONYM EmpDet FOR Emp;

Sql>CREATE SYNONYM emp_syn FOR Scott.Emp;
      (if you are in different user.)

Sql> Insert into EmpDet(Empno,Ename,Job,Deptno)
                              Values(1001,'KRISHNA','MANAGER',20);
Sql> Select *from EmpDet;

Sql> Delete From EmpDet Where Deptno=10;

Sql>Grant all on EmpDet to scott1;

Sql>Select *From Scott.EmpDet;

Sql> Select Synonym_Name From User_Synonyms Where Table_Name='EMP';
```

Public synonym:

- Created by Data Base Administrators.
- We should have CREATE PUBLIC SYNONYM privilege, and it can accessed by all USERS.

Syntax:
Example:

```
Sql>Create PUBLIC SYNONYM EmpDet FOR SCOTT.Emp;

Sql>Select Synonym_Name  From All_Synonyms Where Table_Name='TEMP';
```

- It is created for
  **Table **View **Procedure **Function **Package
- It is used for hiding the information as
  - User Name
  - Object Name
  - Database Link For security purpose.
- It can be also use to provide short name to object.
- It takes minimum space only for name.

Remove:

- **If drop the table the SYNONYM of that table become as Invalid.**

```
Sql>Drop SYNONYM EmpDet;
```

OCP:

1)The database administrator of your company created a public synonym called HR for the HUMAN_RESOURCES table of the GENERAL schema, because many users frequently use this table.
As a user of the database, you created a table called HR in your schema. What happens when you execute
this query?

```
SELECT *
FROM HR;
```

A. You obtain the results retrieved from the public synonym HR created by the database administrator.
B. You obtain the results retrieved from the HR table that belongs to your schema.
C. You get an error message because you cannot retrieve from a table that has the same name as a public synonym.
D. You obtain the results retrieved from both the public synonym HR and the HR table that belongs to your schema, as a Cartesian product.
E. You obtain the results retrieved from both the public synonym HR and the HR table that belongs to your schema, as a FULL JOIN.

2)Mary has a view called EMP_DEPT_LOC_VU that was created based on the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables. She granted SELECT privilege to Scott on this view. Which option enables Scott to eliminate the need to qualify the view with the name MARY .EMP_DEP_LOC_VU each time the view is referenced?

A. Scott can create a synonym for the EMP_DEPT_LOC_VU bus using the command:

```
CREATE PRIVATE SYNONYM EDL_VU
FOR mary.EMP DEPT_LOC_VU;
```

then he can prefix the columns with this synonymn.
B. Scott can create a synonym for the EMP_DEPT_LOC_VU by using the command:

CREATE SYNONYM EDL_VU
FOR mary.EMP_DEPT_LOC_VU;
then he can prefix the columns with this synonym.
C. Scott can create a synonym for the EMP_DEPT_LOC_VU by using the command:
CREATE LOCAL SYNONYM EDL_VU
FOR mary.EMP DEPT_LOC_VU;
then he can prefix the columns with this synonym.
D. Scott can create a synonym for the EMP_DEPT_LOC_VU by using the command:
CREATE SYNONYM EDL_VU
ON mary(EMP_DEPT_LOC_VU);
then he can prefix the columns with this synonym.
E. Scott cannot create a synonym because synonyms can be created only for tables.
F. Scott cannot create any synonym for Mary's view. Mary should create a private synonym for the view and grant SELECT privilege on that synonym to Scott.

3) Evaluate the SQL statement
DROP TABLE DEPT:
Which four statements are true of the SQL statement? (Choose four)
A. You cannot roll back this statement.
B. All pending transactions are committed.
C. All views based on the DEPT table are deleted.
D. All indexes based on the DEPT table are dropped.
E. All data in the table is deleted, and the table structure is also deleted.
F. All data in the table is deleted, but the structure of the table is retained.
G. All synonyms based on the DEPT table are deleted.

Data Manipulation Language

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a transaction.

Update

- Update is used to modify the existing vales in Table or in the base table of View.

Prerequisites

- The table must be in the Own schema.
- UPDATE object privilege should be available.

Syntax

```
Sql>UPDATE table
      SET column = value [, column = value, ...][WHERE  condition];
Sql> UPDATE Emp SET Deptno = 40 WHERE  Empno = 7788;
Sql> UPDATE Emp SET Sal=4000,Comm =NULL WHERE  Job = 'SALESMAN';
```

Updating Two Columns with a Sub query:

```
Sql> UPDATE  emp SET job= (SELECT  job  FROM   emp  WHERE  empno=7788),
      sal= (SELECT  sal  FROM   emp WHERE  empno=7499);
                          WHERE   employee_id   =  7369;
Sql> Update Emp Set (Job,Deptno)=(Select Job,Deptno  From Dept Where
Empno=7499) Where Empno=7698;
Sql> Update Emp Set Deptno=(Select Deptno From Emp Where Empno=7788)
                  Where Job=(Select job From Emp Where Empno=7788);
```

Updating Rows Based on Another Table:

- Use Sub queries in UPDATE statements to update rows in a table based on values from another table.

```
Sql>Update Emp Set Deptno=(Select Deptno  From Dept Where
Dname='ACCOUNTING') Where Deptno=( Select Deptno From Dept Where
Dname='SALES');
```

Appling default Values:

- It is used to UPADETE a value in a column with DEFAULT value set in the constraints.

```
Sql>Update Emp  Set Sal=DEFAULT  Where Ename='FORD';
Sql> Update Emp  Set Hiredate=DEFAULT  Where Empno=7788;
```

Returning Clause:

- The returning clause is introduced from 8.0.
- It is used to return a valve from a aggregate function.
- The clause can be specified for table and materialized views and for views with a single base table.

Restrictions:

- Each express must be a simple expression or a single set aggregate function.
- It cannot be specified for multiple table insert.
- It cannot be specified upon a view upon which a instead of trigger is defined.

```
Sql>VAR sumsal number
Sql>Update Emp Set Sal=Sal*1.2 Where Deptno=10  Returning SUM(Sal) INTO
:sumsal;

Sql> VAR bsal Number

Sql> VAR bname Varchar2(20)

Sql> update emp set sal=sal+sal*.25  where empno=7788 returning ename,sal into
:bname,:bsal;

Sql> print :bname :bsal
```

MERGE Statement:

- The MERGE statement is used to SELECT rows from one or more Sources for UPDATE or INSERT into one or more tables.
- The MERGE statement is convenient to combine multiple operations and avoid multiple INSERT,UPDATE,DELETE.
- MERGE is a deterministic statement, using which the same row of the target table can be transacted multiple times in the same MERGE statement.

Illustrations:

```
Sql>Create Tabie Ebonus (  Empno NUMBER, Bonus NUMBER DEFAULT 100  );
Sql> Insert into Ebonus (Empno) (Select E.Empno  From Emp E  Where
Job='SALESMAN');
Sql> MERGE into Ebonus B
                Using (Select Empno,Sal,Deptno From Emp
                Where Deptno = 30) S
                On (B.Empno = S.Empno)
                When MATCHED THEN
                    UPDATE Set B. Bonus = B.Bonus + S. Sal *0.1
                    DELETE Where (S. Sal > 4000) WHEN NOT MATCHED THEN
                    INSERT (B. Empno , B .Bonus) VALUS ( S . Empno, S. Sal *
0.1)
                   Where ( S. Sal < = 4000);
Sql>MERGE INTO TEST T
          USING EMP E
          ON(E.EMPNO=T.EMPNO)
          WHEN Matched THEN
                UPDATE SET T.JOB=E.JOB, T.SAL=E.SAL, T.DEPTNO=E.DEPTNO,
                T.COMM=E.COMM
          WHEN NOT Matched  THEN
                INSERT VALUES(E.EMPNO,E.ENAME,E.JOB,
               E.MGR,E.HIREDATE,E.SAL,E.COMM,E.DEPTNO)
Sql > Create Table Examtimetable
                (Examname varechar2(30),
                Examtime varechar2(12),
                CONSTRAINT ExamnamePK PRIMARY KEY(Examname);
          Sql> Insert Into Examtimetable VALUES ('PHYSICAL SCINCE','9:00 AM');

    Sql> MERGE Into Examtimetable E1 USING Examtimetable E2
          ON  ( E2. Examname  = E1. Examname AND  E1. Examname = 'PHYSICAL
SCIENCES')
          WHEN MATCHED THEN UPDATE Set E1.Examtime = '10:30 AM'
          WHEN NOT MATCHED THEN
                Insert (E1. Examname , E2. Examntime)
                      VALUES (' PHYSICAL SCIENCES', '10:30 AM');

    Sql> Merge into Examtime  table E2 Using Examtimetable E2
          ON (E2. Examname = E1.Examname AND E1.Examname = 'CHEMICAL
          SCIENCES') WHEN MATCHED THEN
                      UPDATE SET E1.Examtime = '12:30 PM'
               WHEN NOT MATCHED THEN
                Insert (E1. Examname , E2. Examntime)
                          VALUES (' CHEMICAL SCIENCES', '12:30 PM');
```

DELETE Statement:

- It used to remove existing rows from a table.
- An Un partitioned or partitioned table.
- The Un partitioned or partitioned base table of view.
- The Un partitioned or partitioned constraint table of writable Materialized view.

Prerequisites:

- To DELETE rows from a table, the table must be in the USERS SCHEMA.
- DELETE ANY TABLE system privilege allows to DELETE rows from any TABLE OR PARTITION TABLE OR form the base table of any VIEW.

Syntax:

```
Sql>DELETE [FROM] <table Name>
    [WHERE condition];
```

Using Returning Clause

```
Sql>DELETE From Emp Where Empno in(7788,7902);

Sql>Delete From Emp Where Deptno=(Select Deptno  From Dept Where
Dname='ACCOUNTING');

Sql> DELETE FROM EMP P  WHERE ROWID<(SELECT MAX(ROWID)
                    FROM EMP S WHERE P.ENAME=S.ENAME);
Sql>VAR BVSAL number
Sql>Delete From Emp Where Job='SALESMAN' AND Hiredate<Sysdate
                                        Returning SUM(Sal) INTO :BVSAL;
Sql>print :BVSAL
```

Multiple Inserts:

- Supports to insert into more than one table at a time.
- Create 3 tables d1,d2 and d3 same as dept table structure

```
Sql>insert all
          into d1 values(deptno,dname,loc)
          into d2   values(deptno,dname,loc)
           into d3 values(deptno,dname,loc)
           select * from dept;
Sql>Insert ALL
      Into d1(deptno) Values(70)
      Into d2(deptno) Values(80)
      Select deptno  From Dept Where Deptno=40
```

Conditional Insert:

```
Sql>insert all
    when deptno<=40 then Into d1 values(deptno,dname,loc)
    when deptno >40 and deptno<=80 then into d2 values(deptno,dname,loc)
    else into d3(deptno,loc) values(deptno,loc)
        select *from dept
Sql>Select *    From d1;
Sql>Select *   From d2;
Sql>Select *    From d3;
```

OCP

1)Examine the data in the EMPLOYEES and EMP_HIST tables:

EMPLOYEES

EMPLOYEE_ID	NAME	DEPT_ID	MGR_ID	JOB_ID	SALARY
101	Smith	20	120	SA_REP	4000
102	Martin	10	105	CLERK	2500
103	Chris	20	120	IT_ADMIN	4200
104	John	30	108	HR_CLERK	2500
105	Diana	30	108	IT_ADMIN	5000
106	Smith	40	110	AD_ASST	3000
108	Jennifer	30	110	HR_DIR	6500
110	Bob	40		EX_DIR	8000
120	Ravi	20	110	SA_DIR	6500

EMP HIST

EMPLOYEE_ID	NAME	JOB_ID	SALARY
101	Smith	SA_CLERK	2000
103	Chris	IT_CLERK	2200
104	John	HR_CLERK	2000
106	Smith	AD_ASST	3000
108	Jennifer	HR_MGR	4500

The EMP_HIST table is updated at the end of every year. The employee ID, name, job ID, and salary of each existing employee are modified with the latest data. New employee details are added to the table.
Which statement accomplishes this task?

A. UPDATE emp_hist SET employee_id, name, job_id, salary = (SELECT employee_id, name, job_id, salary FROM employees) WHERE employee_id IN (SELECT employee_id FROM employees);

B. MERGE INTO emp_hist eh USING employees e ON (eh.employee_id = e.employee_id)WHEN MATCHED THEN UPDATE SET eh.name = e.name, eh.job_id = e.job_id, eh.salary = e.salary WHEN NOT MATCHED THEN INSERT VALUES (e.employee id, e.name, e.job id, e.salary);

C. MERGE INTO emp_hist eh USING employees e ON(eh.employee_id = e.employee_id) WHEN MATCHED THEN UPDATE emp hist SET eh.name = e.name, eh.job_id = e.job_id, eh.salary = e.salary WHEN NOT MATCHED THEN INSERT INTO emp_hist VALUES (e.employees_id, e.name, e.job_id, e.salary);

D. MERGE INTO emp_hist eh USING employees e WHEN MATCHED THEN UPDATE emp_hist SET eh.name = e.name, eh.job_id = e.job_id, eh.salary = e.salary WHEN NOT MATCHED THEN INSERT INTO emp_hist VALUES (e.employees_id, e.name, e.job_id, e.salary);

2) Examine the structure if the EMPLOYEES and NEW EMPLOYEES tables:

EMPLOYEES

EMPLOYEE_ID NUMBER Primary Key
FIRST_NAME VARCHAR2(25)
LAST_NAME VARCHAR2(25)
HIRE_DATE DATE
NEW EMPLOYEES
EMPLOYEE_ID NUMBER Primary Key

A. SELECT AVERAGE(gpa) FROM student_grades WHERE semester _ end > '01-JAN-2000' and semester end <31-DEC-2000';
B. SELECT COUNT(gpa) FROM student grades WHERE semester _ end > '01-JAN-2000' and semester end <31-DEC-2000';
C. SELECT MIN(gpa) FROM student grades WHERE semester _ end > '01-JAN-2000' and semester end <31-DEC-2000';
D. SELECT AVG(gpa) FROM student_grades WHERE semester _ end BETWEEN '01-JAN-2000' and '31-DEC-2000';
E. SELECT SUM(gpa) FROM student grades WHERE semester _ end > '01-JAN-2000' and semester end <31-DEC-2000';
F. SELECT MEDIAN(gpa) FROM student_grades WHERE semester _ end > '01-JAN-2000' and Semester end <31-DEC-2000';

3)You added a PHONE_NUMBER column of NUMBER data type to an existing EMPLOYEES table.

The EMPLOYEES table already contains records of 100 employees. Now, you want to enter the phone

numbers of each of the 100 employees into the table.
Some of the employees may not have a phone number available.
Which data manipulation operation do you perform?

A. MERGE B. INSERT C. UPDATE D. ADD E. ENTER

F. You cannot enter the phone numbers for the existing employee records.

4) Which are DML statements? (Choose all that apply)

A. COMMIT... B. MERGE... C. UPDATE... D. DELETE...
E. CREATE... F. DROP...

5) Examine the structure of the EMPLOYEES and NEW_EMPLOYEES tables:
EMPLOYEES
EMPLOYEE_ID NUMBER Primary Key
FIRST_NAME VARCHAR2(25)
LAST_NAME VARCHAR2(25)
HIRE_DATE DATE
NEW_EMPLOYEES
EMPLOYEE_ID NUMBER Primary Key
NAME VARCHAR2(60)
Which MERGE statement is valid?

A. MERGE INTO new_employees c USING employees e ON (c.employee_id = e.employee_id) WHEN MATCHED THEN UPDATE SET c.name = e.first_name ||','|| e.last_name WHEN
NOT MATCHED THEN INSERT VALUES (e.employees_id, e.first_name ||','|| e.last_name);

B. MERGE new_employees c USING employees e ON (c.employee_id = e.employee_id) WHEN EXISTS THEN UPDATE SET c.name = e.first_name ||','|| e.last_name WHEN NOT MATCHED THEN INSERT VALUES (e.employee_id, e.first_name ||','|| e.last_name);

C. MERGE INTO new_employees c USING employees e ON (c.employee_id = e.employee_id)
WHEN EXISTS THEN UPDATE SET c.name = e.first_name ||','|| e.last_name WHEN NOT
MATCHED THEN INSERT VALUES (e.employees_id, e.first_name ||','|| e.last_name);

D. MERGE new_employees c FROM employees e ON (c.employee_id = e.employee_id) WHEN MATCHED THEN UPDATE SET c.name = e.first_name ||','|| e.last_name WHEN NOT MATCHED THEN INSERT INTO new _ employees VALUES (e.employees_id, e.first_name ||','|| e.last_name);

Transaction Control Language

Transaction Control

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to a database only if they are committed.

Transaction start and end cases

- A transaction begins when the first executable SQL statement is encountered.
- The Transaction Terminates when the following specifications occur.
  - A COMMIT OR ROLLBACK is issued.
  - A DDL statement issued.
  - A DCL statement issued.
  - The user Exits the SQL *Plus
  - Failure of Machine or System Crashes.

Commit It used to save changes made by DML statements.

Types of Commits:

Implicit Commit

- A DDL statement or a DCL statement is automatically committed by implicitly commit and hence ends a transaction.

Explicit Commit

- It is given by user, valid for only DML operations.

ROLL BACK:-

- It is used to cancel transactions which are not saved.

SAVE POINT:-

- IT is used to mark with current transaction .
- Savepoints are used to identify a point in the Transaction to which you can rollback rather than cancel the complete transaction.

State of data before Commit or Rollback

- Every data change made during the transaction is temporary until the transaction is committed .
- Data manipulation operations primarily do not affect the state of the data, hence the can be recovered.
- The current user can review the results of the Data Manipulation operation by Querying the tables.
- Other users can not view the results of the Data Manipulation made by the current user.

State of the Data after commit is issued

- Data changes are written to the database permanently.
- The previous state of the data in the Database is Permanently Lost.
- All users can view the Results of the recent Transactional change.
- The Locks on the affected rows are automatically released.
- All SAVEPOINT are erased.

```
Sal>COMMIT;
```

State of the Data after Rollback

- Rollback statement is used to Discard all pending changes.
- The Data changes are undone.
- The previous state of the data is returned OR Restored.
- The LOCKS on the affected rows are released automatically

```
Sql>ROLLBACK;
```

Rolling Back Changes to a SAVEPOINT

- SAVEPOINT is used to create a Marked in the current Transaction.
- Using SAVEPOINT the transaction can be discarded up to the marked by using the ROLLBACK statement.

```
Sql>ROLLBACK TO <SAVEPOINTNAME>;
```

- If a second SAVEPOINT is created with the same name as an earlier SAVEPOINT, The Earlier SAVEPOINT is deleted.

OCP

1) CREATE TABLE dept (deptno NUMBER(2), dname VARCNAR2(14), 1oc VARCNAR2 (13));
ROLLBACK;
DESCRIBE DEPT
What is true about the set?

A. The DESCRIBE DEPT statement displays the structure of the DEPT table.
B. The ROLLBACK statement frees the storage space occupies by the DEPT table.
C. The DESCRIBE DEPT statement returns an error ORA-04043: object DEPT does not exist.
D. The DESCRIBE DEPT statement displays the structure of the DEPT table only if there is a COMMIT
statement introduced before the ROLLBACK statement.

2) Which two statements complete a transaction? (Choose two)

A. DELETE employees; B. DESCRIBE employees;
C. ROLLBACK TO SAVE POINT C; D. GRANT SELECT ON employees TO SCOTH
E. ALTER TABLE employees SET UNUSED COLUMN sal;
F. Select MAX(sal) FROM employees WHERE department _ id 20;

3) Evaluate the SQL statement

DROP TABLE DEPT:
Which four statements are true of the SQL statement? (Choose four)

A. You cannot roll back this statement.
B. All pending transactions are committed.
C. All views based on the DEPT table are deleted.
D. All indexes based on the DEPT table are dropped.
E. All data in the table is deleted, and the table structure is also deleted.
F. All data in the table is deleted, but the structure of the table is retained.
G. All synonyms based on the DEPT table are deleted.

Data Definition Language

Altering The Table Definition

1. ALTER to modify the structure of the table(add new columns, remove existing columns).

Alterting for adding Column

2. ADD It is used to add new column(s).

Syntax for Adding Column

Sql>Alter Table <Table name>
ADD
(Column Name Datatype[DEFAULT Exp],
Column Name Datatype].....);

Sql>Alter table faculty ADD phoneno number(10);

Sql>Alter table faculty ADD(City varchar2(10), Bonus number(6,2));

Guidelines For ADDING Column

4 We cannot specify the location where the column can appear, It by default become the Last Column.

5 If the table contains records, before the column is added, the new column contains NULL values.

Modify

6 It is used to modify structure of existing column(s).

Syntax for Modifying Column

Sql>Alter Table <Table name>
MODIFY (Column Name Datatype[DEFAULT Exp],
Column Name Datatype].....);

Sql>Alter Table Faculty MODIF City Varchar2(20);

Guidelines For Modify a Column

7 We can increase the width or precision of a numeric column.

8 We can decrease the width of a column if the column contains only NULL values and if the table has no rows.

9 We can change the data type if the column contains NULL's.

10 We can converts a CHAR column to the VARCHAR2 data type or convert a VARCHAR2 Column to the CHAR data type if the column contains NULL values or if the size is not changed.

Drop

1 It is used to remove column(s).

Guidelines to Drop a Column

2 The column may or may not contain data.

3 We can drop more than one column at a time.

4 The table must have at least one column remaining in it after it is altered.

5 Once a column is drooped it cannot be recovered.

Sql> Alter Table <Table name> DROP Column <ColumnName>;

Sql>Alter Table Temp Drop column Empno;

For more than column

Syntax

Sql> Alter Table <Table name> DROP (ColumnName1,ColumnName2,...);

Sql> Alter Table Temp Drop(JOB,MGR);

Rename columns

1 A column can renamed in table by using ATER RENAME .

2 The Feature is enabled from oracle 9.2 onwards.

3 We can rename only one column at a tiem.

Syntax

Sql>Alter Table <Table Name>
RENAME <Old column name > TO <new column >

Sql>Alter Table Emp Rename Column Empno TO Employ_no;

The SET UNUSED OPTION

1 The SET UNUSED OPTION marks one or more columns as unused such that they can be dropped when the demand on System Resources is less(8i).

2 The response time is faster than the DROP clause.
3 After a column has been marked unused, we cannot have access to that column.
4 The names and types of column marked unused will not be displayed during a describe.
5 We can ADD to a TABLE a new column with the same name as an unused column.

Sql>**Alter Table Temp**
Set UNUSED(Job);

DROP UNUSED Column Option

1 This option removes from the table all columns currently marked as unused.
2 The option is used when we want to reclaim the extra disk space from unused columns in the table.
3 If the table does not Contain UNUSED columns the statement returns with no error.

Sql> **Alter Table Temp Drop UNUSED Columns;**

Dropping A Table

1 It removes the definition of the oracle table.
2 The command not only drops the table and along with the associated INDEXES also.

Syntax

Sql>DROP TBALE<Table name>
[CASCADE CONSTRAINTS];

Sql>Drop Table Temp CASCADE CONSTRAINTS;

Guidelines to Drop Table

1 The data is totally deleted from the table.
2 Any VIEWS and SYNONYAMS will remain but are kept in invalid state.
3 Any pending transactions are committed.
4 Only the create or owner of the table or a USER with DROP any table privilege can remove a table from Database.
1 The DROP Table statement once executed is irreversible.

Changing the name of an Object

2 The RENAME command can be used to change the name of a
3 TABLE
4 VIEW
5 SEQUENCE
6 SYNONYM

To RENAME the OBJECT we must be the OWNER of the OBJECT.

Syntax

Sql>RENAME <Old Name> TO <NewName>;

Example

Sql>RENAME Emp TO Employee;

Truncating A Table

- It is used to remove all rows from a table and to release the STORAGE SPACE used by the specific TABLE.
- The TRUNACTE TABLE will not facilitate for ROLLBACK.

Syntax

Sql>TRUNCATE Table <Table Name>;

Example:

Sql> TRUNCATE Table Employee;

- To TRUNCATE a Table we must be the OWNER of the Table.

Applying Comments Upon a Table

- The COMMENTS command is used to ADD comments to a TABLE or a COLUMN or VIEW etc.
- Each comments can be up to 2000 bytes.
- The data dictionary in which comments are stored are...
  *ALL_COL_COMMENTS *ALL_TAB_COMMENTS
  *USER_COL_COMMENTS *USER_TAB_COMMENTS

Syntax

Sql>COMMENT ON table<TableName>/Column <Tablename.Column> is 'Text';

Example:

Sql>COMMENT ON Table Emp
IS 'The table storing employee Information';

Sql>COMMENT ON Column Emp.MGR
IS 'The Column is actually storing the Registered employee number as Manager Numbers, with a Self relation';

Dropping a Comment

- A comment is dropped from the database by setting it to an empty string.

Sql>COMMENT ON table Emp IS '';

Advanced Table Creation Strategies

Creating a table from an Existing Table

ON THE FLY Tables

- Oracle allows the creation of a new table On-The_Fly, Depending on a SELECT statement on an already existing table.

Syntax

Sql>Create Table <Table Name>AS
Select Columns From TableName [Where Condition];

- The Create Table...As SELECT.... Command will not work if one of the selected columns use LONG data type.
- When the New table is described it reveals that it has "INHERITED" the column definition from the Existing table.
- Using this style we can include all columns using asterisk or a subset of columns from table.
- The New table can contain "INVENTED COLUMNS" which are the product of function of the combination of other columns.
- The column definition will adjust to the size necessary to contain the data in the INVENTED COLUMNS.

Creating an Exact Copy

Sql>Create Table Temp AS Select *From Emp;

Create an Exact Copy with different and required column names

Sql>Create Table Temp1 (Ecode, Name, Basic, Deptno) AS Select empno,ename,sal,deptno From Emp;

Create a Copy with Invented Columns

Sql>Create Table Temp2 (Deptid, Deptname, Deptbudget)
AS Select D.deptno,D.dname,SUM(E.Sal) From Emp E,Dept D Where E.deptno=D.deptno Group by D.deptno,D.dname;

Creating a copy without Data

Sql>Create Table Temp3
As Select Empno "Emp_Num",Ename "Emp_Name",Sal "Basic",Deptno "Dept_Num" From Emp Having 'KRISHNA'='ORACLE';

Creating a table without Generating REDO LOG Entries

- REDO LOG entries are chronological records of database actions used during data bases recoveries.
- The REDO LOG entries generation can be avoided by using the NOLOGGING keyword.
- By circumventing with NOLOGGING key word the performance of the CREATE TABLE command will improve as less work is being done.
- As the new table creation is not being written to the REDO LOG files, the table will not be able to Re-Create, following a Database failure.
- The REDO LOG files are used to recover the Database

Sql>Create Table Tdept NOLOGGING AS Select * From
Sql>Select LOGGING From ALL_TABLES Where Table_Name
Sql>select *from v$logfile

Creating Index Organized Table:

- An Index Organized table keeps its data stored according to the PRIMARY KEY column values in the table.
- An Index Organized table stores its data as if the entire table was stored in an INDEX.
- To create an Index organized table the ORGANIZATION INDEX clause of the CREATE TABLE is used.
- To create a table as an Index Organized table we must create a PRIMARY KEY constraint on it.

Example

```
Sql>Create Table DemoTab ( Stuid Number(4), Name Varchar2(20), Doj Date,
      Constraint sid_pk PRIMARY KEY(Stuid,Name) )ORGANIZATION INDEX;
```

- To Minimize the amount of active management of the Index Organized table, we should create it only if the table's data is very static.
- An Index Organized table is most effective when the Primary key constitutes a large part of the tables columns.

Working with Partitioned Tables

- Dividing the rows of a single table into multiple parts is called Partitioning of a table.
- The table that is Partitioned is called "PARTITIONED TABLE" and the parts are called PARTITIONS.
- The Partitioned is useful for very large Table only.

Important Goals behind Partitioning

- The performance of Query against the tables can improve performance.
- The Management of the table becomes easier.
- As the Partitioned table data is stored in multiple parts, It is Easier to Load and Delete data in Partitions than in the Large Table.
- The backup and recovery operation can be performed better.

Normal un Partitioned table

```
Sql>Create Table TempTable ( Sampleid number(4) Constraint Sampidpk  Primary
key,Sampname Varchar2(20), Sampdate date, Sampdesc Long );
```

Creating Range Partition Table

```
Sql> Create Table TempTablePart ( Empid number(4) Primary key, Empname
Varchar2(20),
      Doj date, Remarks  Varchar2(4000) )
     PARTITION BY RANGE(Empid)
     (    PARTITION Empidpart1
       Values LESS THAN(500),
       PARTITION Empidpart2
       Values LESS THAN(1000),
       PARTITION Empidpart3
       Values LESS THAN(MAXVALUE)  );
```

- The Maximum value need not be specified for the last partition, the MAXVALUE keyword is specified.
- The MAXVALUE specified oracle to use the partition to store any data that could not be stored in the earlier partitions.
- We can create multiple partitions each with its own upper value defined.
- The minimum value for the range is implicitly determined by oracle from the definition of the preceding partition.

Hash Partitions upon a table

- A hash partitions upon a table determines the physical placement of data.
- The Physical placement of data is determined by performed a Hash function on the values of the partition key.
- To create a Hash Partition we use the PARTITION BY HASH Clause.

```
Sql>Create table emptablehash (Empno Number(6) constraint empnopk
Primary Key, Ename Varchar2(20), Job Varchar2(30), Deptno Number(2),
```

CHECK(Deptno IN(10,20,30,40,50,60,70,80,90)), Sal Number(8,2),Constraint Deptnofk_Hash Foreign key (Deptno) References Dept(Deptno)) PARTITION BY HASH(DEPTNO) PARTITIONS 9;

Working with List Partitioning

- In LIST PARTITIONNING we specify oracle all the possible values and designate the partition into which the corresponding rows should be instead.

```
Sql>Create Table Emplistpart (
        EMPNO NUMBER(4) Constraint pk_eno PRIMARY KEY,
        RCHAR2(10), JOB     VARCHAR2(9), MGR    NUMBER(4),
        HIREDATE  DATE, SAL NUMBER(7,2), COMM NUMBER(7,2),
        DEPTNO NUMBER(2), Constraint dnofk  Foreign Key(Deptno) REFERENCES
                Dept(Deptno) )PARTITION BY LIST(Job)
        ( PARTITION jpart1  Values('PRESIDENT','ANALYST'),
          PARTITION jpart2  Values('MANAGER','SALESMAN','CLERK') );
```

Generating Sub Partitions

- Sub partitions are partitions of partitions.
- Sub partition can be used to combine the two types of partitions...
- RANGE Partitions
- HASH Partitions
- In very large tables, the composite partition star strategy is an effective way of separating the data info manageable and tunable divisions.

```
Sql> Create Table Empsubpart  (EMPNO NUMBER(4)Constraint eno_pk PRIMARY
KEY,
        ENAME VARCHAR2(10),JOB     VARCHAR2(9),MGR    NUMBER(4),
        HIREDATE  DATE,SAL NUMBER(7,2),COMM NUMBER(7,2),
        DEPTNO NUMBER(2),Constraint fkdno Foreign Key(Deptno) REFERENCES
                Dept(Deptno) )
        PARTITION BY RANGE(ENAME)
        SUBPARTITION BY HASH(JOB)
        SUBPARTITIONS 5( PARTITION Namep1
                VALUES LESS THAN ('M'),PARTITION Namep2
                 VALUES LESS THAN (MAXVALUE) );
```

Splitting Table Partitions

```
Sql> Alter Table TempTablePart
        SPLIT PARTITION Empidpart3 AT(2000)
        INTO( PARTITION Empidpart3, PARTITION Empidpart4 );
Sql> Alter Table Emplistpart SPLIT PARTITION  jpart2 Values('SALESMAN')
        INTO (PARTITION JSalesMan, PARTITION SSalesMan);
```

Merging Table Partitions

```
Sql>Alter Table Emplistpart MERGE PARTITIONS  Jpart1, JSalesman INTO PARTITION
Jpart1;
Sql> Alter Table Emplistpart MERGE PARTITIONS Jpart1, JSalesman INTO PARTITION
Jpart1;
```

Dropping a table PARTITION:

```
Sql>Alter Table Emplistpart  Drop PARTITION Jpart1;
Sql>Alter Table Emplistpart  Drop PARTITION Jpart1;
Sql> Alter Table TempTablePart  Drop PARTITION Empidpart2
```

Creating Indexes upon Partitions

- Once a Partitioned table is created, we have to create an Index upon that table.
- The Index may be Partitioned according to the same range of values or same list used to Partition the Table.

```
Sql>Create INDEX Emptestlistinx  ON Emptestlist(JOB) LOCAL (PARTITION Jpart1);
```

- The LOCAL keyword tells oracle to create a separate index for each partition of the table.

Oracle Object Oriented Concepts

An Object is a reusable application component that developers need to be aware of, rather than how it works. Object are the basic entities in a system. They could represent a person, place, bank account, or any item that is handled by program. Every object consists of an attribute and one or more methods. An attribute could be any property of the object.

Class:

It is a collection of attributes and functions(method) to plan the object.

Object Table

- Object table are created by using the user defined data types.
- In an Object Table each row or record is treated as an object.
- Each Row in an Object Table has an Object Identifier(OID),Which is unique through out the Database.
- The Rows or Objects of an object table can be referenced by other objects with in the Database.
- An object table is created using the CREATE TABLE command.
- All object types are associated with default method applied upon the relational tables i.e INSERT,DELETE,UPDATE and SELECT.
- The relational DML operation style is accepted only when the user defined data type is a collection of Built-in data types, and the object table does not contain any REF Constraints.

Creating an user defined object type

Syntax

```
Sql>CREATE OR REPLACE TYPE
        <object name>as
        Object (Element1 <data type>(size),
                Element2 <data type>(size),
                ElementN <data type>(size)   );
```

- All user defined data types are schema objects of the database.
- The user defined object data type can be used as Reference in other tables.
- All user defined data types and object are stored permanently in the data dictionaries
- USER_TYPES
- USER_OBJECTS
- We can Query for the user defined data types and objects using the relational SELECT.

```
Sql>Select Type_Name,TypeCode,
    Attributes,Methods
    From USER_TYPES;
Sql>Select Object_Name,
    Object_type
    From USER_OBJECTS;
```

Creating User Defined Address Type

```
Sql>Create or Replace TYPE Addr_type   AS OBJECT ( Hno VARCHAR2(10),
                Street VARCHAR2(20), City VARCHAR2(20) , Pin NUMBER(6));
Sql>Create  or Replace TYPE PF_TYPE AS OBJECT ( PFNO NUMBER(6),
        AMT NUMBER(12,2));
```

- The above statement create the user defined object data type called as Addr_type and PF_TYPE in the data Dictionary called USER_TYPES.
- This data type is also called as collection in oracle, and this collection is used where ever the same data type Collection is expected in projections.

Creating Table with user Defined Data Type

```
Sql>CREATE TABLE EMPLOYEE( ECODE NUMBER(3), NAME VARCHAR2(20),
        ADDRESS ADDR_TYPE, BASIC NUMBER(12,2), PF PF_TYPE);
```

- Once the user defined data types are created we can instantiate them in the normal relational tables.
- These instances look as normal attributes with in the table, but can be operated only with CONSTRUCTOR METHOD or OBJECT VIEWS.
- In any of the operation we have to provide reference to all attributes with in the instance, but partial association is not accepted.

Sql>Create table Supp_Det(Rol number(3), Name varchar2(20), Saddr addr_type,Course varchar2(20));

Inserting the data into Employee Table

```
Sql>INSERT INTO EMPLOYEE VALUES(100,'Vijji', ADDR_TYPE('130 A',
                'JAYA NAGAR','BLORE',516217), 8000,PF_TYPE(1200,400));
```

Select Data from OBJECT Table

```
Sql>SELECT *FROM  EMPLOYEE;
Sql>SELECT  ADDRESS FROM  EMPLOYEE;
Sql>SELECT NAME,E.ADDRESS.CITY FROM EMPLOYEE E;
```

Note:

- Alias is must to retrieve/manipulate Object elements.

Update data from Object Table

```
Sql>UPDATE EMPLOYEE E SET BASIC= BASIC+1000  WHERE
                                        E.ADDRESS.CITY='HYD';
```

Deleting Data from Object table

```
Sql>DELETE FROM EMPLOYEE;
Sql>DELETE FROM EMPLOYEE E   WHERE E.PF.PFNO=1200;
```

Creating User Defined Emp_Det Type

Sql>Create type Info As Object(code Number(4), Name Varchar2(20), Hno VARCHAR2(10),Street VARCHAR2(20), City VARCHAR2(20) , Pin NUMBER(6));

Creating an Object Table

- Is a table which is entirely build from the abstract type.

Syntax:

Create table <Table Name> of <Object Name>;

Illustrate

Sql>Create table Emp_Info of Info;

- The above statement creates the object table Emp as an Abstract data type.
- Each row in the object table has an OID value generated by oracle server.
- The rows in the object table are referenced as OBJECTS.

Inserting rows into Object Tables

- To INSERT a record into an OBJECT TABLE we may use the CONSTRUCTOR METHOD of the actual data type or directly implement the RELATIONL INSERT statement.
- The Normal INSERT OR RELATION INSERT is possible only when the table does not contain any nested data types.

Relational INSERT

Sql>Insert into Emp_info Values(1001,'SANJU','12-134','SRNAG','MGS','Blore',516217);

Inserting Using Construct Method

Sql>Insert into Emp_info Values(Info(1002, 'Kelly','12-135','VIJII NAG','Blore',516217));

Sql> Create table supp_Info of Info;

REF clause:

It is used to retrieve the object reference. The reference of the object is the address of the row object, witch is internally generated by oracle.

The REF() Function

- The REF() function allows to reference existing row objects.
- The OID assigned to each row can be seen by using the REF function.

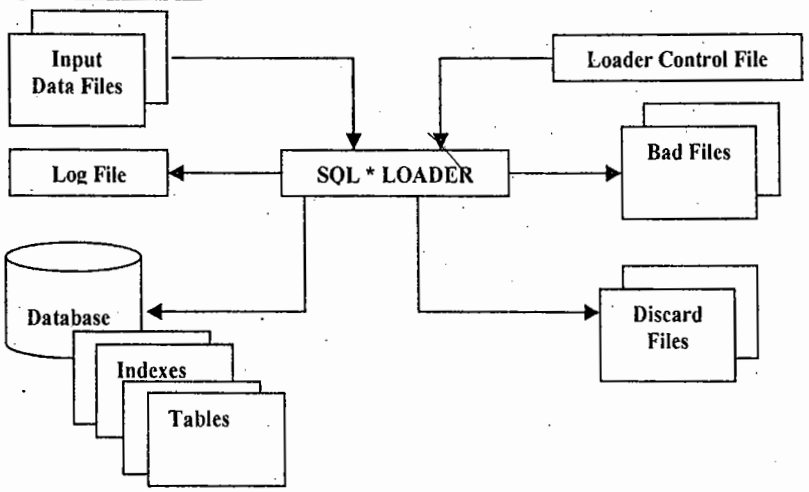
Sql>Select REF(AN) From Etab AN Where City='HYD';

City will not accept null values

```
>create or replace trigger city_chk
 before insert on employ
  for each row
Begin
     if :new.address.city is null then
       raise_application_error(-20101,
                          'city cannot empty');
    end if;
End;
```

SQL* LOADER

SQL * LOADER :



Working Scheme: SQL * Loader uses a control file called Loader control File, which contains instructions for the execution, manipulation, and filtering of the records contained in the input datafiles. During execution, SQL * Loader produces a log file with the operation. If there are records with problems in the input files, a File with the rejected records is produced.

Discarded or Rejected Records: The records read from input files that cannot be inserted in the database are written in separate files for further checking or Correction

Bad File: A bad file contains the records that were rejected by SQL * Loader or by oracle. Normally, this occurs when the record format is invalid. A common situation & quotation marks missing in a field, or a field without the separation comma, after it is approved by SQL * Loader, the record is inserted in the database. However, oracle can reject it violates some constraint

Discard File: SQL * Loader can also create a file called a discard file. The discard file is created when it was specified as activated. It contains the records that did not pass through the selection filter of the control files.

Log File: When executed, SQL * Loader, creates a log file containing detailed information about the data processing, including information about possible errors.

The SQL * Loader Control File: The control File specifies how data in the input file is treated, which tables will receive it, which conversions are made, and so on.

This is a text document file, and the commands to be interpreted by SQL * Loader are written to it. They include the specification of the input file location characteristics of rows and columns, specification of data types, and so on. The contents of the file depend on the type of data configuration. However, we can say that it is divided into three parts.

The first section contains general information, such as the INFILE option to specify the input file, the specification of the set of characters. Or global options such as rows, records to skip, and son.

The second section may contain the INTO TABLE commands each of these commands contains information relating to the table that will receive data.

The third section is optional and can store input data, turning the control file into a datafile.

Load Data: This is a manadatory command that tell SQL * Loader to configure data

Infile: This specifies the input datafile. If the filename extension is omitted, data is assumed. If on asterisk is specified, the program assumes that data is inside its own control file, after the word BEGINDATA.

Examples:

```
INFILE
INFILE test.dat
INFILE 'C:/topdir/subdir/datafile.dat'
```

Append/Replace/Truncate: If the table to be configured are empty, you can use the INSERT option. If the tables have contents, the options APPEND, REPLACE, or TRUNCATE can be used.

APPEND adds the rows to the end of the table, REPLACE deletes all the rows of the table before inserting new ones, and TRUNCATE adds the rows in an attempt to obtain the best performances.

Delimiters: Special Characters can delimit fields at type char, date, or numeric external. The delimited fields can be enclosed in a character or terminated by another character.

Ex:

```
load data
infile "d:\dept.txt"
insert into table dept
fields terminated by ","
(deptno,dname,loc)
-------------------------------------
load data
infile "d:\dept.txt"
append into table dept
fields terminated by ","
(deptno,dname,loc)
-----------------------------------------
load data
infile "d:\dept.txt"
truncate into table dept
fields terminated by ","
(deptno,dname,loc)
---------------------------------
load data
infile "d:\dept1.txt"
replace into table dept
fields terminated by ","
(deptno,dname,loc)
---------------------------------
load data
infile *
truncate into table dept
fields terminated by ","
(deptno,dname,loc)
Begindata
60,SW,BLORE
70,EXP,HYD
-----------------------------------------
load data
infile "d:\dept.txt"
infile "d:\dept1.txt"
insert into table dept
fields terminated by ","
(deptno,dname,loc)
-------------------------
```

Flashback Query: -->Using Flashback feature which was introduced from Oracle 10g, we can quickly rewind a data base to previous time to correct any problems or users errors.
-->It provides point in time recovery without requiring a backup to restore.

Example:

```
Login krishna/oracle
>desc recyclebin
Step1:
 drop table.(Temp)
>select OBJECT_NAME,ORIGINAL_NAME,OPERATION
 from recyclebin
  where ORIGINAL_NAME='TEMP';
To find droptime,createtime  of table.
>select OBJECT_NAME,ORIGINAL_NAME,OPERATION,droptime,createtime
from recyclebin
where ORIGINAL_NAME='TEMP';
To recovery the TEMP table
> flashback table temp to before drop;
```

Query for Practice

1)DEPT table is required with the following structure:
DEPTNO NUMBER(10)
DNAME CHAR(30)
LOC VARCHAR2(33)
2)EMP table is required with the following structure.
EMPNO NUMBER(2)
ENAME VARCHAR2(15)
JOB VARCHAR2(15)
MGR char(4)
HIREDATE DATE
SAL NUMBER(6)
COMM NUMBER(6)
DEPTNO NUMBER(2)
3)Display the DEPT details from department table.
4)Display the EMP details from employees
5)Display empno,ename and job for all employees
6)Reduce the size of DEPTNO column of DEPT table to 2
7)change the data type of DNAME to VARCHAR2 and width to 15.
8)Reduce the with of LOC to 15.
9)Increase the size of EMPNO to 4
10)Reduce the size of JOB to 10
11)Change the data type of MGR to NUMBER(4)
12)SAL should accept 5 integers and 2 decimals
13)COMM should accept 5 integers and 2 decimals
14)Display all the details of all SALESMAN
15)List the emps who joined before 1982.
16)List the empno,ename,deptno, hiredate,and Exp of all Managers.
17)Display all the details of the emps whose Comm is more than their Sal
18)List the emps along with their Exp and whose Daily Sal is more than Rs 200
19)JOB can be duplicate
20)COMM can be NULL
21)HIREDATE should not be SUNDAY
22)SAL should be less than or equals to 10,000
23)EMPNO should be primary key
24)DEPTNO of DEPT table should be PRIMARY KEY
25)DEPTNO of EMP table should be a foreign key to DEPTNO of DEPT table.
26)Display the names of all the employees and their experience in years.
27)Deptno of EMP table should be a foreign key to DEPTNO of DEPT table.
28)List all employees number,name,job and hiredate of employees in department 10.
29)Select the name and salary of all employees who are CLERKS
30)Display the unique Dept's of emps.
31)Display emps whose Salary is more than 2000 after giving 20% increment.
32)List the Empno,Ename and Sal is increased by 20% and expressed as No.of.Rs
33)Display the unique Dept with Jobs.

34)List all MANAGER,CLERKS

35) Display all employee names and their salaries whose salary is not in the range on 1500 and 2850

The Column Heading to be displayed as

Emp Name	Monthly Salary

36) Display the name of the employee who don't have any manager.

37) Display the name ,salary and commission for all employees whose commission amount is greater than their salary increased by 10%

38) Display the Ename,salary and salary increased by 15% expressed as a whole number.Label the column New Salary.In addition to this display the Increase of salary over the previous one.

For Example.

ENAME	SAL	NEWSAL	INCREASE
KING	5000	5500	500

39)Display the Employee name ,hiredate, salary review date which is the first Monday after 6 months of the service.

Ename	Hiredate	Review
KING	17-NOV-81	Monday, the Twenty-Forth of May,1982

40)Display employee number and total salary for each employees.
41)Display employee number and annual salary for each employees.
42)Display the names of all employees who are working in department number 10
43)Display the names of all employees working as MAMAGER and drawing a salary more than 2000.
44)Display employee number and names for employees who earn commission.
45)Display names of employees who do not earn any commission
46)Display the names of employees who are working as CLERK,SALESMAN or ANALYST and drawing a salary more than 2000.
47)Display the names of employees who are working in the company for the past 5 years.
48)Display the list of employees who have joined the company before 31th JAN 81 or 30 NOV 81.
49)Display current user
50)Display current date
51)list the first 10 records of the EMP table.
52)List the emps empno,ename,sal and the column heading as Emp_Num,Emp_Name,Salary.
53)List all the unique deptno of emps.
54)Display the Job and the no. of persons working in each job type.
55)Display the name and salary of all employees who are CLERKS.
56)List the name,job and salary of everyone hired on December 17,1980.
57)List the Department name and department number for departments with numbers greater than or equals to 20.
58)Select the name, salary and commission of employees whose commission is greater than their salary.
59)Display the names of employees working in department number 10 or 20 or and employees working as CLERK,MANAGER or SALESMAN.
60)Display the name, monthly salary, daily salary and hourly salary for all employees. Assume that SAL column in the table is monthly salary; that there are 22 Working days in a month; and that there are 8 working hours in a day. Rename the columns as MONTHLY, DAILY and HOURLY.

61)Display details of FORD.
62)Display the names of employees whose name starts with alphabet M
63)Display the names,job of employees whose job ends with alphabet R
64)List the names of employees whose name is exactly four character in length
65)List the names, jobs of employees whose name have second alphabet
66)Display the names of employees who are not working as manegers
67)Display all employees joined on 1 MAY 1981.
68)Display empno, ename, hiredate of all employees joined before 1 JAN 1983.
69)Display employee number and total salary for each employee
70)Display employee name and annual salary for all employees
71)Display the names of all employees who are working in department number 10
72)Display the names of all employees working as clerks and drawing a salary more than 2000
73)Display the employees whose names start with 'T' and end with 'R'
74)Display the employees whose exp is more than 10 years
75)Display the who are working as clerks and exp is more than 15 Years
76)How to create .LST file
77)Display the names of employees who are working as clerk , salesman or analyst and drawing a salary more than 3000
78)Display the total number employees working in the company
79)Display the total salary and total commission to all employees.
80)Display all rows from emp table. The system should wait after every screen full of information.
81)List the names and employee numbers of managers who earn more than 2600,in the order of names.
82)select the information about managers and the president from the column job in the EMP table. order the result by department number.
83)List all the employee names that do not end with 'S'.
84) Display the maximum salary from emp table
85) Display the minimum salary from emp table
86) Display the average salary from emp table
87) Display the maximum salary being paid to CLERK
88) Display the maximum salary being paid in dept no 20 Display the 89) Display the minimum salary being paid to any SALESMAN
90)List the name,job and department of everyone whose name falls in the alphabetical range from 'C'to 'F'
91) List the empno,ename,sal,job of emps with the annSal<3400 but receiving some comm. Which should not be greater than Sal and the desg should be Salesman working for dept 20.
92) List all the unique jobs along with deptno.
93) Display the average salary drawn by managers
94) Display the total salary drawn by analyst working in dept no 40
95) Display the names of employees in order of salary i.e. the name of the employee earning lowest salary should appear first.
96) Display the names of employees in descending order of salary.
97) Display the details from emp table in order of emp name.
98) Display empnno,ename,deptno and sal. Sort the output first based on name and within name by deptno and witdhin deptno by sal.
99) Display the name of employees along with their annual salary(sal*12). The name of the employee earning highest annual salary should appear first?
100)Display all employee names start with 'TH' or 'LL'.
101)Display all employees who are hired during 1983.
102)Display the data as Who..What..When. Ex SMITH HAS HELD THE POSIT OF CLERK IN DEPT 20 SINCE 13-JUN-83.
103)List the details of the employees in department 10 and 20 in alphabetical order of names.
104)List all rows from EMP table by converting the NULL values in COMM col to 0.

105)List all SALESMAN
106)List all managers and salesman with salaries over 1500.
107)Write a query that will accept a given job and displays all records according to the job.
108)List the names and hiredates of employees in department 20.Display the hiredates as 'DD/MM/YY'.
109)How many months has the president worked for the company?Round to the nearest whole number.
110)List all the employee names, jobs and a job classification ,which you will supply. Translate the value stored in each job filed(CLERK,MANAGER,etc) to a job Classification number(1,2,etc). Translate CLERK to 1,MANAGER to 3,PRESIDENT to 5 and all other jobs to 2.Name the job classification column JOB-CLASS.
111) List all the emps who are working since 1st April 1982.
112) List the emps who joined before 1985 and salary is more than 3000.
113) Give SQL commands to find the average annual salary per jobs in each department. The SAL figures in the EMP table are for each month.
114) In one query, count the number of people in department 30 who can receive a salary and the number of people who receive the commission.
115) Display name,salary,Hra,pf,da,TotalSalary for each employee. The out put should be in the order of total salary,hra 15% of salary,DA 10% of salary .pf 5% salaryTotal Salary will be salary+hra+da)-pf?
116) Display Department numbers and total number of employees working in each Department?
117) Display the various jobs and total number of employees working in each job group?
118) Display department numbers and Total Salary for each Department?
119) Display department numbers and Maximum Salary from each Department?
120) Display various jobs and Total Salary for each job?
121) Display each job along with min of salary being paid in each job group?
122) Display the department Number with more than three employees in each department?
123) Display the department Number with more than three employees in each department?
124) Display various jobs along with total salary for each of the job where total salary is greater than 4000?
125) Display the various jobs along with total number of employees in each job.The output should contain only those jobs with more than three employees?
126) Display the emps whose job same as FORD.
127) Display the emps who are senior to KING.
128) Display the name of employees who earn Highest Salary?
129) Display the employee Number and name for employee working as clerk and earning highest salary among the clerks?
130) Display the names of salesman who earns a salary more than the Highest Salary of the clerk?
131) Display the names of clerks who earn a salary more than the lowest Salary of any salesman?
132) Computer Average, Minimum and Maximum salaries of those group of employees having the job of CLERK or MANGER.
133) Display the department number where more than two clerks are working.
134) Calculate the total compensation expense for each department for one year. The sal and comm figures in the EMP tables are for each month. Assume that employees who don't earn a commission receive non-monetary benefits that are worth Rs.100/-.
135) Do a case sensitive search for a list of employees with a job that a user enters.
136) Display the names of employees who earn a salary more than that of jones or that of salary greater than that of scott?
137) Display the names of employees who earn Highest salary in their respective departments?
138) Produce the names and jobs of employees as Ex:-SMITH(CLERK).

139) Which employees earn less than 30% of the president's salary?
140) Display the names of employees who earn Highest salary in their respective departments?
141) Display the names of employees who earn Highest salaries in their respective job Groups?
142) Display employee names who are working in Accounting department?
143) create a view consisting of employees and their total sum of salary grouped by department number wise.
144) Display the employee names who are Working in Chicago?
145) Transfer SMITH to deptno 10.
146) Transfer the emps of dept 20 to 30.
147) Transfer the emps CHICAGO to DALLAS.
148) Create a view consisting of all the columns from EMP table and their corresponding records from dept table consisting of department name & location.
149) How many employees are working in NEW YORK.
150)Update the Salary of 'ALLEN' with the highest paid emp of Grade 2 and transfer him to BLAKEs dept and change the MGR to BLAKE.
151) Create a query that will display the employee name ,dept number and all the employees name that work in tha same dept as a given employee.
152) Display dept name,location,number of employees and the average salary for all employees in that dept.
153) Display the employee name and his manager's name.
154) Display the employee's name, department's name ,grade and manager's name.
155) Print the following: salary itself if it is 1500.'HIGH' if it is more than 1500.'LOW' if it is less than 1500.
156) Display the job groups having Total Salary greater than the maximum salary for Managers?
157) Display the names of employees from department number 10 with salary greater than that of ANY employee working in other departments?
158) Display the names of mployees in Upper Case?
159) Display the names,jobs of employees in Lower Case?
160) Display the names of employees in Proper case?
161)Find the length of your name using Appropriate Function?
162) Display the length of all the employee names?
163) List department number, department name, location, commission paid and total salary of each department.
164) Display the average monthly salary bill for each job type with in a department.
165) Display those jobs where the minimum salary is greater than or equal to 3000.
166) Display the name of employee Concatinate with Employee Number?
167) Use appropriate function and extract 3 characters starting from 2 characters from the following string 'Oracle' i.e., the out put should be ac?
168) Find the first occurance of character o from the following string Naresh iTechnologies .
169) Increment the Salaries by 5% and add a comm. of 300 to the existing comm. and changing the clerk to jobs of all 'salesman' whose salaries is more than or equal to 1500.
170) Increment the sals of the emps by 5%
171)Increment the sal of all the by Rs 500 and change their MGR to 7902
172) Change the MGR of all BOSTON related emps FORD.
173)Transfer the emps to analyst dept and give the sal of smith plus 500 to those belongs to grade 3 working at NEW YORK or DALLAS with an exp>7y whose name should not be 4 chars.
174)Find out the difference between highest and lowest salaries.
175)Find all department, which have more than 3 employees.
176) List lowest paid employee working for each manager, exclude any group where the minimum salary is less than 1000,in the reverse order of salaries.
177) Display all employee names and their department names in the order of

department names.
178) Display all employee name, department numbers, and department names
179) Replace every occurance of alphabet A with B in the string .Alliens (Use Translate function)?
180) Display the information from the employee table . where ever job Manager is found it should be displayed as Boss?
181) Display empno,ename,deptno from emp table. Instead of display department numbers display the related department name(Use decode function)?
182) Display your Age in Days?
183) Display your Age in Months?
184) Display current date as 15th August Friday Nineteen Nienty Seven?
185) Display the following output for each row from emp table?
Scott has joined the company on 13th August ninteen ninety?
186) Update the deptno of sales dept for those emps working at deptno 10.
187) update the salary of FORD with the highest paid salary emp of SALESMAN more than 8 Y exp.
188) Change the mgr to 7788 for those working for the mgr 7369.
189) Display the departments that have no employees.
190) Find all employees who joined the company before their managers.
191) Find the employees who earn more than the lowest salary in each department.
192) Add 300 to the comm. of all SALESMAN who are receiving some comm.
193) Change the deptno of FORD to 30 and also change the job as SALEMAN with an increment in the Salary 8%.
194) Display employees who earn more than the lowest salary in each department 30.
195) Find the job with highest average salary.
196) Display the name, job, and hiredate for employees whose salary is more than the highest salary in SALES dept.
197) Transfer the emps of dept 20 to SALES dept.
198) Give an increment of 300 to all emps of grade 2.
199) Replace the sal of SMITH with the highest paid CLERK of NEW YORK or DALLAS.
200) Display all the employees whose grade is the same as that of JONES.
201) Display the empname,salary of all employees who report to KING.
202) Display the empname,gross,job,loc who are having more than JONES Gross.
203) Display current date and 78 days after.
204) Find the nearest Saturday after Current date?
205) Display the current time?
206) delete all the information of CLERKS.
207)Delete all Grade 2 emps
208)Delete all employees who joined in the month of DEC 1981.
209)Delete all the emps working in NEW YORK.
210)Delete all the emps who are senior than their manager.
211) Display the date three months before the Current date?
212) Display the common jobs from department number 10 and 20?
213) Display current date and 78 days after.
214) Display the current date in the following fashion.
215) Display Employee name and Job information from employee in the following fashion.
216) Display all the information from dept table where second character of LOC name is 'A'.
217) Display the last date in Feb-97.
218) Display logged on user name in the following fashion.Current User Name is : SCOTT
219) Display 'NARESH TECHNOLOGY IS VERY GOOD FOR ORACLE' text as 'Naresh Technology Is Very Good For Oracle' in a Select statement.
220) Select employee name ,Hiredate in the format of "2nd of july 1997" for deptno 10 and 20.
221) Display employee names in lower case whose salary is greater than 2000 and less than 2800.

222) Display the jobs found in department 10 and 20 Eliminate duplicate jobs?
223) Display the details of those employees who do not have any person working under him?
224) Display the details of those employees who are in sales department and grade is 3?
225) Display those department whose name start with"S" while location name ends with "K"?
226) Display those employees whose manager name is Jones?
227) Display those employees whose salary is more than 3000 after giving 20% increment?
228) Display all employees with their department names?
229) Delete all the emps working under FORD with exp>5year.
230) Delete the Information FORD,SCOTT.
231) Delete the emps who joined most recently under FORD.
232) create a unique index for employee names of emp table.
233) Assuming the salary of emp table as for a month, produce annual salary with haeding as ANNUAL SALARY'.
234) S lect employee number and name combined to together to together with a heading 'EMPLOYEE'.
235) Display in how many department employees are working.
236) List the employees who are having experience of more than 10 years
237) List the employees who have 'I' or 'LL' as the exact middle character(s) of their names.
238) List the employees whose name bigger than their manager's name.
239) List the employees who are managers of clerk with salary more than 2000.
240) Find all the managers in any department and all clerks in department 10 only.
241) List the employee names by adding '-' on left side, to the double of the actual size of the name.
242) Display ename who are working in sales department?
243) List the total information of EMP table along with dname and loc of all the emps working under 'SALES' or 'ACCOUNTING' in the asc deptno.
244) Display employee number, employee name, department name, location from employee and department tables.
245) Display employee number, employee name, salary + comm as TOTAL, location from employee and department tables.
246) Display the empno,ename,sal,dname,loc of all the CLERK or MANAGER working in NEW YORK or CHICAGO with an exp more than 8 years without receiving the COMM.
247) Diplay employee name, department name, job from employee and department tables who job is CLERK.
248) Display employee name, deptno, dname from employee and department table where location is DALLAS.
249) Display empno,ename,deptno,dname from emp and dept who empno is 7788 or 7902.
250) Display first 3 characters of employee name, job, sal, comm, location from employee and department tables who works in department 20 or 30, salary should be more than 1000 comm as not null.
251) Delete the emps who joined most recently under KING.
252) Delete grade 3 and 4 emps.
253) List the employee names where the second occurrence from second position is 'A'.
254) Display employee name,job,hiredate and salary combined together without using the '||' operator.
255) Display employee name,dept name,salary,and commission for those sal in between 2000 to 5000 while location is Chicago?
256) Display those employees whose salary is greater than his managers salary?
257) Display those employees who are working in the same dept where his manager is work?
258) Display those employees who are not working under any Manager?

259) Display the grade and employees name for the deptno 10 or 30 but grade is not 4 while joined the company before 31-DEC-82?
260) Update the salary of each employee by 10% increment who are not eligible for commission?
261) Delete those employees who joined the company before 31-Dec-82 while their department Location is New York or Chicago?
262) Delete the emps who belongs to Grade 2 or 3 working at Chivago joined in any month of the first half of 82.
263) Delete duplicate records in the emp table.
264) Delete those dept's where no employee is working.
265) Delete the emps who are senior to their own managers.
266) List all managers from all departments and all clerks from department 10 only.
267) List all employee names from from EMP table if the name contains second and last-but-one character as W.
268) List all columns from both the DEPT and EMP tables
269) Display different jobs and the number of employees working from EMP table.
270) Display the difference between total salary of department 10 and department 20.
271) Display first 3 characters of employee name, job, sal, comm, location from employee and department tables who works in department 20 or 30, salary should be more than 1000 comm as not null.
272) Display location, department name, length of location, length of department name for employee numbers 7788, 7902,7346.
273) Display location and sum of the salary spending for employees on that location respectively.
274) Select department number, dname, average salary for each department.
275) Select department number, name where atleast 3 employees works.
276) Select department name, location where the average salary in each department more than 1600.
277) Count how many employees have 'S' in their names.
278) Produce this formant

EMPLOYEE..... JOB
SMITH.....CLERK
ALLEN.....SALESMAN

279) Display employee name ,job,deptname,loc for all who are working as manager?
280) Display those employees whose manager name is jones and also display their manager
281) Display name and salary of ford if his salary is equal to hisal of his grade?
282) Display employee name ,job,deptname,his manager name ,his grade and make an under department wise?
283) List out all the employee names ,job,salary,grade and deptname for every one in a company except 'CLERK' . Sort on salary display the highest salary?
284) Display employee name,job and his manager .Display also employees who are with out managers?
285) Display Top 5 employee of a Company?
286) Display those employees whose salary is equal to average of maximum and minimum?
287) Select count of employees in each department where count >3?
288) Display dname where atleast three are working and display only deptname?
289) Display name of those managers name whose salary is more than average salary of Company?
290) List employee names and hiredates by adding one year to the date.
291) List the employee names who have minimum and maximum experience.
292)Display those managers name whose salary is more than average salary of his employees?
293) Display employee name,sal,comm and netpay for those employees whose netpay is greater than or equal to any other employee salary of the company?
294) Display those employees whose salary is less than his manager but more than

salary of other managers?
295) Find the last 5(least) employees of company?
296) Find out the number of employees whose salary is greater than their managers salary?
297) Display the manager who are not working under president but they are working under any other manager?
298) Delete those records from emp table whose deptno not available in dept table?
299) Display those enames whose salary is out of grade available in salgrade table?
300) Display employee name,sal,comm and whose netpay is greaterthan any othere in the company?
301) Display name of those employees who are going to retire 31-Dec-99 if maximum job period is 30 years?
302) Display those employees whose salary is odd value?
303) Display those employees whose salary contains atleast 3 digits?
304) Display those employees who joined in the company in the month of Dec?
305) Display the Cartesian product of EMP and DEPT tables.
306) Display those employees whose name contains A?
307) Display those employees whose deptno is available in salary?
308) Display those employees whose first 2 characters from hiredate – last 2 characters sal?
309) Count how many duplicate records are there in the EMP table.
310) List ROWIDs from EMP table.
311) Create a view with different job and sum of salary from EMP table. Is manipulation possible in the view?
312)Create a view with EMPNO,ENAME,JOB and SAL. Is manipulation possible in the view directly?
313) Display those employeess whose 10% of salary is equal to the year joining?
314) Display those employees who are working in sales or research?
315) Display those employees who joined the company before 15th of the month?
316) Create a view with DNAME,ENAME,SAL .Is it possible to manipulate the view directly?
317) Get all department numbers, which are present in DEPT table but not in EMP table.
318) List all employees who are hired in last week of a month.
319) Delete those employee who joined the company 10 years back from today?
320) who are the top three earners of the company?
321) In the employee table if the job is 'CLERK' change it to 'WORKER' if job is 'MANAGER' change it to 'BOSS', otherwise change it to 'EMPLOYEE'.
322) Find all employees who have the job equal to that of SMITH.
323) Create a sequence and change the employees' number of EMP table into serial numbers.
324) Display the experience of employees in years, months, weeks and days.
325) Write a query to display a character in Pyramid format.
326) Count th number of employees who are working as managers (Using set opetrator)?
327) Display the name of employees who joined the company on the same date?
328) Display the name of the dept those employees who joined the company on the same date?
329) display those employees whose grade is equal to any number of sal but not equal to first number of sal?
330) Count the no of employees working as manager using set operation?
331) Display the name of employees who joined the company on the same date?
332) Display the manager who is having maximum number of employees working under him?
333) What is name of Default Index(in case of Primary key, Unique).
334) List out the employee name and salary increased by 15% and express as whole number of Dollars?
335) Produce the output of the emp table "Employee_and Job" for ename and job ?

336) List of employees with hiredate in the format of 'June 4 1988'?
337) Print list of employees displaying 'Just salary' if more than 1500 if exactly 1500 display 'on taget' if less than 1500 display below 1500?
338) Which query to calculate the length of time any employee has been with the company.
339) Given a string of the format 'nn/nn' . Verify that the first and last 2 characters are numbers .And that the middle character is '/' Print the expressions 'Yes' IF valid NO' of not valid . Use the following values to test your solution'12/54',01/1a,'99/98'?
340) Employes hire on OR Before 15th of any month are paid on the last friday of that month those hired after 15th are paid the last friday of th following month .print a list of employees .their hiredate and first pay date sort those who se salary contains first digit of their deptno?
341) Print the details of employees who are subordinates to BLAKE?
342) Display those who working as manager using co related sub query.
343) Write an SQL statement to refresh the salaries, according to experience of the employees. For one year of experience, the salary will be 1000.
344) Write an SQL statement to accept date of birth and display the age in years, months, weeks, days, hours, minutes and seconds separately .
345) Write an SQL statement to insert a record into DEPT table. If the user provides a value for LOC It should be inserted; Otherwise 'HYDERABAD' should be inserted as default.
346) Find out how many managers are there with out listing them
347) .Use the variable in a statement which finds all employees who can earn 30000 a year or more(i.c using &).
348) In which year did most people join the company. Display the year and number of employees.
349) create a copy of emp table.
350) Display those employee whose joining of month and grade is equal.

NARESH i™
technologies

Rs =50/=

PL/SQL

Course Material

By Mr. Krishna Reddy

Opp. Satyam Theatre, Ameerpet, Hyd.
Office: 040- 23746666 Mobile: 9000994007/8

- PL/SQL is a procedural extension to a non-procedural language SQL.
- PL/SQL is one of the utility of Oracle dbserver and it is not a tool like SQL*plus.
- It is a block-structured language.
- PL/SQL block is a logical collection of procedural as well as non-procedural statements. It is a data processing language, but SQL is a powerful Data Manipulation Language.
- PL/SQL is having all the features of procedural language as well as all non-procedural features(because of SQL support).
- PL/SQL can used for both Server-Side and Client-Side Development.
- PL/SQL block is a logical collection of procedural as well as non-procedural statements.

Types of valid statements of PL/SQL block

- SQL statements(non procedural statements)
- Non-SQL statements(procedural statements)

Block structure approach

- The basic units that make up a PL/SQL program are logical blocks.
- The block in PL/SQL can be nested with one another.
- A block groups related declarations and statements into one single unit.
- The basic parts of a PL/SQL block are
    - Declarative part(optional)
    - Executable part(Mandatory)
    - Exception Handling(optional)

Declarative part

- It is used to define user defined types, variables which can be used, in the executable part for further manipulations.

Executable Part

- All procedural statements are included between the BEGIN and END statement.
- It must have one executable statement.

Exception Handling Part

- Error that occur during execution are dealt in the exception handling part.

Notes:

- A PL/SQL program is a logical block, which contain any number of nested sub blocks.
- Block can be nested in the executable and exception handling parts of a PL/SQL block, or a sub program.
- A PL/SQL is marked with either a "DECLARE" or "BEGIN" keyword and ends with the keyword "END"

- Only BEGIN and END keywords are mandatory.
- A semicolon(;) has to be placed after the "END" keyword.

Block syntax

```
SQL>DECLARE
        [Variable declarations;
        Cursor declaration;
        User_defined exception;]
    BEGIN
    <SQL statements;>
    PL/SQL statement
    EXCEPTION
    Action to perform when errors occurs.
    END;
```

- Only executable section is required.
- The declarative and exception handling section are optional.
- We can define local sub programs in the declarative part of any block.
- However , we can call local subprograms only from the block in which they are define.

A simple PL/SQL Block

```
SQL> BEGIN
        NULL;
    END;
```

NULL block.

```
SQL> BEGIN
        RETURN;
    END;
```

→BLOCK with RETURN clause.

```
SQL>DECLARE
    BEGIN
        NULL;
    END;
```

```
SQL>DECLARE
    BEGIN
        NULL;
        EXCEPTION
        WHEN OTHERS THEN
        NULL;
    END;
```

Executing statements and PL/SQL blocks from SQL*Plus

- Place a semicolon(;) at the end of the SQL statement or PL/SQL control statement.
- Use a Forward slash(/) to run the Anonymous PL/SQL block in SQL*plus buffer.
- Place a period(.) to close a SQL*Plus Buffer with out running the PL/SQL program.
- A PL/SQL block is treated as one continuous statement in the SQL*plus buffer.
- Semicolon within the PL/SQL block does not closes or Runs the SQL Buffer.
- In PL/SQL an Error is called as an Exception.
- Sections keywords "DECLARE","BEGIN", and "EXCEPTION" should not contain A semicolon.
- "END" and all other PL/SQL statements should be applied with a semicolon at the End.

Note:

- Declaration part of PL/SQL block is optional ,but the body of the PL/SQL is must.(The area between begin and end is called as body of the Pl/SQL block.
- Every PL/SQL block is terminated by **END** statement followed by a semicolon.
- PL/SQL is not a data formatting language.

Types of blocks in PL/SQL

- A PL/SQL program can be written in various types of blocks, they are

Anonymous block

- **The block having no name.**
- They are declared at the point in an application, where they are to be executed and are passed to PL/SQL engine for execution at runtime.
- It cannot be called.
- There are used in D2K form.

```
SQL>DECLARE
    BEGIN
    NULL;
    EXCEPTION
    WHEN OTHERS THEN
    NULL;
    END;
```

Named Block

- The block having name and they have all the features as specified for the anonymous blocks.
- Named blocks help in associating with the scope and resolution of variables in different blocks.
- They give the specifications of the named spaces as provided in high level OOPs languages like C++ and JAVA.

- Named blocks are conveniences for variable management.

```
SQL><<FirstBlock>>
        DECLARE
        BEGIN
        NULL;
        EXCEPTION
        WHEN OTHERS THEN
        NULL;
        END FirstBlock;
```

- Named blocks make the PL/SQL blocks more clear and readable.
- Named blocks increase the clarity of programming when we attempt the nesting process, and control structures.

Sub-Programmed blocks

- These are named PL/SQL blocks that can take parameters and can be invoked with in the other anonymous or sub-programmed PL/SQL blocks.
- These blocks are either is declared as procedures or functions.
- A procedure block is used for performing an action, and a functional block is used for performing calculation.
- These sub-programs provides modularity and reusability.

Comments in PL/SQL

- PL/SQL compiler ignores comments.
- Comments promote readability and aids understing.
- PL/SQL supports two types comments
  - Single line comments(--)
  - /*Multi
    Line
    Comments */

Variables in PL/SQL

Use of Variables

Variables can be used for:

Temporary storage of data

- Data can be temporarily stored in one or more variables for use when validating data Input for Processing the Data in the Data flow process.

Manipulation of stored values

- Information is transmitted between PL/SQL and database with variables.
- Variables help in the data manipulation of stored values.
- Variables can be used for calculations and others data manipulations without accessing the database each and every time, Hence they help in reducing the I/O Cycles.

Reusability

- Once declared can be used repeatedly by even in other blocks.

Ease of maintenance

- Variables can be declared based on the declarations of the definitions of database columns.
- Provides Data Independence, reduces maintenance costs and allows programs to adapt the changes as the Database changes.

Handling Variables in PL/SQL

- Declare and initialize variables in the declaration section.
- Declarations allocate storage space for a value according to its defined size.
- Declarations can be assigned with an Initial value and can be imposed with a NOT NULL constraint or DEFAULT option.
- We reassign new values to variables in the executable section.
- Pass values into PL/SQL blocks through parameters.
- There are Three parameter modes, They are IN(The default),OUT and IN OUT.
- View results from a PL/SQL through output of variables.
- Reference variables can be used for Input or Output in SQL data manipulation statements.

Types of Variables

- PL/SQL variables
  - Scalar
  - Composite
  - Reference
  - LOB (large objects)

Scalar Data Types:

- They holds a single value.
- Main data types are those that correspond to column types in oracle server tables.
- Supports Boolean variables.

Composite Data types:

- A variable of a composite type contains one or more scalar variables.
- Composite data types are similar to structures in 'C' language.
- They help in keeping all the related data items together as one collection.

Reference Data types:

- They hold values, acting as pointers, which designate other program items.
- They are very essential when manipulating collection of data items in sub programs.

LOB Data types:

- They holds values called locations, specifying the location of large objects that are stored out of line in the PL/SQL program.

Operators in PL/SQL

- Logical :**AND,OR,NOT**
- Arithmetic :+,-,*,/
- Concatenation : ||
- Comparison : =,!=,<,>,<=,>=
- SQL*Plus : ISNULL,LIKE,BETWEEN,IN

Assigning values to a variable

- In PL/SQL a variable can be assigned in three ways
- By using assignment operator(:=)
- By selecting or fetching database values(INTO).
- By passing it as IN or OUT or INOUT parameter to a subprogram.

Declaring PL/SQL Variables

Syntax

Identifier name [CONSTANT] datatype [NOT NULL][:=DEFAULT] Expr];

Identifier name

- Specifies the name of the relevant variable for that block.

Constant

- Constant the variable such that its value cannot change during the program process.
- Constants must be initialized else raise an exception.

Datatype

- It is a scalar, composite, reference of LOB data type as applicable to the required situation.

NOTNULL

- Constraints a variable such that it must contain a value and raises if NULL is identified.
- NOT NULL variables should be initialized else raise an exception.

DEFAULT

- Sets the default value for the value in the PL/SQL program if not attended.

Expr

- It is any PL/SQL expression that can be a literal, another variable or an expression involving operations and functions for initialization.

Illustration

```
DECLARE
Vno number(4):=7788;
Vname varchar2(20) not null:='SCOTT';
--must not hold null value through out program
Doj date default sysdate;
Flag boolean:= true;
```

```
--for boolean variable accept only TRUE or FALSE value and
--value should not be in single code
pcode constant number(6):=567893;
--cannot be used as an assignment target.
```

Debugging statement

- The debugging statement of PL/SQL is DBMS_OUTPUT.PUT_LINE();
- For producing outputs on the video device we need the assistance of DBMS_OUTPUT package.
- The package enabled to display output from PL/SQL blocks and sub programs.
- The procedure PUT_LINE() outputs information to a buffer in the SGA.
- This procedures is used to identify the error during runtime of the PL/SQL block.
- This procedure requires a single parameter of any of the data types varchar2 or number or date.
- To see the output of PUT_LINE() on the console of the client side enter the following SQL plus command on the SQL prompt.

Syntax

DBMS_OUTPUT.PUT_LINE('Message'||variablename);

- The DBMS_OUTPUT packages is owned by the oracle user SYS.
- The size of the buffer can be set between 2000 to 10,00,000 bytes.
- The specification to set buffers are
  - SET SERVEROUTPUT ON
  - SET SERVEROUTPUT ON SIZE 5000
  - SET SERVEROUTPUT OFF

```
SQL>BEGIN
    DBMS_OUTPUT.ENABLE;
    DBMS_OUTPUT.PUT_LINE('Frist program in PL/SQL');
    DBMS_OUTPUT.PUT_LINE('Illustraind  by');
    DBMS_OUTPUT.PUT_LINE('''Krishna reddy ''');
    END;
```

- DBMS_OUTPUT.ENABLE; Once declare will make the SERVEROUTPUT process to get activated.

```
SQL> DECLARE
    V_FirstNum NUMBER:=&fno;
    V_SecondNum NUMBER:=&sno;
    V_tol NUMBER;
    BEGIN
    V_tol:=v_FirstNum+V_SecondNum;
    DBMS_OUTPUT.PUT_LINE('The sum of Frist and Second num is '||v_tol);
    END;
```

```
SQL> BEGIN
       DBMS_OUTPUT.PUT_LINE('The sum of Frist and Second num is
       '||(&FirstNum+&SecondNum));
END;
```

Variable Scope and Nested Blocks

Scope:

- The scope of an identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.
- Once the scope of the variable is lost it means that the life of the variable no more.
- All variables loose their scope as soon as the PL/SQL block ends or terminates.
- Oracle automatically release the space upon that variable once its scope is closed.

Concepts of scoping

- With in the same scope, all identifiers must be unique.
- Even when the data types are differing, variables and parameters cannot share the same name.

Nested Blocks and variables scope

- PL/SQL blocks can be nested wherever an executable statement is allowed.
- A nested block becomes a statement.
- An exception section can contain nested blocks.

Scope and visibility diagram

Ex:

```
       DECLARE
       x NUMBER;
       BEGIN
       DECLARE
        y NUMBER;
       BEGIN
       y:= x;
       END;
       ...
       END;
SQL>DECLARE
       m NUMBER:=250;
       BEGIN
       DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
             DECLARE
             m NUMBER :=550;
             BEGIN
             DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
```

```
            END;
        DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
    END;
SQL> DECLARE
m NUMBER:=100;
BEGIN
m:=200;
DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
        DECLARE
        n NUMBER :=300;
        v_tol NUMBER;
        BEGIN
        m:=500;
        n:=600;
        v_tol:=m+n;
        DBMS_OUTPUT.PUT_LINE('The sum of m,n is :'||v_tol);
        END;
DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
END;
```

Labeled Block

- Represent blocks with a label associated to them.

How to convert the anonymous block to Label block

- we need to place a label before the DECLARE keyword.
- The label can also can use appear after the END keyword and it is optional.
- The advantage of of labeled block is that labeled blocks allow you to access those variables that would not be visible when using anonymous block.

```
SQL> <<Block1>>DECLARE
m NUMBER:=100;
BEGIN
m:=200;
DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
        <<block2>>DECLARE
        n NUMBER :=300;
        v_tol NUMBER;
        m number:=400;
        BEGIN
        block1.m:=500;
        n:=600;
        v_tol:=block1.m+n;
        DBMS_OUTPUT.PUT_LINE('The sum of block1 m,n is :'||v_tol);
        DBMS_OUTPUT.PUT_LINE('The value of Inner block is:'||m);
        END block2;
DBMS_OUTPUT.PUT_LINE('The value of Outer m:'||m);
```

```
END block1;
```

```
>CREATE TABLE kcb_acc_tab(
 accno VARCHAR2(20) CONSTRAINT pk_accno PRIMARY KEY,
 name VARCHAR2(20) CONSTRAINT nn_name NOT NULL,
 acctype CHAR CONSTRAINT chk_atype CHECK (acctype IN('C','S','R')),
 doo timestamp DEFAULT sysdate,
 bal NUMBER(7,2)   CONSTRAINT nn_BAL NOT NULL
);
```

Writing Control Structures

- PL/SQL can also process data using flow of control statements.
- PL/SQL helps us manipulates and process oracle data using control statements.
- The flow of control statements can be classified under the following categories:
  - Conditional control
  - Iterative control
  - Sequential control

The statement structures provided by PL/SQL for this purpose are

- IF-THEN-ELSE
- END IF
- ELSIF
- CASE
- END CASE
- LOOP
- FOR-LOOP
- WHILE-LOOP
- END LOOP
- EXIT-WHEN and
- GOTO

- By integration all the above statements in a proper way a PL/SQL programmer can handle any real time situation.

Conditional Control

- Sequence of statement can be executed based on a certain condition using the **IF** statement.
- There are three forms of **IF** statements
  - IF-THEN-END IF
  - IF-THEN-ELSE-END IF
  - IF-THEN-ELSIF-END IF

General Syntax

```
IF condition THEN
    sequences of statements;
END IF;
```

Points to Ponder

- IF...THEN is a reserved word and marks the beginning of the "IF" statement.
- The "END IF" is a reserved phrase that indicates the end of the "IF...THEN" construct.
- When "IF...THEN" is executed, A condition is evaluated to either "TRUE" or "FALSE".

- Conditions are compared by using either the comparison operators or SQL*plus operators.
- One "IF" keyword can manage any number of conditions at a point of time using the LOGICAL CONNECTIVITIES like "AND","OR".
- Even though the multiple conditions need not be constrained by using brackets, It is better to control their precedence using the proper implementations of brackets to avoid ambiguity.
- Every "IF" that is implemented needs compulsorily a "TRUE" state evaluation for its successful execution, But an evaluation state is not compulsory when the condition is "FALSE".
- The sequence of statements is executed only if the condition evaluates to true.
- If it is false or null, then the control passes to the statement after 'END IF'.

```
Sql>BEGIN
        IF  ascii('A')=65 THEN
              DBMS_OUTPUT.PUT_LINE('This is true');
        END IF;
  END;
DECLARE
              v_Firstnum NUMBER:=&Fnum;
              v_Secondnum NUMBER:=&Snum;
              v_Temp NUMBER;
        BEGIN
        DBMS_OUTPUT.PUT_LINE('Original v_Firstnum='||v_Firstnum);
        DBMS_OUTPUT.PUT_LINE('Original v_Secondnum='||v_Secondnum);
        IF v_Firstnum>v_Secondnum THEN
              v_Temp:=v_Firstnum;
        v_Firstnum:=v_Secondnum;
        v_Secondnum:=v_Temp;
        END IF;
        DBMS_OUTPUT.PUT_LINE('Swapped v_Firstnum=  '||v_Firstnum);
        DBMS_OUTPUT.PUT_LINE('Swapped v_Firstnum= '||v_Secondnum);
END;
```

IF...THEN...ELSE...END IF statement

- IF...THEN...ELSE statement enable us to specify two different groups of statements for execution.
- One group is evaluated when the condition evaluates to "TRUE", then next group is evaluated when the condition evaluates to "FALSE"

Syntax

```
IF condition THEN
        sequences of statements;
ELSE
        sequences of statements;
END IF;
```

```
Sql>DECLARE
        v_Firstnum NUMBER:=&Fnum;
        v_Secondnum NUMBER:=&Snum;
    BEGIN
    IF v_Firstnum<v_Secondnum THEN
    DBMS_OUTPUT.PUT_LINE('Smallest of two number is '||v_Firstnum);
    ELSE
    DBMS_OUTPUT.PUT_LINE('Smallest of two number is '||v_Secondnum);
    END IF;
    END;
```

```
SQL>DECLARE
      v_Num NUMBER:=&Enternumber;
    BEGIN
    IF MOD(v_Num,2)=0 THEN

    DBMS_OUTPUT.PUT_LINE(v_Num||' is an Even Number.');
    ELSE
    DBMS_OUTPUT.PUT_LINE(v_Num||' is an Odd Number.');
    END IF;
    END;
```

```
SQL> DECLARE
 v_Number1 NUMBER:= &Number1;
 v_Number2 NUMBER:= &Number2;
BGEIN
IF v_Number1>v_Number2 THEN
DBMS_OUTPUT.PUT_LINE('The Greatest Number is: '||v_Number1);
ELSE
  IF v_Number2>v_Number1 THEN
  DBMS_OUTPUT.PUT_LINE('The Greatest Number is: '||v_Number2);
  ELSE
     DBMS_OUTPUT.PUT_LINE('The Number are equal '||v_Number1||' and
'||v_Number2);
        END IF;
    END IF;
END;
```

Behavior of NULL's

- In simple "IF" when a condition is evaluated to NULL, then the statements in "TRUE" state will not be executed, instead the control will be passed to the first executable statement after the "END IF".
- In IF...THEN...ELSE construct the FALSE block is executed whenever the condition evaluates to NULL.
- Hence when ever a conditional process is executed it is better to cross verify the NULL status of any variable or value before execution.

```
SQL> DECLARE
        v_Firstnum NUMBER:=&Fnum;
        v_Secondnum NUMBER:=&Snum;
   BEGIN
   IF v_Firstnum=v_Secondnum THEN
   DBMS_OUTPUT.PUT_LINE('Given numbers are equal');
            END IF;
   DBMS_OUTPUT.PUT_LINE('Did you watch the NULL effect.');
END;
```

Note:Supply NULL at runtime to check the affect of NULL

```
DECLARE
        v_num NUMBER:=&Enternumber;
   BEGIN
   IF MOD(v_num,2)=0 THEN
   DBMS_OUTPUT.PUT_LINE(v_num||' is an Even number.');
   ELSE
   DBMS_OUTPUT.PUT_LINE(v_num||' is an Odd number.');
            END IF;
   DBMS_OUTPUT.PUT_LINE('Did you watch the Difference...');
END;
```

Nested "IF" Statements

- The "IF" statements can be nested into one another as per requirements.
- Nested "IF" is a situation in which an "IF" follows another "IF" immediately for every "TRUE" state of an "IF" condition.
- Each "IF" is considered as an individual block of "IF" and needs proper nesting.
- Each "IF" block that is opened needs a proper close with "END IF". Else PL/SQL raises exceptions.
- Nested "IF" situations have to be planned when we have a series of conditions fall sequentially.

Syntax:

```
IF condition THEN
    IF condition THEN
        IF condition THEN
        Statement1;
        ELSE
        Statement2;
        END IF;
    ELSE
    Statement3;
    END IF;
ELSE
Statement4;
END IF;
```

```
SQL>DECLARE
v_year NUMBER:=&year;
BEGIN
IF mod(v_year,4)=0 THEN
IF mod(v_year,100)<>0 THEN
DBMS_OUTPUT.PUT_LINE(v_year||' is a leap year');
ELSE
      IF mod(v_year,400)=0 THEN
      DBMS_OUTPUT.PUT_LINE(v_year||' is a leap year');
      ELSE
      DBMS_OUTPUT.PUT_LINE(v_year||' is a not a leap year');
      END IF;
     END IF;
ELSE
DBMS_OUTPUT.PUT_LINE(v_year||' is a not leap year');
END IF;
END;
```

Branching with Logical connectivity's

- In this situation one "IF" is associated with a collection of conditions using either logical either logical "AND" or logical "OR" operator.

Syntax1

IF condition1 AND condition2 THEN

```
        Statement1;
       Statement2;
ELSE
       Statement1;
       Statement2;
END IF;
```

Syntax2

```
IF condition1 OR condition2 THEN
        Statement1;
       Statement2;
ELSE
       Statement1;
       Statement2;
END IF;
```

Syntax3

```
IF condition1 AND condition2 OR Condition3  xTHEN
        Statement1;
       Statement2;
ELSE
       Statement1;
       Statement2;
END IF;
```

ELSIF statements

Syntax:

```
IF Condition1 THEN
  Statements1;
ESLIF Condition2 THEN
    Statement2;
ELSIF Condition3 THEN
   Statement3;
ELSE
    Statement n;
END IF;

SQL>DECLARE
 v_TotalEmps NUMBER;
BEGIN
     SELECT COUNT(*)
     INTO v_TotalEmps
     FROM emp;
     IF v_TotalEmps=0 THEN
      INSERT INTO temp_table values('There are no emps');
     ELSIF v_TotalEmps <5 THEN
```

```
    INSERT INTO temp_table values('There are few emps joined');
  ELSIF v_TotalEmps <10 THEN
 INSERT INTO temp_table values('There are little more emps joined');
ELSE
  INSERT INTO temp_table values('There are many emps joined');
  END IF;
END;
```

CASE Expressions

- It is similar to a switch statement in C.
- When a particular search condition evaluates to TRUE, the group of statements associated with this condition are executed.
- We can evaluate conditions using the normal conditional operators or SQL*Plus operators.

Case expression Syntax

```
CASE test_var
      WHEN value1 THEN
            Sequence_of_statements1;
       WHEN value2 THEN
            Sequence_of_statements2;
      ...
       WHEN valuen THEN
            Sequence_of_statementsn;
      [ELSE
            else_sequence;]
  END CASE;
```

- The reserved word "CASE" marks the beginning of the CASE statement.
- The selector is a value that determines, which "WHEN" clause should be executed.
- Each when clause contains as expression and or more executable statements associated with it.
- The "ELSE" clause is optional.
- Each "CASE" statement is marked with "END CASE".
- Where test_var is the variable or expression to be tested, value1 through valuen are the comparison values.
- If none of the values are equal, then else_sequence will be executed.

```
SQL>DECLARE
      v_Dname varchar2(20);
            v_Deptno number;
BEGIN
      SELECT deptno into v_Deptno
      FROM emp
```

```
        WHERE empno=&eno;
        CASE v_Deptno
        WHEN 10 THEN
         v_Dname:='ACCOUNTING';
        WHEN 20 THEN
         v_Dname:='RESEARCH';
        WHEN 30 THEN
         v_Dname:='SALES';
        WHEN 40 THEN
         v_Dname:='OPERATIONS';
        ELSE
          v_Dname:='UNKNOW';
        END CASE;
        DBMS_OUTPUT.PUT_LINE('Emp  dept name is:'||v_Dname);
END;
```

Labeled CASE statements

- A CASE statement can optionally be labeled, like a PL/SQL block.
- If a CASE statement is labeled, then the label can also appear after the END CASE clause.

```
SQL>DECLARE
    v_TestVar NUMBER:=10;
BEGIN
    <<Mycase>>
    CASE v_TestVar
    WHEN 10 THEN
    DBMS_OUTPUT.PUT_LINE('ACCOUNTING');
    WHEN 20 THEN
    DBMS_OUTPUT.PUT_LINE('RESEARCH');
    WHEN 30 THEN
    DBMS_OUTPUT.PUT_LINE('SALES');
    WHEN 40 THEN
    DBMS_OUTPUT.PUT_LINE('OPERATIONS');
    END CASE Mycase;
END;
```

```
SQL>Create  or replace procedure display(s varchar2)
Is
Begin
Dbms_output.put_line(s);
End;
```

Loops(Iterations in PL/SQL)

- Loops repeat a statement or sequence of statements multiple times.
- There are three loop types:

- Basic loop or Simple Loop
- FOR loop
- WHILE loop

Simple Loop

- The most basic kind of loop, simple loops, have the syntax

```
LOOP
    Sequence_of_statemnets;
END LOOP;
```

- The sequence_of_statements will be executed between the keywords "LOOP" and "END LOOP".
- To keep the loop in finite state the "EXIT" statement is used.
- We can add one with the EXIT statement which has the following syntax:

```
EXIT [WHEN condition];
```

EXIT Statement

- **"Exit"** Statement is used to terminate a LOOP.
- Once the loop is terminated, the control passes to the next statement after the "END LOOP".
- The "EXIT" statement should always be placed inside the Loop only.
- "EXIT" can be associate with a "WHEN" clause to allow conditional termination of the loop.
- The "EXIT" condition can be at the top of the loop or at the end of the loop as per logical convenience.
- Depending upon the circumstances we can make use of this LOOP as Pre - Tested loop or Post-Tested loop construct.
- The loop terminates its process when the conditional state is "TRUE".
- The statement EXIT WHEN condition is equivalent to

```
IF condition THEN
  EXIT;
END IF;
```

```
SQL> DECLARE
      v_counter Number:=1;
     BEGIN
         LOOP
       INSERT INTO temp_table
       VALUES(v_Counter,'Loop index');
       v_Counter:=v_Counter+1;
       EXIT WHEN v_Counter>50;
         END LOOP;
     END;

SQL>DECLARE
      a Number:=100;
     BEGIN
```

```
LOOP
   IF a=250 THEN
     EXIT;
   END IF;

    a:=a+25;

  display(a);
 END LOOP;
END;
```

Nested Loops:

- It is a situation where one loop is embedded into the other.
- The outer loop and the inner loop get associated with one another and execute simultaneously.
- The overall loop terminates is dictated by the outer loop's "EXIT WHEN " condition or "EXIT" condition.
- In nested loop's the outer loops condition evaluated as TRUE, always makes the inner loop to resume its process and the inner loop's termination actually makes the outer loop to update its process.

```
SQL> DECLARE
v_num NUMBER:=1;
BEGIN
LOOP
EXIT WHEN v_num>10;
LOOP
EXIT WHEN v_num>5;
display('Inner loop:'||v_num);
v_num:=v_num+1;
END LOOP;
display('Outer loop:'||v_num);
v_num:=v_num+1;
END LOOP;
END;
```

Nested loops and labels:

- Loops can be nested to multiple levels.
- All the loops can be nested into one another.
- Loops can be labeled as per the requirements.
- Label loops by placing the label before the word loop within the label delimiters.
- When the loop is labeled, the label name can be optionally included after the END LOOP statement for clarity.

```
DECLARE
```

```
v_num NUMBER:=1;
BEGIN
<<outerloop>>LOOP
<<innerloop>>LOOP
EXIT WHEN v_num>5;
display('Inner loop:'||v_num);
v_num:=v_num+1;
END LOOP innerloop;
display('Outer loop:'||v_num);
v_num:=v_num+1;
EXIT WHEN v_num>10;
END LOOP outerloop;
END;
```

GOTOs and Labels

- The "**GOTO**" statements allows us to branch to a label unconditionally.
- The label, which is enclosed within double angle brackets must precede an executable SQL statement or a PL/SQL block.
- When executed, the **GOTO** statements transfers control to the labeled statement or block.

Syntax

GOTO LabelName;

Scenario

```
SQL>CREATE TABLE product_master
(product_no varchar2(6) constraint
pk_product_pk  primary key,
Description varchar2(25),
unit_measure varchar2(10),
Qty_on_hand number(8),
reorder_lvl number(8),
cost_price number(10,2),
selling_price number(8,2));
```

```
SQL> DECLARE
 v_Qtyhand product_master.qty_on_hand%type;
 v_Relevel product_master.reorder_lvl%type;
 v_product_no product_master.product_no%type;
 BEGIN
   v_product_no:='&prodno';
   SELECT qty_on_hand,reorder_lvl INTO
     v_Qtyhand,v_Relevel
   FROM product_master
     WHERE product_no=v_product_no;
```

```
 IF v_Qtyhand<v_Relevel THEN
        GOTO updation;
  ELSE
        GOTO noupdation;
   END IF;
   <<updation>>
  UPDATE product_master SET
qty_on_hand=qty_on_hand+reorder_lvl
  WHERE product_no=v_product_no;
    RETURN;
   <<noupdation>>
   display('There are enogh product');
  RETURN;
 END;
```

Note:

- Hence to keeps proper meaning within the sequence "RETURN" should be used..

```
SQL>DECLARE
        v_Counter NUMBER:=1;
      BEGIN
        LOOP
        INSERT INTO temp_table
        VALUES(v_Counter,'Loop Count');
        v_Counter:=v_Counter+1;
        IF v_Counter>=50 THEN
         GOTO EndOfLoop;
        END IF;
              END LOOP;
        <<EndOfLoop>>
        INSERT INTO temp_table(Ind) VALUES('Done!');
      END;
```

Restrictions on GOTO

- It is illegal to branch into an inner block, loop , or IF statement.
- "GOTO" cannot navigate from the EXCEPTION selection to any other section of the PL/SQL block.
- **"GOTO "** cannot reference a LABLE in a nested block.
- "GOTO" cannot be executed outside an "IF" clause to LABLE inside "IF" clause.
- It is better to have a limited usage of "GOTO" in programming block.
- "GOTO" cannot navigate from the EXCEPTION selection to any other section of the PL/SQL block.

```
SQL>DECLARE
      n NUMBER:=10;
      BEGIN
          GOTO l_innerblock;
           IF  n>20 THEN
            <<l_innerblock>>
            INSERT INTO dept VALUES(50,'SW','HYD');
          END IF;
      END;

 SQL> BEGIN
         GOTO l_innerblock;
      BEGIN
      <<l_innerblock>>
      INSERT INTO dept VALUES(50,'SW','HYD');
      END;
      END;
```

WHILE Loops

- The WHILE LOOP statement includes a condition associates with a sequence of statement.
- If the condition evaluates to true, then the sequence of statements will be executed, and again control resumes at the beginning of the loop.
- IF the condition evaluates to false or NULL, then the loop is bypassed and the control passes to the next statement.

Syntax

```
WHILE condition LOOP
statement1;
statement2;
. . .
END LOOP;
```

Condition is evaluated at the beginning of each iteration.

```
SQL>DECLARE
      n NUMBER(3):=1;
      v VARCHAR2(100);
      BEGIN
            WHILE n<=10
            LOOP
                  v:=v||' '||n;
                  n:=n+1;
            END LOOP;
```

```
        display(v);
    END;

SQL>DECLARE
n NUMBER(5):=&n;
s NUMBER(5):=0;
r NUMBER(2):=0;
BEGIN
WHILE n!=0
lOOP
r:=mod(n,10);
s:=s+r;
n:=trunc(n/10);
END LOOP;
display('The sum of digit of given number is'||s);
END;
```

FOR LOOP

- It has the same general structure as the basic loop.
- "FOR LOOP" contains a control statement at the front of the lOOP keyword, to determine the number of iterations that PL/SQL performs.

Syntax:

```
FOR loop_counter IN [REVERSE] Lowerbound..Upperbound
 LOOP
Statement1;
Statement2;
END LOOP
```

Counter:

- It is an implicitly declared INTEGER whose value is automatically increased or decreased by 1 on each iteration of the LOOP until the upper bound or lower bound is reached.

Reverse:

- It is a keyword, and causes the counter to decrement with each iteration from the upper bound to the lower bound.
- The loop_counter need not be declared, as it is implicitly declared as an integer.
- The bounds of the loop are evaluated once.
- This determines the total number of iterations that loop_counter will take on the values ranging from low_bound to high_bound, incrementing by 1 each time until the loop is complete.

```
DECLARE
    v_FactNum NUMBER:=&No;
```

```
        v_Factorial NUMBER:=1;
BEGIN
        FOR v_Counter IN REVERSE 1..v_FactNum
        LOOP
                v_Factorial:=v_Factorial*v_Counter;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('The Factorial of '||v_FactNum||' is
                                        :'||v_Factorial);
END;

SQL>DECLARE
J NUMBER(2):=&J;
v VARCHAR2(100);
k NUMBER(3);
BEGIN
FOR  i in 1..10
loop
k:=j*i;
v:=v||j||'*'||i||'='||k||' ';
end loop;
display(v);
END;
```

Nested FOR loops

```
SQL>DECLARE
        n NUMBER:=&no;
        v VARCHAR2(100);
        BEGIN
        FOR i IN 1..n
        LOOP
        FOR k IN 1..i
        LOOP
        v:=v||'  '||k;
        END LOOP;
        display(v);
         v:=null;
        END LOOP;
        END;
```

CURSOR MANAGEMENT

- In order to process a SQL statements, Oracle will allocate an area of memory know as the context area.
- PL/SQL uses this context area to store and to execute the SQL statements.
- The information(rows) retrieved from database table, which is available in context area, is known 'Active Set'.
- A Cursor is a pointer, which works on active set i.e., which points one row at a time in the context area.
- A cursor is used to process multiple rows using PL/SQL.

Types of Cursors

- There are two types of cursors
  - Implicit cursors
  - Explicit cursors

Implicit cursors

- It is a CURSOR that is automatically declared by Oracle every time an SQL statement is executed.
- These are created or erased automatically.
- These are identified by SQL%<cursor attribute>
- Whenever you issue a SQL statement, the Oracle server opens an area of memory in which the command is parsed and executed. This area is called a cursor.
- When the executable part of a block issues a SQL statement, PL/SQL creates an implicit cursor, which PL/SQL manages automatically.
- A CURSOR is automatically associated with every DML statement.
- All UPDATE and DELETE statements have cursors that identify the set of rows that will be affected by the operations.
- During processing of an IMPLICIT cursor, oracle automatically performs the operations like "OPEN","FETCH", and "CLOSE" of the context area.

Explicit Cursor

- Explicit cursors are explicitly declared by the programmer.
- This cursor is declared within the PL/SQL block, and allows sequential process of each row of the returned data from database.

Cursor Attributes

- There are four attributes available in PL/SQL that can be applied to cursor.
- Cursor attributes are appended to a cursor name in a PL/SQL block, similar to %TYPE and %ROWTYPE.
- The attributes are
  - %FOUND
  - %NOTFOUND
  - %ISOPEN
  - %ROWCOUNT

%FOUND

- Boolean attribute that evaluates to TRUE If the most recent SQL statement affects one or more rows.

%NOTFOUND

- Boolean attribute that evaluates to TRUE if the most recent SQL statement does not affect any rows.

%ISOPEN

- Boolean attribute that evaluates to TRUE if the cursor is open else evaluates to FALSE.

%ROWCOUNT

- Number of rows affected by the most recent SQL statement (an integer value).

```
SQL>var rows_deleted varchar2(20)
SQL>DECLARE
   v_Empno emp.empno%TYPE := 7788;
   BEGIN
   DELETE FROM  emp
   WHERE  empno = v_Empno;
   :rows_deleted:= (SQL%ROWCOUNT ||' row deleted.');
   END;
SQL> DECLARE
   v_Job Emp.Job%TYPE:='&Job';
   BEGIN
    UPDATE emp
    SET sal=sal+1000
    WHERE job=v_Job;
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT||'Rows were Updated.');
    IF SQL%NOTFOUND THEN
      DBMS_OUTPUT.PUT_LINE('Data Not Found So, No updation.');
    END IF;
   END;
```

Implicit FOR Loops

```
          SQL>DECLARE
   FOR v_Empdata IN(SELECT ename,job,sal,comm
      FROM emp
      WHERE deptno=30)
   LOOP
   INSERT INTO bonus
    VALUES(v_Empdata.ename,v_Empdata.job,v_Empdata.sal,
     v_Empdata.comm);
   END LOOP;
  END;
```

Explicit Cursor

- An EXPLICT CURSOR is generated using a name with association of select statement in the DECLARE section of the PL/SQL block.

Advantages

- Explicit cursor provide more programmatic control for programmers.
- Explicit cursor are more efficient in implementation, hence easy to trap errors.

How cursor will work

The life cycle of the explicit cursor goes through four stages.
DECALRE
OPEN
FETCH
CLOSE

DECLARE:

- The CURSOR is declared in the declarative block and is provided with a name and a SELECT statement.

Syntax:-
CURSOR <cursor name is
Select_statement.

Note:-in <select statement> include most of the usual clauses, except INTO clause.(if use It there is no use).

- The cursor name can be any valid identifier.
- Any SELECT statements are legal, including joins and statements with the SET OPERATORS.

Illustration

```
>declare
vsal number:=2000;
cursor c1 is
select sal into vsal
from emp
where empno=7902;
begin
display('The vsal val is '||vsal);
end;
```

Opening a Cursor:

Syntax:

OPEN cursor_name;

- The opening of cursor executes the query and retrieves the information from the database and dumps it into the context area to foam active set.
- Cursor can be opened only in the EXECUTION OR EXCEPTION section of the PL/SQL block.
- The active set pointer is set to the first row.
- Once a cursor has been opened, it cannot be reopened unless it is first closed.

Fetching from a Cursor

- Fetch the record from CONTEXT AREA into cursor variable.
- This fetches 1 row at a time into the cursor variable from the active set.
- The INTO clause for the query is part of the FETCH statement.

The FETCH statement has two forms:

Syntax:

1) FETCH cursor_name INTO list_of_variables;
2) FETCH cursor_name INTO PL/SQL_record;

- The list_of_variable is a comma-separated list of previously declared PL/SQL variable, and PL/SQL_record is a previously declared PL/SQL record.

Illustration:

FETCH Empcursor INTO v_emprecord;

Closing a Cursor

- This tells PL/SQL engine that the program is finished with the cursor, and the resources associated with it can be freed.
- These resource include the storage used to hold the active set, as well as any temporary space used for determining the active set.
- The active set can be re-established several times.

Syntax

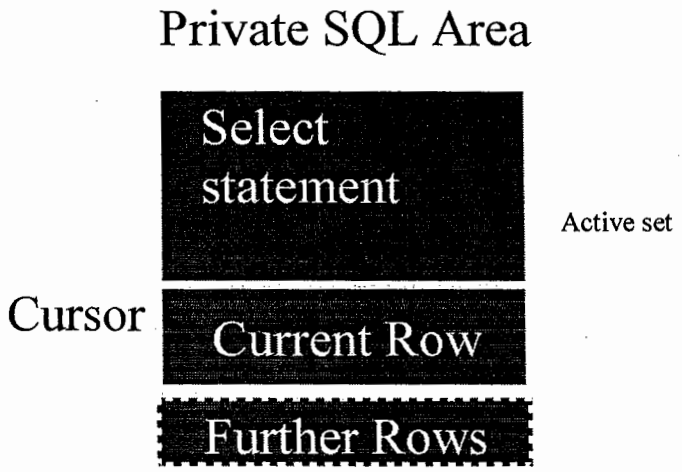
CLOSE <cursor name>;

Cursor Attributes

- There are four attributes available in PL/SQL that can be applied to cursors. cursor attributes are appended to a cursor name in PL/SQL block and then used in expressions.
- The attributes are
  1) <cursor name>%Found
  2) <cursor name>%NotFound
  3) <cursor name>%ISOPEN
  4) <cursor name>%Rowcount

Active Set:

The set of rows returned by a multiple-row query is called the Active Set.




```
SQL>declare
cursor c1 is
select empno,ename,sal
from emp
where deptno=20;
vempno emp.empno%type;
vename emp.ename%type;
vsal emp.sal%type;
begin
open c1;
loop
fetch c1 into vempno,vename,vsal;
exit when c1%notfound;
if vsal between 0 and 1000 then
  vsal:=vsal+vsal*0.15;
elsif vsal between 1001 and 2000 then
   vsal:=vsal+vsal*0.25;
else
   vsal:=vsal+vsal*0.35;
end if;
update emp set sal=vsal
where empno=vempno;
display(rpad(vename,8)||'  '||vsal);
end loop;
close c1;
end;
```

```
SQL>declare
cursor jc is
select empno,ename,job,sal
from emp;
i emp%rowtype;
begin
open jc;
loop
fetch jc into i.empno,i.ename,i.job,i.sal;
exit when jc%notfound;
if i.job='CLERK' then
   i.sal:=i.sal+100;
elsif i.job='SALESMAN' then
   i.sal:=i.sal+200;
else
   i.sal:=i.sal+300;
end if;
update emp set sal=i.sal
where empno=i.empno;
display(rpad(i.ename,8)||'  '||rpad(i.job,10)||'  '||i.sal);
end loop;
close jc;
end;
```

```
>declare
cursor cc is
select empno,ename,comm
from emp;
i cc%rowtype;
begin
open cc;
loop
fetch cc into i;
exit when cc%notfound;
if i.comm is null then
    i.comm:=300;
elsif i.comm=0 then
   i.comm:=250;
else
  i.comm:=i.comm+i.comm*0.35;
end if;
update emp set comm=i.comm
where empno=i.empno;
display(rpad(i.ename,8)||'  '||i.comm);
```

```
end loop;
close cc;
end;
```

```
SQL>declare
cursor cpf is
select empno,ename,sal basic,sal*0.45 hra,sal*0.35 da,sal*0.15 pf,deptno
from emp
where sal>4000;
i cpf%rowtype;
vgross number;
begin
open cpf;
loop
fetch cpf into i;
if cpf%found then
 vgross:=i.basic+i.hra+i.da-i.pf;
display(i.empno||' '||rpad(i.ename,8)||' '||rpad(i.basic,6)||' '||rpad(i.hra,5)||' '||
rpad(i.da,5)||' '||rpad(i.pf,5)||' '||
rpad(vgross,5)||' '||i.deptno);
else
  exit;
end if;
end loop;
display('No of emps eligible for pf is'||cpf%rowcount);
close cpf;
end;
```

```
> insert into utab
values(upper('&uid'),
Translate(  upper('&password'),
'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'1234567890!@#$%^&*()-=_+;,.')
);

SQL>declare
cursor primary_cur is
select 'X'
from utab
where userid=upper('&uid') and
password=Translate(upper('&password'),
'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'1234567890!@#$%^&*()-=_+;,.');

dummy_var char;
```

```
begin
open primary_cur;
fetch primary_cur into dummy_var;
if primary_cur%found then
  display('The userid and password corr');
else
  display('Unknow userid and password');
end if;
close primary_cur;
end;
```

Cursor with For loops

- PL/SQL provides a simple type of loop, which implicitly handles the cursor processing.
- Within the loop each record in the Active set is retrieved and used.
- Each loop iteration advances the cursor pointer by one record in the Active set.
- The loop works on the range oriented operational logic.
- The loop is very useful when traveling the entire data in the database table.
- It is more Dynamic in operation than the Simple loop.

Syntax

```
For <variable> IN <cursor name>
LOOP
        <Ex Stmt>;
END LOOP;
```

Advantages

No need to

→Declare INDEX variable explicitly.
→OPEN a cursor .
→FETCH the rows from cursor.
→Terminate the loop explicitly.
→CLOSE the cursor.

```
SQL>declare
cursor cpf is
select empno,ename,sal basic,sal*0.45 hra,sal*0.35 da,sal*0.15 pf,deptno
from emp
where sal>4000;
vgross number;
cnt number;
begin
for i in cpf
loop
vgross:=i.basic+i.hra+i.da-i.pf;
```

```
display(i.empno||' '||rpad(i.ename,8)||' '||rpad(i.basic,6)||' '||rpad(i.hra,5)||' '||
rpad(i.da,5)||' '||rpad(i.pf,5)||' '||rpad(vgross,5)||' '||i.deptno);
cnt:=cpf%rowcount;
end loop;
display('No of emps eligible for pf is'||cnt);
end;
```

parametric Cursor

- The cursor defined with parameter is called parametric Cursor.
- Cursor parameters can be assigned with default values.
- The Mode of cursor parameters can be only "IN" Mode.

Syntax

```
CURSOR Cursorname(ParameterName Datatype,...)
IS
SELECT statement;
```

Illustration

```
SQL> DECLARE
    CURSOR Empcursor
                    (Pdeptno NUMBER,
                     Pjob VARCHAR2) IS
    SELECT empno, ename
    FROM emp
    WHERE deptno = Pdeptno AND job = Pjob;
```

Methods of Opening a Parametric Cursor

```
1.Open Empcursor(30,'SALESMAN');
2.DECLARE
  v_empdeptno Emp.deptno%TYPE:=&Givedeptno;
  v_empjob Emp.job%TYPE:=&GiveJob;
  BEGIN
  OPEN  Empcursor(v_empdeptno,v_empjob);
3. DECLARE
    CURSOR Empcursor
                    (Pdeptno NUMBER,
                    Pjob VARCHAR2) IS
    SELECT empno, ename
    FROM emp
    WHERE deptno = Pdeptno AND job = Pjob;
    BEGIN
     FOR Emprecord IN Empcursor(20,'CLERK')
    LOOP
```

Import Points

- Unless you want to accept default values, each formal parameter in the cursor declaration must have a corresponding actual parameter in the OPEN statement.

- A parametric cursor can be opened and closed explicitly several times in a PL/SQL Block.
- A parametric cursor returns a different Active Set on each occasion.
- The concept is more useful when the same Cursor is Referenced repeatedly.

```
SQL>declare
cursor dc is
select deptno
from dept;
cursor ec(pdno in dept.deptno%type)
is
select empno,ename,sal basic,sal*0.45 hra,
sal*0.35 da,sal*0.15 pf,deptno
from emp
where deptno=pdno;
vdno dept.deptno%type;
i ec%rowtype;
vgross number;
begin
open dc;
loop
fetch dc into vdno;
exit when dc%notfound;
open ec(vdno);
loop
fetch ec into i;
exit when ec%notfound;
vgross:=i.basic+i.hra+i.da-i.pf;
display(i.empno||' '||rpad(i.ename,8)||' '||rpad(i.basic,5)||' '||rpad(i.hra,5)||'
'||
rpad(i.da,5)||' '||rpad(i.pf,5)||' '||
rpad(vgross,5)||' '||i.deptno);
end loop;
if ec%rowcount>0 then
display('--------------------------------------------------');
end if;
close ec;
end loop;
end;
```

```
SQL> declare
 cursor dc is
 select unique deptno
```

```
from emp;
cursor ec(pdno in emp.deptno%type)
is
select empno,ename,sal basic,sal*0.45 hra,
sal*0.35 da,sal*0.15 pf,deptno
from emp
where deptno=pdno;
vdno emp.deptno%type;
i ec%rowtype;
vgross number;
begin
delete from emp_report;
open dc;
loop
fetch dc into vdno;
exit when dc%notfound;
open ec(vdno);
loop
fetch ec into i;
exit when ec%notfound;
vgross:=i.basic+i.hra+i.da-i.pf;
insert into emp_report values(i.empno,i.ename,i.basic,i.hra,i.da,i.pf,vgross,
i.deptno);
end loop;
if ec%rowcount>0 then
insert into emp_report(ecode) values(null);
end if;
close ec;
end loop;
end;
```

FOR UPDATE Clause

- FOR UPDATE clause explicitly locks the records Stored in the private work area.
- The FOR UPDATE clause in the cursor query is used to LOCK the affected rows while the cursor is opened.
- You need not need provide an explicit COMMIT command to release the lock acquired by using the FOR UPDATE clause.

Using WHERE CURRENT OF clause

- WHERE CURRENT OF clause to refer the current record, fetched from the explicit cursor.
- we need to suffix the name of the explicit cursor with the CURRENT OF clause to refer to current record.

- In order to use the WHER CURRENT OF clause, you need to lock the record fetched from the cursor.

```
SQL>declare
cursor lc is
select ename,sal,deptno
from emp
for update;
i lc%rowtype;
begin
display('The emp det are');
display('EmpName'||'  '||'Salary'||'  '||'DeptNum');
display('-------'||'  '||'------'||'  '||'-------');
open lc;
loop
fetch lc into i;
exit when lc%notfound;
display(rpad(i.ename,8)||'  '||rpad(i.sal,5)||'  '||i.deptno);
if i.deptno=50 then
  delete from emp
  where current of lc;
end if;
end loop;
close lc;
display('After delete the 50th dept emps');
open lc;
loop
fetch lc into i;
exit when lc%notfound;
display(rpad(i.ename,8)||'  '||rpad(i.sal,5)||'  '||i.deptno);
end loop;
close lc;
end;
```

Cursors with sub Queries

- A CURSOR can be constructed upon the result provided through a SUBQUERY.
- The sub query can be an ordinary one or a correlated sub query.

```
SQL>DECLARE
CURSOR  cs is
SELECT d.deptno,d.dname,d.loc,v.noe
FROM dept d,(SELECT deptno,count(*) noe
```

```
            FROM emp
            group by deptno) v
    WHERE  v.noe>3 AND
    d.deptno=v.deptno;
    BEGIN
    FOR i in cs
    LOOP
    display(i.deptno||' '||i.dname||'  '||i.loc||' '||i.noe);
    END LOOP;
    END;
```

Dynamic Cursor Ref Cursor or Cursor Variable

- Explicit cursor is Static Cursor.
- Explicit cursor referrers always one work area associated with cursor.
- Dynamic cursor is Ref Cursor, It referrers different work area in memory.
- Cursor variables are analogous to PL/SQL variables, which can hold different values at runtime.
- It is used to declare a cursor with out select statement.
- A Ref Cursor can be reused if it is declared in package.
- A Ref Cursor support to return more than 1 row from sub program.
- Deferent select statement can be associated with cursor variable at run time.

Types of cursor variable

- There are two types of Ref Cursor.

weak cursor variable or Weak Ref Cursor

- If we do not include a RETURN clause in the Cursor, then it is a weak REF Cursor.
- Cursor variables declared from weak REF cursors can be associated with any query at runtime.

Syntax:

step1:

Type <type name> is ref cursor;

step2:

<cursor variable name> <type name>;

step3:

open<cursor variable name>
for(select statement);
fetch
exit
close;

strong cursor variable

- A REF CURSOR with a RETUN clause define a "Strong" REF CURSOR.

Syntax:

step1:

```
Type <type name> is ref cursor
RETURN record_type;
```

step2:

```
<cursor variable name> <type name>;
```

step3:

```
open<cursor variable name>
for(select statement);
fetch
exit
close;
SQL>DECLARE
vempno emp.empno%type;
vename emp.ename%type;
vsal emp.sal%type;
vjob emp.job%type;
vdeptno emp.deptno%type;
TYPE ref_c IS REF CURSOR;
c1 ref_c;  --it is data type of ref_c
begin
vdeptno:=&dno;
IF vdeptno=10 THEN
  OPEN c1 FOR SELECT empno,ename,sal
  FROM EMP WHERE deptno=vdeptno;
ELSIF vdeptno=20 THEN
  OPEN c1 FOR SELECT ename,job,sal
  FROM emp WHERE deptno=vdeptno;
ELSE
   OPEN c1 FOR SELECT ename,sal,deptno
        FROM emp
   WHERE deptno=vdeptno;
END IF;
IF vdeptno=10 THEN
LOOP
       FETCH c1 INTO vempno,vename,vsal;
       EXIT WHEN C1%NOTFOUND;
       display(vempno||' '||vename||' '||vsal);
END LOOP;
CLOSE c1;
```

```
ELSIF vdeptno=20 THEN
 LOOP
       FETCH c1 INTO vename,vjob,vsal;
       EXIT WHEN C1%NOTFOUND;
       display(vename||' '||vjob||' '||vsal);
 END LOOP;
 CLOSE c1;
ELSE
   LOOP
       FETCH c1 INTO vename,vsal,vdeptno;
       EXIT WHEN c1%NOTFOUND;
            display(vename||' '||vsal||' '||vdeptno);
    END LOOP;
    END IF;
END;
```

Exception Handling

- An Exception in PL /SQL Block is Raised during Execution of a Block.
- Once an EXCEPTION arises it terminates the main body of actions performed by the PL/SQL block.
- A Block always terminates when PL / SQL Raises an Exception.
- WE can specify an exception handler to perform
- Final action.

Note :

- If the exception is raised in the executable section of the Block and there is no corresponding Handler, the PL / SQL Block Terminates with Failure.
- If the exception is handled then PL / SQL Block terminates successfully.

Exception Handling

- An Exception is raised when an error occurs.
- In case of an error, normal exception stops and the control is immediately transferred to the exception handling part of the PL/SQL block.

Trapping Exceptions

- The exception handling section consists of handlers for all the exceptions.
- An exception handler contains the code that is executed when the error associated with the exception occurs, and the exception raised.
- Each exception handler consists of a WHEN clause which specifies an EXCEPTION that has to be handled.

Syntax

```
Exception
When <exception1> [or exception2...]  Then
SQL statement1
SQL statement2
When <exception3> [or exception4...]  Then
SQL statement1
SQL statement2
When Others  Then
SQL statement1
SQL statement2
End;
```

The Others Exception Handler

- The OTHERS exception is an optional EXCEPTION handling clause that traps unspecified exceptions.
- The OTHERS should always be the last handler in the block.
- It is good programming practice to have an OTHERS handler at the top level of your program to ensure that no errors go undetected.

Exception Guidelines

- Begin the EXCEPTION handling section of the block with the keywords EXCEPTION.

- Define several EXCEPTION HANDLERS ,each with its own set of action for the block.
- Avoiding unhandled exceptions, this can be done via an OTHERS handler at the topmost level program.
- When an EXCEPTION occurs PL/SQL processes only one handle before leaving the block.

Predefined Exception

- Oracle has predefined several exceptions that correspond to the most common oracle errors.
- A predefined oracle server error is trapped by referencing its standard name within the corresponding exception handling at runtime.
- Each Predefined Exception begins With ORA followed by a number.
- As soon As the Exception occurs oracle implicitly raise the exception and jumps into the EXCEPTION handle block, if proper EXCEPTION handle is found manages the specified steps.
- In predefined oracle server exceptions only one exception is raised and handled at any time.

Types of Exception

Pre defined(System defined)

- Predefined exception are raised automatically by the system during run time.
- Predefined exception are already available in the program it is not necessary to declare them in the declarative section like user_defined exception.

Predefined Exception List

Exception Name	Error No	Description
ORA-0001	DUP_VAL_ON_INDEX	Unique constraint violated.
ORA-1001	INVALID_CURSOR	Illegal Cursor operation.
ORA-1403	NO_DATA_FOUND	No data found.
ORA-1422	TOO_MANY_ROWS	A SELECT...INTO statement matches more than one row.
ORA-1722	INVALID_NUMBER	Conversion to a number failed For example,'krishna street 1' not valid.
ORA-6502	VALUE_ERROR	Truncation, arithmetic, or conversion error.
ORA-01476	ZERO_DIVIDE	Divisor is equal to zero
ORA-06511	CURSOR_ALREADY_OPEN	This exception is raised when We try to open a cursor which is already opened.
ORA-01017	LOGIN_DENIED	This exception is raised when we try to enter oracle using invalid username/password.

```
>DECLARE
      v_empno emp.empno%TYPE:=&empno;
      v_ename emp.ename%TYPE;
      v_job emp.job%TYPE;
BEGIN
      SELECT ename,job INTO v_ename,v_job
      FROM emp
      WHERE empno=v_empno;
      DBMS_OUTPUT.PUT_LINE('The empno detail are '||v_ename||'  '||v_job);
      EXCEPTION
      WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('The empno is not found.');
END;

SQL>DECLARE
       v_accno kcb_acc_tab.accno%TYPE:=&accno;
       v_name kcb_acc_tab.name%TYPE:='&name';
       v_bal kcb_acc_tab.bal%TYPE:=&bal;
     BEGIN
            INSERT INTO kcb_acc_tab(accno,name,bal)
            VALUES(v_accno,v_name,v_bal);
             DBMS_OUTPUT.PUT_LINE('Account detailes are inserted
      successfully.');
      EXCEPTION
              WHEN DUP_VAL_ON_INDEX THEN
              DBMS_OUTPUT.PUT_LINE('accno already exists');
     END;
SQL> DECLARE
       v_empno emp.empno%TYPE;
       v_ename emp.ename%TYPE;
       v_deptno emp.deptno%TYPE;

     BEGIN
        SELECT empno,ename,deptno INTO v_empno,v_ename,v_deptno
        FROM emp
        WHERE empno=7788 AND ename='SCOTT';
        DBMS_OUTPUT.PUT_LINE('The scott works in department
      number:'||v_deptno);
       Select empno,ename,deptno into v_empno,v_ename,v_deptno
        FROM emp
       Where deptno=10;
       DBMS_OUTPUT.PUT_LINE('The Employee number:'||v_empno);
       DBMS_OUTPUT.PUT_LINE('The Employee name:'||v_ename);
```

```
Exception
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error:There is no such empno or ename or
        deptno');
WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Error:More than one Employee works in
        department number 10');
WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred while processing the
program');
END;
```

User-defined Exception:

- A user-defined exception is an error that is defined by the program
- The developers to handle the business situations define user-defined exceptions during the execution of the PL/SQL block.
- User defined exceptions are defined by the following two techniques:

→Using a flow control statement RAISE

Raise statement transfer the control of the block from the execution part of the PL/SQL block to the exception handing part of the block.

Step

1. **Declare Exception**
2. **Raise in Executable section explicitly using**
   RAISE <Exception_handler_name>;
3. **Handle the raised exception**

→Using a packaged procedure

RAISE_APPLICATION_ERROR(error number,'Error message');

RAISE_APPLICATION_ERROR

- This Built-in procedure is used to create your own error message, which can be more descriptive than named exceptions.
- It is used to communicate a predefined exception interactively by returning a non standard error code and error message.
- Using this procedure we can report error to application and avoid returning unhandled exception.

Note:

- Error number must exists between -20,000 and -20,999

- Error_message is the text associate with this error, and keep_errors is Boolean value.
- The error_message parameter must be less than 512 characters.

SQLCODE FUNCTION

- It returns the current error code.
- For a user defined exception it returns 1, +100 NO_DATA_FOUND exception.

SQLERRM

- It returns the current error message text.
- SQLERRM returns the message associated with the error number.
- The maximum length of a message returned by the SQLERRM functions is 512 bytes.

```
SQL> DECLARE
  i emp%rowtype;
  BEGIN
 i.empno:=&eno;
  SELECT ename,sal  into i.ename,i.sal
  from emp
  where empno=i.empno;
  IF i.sal<2000 THEN
Raise_application_error(-20345,'The emp sal is less than 2000 so no updation');
    i.sal:=i.sal+i.sal*0.35;
  UPDATE emp set sal=i.sal
  WHERE empno=i.empno;
  ELSE
    i.sal:=i.sal+i.sal*0.35;
  UPDATE emp set sal=i.sal
  WHERE empno=i.empno;
  display('The emp det are '||i.ename||'  '||i.sal);
  END IF;
  END;


QL>DECLARE
salary_missing EXCEPTION;
i emp%rowtype;
BEGIN
i.empno:=&eno;
SELECT ename,sal into i.ename,i.sal
FROM emp
WHERE empno=i.empno;
IF i.sal IS NULL  THEN
```

```
  RAISE salary_missing;
ELSE
  i.sal:=i.sal+i.sal*0.25;
UPDATE emp SET sal=i.sal
WHERE empno=i.empno;
display('The emp det are '||i.ename||'  '||i.sal);
END IF;
EXCEPTION
WHEN no_data_found THEN
   display(i.empno||' is not exists');
display(SQLCODE||'  '||SQLERRM);
WHEN salary_missing THEN
display('The emp is not having any salary so give salary as 3000');
UPDATE emp SET sal=i.sal
WHERE empno=i.empno;
display(SQLCODE||'  '||SQLERRM);
WHEN others then
  display('The SUHE');
END;
```

Trapping Non-Predefined oracle server errors

- We can associate a named exception with a particular oracle error.
- The Non-predefined oracle server error is trapped by declaring it first or by using the OTHERS exception handle.
- The declare EXCEPTION is RAISED implicitly by the oracle server.
- The PL/SQL PRAGMA EXCEPTION_INIT() can be used for associating EXCEPTION name with an oracle error number.
- The PRAGMA EXCEPTION_INIT() tells the PL/SQL engine completely to associate an EXCEPTION name with an oracle error number.
- The PRAGMA EXCEPTION_INIT() allows programmer to refer to any internal EXCEPTION by the name and associate that to specific handles.
- Pragma is a directive of compiler which tells compiler to associate error no with user declared exception at compile time.

Steps

1.Declare Exception

2. Associate Exception with Oracle error No.
Using
Pragma
exception_init(exception_name,oracle_error_number);

3. Handle the raised exception

- Exeption_name is the name of an exception declare prior to the pragma.

- Oracle _erroe_number is the desied error code to be associate with this named exception.

```
SQL> DECLARE
 pk_vio EXCEPTION;
 PRAGMA EXCEPTION_INIT(pk_vio,-00001);
 BEGIN
 insert into emp(empno,ename,sal,deptno)
values(&eno,'&ename',&sal,&deptno);
 EXCEPTION
 when pk_vio then
 dbms_output.put_line('Duplicate empno is not  allowed here');
 END;
```

Understanding PL/SQL Collections

- PL/SQL, similar to other programming languages such as C,C++, allows using arrays and records.
- PL/SQL has two composite types :records and collections.

PL/SQL Records

- **A PL/SQL Record** is allows you to treat several variables as a unit.
- **PL/SQL Record** are similar to structure in C.
- When a "RECORD TYPE" of fields are declared then they can be manipulated as a Unit through out the Application.

Note:

- In the composite data type **RECORD, we can specify the data type of the column.**
- Each **RECORD** defined can have as many Fields as necessary.
- Fields declared as 'not null' must be initialized in the declaration part.
- A record can be initialized in its declaration part unlike PL/SQL tables, which doesn't allow initialization in the declaration part.
- The **DEFAULT** key word can also be used when defining fields.
- A **RECORD** can be the component of another **RECORD**.

Syntax

```
TYPE type_name IS RECORD
(Element1 <data type>,
 Element2 <data type>,
Element3 <data type>,
 Element4 <data type>,
Elementn <data type>);
```

- field_declaration syntax is

```
Elementname{ Elementdatatype(size) OR
             Recordvariable%TYPE OR
             Table.column%TYPE OR
             Table%ROWTYPE}
        [[NOT NULL] {:= OR DEFAULT} expr]
```

Defining PL/SQL Record

- To define a user defined PL/SQL data type
  - Type name→is the name of the Record type.
  - Element name→It is the name of the field within the Record.
  - Element Data Type→It is the data type of the Element.
  - Expr →It is the Field type OR an Initial Value.

→The **NOT NULL** constraint prevents the Assigning of NULL's to those Fields.
→Element declaration are like variable declaration each Element has a unique name and A specific data type.
→We must create the data type first and then declare an identifier using the declared data type.

Illustration

```
SQL>DECLARE
  TYPE erec IS RECORD
(veno NUMBER(4),
 vname emp.ename%TYPE,
 basic emp.sal%TYPE,
  i dept%ROWTYPE,
 vgross number(16,2));
 e erec;
BEGIN
   e.veno:=&employe;
   select ename,sal,dept.deptno,dname into
   e.vname,e.basic,e.i.deptno,e.i.dname
   from emp,dept
   where emp.deptno=dept.deptno and empno=e.veno;
   e.vgross:=e.basic+e.basic*.25+e.basic*.35-e.basic*.12;
display(e.veno||' '||e.basic||'  '||e.i.deptno||'  '||e.i.dname||' '||e.vgross);
END;
```

PL/SQL Tables

- PL/SQL tables are temporary array-like objects used in a PL/SQL block.
- They are modeled as database tables, but are not same.
- PL/SQL tables are very dynamic in operation, giving the simulation to pointers in 'C' language.
- PL/SQL TABLES use a "PRIMARY KEY" to give array like access to rows.
- PL/SQL table can be declared in the declarative part of any block. Subprogram or package.
- It is similar to an array in Third generation language.
- PL/SQL table should contain two components.
  - A "PRIMARY KEY" of data type BINARY_INTEGER, that indexes the PL/SQL table.
  - A column of a scalar or Record data type which stores the PL/SQL table elements.
- PL/SQL tables can increase in size dynamically as they are unconstrainted.
- A PL/SQL TABLE must be declare in two steps.
-

1) First we define a table type
2)We declare a variable of PL/SQL table as data type.

Syntax

```
TYPE  <Type name> IS TABLE OF
{column type OR Table.column%TYPE OR PL/SQL RECORD}
 INDEX BY BINARY_INTEGER;
```

- The number of rows in PL/SQL table can increase dynamically, hence a PL/SQL table can grow as new rows are added.

Referencing PL/SQL Table

- PLSQL_Tablename(Primary_key_value);
- PRIMARY_KEY_VALUE belongs to type BINARY_INTEGER.
- The primary key value can be negative indexing need not start with 1.
- The method make PL/SQL tables easier to use are.

COUNT

- Returns the number of elements that a PL/SQL table currently contais.

```
SQL>DECLARE
                TYPE name IS TABLE OF
                VARCHAR2(50) INDEX BY BINARY_INTEGER;
                n name;
BEGIN
                n(0):='Siva';
                n(1):='Rama';
                n(2):='Krishna';
                display('The full name is '||n(0)||' '||n(1)||'  '||n(2));
END;
```

Note: In oracle 11g Negative index are not allowed.

Trace table:

```
---------------
ename varchar2(20),
 usal number(7,2),
 dou timestamp
```

```
SQL>DECLARE
                TYPE eno IS TABLE OF
                emp.empno%TYPE INDEX BY BINARY_INTEGER;
                TYPE name IS TABLE OF
                emp.ename%TYPE  INDEX BY BINARY_INTEGER;
                TYPE pays IS TABLE OF
                emp.sal%TYPE  INDEX BY BINARY_INTEGER;
                e eno;
                n name;
                p pays;
                ctl number:=1;
BEGIN
FOR i IN(SELECT empno,ename,sal FROM emp)
LOOP
e(ctl):=i.empno;
n(ctl):=i.ename;
p(ctl):=i.sal;
ctl:=ctl+1;
END LOOP;
for MyIndex IN 1..e.count
```

```
LOOP
IF p(MyIndex) BETWEEN 0 AND 1000 THEN
p(MyIndex):=p(MyIndex)+100;
ELSIF p(MyIndex) BETWEEN 1001 AND 2000 THEN
p(MyIndex):=p(MyIndex)+200;
ELSE
p(MyIndex):=p(MyIndex)+300;
END IF;
UPDATE emp SET sal=p(MyIndex)
WHERE empno=e(MyIndex);
INSERT INTO trace VALUES(n(MyIndex),p(MyIndex),sysdate);
Display(n(MyIndex)||' '||p(MyIndex));
END LOOP;
END;
```

Nested type Collection

```
TYPE type_name IS RECORD
(
 EmpRecord Emp%ROWTYPE,
 DeptRecord Dept%ROWTYPE,
 Element  RECORD);

SQL>DECLARE
TYPE pf_info is RECORD
                    (pfno NUMBER(4),
                    amount NUMBER(14,2));
TYPE emp_rec IS RECORD
                    (eid NUMBER(4),
                     name VARCHAR2(20),
                     basic NUMBER(12,2),
                     pf pf_info);
TYPE etab IS TABLE OF emp_rec
INDEX BY BINARY_INTEGER;
ctr NUMBER(3):=1;
e etab;
BEGIN
                    FOR i IN(SELECT empno,ename,sal basic,
                        sal*.12 pamt FROM  emp
                        WHERE  sal>2000)
      LOOP
                    e(ctr).eid:=i.empno;
                    e(ctr).name:=i.ename;
                    e(ctr).basic:=i.basic;
                    e(ctr).pf.pfno:=i.empno+5;
                    e(ctr).pf.amount:=i.pamt;
                    ctr:=ctr+I;
END LOOP;
```

```
                display('employee detailes are:');
                FOR  MyIndex in 1..e.count
                loop
                   display(e(MyIndex).eid||'  ''||e(MyIndex).name||'
                         '||e(MyIndex).basic||'  '||e(MyIndex).pf.pfno||'
                         '||e(MyIndex).pf.amount);
                END LOOP;
END;
```

RETURNING Clause

- This clause is valid at the end of any DML statement.
- It is used to get information about the row or rows just processed.

Syntax

→ REURNING → expr → INTO → variable

- expr is a valid PL/SQL or SQL expression, which can include columns or pseudocolumns of the current table.
- Variable is the PL/SQL variable into which the result will be stored.

BULK COLLECT clause

- It used to collect more than 1 row at a time.
- It is used as part of the SELECT INTO,FETCH INTO , or RETURNING INTO clause and will retrieve rows from the query into the indicated collections.

BULK COLLECT clause with SELECT

Illustrations

```
SQL> DECLARE
                TYPE name IS TABLE OF
                emp.ename%TYPE  INDEX BY BINARY_INTEGER;
                TYPE pays IS TABLE OF
                emp.sal%TYPE  INDEX BY BINARY_INTEGER;
                n name;
                p pays;
        BEGIN
        SELECT  ename,sal BULK COLLECT  INTO n,p
         FROM emp;
        FOR  i IN 1..n.COUNT
         LOOP
                Display(RPAD(n(i),9,' ')||' '||p(i));
         END LOOP;
        END;
```

BULK COLLECT clause DELETE

Table delete_log

```
ename varchar2(20),
 basic number(7,2),
 dod timestamp
SQL> DECLARE
```

```
                TYPE name IS TABLE OF
                emp.ename%TYPE  INDEX BY BINARY_INTEGER;
                TYPE pays IS TABLE OF
                emp.sal%TYPE  INDEX BY BINARY_INTEGER;
                n name;
                p pays;
BEGIN
DELETE FROM  emp
WHERE  deptno=30
RETURNING  ename,sal BULK COLLECT INTO  n,p;
FOR  i IN 1..n.COUNT
 LOOP
                Display(RPAD(n(i),9,' ')||' '||p(i));
IF p(i)>2500 THEN
INSERT INTO delete_log VALUES(n(i),p(i),sysdate);
END IF;
 END LOOP;
END;
```

BULK COLLECT clause UPDATE

```
SQL> DECLARE
                TYPE name IS TABLE OF
                emp.ename%TYPE  INDEX BY BINARY_INTEGER;
                TYPE pays IS TABLE OF
                emp.sal%TYPE  INDEX BY BINARY_INTEGER;
                n name;
                p pays;
BEGIN
UPDATE emp SET sal=sal+sal*0.35
WHERE deptno=20
RETURNING  ename,sal BULK COLLECT INTO  n,p;
FOR  i IN 1..n.COUNT
 LOOP
                Display(RPAD(n(i),9,' ')||' '||p(i));
IF p(i)>2500 THEN
INSERT INTO trace VALUES(n(i),p(i),sysdate);
END IF;
 END LOOP;
END;
```

BULK COLLECT clause CURSOR

```
SQL> DECLARE
                TYPE name IS TABLE OF
                emp.ename%TYPE  INDEX BY BINARY_INTEGER;
                TYPE pays IS TABLE OF
                emp.sal%TYPE  INDEX BY BINARY_INTEGER;
```

```
                CURSOR  c_bulkcollect IS
                SELECT ename,sal FROM emp;
                n name;
                p pays;
BEGIN
OPEN c_bulkcollect;
FETCH c_bulkcollect BULK COLLECT INTO  n,p;
FOR  i IN 1..n.COUNT
 LOOP
                Display(RPAD(n(i),9,' ')||' '||p(i));
 END LOOP;
END;
```

Differences between Anonymous and Named PL/SQL block.

Anonymous PL/SQL Block	Named PL/SQL Block
1.This is an unnamed PL/SQL block	1.This is a named PL/SQL block
2.These are stored in operating system	2.These are stored in oracle database.
3.There is no security	3.Oracle is providing security
4.There is no information hiding facility	4.There is a hiding facility.
5. It requires every time compilation process.	5.It is compiled once and ready to execute
6.Granting privileges on this is not possible.	6.Granting privileges is possible.
7.These are not accessible to other program.	7. These are accessible by other oracle tools like SQL*PLUS ,Oracle Forms ,Reports

Subprograms In PL/SQL

- Subprograms are used to provide modularity and encapsulate a sequence of statements.
- once subprograms are built and validated, they can be used in a number of applications.
- Subprogram also provide abstraction.
- Subprograms are named PL/SQL blocks that can accepts parameters.
- A Subprogram can also have a declarative part, an executable part and an exception handling part.

Important features of subprogram

Modularity

- Subprograms allow us to break a program into manageable, well-defined logical modules.

Reusability

- Subprograms once executed can be used in any number of application.

Maintainability

- Subprograms can simplify maintenance, because if a subprogram is affected, only its definition changes.

PROCEDURES

- A procedure is a subprogram that performs a specific action.
- A procedure may or may not return value.

Syntax

```
CREATE OR REPLACE  PROCEDURE  procedureName(parname1[MODE] parType,..)
IS
  [Local variable declaration;]
BEGIN
  Executable Stage
EXCEPTION
  Exception Handlers;
END [ProcedureName];
```

- Procedure can have o or many parameters.
- Every PROCEDURE consists of TWO parts
  - The header of the procedure.
  - The body of the procedure.

The Header

- It comes before the AS/IS keyword.
- It contains the PROCEDURE NAME and the PARAMETER LIST.

The Body

- It is any thing or every thing that is existing after the AS keyword.
- The word REPLACE is optional.

Execute a Procedure
At SQL Prompt
>EXECUTE/EXEC PROCEDURE_NAME
Call the Procedure in another PL/SQL Block

```
SQL>BEGIN
        PROCEDURE_NAME;
    END;
```

- **The procedure details are stored in**
  - **USER_SOURCE**
  - **USER_OBJECTS**

```
CREATE OR REPLACE PROCEDURE cal_intr
                    (P number,N number,R number)
IS
si number(14,2);
ci number(16,2);
BEGIN
    si:=(P*N*R)/100;
    ci:=power((1+r/100),n);--not case sensitive
    ci:=p*ci;
    dbms_output.put_line('simple interest is:'||si);
    dbms_output.put_line('compound interest is:'||ci);
END cal_intr;
```

To view the Procedure availability

```
SQL>SELECT object_name ,object_type,status
    FROM USER_OBJECTS
    WHERE object_name='PROC_NAME';
```

Illustration

```
SQL> SELECT object_name ,object_type,status
    FROM USER_OBJECTS
    WHERE object_name='CAL_INTR';

SQL> SELECT line||'>',text FROM  user_source
    WHERE name='PROC_NAME';
```

Illustration

```
    SQL> SELECT line||'>',text FROM user_source
        WHERE name='CAL_INTR';
```

Recompile an Existing Procedure
Syntax
SQL>ALTER PROCEDURE procedurename compile;
Illustration
SQL>ALTER PROCEDURE cal_intr COMPILE

```
    SQL>CREATE OR REPALCE  procedure proc_bonus
        as
```

```
cursor cb is
select empno,ename,job,sal+nvl(comm,0) netsal
from emp;
i cb%rowtype;
bonus number;
begin
open cb;
loop
fetch cb into i;
exit when cb%notfound;
if i.job='CLERK' then
  bonus:=i.netsal*0.15;
elsif i.job='SALESMAN' then
  bonus:=i.netsal*0.25;
else
  bonus:=i.netsal*0.35;
end if;
update emp set sal=sal+bonus
where empno=i.empno;
display(rpad(i.ename,8)||'  '||rpad(i.job,8)||'  '||bonus);
end loop;
close cb;
END proc_bonus;
```

Subprogram Parameters

- PARAMETERS are the mean to pass values TO and FROM the calling environments to the oracle server.
- PARAMETERS are the values that will be processed or returned via the EXECUTION of the PROCEDURE or FUNCTION.
- These PARAMETERS can have different modes.

Parameter Modes

- There are three types of modes
  - IN Mode
  - OUT Mode
  - INOUT Mode

IN MODE

- It is the Default mode of subprogram.
- Passes a value into the program , from the calling environment.
- It is READ ONLY value.

OUT MODE

- Used to return a value from subprogram.
- It is WRITE ONLY value.
- Passes a valued back from the program to the calling environment.
- Cannot be assigned default values.
- Value assigned only if the program is successful.

INOUT MODE

- Carriers a value into a subprogram.
- Returns value from sub program.
- Values will be READ from the calling environment and then WRITTEN to the calling environment.

```
SQL> CREATE OR REPLACE PROCEDURE
 proc_sc(n in NUMBER,s OUT NUMBER,c OUT NUMBER)
 IS
 BEGIN
 s:=n*n;
 c:=s*n;
 END proc_sc;  ..
> var sr NUMBER
> VAR cr NUMBER
> EXEC proc_sc(10,:sr,:cr)
```

Calling in PL/SQL block

```
SQL>DECLARE
n NUMBER:=&no;
vs NUMBER;
vc NUMBER;
BEGIN
proc_sc(n,vs,vc);
display('The '||n||' seq and cube values are '||vs||'  '||vc);
END;
```

```
SQL>DECLARE
n NUMBER:=&no;
BEGIN
proc_sc(n,:sr,:cr);
display('The '||n||' seq and cube values are '||:sr||'  '||:cr);
END;
```

```
SQL>CREATE OR REPLACE PROCEDURE
proc_dtsal(pdts IN OUT NUMBER

)
IS
BEGIN
SELECT SUM(sal) INTO pdts
FROM emp
WHERE deptno=pdts;
END proc_dtsal;
SQL> CREATE OR REPLACE PROCEDURE
 proc_dtsal(pdts IN OUT NUMBER)
```

```
IS
BEGIN
SELECT SUM(sal) INTO pdts
FROM emp
WHERE deptno=pdts;
END proc_dtsal;
> var bdts NUMBER
>EXEC :bdts:=10
```

```
SQL> CREATE OR REPLACE PROCEDURE
   add_dept(pdname IN dept.dname%TYPE DEFAULT 'UNKNOW',
        ploc IN dept.loc%TYPE DEFAULT 'UNKNOW')
IS
BEGIN
INSERT INTO dept VALUES(ds.nextval,pdname,ploc);
END add_dept;
```

Note:

If want insert value only for 'loc'

Types of sending arguments to the subprogram

Position Notation

- It is simple an association of the values by POSITION of the arguments at call time with that of declaration in the header of the procedure creation.
- The order of the parameters used when executing the procedure should match the order in the PROCEDURES HEADER exactly.

Named Notions(=>)

- It is an explicit association using the symbol =>

Syntax

Formalparametername=>ArgValue

- In named notation the order of parameters in does not matter.
- If the notation is mixed then position notation should be used first then the named notation should be used.

```
SQL>GRANT EXECUTE ON  cal_intr to miller;
```

MILLER

```
SQL>EXEC  scott.cal_intr(1000,12,2)
```

Synonym for Procedure

SQL>Create [public] Synonym <Synonym Name> for <Procedure Name>;

SQL> Create public Synonym sci FOR cal_intr;

SQL> GRANT EXECUTE ON sci TO PUBLIC;

MILLER

SQL>exec sci(1000,12,2)

Note: Synonym can't be created for Procedure which is defined in Package.

pragma autonomous_transaction

- Prior to Oracle 8i and higher .(Autonomous--in_depended).
- It Used in nested procedure to make each procedure in_depended for TCL.
- Allow to write TCL in Trigger.
- It must appear in the declarative section of the block ,and only one pragma is allowed in the block.
- It can go anywhere in the declarative section ,but is it good style to put it at the beginning.

```
SQL>CREATE OR REPLACE PROCEDURE add_emp
AS
BEGIN
INSERT INTO emp(empno,ename,job,sal,deptno)
 VALUES(7001,'EFCODD','MANAGER',2000,50);
COMMIT;
END add_emp;
SQL>CREATE OR REPLACE PROCEDURE add_dept
        AS
        BEGIN
        INSERT INTO dept
         VALUES(50,'EXPORT','BLORE');
        add_emp;
        END add_dept;
```

How to store Images

```
SQL>CREATE OR REPLACE procedure load(fname varchar2)
as
f_lob bfile;
b_lob blob;
Begin
 insert into Images
values(fname,substr(fname,instr(fname,'.')+1)
,empty_blob())
      RETURN image into b_lob;
 f_lob:=BFILENAME('KRISHNA',fname);
 dbms_lob.fileopen(f_lob,dbms_lob.file_readonly);     dbms_lob.loadfromfile(b_lob,f_lob,
dbms_lob.getlength(f_lob));
 dbms_lob.fileclose(f_lob);
commit;
end load;
```

Dropping Procedure

- Similar to dropping a table procedure can also be dropped.

Syntax

SQL> DROP PROCEDUE procedure_name;

Illustration

SQL>DROP PROCEDURE cal_intr;

USER DEFINED FUNCTIONS

- A function is very similar to a procedure.
- Function is a named PL/SQL block and it may or my not take the parameters but it must return a value.
- Function must have a RETURN clause in the EXECUTABLE section of a FUNCTION.
- Functions are mainly used to perform the calculation.
- The data type of the return values must be declared in the header of the function.
- Procedures and Functions are different forms of PL/SQL blocks, with a declared, executable, and exception section.
- Procedures and Functions can be stored in the database or declared within a block.
- A procedure call is a PL/SQL statement by itself, while a function call is called as part of an expression.
- A function has output that needs to be assigned to a variable or it can be used in a SELECT statement.
- Function can not call when it has RETURN data type as Boolean.
- A Function can contain more than one RETURN statement, each Exception should have a RETURN statement.

Syntax

```
CREATE [OR REPLACE]
 FUNCTION <Function name>(parameter List)
 RETURN data type
 IS
  [Local variables]
 BEGIN
  <Body>
  RETURN(value);
  EXCEPTION
  <Defined pragmas>
 END [Function name];
```

```
>CREATE OR  REPLACE FUNCTION cal_intr
                    (P NUMBER,N NUMBER,R NUMBER)
  RETURN NUMBER
   IS
    v_ci NUMBER(16,2);
  BEGIN
   v_ci:=POWER((1+r/100),n);
   v_ci:=p*v_ci;
   RETURN(v_ci);
 END;
```

Calling function at SQL:-

```
>var cint number;
>exec :cint:=cal_intr(1000,12,2);
>print :cint;
```

Calling function with SELECT statement

```
>SELECT  cal_intr(1000,12,2) FROM dual;
> SELECT ename,hiredate,sal,ROUND(cal_intr(sal,12,2)) "cintr"
  FROM emp
```

Calling Function In PL/SQL block

```
DECLARE
v_cintr NUMBER;
BEGIN
V_cintr:= cal_intr(1000,12,2);
DISPLAY('The Compound Intr is: '||v_cintr);
END;
/
CREATE OR  REPLACE FUNCTION leapyear
                    (y  NUMBER)
 RETURN VARCHAR2
 IS
BEGIN
   IF (MOD(y,400)=0)OR (MOD(y,100)!=0 and MOD(y,4)=0) THEN
        RETURN('Leap Year');
   ELSE
         RETURN('Not Leap Year');
   END IF;
END;
/
```

Q)Write a function to accept the empno and return exp with minimum 3 decimal?

```
CREATE OR REPLACE FUNCTION empexp(p_empno emp.empno%TYPE)
 RETURN NUMBER
IS
v_hiredate emp.hiredate%TYPE;
v_exp NUMBER(6,3);
BEGIN
SELECT hiredate INTO v_hiredate
FROM emp
WHERE empno=p_empno;
v_exp:=MONTHS_BETWEEN(SYSDATE,v_hiredate)/12;
RETURN v_exp;
END;
```

Q) Write a function to accept the deptno and return the No of emps in that deptno.

```
CREATE OR REPLACE FUNCTION noe(p_deptno emp.deptno%TYPE)
 RETURN NUMBER
IS
v_noe NUMBER(6,3);
BEGIN
SELECT count(empno) INTO v_noe
FROM emp
WHERE deptno=p_deptno;
RETURN v_noe;
END;
> select unique deptno,noe(deptno)
  from emp
  where noe(deptno)>3;
```

Database TRIGGERS

Database TRIGGERS

- A Trigger set of PL/SQL statements automatically executed whenever an DML statement is performed on table.
- It is a PL / SQL Block like a Procedure. i.e., to perform some specific task. But a procedure always requires an explicit call for execution but triggers are executed automatically when any triggering event occurs.
- It is Associated with a Table or View.
- INSERT,UPDATE,DELETE are considered as the triggering events of the database trigger. These events initiate the firing of trigger.
- The firing of trigger is nothing but the execution of the PL/SQL code associated to that trigger.
- It is also a Database Object.

Advantages:

- A database trigger is a security object to provide security to the table like tracking the transaction.
- A database trigger is also used to define the complex business constraints that cannot be defined by using integrity constraints.
- Automatically generating values for derived columns or PRIMARY KEY columns.
- Used to implement user defined restrictions on table.
- provides high security.
- Activated when table are manipulated from other application software also.

Syntax:

```
SQL>CRAETE [OR REPLACE ] TRIGGER triggername
      AFTER/BEFORE  INSERT or UPDATE or DELETE
      [OF columnname] ON tablename
      [FOR EACH ROW]
      [WHEN condition]
      DECLARE
            Declaration statements;
      BEGIN
            Executable statements;
      [EXCEPTION
            Exception handling;]
      END [trg_name];
```

- Trigger can not be duplicate trigger name .
- Trigger can be attached one table
- When condition is true, trigger will be executed otherwise not executed.

The specification

New&Old

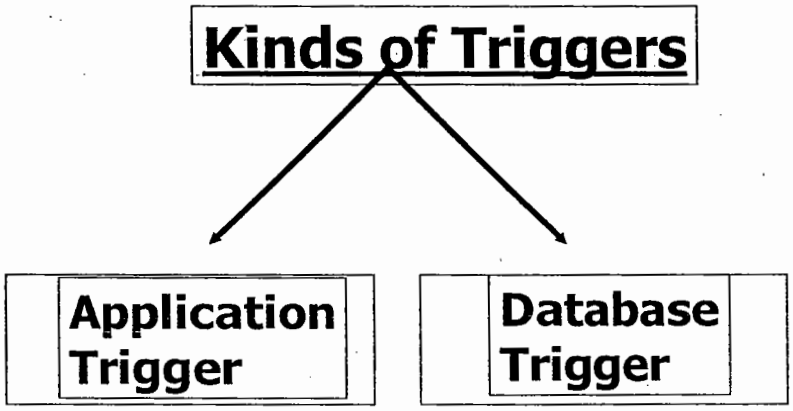
- Refers to the values available in DML statements.
- Valid in row trigger only.

New&Old:-

→Refers to the values available in DML stmts.

→Valid in row trigger only

	New	Old
Insert	✓	✗
Update	✓	✓
Delete	✗	✓

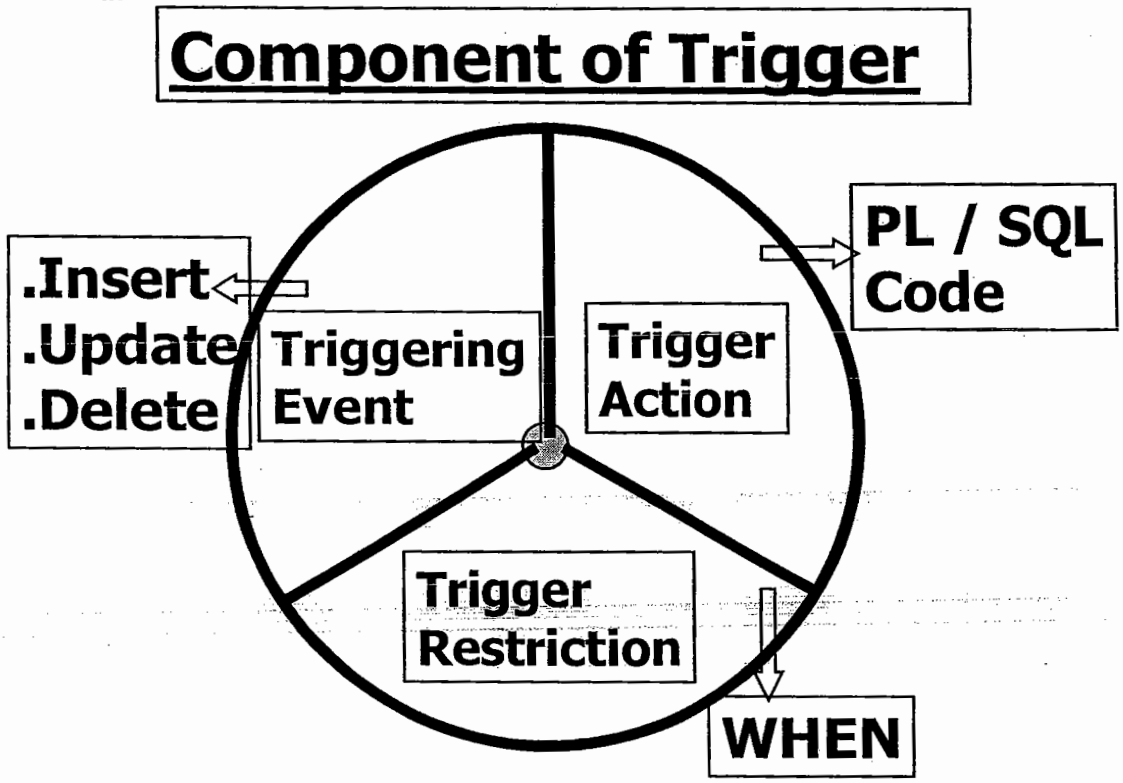


Application Trigger

- **It is fired when DML Event occurs Within the Application.**
  Example : Oracle Forms

Database Trigger

- **It is fired when DML Event occurs on a Table / View, no matter which User is connected.**



Trigger Parts

- Indicates when to activate the trigger. i.e., defines whether the trigger fires before or after the statement is executed.

Before Triggers

- These TRIGGERS fire BEFORE any transactions are implemented.
- These TRIGGERS can be classified as
  - BEFORE INSERT
  - BEFORE UPDATE
  - BEFORE DELETE

Usage

- When a trigger provides values for derived columns BEFORE the INSERT OR UPDATE statement is completed.
- When a trigger determines whether an INSERT,UPDATE or DELETE statement should be allowed to completed.

After Triggers

- These TRIGGERS fire AFTER any transaction is implemented.
- These TRIGGERS can be classified as
  - AFTER INSERT
  - AFTER UPDATE

- AFTER DELETE

Usage

- When a TRIGGER should fire after a DML statement is executed for acknowledgement purpose or auditing.
- When a TRIGGER should perform action not specified in a BEFORE trigger.

Restrictions

- A trigger may not issue a transactional control statement like COMMIT,SAVEPOINT and ROLLBACK.
- Any FUNCTION or PROCEDURE called by a trigger cannot issues a transaction control statement.

Level of Triggers

- Trigger can be define at two different levels.
- They are
  1)Row level
  2)Statement or table level

Row level Trigger

- A row trigger is fired as many times as there are rows affected by triggering event.
- When the statement FOR EACH ROW is present in the CREATE TRIGGER clause, the trigger is a ROW trigger.

Statement Level

- Trigger will be fired only once for DML statement.

Trigger Body

- A set of PL/SQL statements.

The WHEN clause

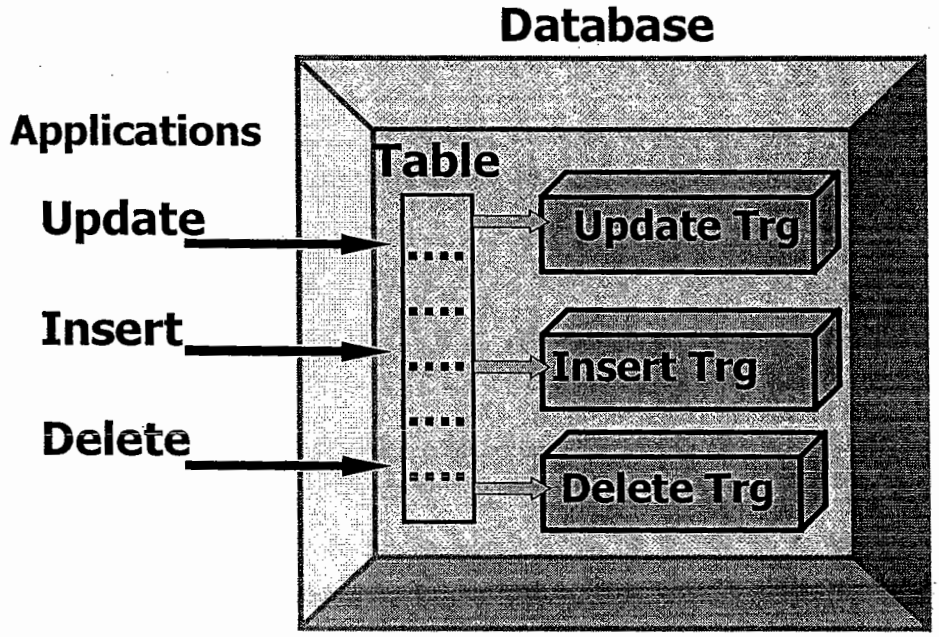
- The WHEN clause is valid for row-level triggers only.
- The trigger body will be executed only for those rows that meet the condition.

Syntax:

WHEN trigger_condition

- The :new and :old records can be referenced inside trigger_condition as well.
- The colon is only valid in the trigger body.

- **Activated when table are manipulated from other application software also.**



How many Trigger are there
Mxm-12 Trigger.

Row level

After Insert	Before Insert
After Update	Before Update
After Delete	Before Delete

Table Level

After Insert	Before Insert
After Update	Before Update
After Delete	Before Delete

```
SQL>create or replace trigger trg_upp_con
before insert or update
of name on kcb_acc_tab
FOR EACH ROW
BEGIN
:NEW.name:=UPPER(:NEW.name);
END trg_upp_con;

SQL> CREATE OR REPLACE TRIGGER trg_nnd
```

```
BEFORE INSERT OR UPDATE
OF dname ON dept
FOR EACH ROW
DECLARE
cnt NUMBER;
pragma autonomous_transaction;
BEGIN
SELECT count(*) INTO cnt
FROM DEPT
WHERE dname=:NEW.DNAME;
IF cnt>0 THEN
  RAISE_APPLICATION_ERROR
          (-20345,'The dname con not be dup');
ELSIF :NEW.dname IS NULL THEN
RAISE_APPLICATION_ERROR
     (-20346,'The dname con not be null');
END IF;
END trg_nnd;
```

```
Table          delete_log
ecode NUMBER(4),
 ename VARCHAR2(20),
 basic NUMBER(7,2),
 dod timestamp
SQL>CREATE OR REPLACE TRIGGER trg_del
AFTER delete
ON emp
FOR EACH ROW
BEGIN
INSERT INTO delete_log VALUES(:OLD.empno,:OLD.ename,:OLD.sal,sysdate);
END trg_del;
```

```
Table Trace
----------------
 Userid  varchar2(20),
 dod    timestamp
SQL>CREATE OR REPLACE TRIGGER trg_del
AFTER delete
ON emp
FOR EACH ROW
BEGIN
INSERT INTO trace VALUES(user,sysdate);
END trg_del;
SQL>CREATE OR REPLACE TRIGGER trg_bonus
 AFTER INSERT
 ON emp
```

```
FOR EACH ROW
DECLARE
pragma autonomous_transaction;
BEGIN
INSERT INTO bonus VALUES(:NEW.ename,:NEW.job,:NEW.sal,:NEW.comm);
COMMIT;
END trg_bonus;
```

It gives Run time error .

ORA-04092: cannot COMMIT in a trigger

SQL> rollback;

Emp table transactions only will rollback;

```
SQL>CREATE OR REPLACE TRIGGER trg_sal
AFTER delete
ON EMP
FOR EACH ROW
WHEN (old.sal>1500)
BEGIN
INSERT INTO delete_log
 VALUES(:OLD.empno,:OLD.ename,:OLD.sal,sysdate);
END trg_sal;
```

3 events on single Trigger:-
(insert,update,delete)

```
CREATE [OR REPLACE ] TRIGGER <trg_name>
BEFORE/AFTER   insert OR update OR delete
ON <tablename>
[FOR EACH ROW]
[When  condition]

[DECLARE]

BEGIN
        IF Inserting THEN
                SQL Statements
        END IF;
        IF updating THEN
                SQL Statements
        END IF;
        IF deleting THEN
                SQL Statements
        END IF;

END [trg_name];
```

```
> desc INT_TAB
 Name                                      Type
 ----------------------------------------- -------- ----------------------
 ECODE                                     NUMBER(4)
 NAME                                      VARCHAR2(10)
 DEPTNO                                    NUMBER(2)

SQL> desc UPD_TAB
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 ECODE                                     NUMBER(4)
 NAME                                      VARCHAR2(10)
 USAL                                      NUMBER(7,2)

SQL> desc del_tab
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 ECODE                                     NUMBER(4)
 NAME                                      VARCHAR2(10)
 JOB                                       VARCHAR2(9)

SQL> CREATE OR REPLACE  TRIGGER trg_iud
AFTER   insert OR update OR delete
ON emp
FOR EACH ROW
BEGIN
IF Inserting THEN
        INSERT INTO int_tab
  VALUES(:NEW.empno,:NEW.ename,:NEW.deptno);
END IF;
IF updating THEN
 INSERT INTO upd_tab
  VALUES(:NEW.empno,:NEW.ename,:NEW.sal);
END IF;
IF deleting THEN
 INSERT INTO del_tab
  VALUES(:OLD.empno,:OLD.ename,:OLD.job);
END IF;
END trg_uid;
```

Table job_list

Job	low_pay	high_pay
Clerk	3000	8000
Salesman	4000	12000
Annalist	8000	25000

Manager 6000 20000

```
SQL>CREATE OR REPLACE TRIGGER trg_job
BEFORE insert OR update
ON EMP
for each row
WHEN (NEW.job<>'PRESIDENT')
DECLARE
i job_list%ROWTYPE;
BEGIN
SELECT low_pay,high_pay INTO i.low_pay,i.high_pay
FROM job_list
WHERE job=INITCAP(:NEW.job);
IF :NEW.sal NOT BETWEEN i.low_pay AND i.high_pay THEN
 RAISE_APPLICATION_ERROR
        (-20345,:NEW.job ||' sal in between '||i.low_pay||' and '||i.high_pay);
END IF;
exception
WHEN no_data_found THEN
 RAISE_APPLICATION_ERROR
(-20346,:NEW.job||' is not exists in job_list table');
END trg_job;

SQL>CREATE OR REPLACE TRIGGER trg_wht
       BEFORE INSERT OR UPDATE OR DELETE
         ON emp
         DECLARE
         cnt NUMBER;
 BEGIN
 SELECT COUNT(*) INTO cnt
 FROM htab
 WHERE TO_CHAR(HDAY,'DD-MON-YY')=TO_CHAR(SYSDATE,'DD-MON-YY');
 IF TO_CHAR(SYSDATE,'DY') IN('SAT','SUN') THEN
  RAISE_APPLICATION_ERROR(-20345,'No transaction in week end');
 ELSIF cnt>0 THEN
  RAISE_APPLICATION_ERROR(-20347,'No transaction in Hday');
 ELSIF TO_CHAR(sysdate,'HH24') NOT BETWEEN 9 AND 19 THEN
  RAISE_APPLICATION_ERROR(-20348,'No transaction in this time');
 END IF;
 END trg_wht;
```

Remove trigger

Syntax

```
SQL>DROP TRIGGER <trg_name>;
SQL>DROP TRIGGER  trg_wht;
```

To disable all triggers on Table

SQL> ALTER TABLE <tabname> DISABLE ALL TRIGGERS;

SQL>ALTER TABLE emp DISABLE ALL TRIGGERS;

To Enable all triggers on Table

SQL> ALTER TABLE <tabname> ENABLE ALL TRIGGERS;

SQL > ALTER TABLE emp ENABLE ALL TRIGGERS;

To disable single trigger on Table

Syntax

SQL>ALTER TRIGGER trg_name DISABLE;

SQL>ALTER TRIGGER trg_wht DISABLE;

To enable single trigger on Table

Syntax

SQL>ALTER TRIGGER trg_name ENABLE;

SQL>ALTER TRIGGER trg_wht ENABLE;

```
SQL> SELECT TRIGGER_TYPE,TRIGGERING_EVENT,STATUS
        from user_triggers
      WHERE TRIGGER_NAME='TRG_WHT';
```

To view the source code of Trigger

```
SQL>SELECT text FROM user_source
        WHERE name='TRG_WHT';
```

Instead of Triggers(8i)

→Trigger on view

→used to manipulate join views

Instead of →insert

→update

→delete

```
SQL> CREATE VIEW ev
        AS
        SELECT *FROM emp;
SQL> CREATE OR REPLACE TRIGGER trg_ev
        Instead of DELETE
        ON ev
        BEGIN
        display('Record is removed');
        END trg_ev;
SQL> CREATE OR REPLACE VIEW ed_view
AS
SELECT empno,ename,job,sal,e.deptno,dname,loc
FROM emp e,dept d
WHERE e.deptno=d.deptno;

SQL> CREATE OR REPLACE TRIGGER trg_ed
 instead of DELETE
ON ed_view
BEGIN
DELETE FROM emp
```

```
WHERE deptno=:OLD.deptno;
DELETE FROM dept
WHERE deptno=:OLD.deptno;
END trg_ed;
```

DDL TRIGGERS

- The only difference between DML and DDL triggers is the firing event.
- You use DDL triggers to enforce rules or audit the creation of database objects.
- We create a table name AUDIT_CREATION.

```
SQL>Create table audit_creation
        (audit_creation_id number constraint acid primary key,
          audit_owner_name  varchar2(30) constraint audit_creation_nn1 not null,
           audit_obj_name  varchar2(30) constraint audit_creation_nn2 not null,
          audit_date  date constraint audit_creation_nn3 not null);
```

Even attribute Function

ORG_DICT_OBJ_OWNER

- This function takes no formal parameter.
- It Returns no owner of the object acted upon by the event as a VARCHAR2 datatype.

ORA_DICT_OBJ_NAME

- It return an object name as a VARCHAR2 datatype.
- The object name represents the target of the DDL statement.

This trigger tracks the creation all objects, including which user issued the CREATE statement.

```
>CREATE OR REPLACE TRIGGER audit_creation
Before create on SCHEMA
BEGIN
 INSERT INTO audit_creation
 VALUES(audit_creation_s1.nextval,
   ORA_DICT_OBJ_OWNER,
   ORA_DICT_OBJ_NAME,
    SYSDATE);
END;
```

Database Packages

Packages

- It is logical group of objects such as
  - Functions
  - Procedures
  - Global variables
  - Cursors

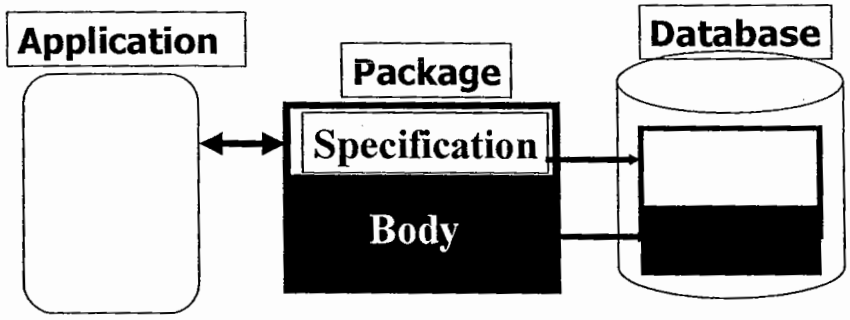
Advantages

- Packages allow us to organize our application development more efficiently.
- Packages allow us to grant the privileges more efficiently.
- It allows the Oracle Server to Read Multiple Objects into Memory at Once.
- Packages can contain global variables and cursors that are available to all procedures and functions in the packages.
- Packages allow us to overload procedure and functions.
- All related program objects are stored in one particular location in the memory (which reduce the search process) stored in USER_SOURCE.

- Package has two parts, with are individually coded.
  - Package specification
  - Package body

Components of a Package

--Specification
--Body



Package Specification

- The PACKAGE SPECIFICATION contains information about the contents of the package.
- PACKAGE SPECIFICATION contains declarations of GLOBAL/PUBLIC variables.

- All object placed in the PACKAGE SPECIFICATION are called PUBLIC OBJECTS. such objects are called public objects

Syntax for Specification:
CREATE [OR REPLACE] PACKAGE <Pkg Name>
IS/AS
type_definition
PROCEDURE <proc name>(parameter)
FUNCTION <fun name>(parameter) RETURN datatype;
Variable_declaration;
Exception_declaration;
Cursor_declaration;
END [Pkg Name];

Package Body

- The PACKAGE BODY is a separate data dictionary object from the package header.
- The PACKAGE BODY contains the actual executable code for the OBJECTS described in the PACKAGE SPECIFICATION.
- The PACKAGE BODY contain code for all procedures and function described in the specification.
- When creating stored package the package specification and body can be complied separately.

Syntax for Body:
CREATE [OR REPLACE] PACKAGE BODY <Pkg Name>
IS/AS
[Private variable declaration];
PROCEDURE <procedure name>(arg list)
IS
.....
.....
END [procedure name];
FUNCTION <function name>(arg list)
RETURN datatype
IS
BEGIN
.....
.....
END [function name];

END <Pkg Name>;

Invoking Package Constructs
>Execute <Package Name>.<Prog Name>;

Bodiless Package

- We can declare Package Specification that does not need Package body.

SQL>CREATE OR REPLACE PACKAGE pack_am

```
AS
PROCEDURE add_num(m IN NUMBER,n IN NUMBER);
FUNCTION mul_num(x IN NUMBER,y IN NUMBER)
RETURN number;
result NUMBER;
END pack_am;
SQL> CREATE OR REPLACE PACKAGE BODY pack_am
 AS
 PROCEDURE add_num(m IN NUMBER,n IN NUMBER)
 IS
 BEGIN
 result:=m+n;
 display('The sum of m,n is '||result);
 END add_num;
 FUNCTION mul_num(x IN NUMBER,y IN NUMBER)
 RETURN number
 IS
 BEGIN
 result:=x*y;
 RETURN(result);
 END mul_num;
 END pack_am;

SQL>CREATE OR REPLACE PACKAGE pack_eg
AS
FUNCTION exp(pdoj IN DATE) RETURN  NUMBER;
FUNCTION gross(pbasic IN NUMBER) RETURN NUMBER;
END pack_eg;
SQL>CREATE OR REPLACE PACKAGE BODY pack_eg
AS
FUNCTION exp(pdoj IN DATE)
RETURN  NUMBER
IS
BEGIN
RETURN(round(MONTHS_between(SYSDATE,pdoj)/12));
END exp;
FUNCTION gross(pbasic IN NUMBER)
RETURN NUMBER
IS
BEGIN
RETURN(pbasic+pbasic*0.45+pbasic*0.35-pbasic*0.15);
END gross;
END pack_eg;

SQL>select pack_eg.exp('14-jun-95'),pack_eg.gross(1000) from dual;
SQL> select ename,hiredate,pack_eg.exp(hiredate),
```

```
        sal,pack_eg.gross(sal)
         from emp
        WHERE pack_eg.exp(hiredate)>30;
```

Function over loading

```
        SQL>CREATE OR REPLACE PACKAGE pack_fo
        AS
        FUNCTION fun_add(m IN NUMBER,n IN  NUMBER)
        RETURN NUMBER;
        FUNCTION fun_add(x IN VARCHAR2,y IN  VARCHAR2)
        RETURN VARCHAR2;
        END pack_fo;
        SQL>CREATE OR REPLACE PACKAGE BODY pack_fo
        AS
        FUNCTION fun_add(m IN NUMBER,n IN  NUMBER)
        RETURN NUMBER
        IS
        BEGIN
        RETURN(m+n);
        END fun_add;
        FUNCTION fun_add(x IN VARCHAR2,y IN  VARCHAR2)
        RETURN VARCHAR2
        IS
        BEGIN
        RETURN(x||' '||y);
        END fun_add;
        END pack_fo;
SQL> SELECT pack_fo.fun_add(100,200)
        "Res1",pack_fo.fun_add('KRISHNA','ORACLE') "Res2"
        from dual;
```

→Package will be stored in USER_SOURCE

```
SQL> SELECT TEXT FROM USER_SOURCE
          WHERE NAME='PACK_EG';
```

Remove the package:-

```
SQL>DROP PACKAGE <packname>;

SQL>CREATE OR REPLACE PACKAGE pack_bal
AS
PROCEDURE upd_bal(paccno IN kcb_acc_tab.accno%TYPE,
                  pttype IN kcb_tran_tab.ttype%TYPE,
                  pamt IN kcb_tran_tab.amt%TYPE);
FUNCTION chk_bal(paccno IN kcb_acc_tab.accno%TYPE,
                  pamt IN kcb_tran_tab.amt%TYPE)
RETURN BOOLEAN;
```

```
FUNCTION fnow(paccno IN kcb_acc_tab.accno%TYPE)
RETURN NUMBER;
FUNCTION ftwa(paccno IN kcb_acc_tab.accno%TYPE)
RETURN NUMBER;
cbal kcb_acc_tab.bal%TYPE;
END pack_bal;

SQL> PACKAGE BODY pack_bal
AS
PROCEDURE upd_bal(paccno IN kcb_acc_tab.accno%TYPE,
            pttype IN kcb_tran_tab.ttype%TYPE,
            pamt IN kcb_tran_tab.amt%TYPE)
IS
BEGIN
SELECT bal INTO cbal
FROM kcb_acc_tab
WHERE accno=paccno;
IF upper(pttype)='D' THEN
cbal:=cbal+pamt;
elsif upper(pttype)='W' THEN
cbal:=cbal-pamt;
END IF;
UPDATE kcb_acc_tab SET bal=cbal
WHERE accno=paccno;
END upd_bal;
FUNCTION chk_bal(paccno IN kcb_acc_tab.accno%TYPE,
            pamt IN kcb_tran_tab.amt%TYPE)
RETURN BOOLEAN
IS
vacctype kcb_acc_tab.acctype%type;
BEGIN
SELECT acctype,bal INTO vacctype,cbal
FROM kcb_acc_tab
WHERE accno=paccno;
cbal:=cbal-pamt;
IF cbal<5000 AND vacctype='S' THEN
 RETURN(false);
ELSIF cbal<10000 AND vacctype='C' THEN
 RETURN(false);
ELSE
 RETURN(true);
END IF;
END chk_bal;
FUNCTION fnow(paccno IN kcb_acc_tab.accno%TYPE)
RETURN NUMBER
IS
```

```
vnow NUMBER;
BEGIN
 SELECT count(*) INTO vnow
 FROM kcb_tran_tab
 WHERE accno=paccno AND
   ttype='W' AND              TO_CHAR(dot,'DD-MON-YY')=
TO_CHAR(sysdate,'DD-MON-YY') AND
     EXISTS (SELECT 'krishna'
      FROM kcb_acc_tab
      WHERE accno=paccno AND
      acctype='S');
RETURN(vnow);
END fnow;
FUNCTION ftwa(paccno IN kcb_acc_tab.accno%TYPE)
RETURN NUMBER
IS
vtwa NUMBER;
BEGIN
 SELECT sum(amt) INTO vtwa
 FROM kcb_tran_tab
 WHERE accno=paccno AND
   ttype='W' AND
 TO_CHAR(dot,'DD-MON-YY')=
TO_CHAR(sysdate,'DD-MON-YY') AND
     EXISTS (SELECT 'krishna'
      FROM kcb_acc_tab
      WHERE accno=paccno AND
      acctype='S');
IF vtwa IS NULL THEN
  vtwa:=0;
END IF;
RETURN(vtwa);
END ftwa;
END pack_bal;
```

Dynamic Cursor in package

```
SQL>CREATE OR REPLACE PACKAGE sc_fm
 IS
  type r_comp is record
  (deptno emp.deptno%type,
   ename emp.ename%type,
   hiredate emp.hiredate%type,
   sal emp.sal%type,
   total_sal emp.sal%type);
   TYPE comp_rc IS REF CURSOR
   RETURN r_comp;
```

```
 FUNCTION fun_emp(p_deptno NUMBER)
  RETURN comp_rc;
END sc_fm ;

SQL> CREATE OR REPLACE PACKAGE BODY sc_fm
        IS
  FUNCTION fun_emp(p_deptno NUMBER)
  RETURN comp_rc
 IS
 c_emp sc_fm.comp_rc;
 v_raise NUMBER;
 BEGIN
        IF p_deptno>=40 THEN
              v_raise :=1.4;
        ELSIF p_deptno=30   THEN
              v_raise :=1.3;
        ELSIF p_deptno=20   THEN
              v_raise :=1.2;
        ELSE
               v_raise:=1.1;
        END IF;
 OPEN
        c_emp FOR
        SELECT deptno,ename,hiredate,sal,
                 v_raise*(sal+nvl(comm,0)) total_sal
        FROM emp
        WHERE deptno=p_deptno;
        RETURN c_emp;
  END fun_emp;
END sc_fm;
```

Dynamic SQL

- Within PL/SQL, you can execute any kind of SQL statement(even data definition and data control statements)
- In PL/SQL, such statements cannot be executed statically.
- when we want to execute a SQL data definition statement (such as CREATE)
- A data control statement (such as GRANT).

EXECUTE IMMEDIATE :-

- The EXECUTE IMMEDIATE statement prepares and immediately executes a dynamic SQL statement or an anonymous PL/SQL block.

syntax

```
EXECUTE IMMEDIATE dynamic_string
```

```
    SQL>CREATE OR REPLACE PROCEDURE
        drop_table (table_name IN VARCHAR2)
    AS
    BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE '|| table_name||' PURGE';
    END drop_table;
SQL>CREATE OR REPLACE PROCEDURE proc_tg
AS
BEGIN
 FOR i  IN (SELECT  'GRANT  SELECT ON '||table_name||' to krishna ' cmd
                FROM user_tables)
 LOOP
  execute immediate i.cmd;
 END LOOP;
EXCEPTION
  when others then
 dbms_output.put_line('error'||substr(sqlerrm,1,250));
END PROC_TG;

SQL> CREATE OR REPLACE PROCEDURE proc_tab_revoke
AS
BEGIN
 FOR i  IN (SELECT  'REVOKE  SELECT ON '||table_name||' FROM krishna ' cmd
                FROM user_tables)
 LOOP
  execute immediate i.cmd;
 END LOOP;
EXCEPTION
  when others then
 dbms_output.put_line('error'||substr(sqlerrm,1,250));
END proc_tab_revoke;
```

UTL_FILE package(Out bound Interface)

UTL_FILE :

- This package will be used for inserting the data from flat file to table and table to flat file.
- It has functions

UTL_FILE.FOPEN() : Using this function we will create a new file in the UTL_FILE access directories.

UTL_FILE.PUTLINE() : using this function we inserting the data in file.

UTL_FILE.FCLOSE() : Using this function we will close the file after inserting the data.

```
SQL>CREATE OR REPLACE PROCEDURE  outbound
AS
 CURSOR c1 IS
 select empno,ename,job,hiredate,sal,deptno
```

```
FROM EMP;
Id UTL_FILE.FILE_TYPE;
BEGIN
Id:=UTL_FILE.FOPEN('KRISHNA','empf.txt','w');
FOR i IN c1
LOOP
UTL_FILE.PUT_LINE(id,i.EMPNO||','||i.ENAME||','||i.job||','||i.hiredate||','||i.sal||','||i.deptno);
END LOOP;
UTL_FILE.FCLOSE(id);
END  outbound;
```