**Oracle Introduction**

Oracle Corporation is the largest software company to develop and markets computer software applications for business. The company is best known for its Oracle database products and, more recently, cloud products and services. Its relational database was the first to support SQL, which has since become the industry standard.

Oracle database is one of the most trusted and widely used relational database engines. The biggest rival of Oracle database is Microsoft's SQL Server.

**Oracle Database Features**

Oracle database manages data with the help of an open, complete, and integrated approach. The following are features that complete the demand for powerful database management:

**Availability**: It is never offline or out of service that means supported 24\*7 availability of the database. It provides high availability of databases because of the Oracle Data Guard functionality. This functionality allows using of the secondary database as a copy of the primary database during any failure. As a result, all normal processes such as backups and partial failures do not interrupt the database from being used.

**Security:** Oracle has a mechanism for controlling and accessing the database to prevent unauthorized access. It provides high security because of the Oracle Advanced Security features.

**Scalability:** It provides features like RAC (Real Application Cluster) and Portability, which makes an Oracle database scalable based on usage. In a clustered environment, it includes capabilities such as rolling instance migrations, performing upgrades, maintaining application continuity, quality of service management, etc.

**Performance:** Oracle provides performance optimization tools such as Oracle Advanced Compression, Oracle Database In-Memory, Oracle Real Application Testing, and Oracle Times Ten Application-Tier Database Cache. Their main objective is to improve system performance to the highest possible level.

**Importance of Oracle**

It is among the oldest companies which provide database management solutions. The company has always focused on Enterprise requirements and acknowledged the latest technology trends. That’s why its products are always embellished with new features. For instance, the latest Oracle database 21C is also available on Oracle Cloud. It offers users to choose from the different database editions that suit their needs to provide a cost-effective solution.

**Benefits of Oracle**

1.**Performance**: It has methodologies and principles to achieve high performance. We can implement performance tuning in its database to retrieve and alter data faster, in order to improve query execution time and hence application operations.

2. **Multiple Database:** Its database supports managing multiple database instances on a single server. It provides an Instance Caging method to manage CPU allocations on a server running the database instances. Instance caging works with the database resource manager to manage services over multiple instances.

3. **Editions:** As we discussed above, about the different editions Oracle offers, it benefits the users to purchase editions as per their application requirements. They can seamlessly update the edition if their requirements change in the future. If you want to learn and do some hands-on Oracle, you can download and install the express edition database which is absolutely free.

4. **Clusters:** It uses Real Application Clusters to provide a high data availability system. The database with RAC has benefits over traditional database servers :

1. Scaling the database over multiple instances.
2. Load balancing
3. Data redundancy and availability
4. Flexible to increase processing capacity

5. **Failure Recovery:** RMAN (Recovery Manager) is the feature of an Oracle DB which recover or restore the database files during downtimes and outages. It supports online, archived backups and continuous archiving. Users can also SQL\* PLUS for recovery, called user-managed recovery, which is supported by it. There is an export utility available in the database to add user-managed backups.

6. **PL/SQL**: The database supports PL/SQL extension for procedural programming.

**Introduction to SQL**

SQL stands for **Structured Query Language**. It is used for storing and managing data in relational database management system (RDMS).It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

SQL allows users to query the database in a number of ways, using *English-like statements*.

Structure Query Language (SQL) is a language used for storing and managing data in RDBMS. SQL was the first commercial language introduced for E.F Codd's Relational model. Today almost all RDBMS (MySql, Oracle, Informix, Sybase, MS Access) uses SQL as the standard database language. SQL is used to perform all type of data operations in RDBMS.

**Characteristics** of SQL

* Easy to learn.
* To access data from relational database management systems.
* Can execute queries against the database.
* To describe the data.
* To define the data in the database and manipulate it when needed.
* To create and drop the database and table.
* To create a view, stored procedure, function in a database.
* Allows users to set permissions on tables, procedures, and views.

**SQL Commands**

SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

**Types** of SQL Commands

There are **five** types of SQL commands: DDL, DML, DCL, TCL, and DQL



**SQL Data Types**

Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.

Classified into three categories for every database.

* String
* Numeric
* Date and time

String data types

CHAR(size) : It is used to store character data within the predefined length. It can be stored up to 2000 bytes.

VARCHAR2(size) : It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.

VARCHAR(SIZE) : It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)

**Difference** between char and varchar

There is **no difference** between VarChar and VarChar2 in Oracle. However, it is advised not to use VarChar for storing data as it is reserved for future use for storing some other type of variable. Hence, always use VarChar2 in place of VarChar.

Hence it is advised to use Char datatype when the length of the character string is fixed and will not change in the future.

If the length of the character string is not fixed, then VarChar2 is preferred.

**Numeric**

**NUMBER(p, s)** : It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.

**FLOAT(p) :** It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.

**Date and Time**

**DATE :** It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.

**TIMESTAMP:** It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Large Object Data Types (**LOB Types**)

**BLOB** : It is used to specify unstructured binary data. Its range goes up to 232-1 bytes or 4 GB.

**BFILE**: It is used to store binary data in an external file. Its range goes up to 232-1 bytes or 4 GB.

**CLOB** : It is used for single-byte character data. Its range goes up to 232-1 bytes or 4 GB.

**NCLOB** : It is used to specify single byte or fixed length multibyte national character set (NCHAR) data. Its range is up to 232-1 bytes or 4 GB.

**RAW(size)** : It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.

**LONG RAW** : It is used to specify variable length raw binary data. Its range up to 231-1 bytes or 2 GB, per row.

**DDL (Data Definition Language)**

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.

**CREATE** – create a new Table, database,schema

**ALTER** – alter existing table, column description

**DROP** – delete existing objects from database

Create Table Command – The CREATE TABLE command is used to implement the schemas of individual relations.

**Syntax**

CREATE TABLE table name (column name1 data type (size), column name2 data type (size), column nameN data type (size));

The example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName,Address:

CREATE TABLE Persons ( PersonID NUMBER(5),LastName varchar(10), FirstName varchar(10), Address varchar(15) );

Syntax: DESC table name;

The **DESC command** returns the attributes (columns) of the table, the datatype associated with the column, and also any constraint (if any) imposed on the column.

**ALTER TABLE** Statement

The ALTER TABLE statement is used to **add, delete, or modify** columns in an existing table.

ALTER TABLE statement is also used to **add and drop** various constraints on an existing table.

To **add** a column in a table, use the following syntax:

ALTER TABLE table\_name ADD column\_name datatype;

EX:

SQL> ALTER TABLE PERSON ADD CITY VARCHAR (10);

To **delete** a column in a table, use the following syntax:

ALTER TABLE table\_name DROP COLUMN column\_name;

EX:

SQL> ALTER TABLE PERSON DROP CITY;

To **change** the data type of a column in a table

ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;

EX: ALTER TABLE MODIFY COLUMN ADDRESS CHAR(10);

**DROP TABLE Statement**

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

drop table <table name>;

If this statement is successfully executed then the message “table dropped” would be displayed.

**Truncate:** A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

**TRUNCATE TABLE Vs DROP TABLE**

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

TRUNCATE TABLE table\_name;

Ex: TRUNCATE TABLE Emp

**Note:** *The rollback process is not possible after truncate table statement. Once you truncate a table you cannot use a flashback table statement to retrieve the content of the table.*

**SQL Constraints**

Constraints are used to limit the type of data that can go into a table. Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following **constraints**:

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT

**Not Null**

The NOT NULL constraint enforces a column to NOT accept NULL values. The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P\_Id" column and the "LastName" column to not accept NULL values:

CREATE TABLE Persons ( P\_Id number(3) NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Address varchar(255), City varchar(255));

**Unique:**

The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

**Note** that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

CREATE TABLE Persons ( P\_Id number(4) NOT NULL UNIQUE, LastName varchar(255) NOT NULL, FirstName varchar(255), Address varchar(255), City varchar(255) ) ;

**Primary Key**

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values. A primary key column cannot contain NULL values. Each table should have a primary key, and ***each table can have only ONE primary key***.

CREATE TABLE Persons ( P\_Id number(3) NOT NULL PRIMARY KEY, LastName varchar(255) NOT NULL, FirstName varchar(255), Address varchar(255), City varchar(255) ) ;

**Foreign Key**

A FOREIGN KEY in one table points to a PRIMARY KEY in another table. Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P\_Id LastName FirstName Address City

1 Hansen Ola Timoteivn 10 Sandnes

2 Svendson Tove Borgvn 23 Sandnes

3 Pettersen Kari Storgt 20 Stavanger

The "Orders" table:

O\_Id OrderNo P\_Id

1 77895 3

2 44678 3

3 22456 2

4 24562 1

The Foreign Key constraint is used to prevent actions that would destroy links between tables. The Foreign Key constraint also prevents that invalid data form being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

CREATE TABLE Orders ( O\_Id number(4) NOT NULL PRIMARY KEY, OrderNo number(3) NOT NULL, P\_Id number(3) FOREIGN KEY REFERENCES Persons(P\_Id) ) ;

**CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CREATE TABLE Persons ( P\_Id number(3) NOT NULL CHECK (P\_Id>0), LastName varchar2(15) NOT NULL,FirstName varchar2(15), Address varchar2(15), City varchar2(15)) ;

**DEFAULT Constraint**

The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified. The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:

CREATE TABLE Persons ( P\_Id number(3) NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Address varchar(255), City varchar(255) DEFAULT 'HYDERABAD’ ) ;

**DML (Data Manipulation Language) Commands**

The data manipulation language is used to add, update, and delete data in the database. The SQL commands

**INSERT** is used to add data into the database

**UPDAT**E is used to modify the data in the database,

**DELETE** is used to delete data in the database,

**Insert**

The INSERT command is to add new row to the table.

INSERT INTO table name VALUES (‗column1-name‘, ‗column2-name‘. . . columnN-name);

Ex: Insert into persons(1234,‘RAO‘,‘RAMANA‘,‘KURNOOL‘);

Inserting specified column values to a table

INSERT INTO table name(COL1,COL2,…) VALUES (‗column1-name‘, ‗column2-name‘. . . columnN-name);

Ex: Insert into persons(person\_id,last\_name,addr) values (1234,‘RAMANA‘,‘KURNOOL‘);

**Inserting rows at run time**

INSERT INTO table name VALUES (‗&column1-name‘, ‗&column2-name‘. . . &columnN-name);

**NOTE**: Here & is used to accept values at runtime **Character, date type** values has to represent in ‘ ‘ (single quotes)

Ex: Insert into persons(&person\_id,‘&last\_name‘,‘&first\_name‘,‘&addr‘);

**UPDATE Command**

The data in the table can be updated by using UPDATE command. The syntax of the UPDATE command is:

UPDATE table name SET attribute value=new value WHERE condition;

Ex: To change the addr of all the persons to NANDYAL

UPDATE persons set addr=‘NANDYAL‘;

Ex: To change first\_name of 1234 from ‗RAMANA‘ TO ‗RAM‘

UPDATE persons set FIRST\_NAME=‘RAM‘ WHERE PERSON\_ID=1234;

**DELETE Command**

The DELETE command in SQL is used to delete row(s) from the table. The syntax of DELETE command is

DELETE FROM table name WHERE condition;

Ex: TO delete all rows from a table

DELETE FROM PERSONS;

Ex: to delete a particular row

DELETE from PERSONS where person\_id=1234;

**Transaction Control Language (TCL)**

TCL is a computer language and a subset of SQL, used to control transactional processing in a database. A transaction is logical unit of work that comprises one or more SQL statements, usually a group of DML statements.

TCL commands include:

**▪ COMMIT** This command is used to end a transaction. Only with the help of the commit command, transaction changes can be made permanent to the database. and also apply the transaction by saving the database changes.

**▪ ROLLBACK** this command is used to undo all changes of a transaction. we can either rollback the entire transaction so that all changes made by SQL statements are undone.

**▪ SAVE POINT** save points are like markers to divide a very lengthy transaction to smaller ones, They are used to identify a point in a transaction to which we can later rollback.

The general format is to issue a BEGIN WORK statement, one or more SQL statements, and then the COMMIT statement. Alternatively, a ROLLBACK statement can be issued, which undoes all the work performed since BEGIN WORK was issued. In terms of transactions, the opposite of commit is to discard the tentative changes of a transaction, a rollback.

**Data Query Language**

DQL is used to fetch the data from the database.

It uses only one command: **SELECT**

**SELECT Statement**

The SELECT command is the ***most used*** command in SQL. It allows database users to retrieve the specific information they desire from an operational database.

**Syntax**

SELECT [ \* | DISTINCT] select\_list [INTO [new\_table\_name]] [FROM table\_name [, table\_name2][..., table\_name16]]] [WHERE clause] [GROUP BY clause] [HAVING clause] [ORDER BY clause];

**\*(ALL)**: Retrieves all rows in the results. ALL is the default.

**DISTINCT**: Includes only unique rows in the results. Null values are considered equal for the purposes of the DISTINCT keyword; only one NULL is selected no matter how many are encountered.

**Select list**: Specifies the columns to select. Can be one or more of the following:

**Asterisk (\*)** representing all columns listed in the order in which they were specified in the CREATE TABLE statement for all tables in the FROM clause, in the order they appear.

- A list of column names, specified in the order in which you want to see them. If the select\_list contains multiple column names, separate the names with commas.

The command shown below retrieves all of the information contained within the EMP table. Note that the asterisk is used as a wildcard in SQL.

SELECT \* FROM EMP;

Alternatively, users may want to limit the attributes that are retrieved from the database.

For example, the Human Resources department may require a list of the last names of all employees in the company. The following SQL command would retrieve only that information:

SELECT last\_name FROM personal\_info;

**WHERE Clause:**

WHERE clause is used to extract only those records that fulfill a specified criterion.

SELECT \* FROM Persons WHERE City='KURNOOL';

**Clauses in SQL**

One of the most important advantages of SQL is its ability to produce complex free-form queries. SQL provides useful functions that count, find minimum and maximum values, calculate averages, and so on.

Better yet, SQL allows the user to limit queries to only those entries that have no duplicates or entries whose duplicates can be grouped.

**ORDER BY**

The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default. If you want to sort the records in a descending order, you can use the **DESC** keyword.

SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s) ASC|DESC;

EX1: SELECT \* FROM Persons ORDER BY LastName;

EX 2: SELECT \* FROM Persons ORDER BY LastName DESC;

Multilevel ordered sequence is known as a cascading order sequence, and it can be created easily by listing several attributes, separated by commas, after the ORDER By clause.

**DISTINCT clause**

Sometimes you will want to list only the different (distinct) values in a table. The DISTINCT keyword can be used to return only distinct (different) values.

Syntax: SELECT DISTINCT column\_name(s) FROM table name ;

Ex: SELECT DISTINCT City FROM Persons;

**Group by clause**

GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

SELECT Column1, Column2, …, Column\_N, Aggregate Function (Expression) FROM Tables WHERE <cond> GROUP BY <Column1, Column2, ...> HAVING <COND>;

Aggregate function can be a function such as SUM, COUNT, MIN, or MAX.

Group By Having will group values that have a particular value. This can be used in conjunction with other logical functions such as MIN, MAX, COUNT, and SUM.

The **HAVING clause** filters rows after the grouping with the Oracle GROUP BY clause. Oracle GROUP BY HAVING can be used to limit the returned rows after the grouping.

Ex: SUM function to return the name of the department and the total sales

SELECT department, SUM (sales) as "Total sales" FROM order details GROUP BY department;

Ex2: Select job, sum(sal) from emp group by job having job in(‗SALESMAN‘,‘CLERK‘);

**SQL Aggregate Functions**

SQL Aggregate Function is built-in functions for counting and calculations (perform a calculation on a set of values and return a single value).

Syntax SELECT function (column) FROM table

**AVG** - Average value of columns

Select AVG (Sal) FROM EMP;

**COUNT** - number of rows

Select COUNT (\*) FROM EMP;

SELECT COUNT (DISTINCT Sal) FROM CUSTOMERS ;

**MAX** - Maximum or Highest number in a column

SELECT MAX (Salary) FROM CUSTOMERS ;

**MIN -** Minimum or Lowest number in a column

SELECT MIN (Salary) FROM CUSTOMERS ;

**SUM** - Total number in a column

SELECT SUM (Salary) FROM CUSTOMERS ;

**Operators**

SQL supports different type of operators

* Arithmetic operators

+ - \* /

* Relational operators

=(equal to) **< > or !- or ~=** (not equal) > < >= <=

* Logical operators

AND OR NOT

* Comparison operators

IN LIKE IS BETWEEN

**Logical Operators**

The AND operator displays a record if both the first condition and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

AND Ex: SELECT \* FROM Persons WHERE FirstName='ANAND' AND LastName='SAI' ;

OR Ex: SELECT \* FROM Persons WHERE FirstName='RAVI' OR FirstName='RAM' ;

**Comparison operators**

**IN** Operator

The IN operator allows you to specify multiple values in a WHERE clause. In is used as a shorthand for **multiple OR** conditions.

Syntax

Where col IN (VAL1,VAL2,…)

Ex:

Q. List the emps who are working in 30,20 depts

select \* from emp where deptno=20 or deptno=30;

(or)

select \* from emp where deptno IN(20,30);

Q. LIST the details of WARD,JONES,SMITH

Select\*from EMP WHERE ename in('WARD' , 'JONES', 'SMITH');

Q . List the emps who are not working as CLERK,SALESMAN

select ENAME,JOB,SAL from emp where job NOT IN( 'CLERK','SALESMAN');

(or)

select \* from emp where job!='CLERK' AND JOB!='SALESMAN';

**Between & and Operator:** To check the range of values.

The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

syntax

col between min and max

Q List the emps who are earning 1000 to 2000

select EMPNO,ENAME,SAL from emp where sal>=1000 and sal<=2000;

or

select \* from emp where sal between 1000 and 2000;

Q. list the emps who are not earning between 1400 to 2500

select \* from emp where sal not between 1400 and 2500;

Q list the emps who are joined not in 81

'01-jan-81' to '31-dec-81'

select \* from emp where hiredate <= '01-jan-81' and hiredate >='31-dec-81';

between and

select ename,job,sal,hiredate from emp where hiredate NOT between '01-jan-81' and '31-dec-81';

Q List the emps who are not joined in 82 and also clerk

select \* from emp where hiredate not between '01-jan-82' and '31-dec-82' AND job='CLERK' ;

**LIKE** Operator

similar to equal to ' = ', to check char and date types

syntax

col like value

ex: list the details of ALLEN

select \* from emp where ename ='ALLEN';

OR

select \* from emp where ename LIKE 'ALLEN';

Q LIST emps who are not working as clerk

select \* from emp where job not like 'CLERK';

OR

select \* from emp where job != 'CLERK';

**Wild cards**

There are two wildcards often used in conjunction with the LIKE operator:

* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

To represent unknown chars

clerk

clark phanindra phanendra

cl\_rk phan\_ndra

SELECT JOB FROM EMP WHERE JOB LIKE 'CL\_RK';

Q. List the emps who are having max 4 chars in their names

SELECT ENAME FROM EMP WHERE ENAME LIKE '\_\_\_\_';

list the emps who are having A as 3rd char

select ename from emp where ename like '\_\_A%';

Q List the emps who are having A as first char

select ename from emp where ename = 'A%';

Q List the emps who are having A as first char and also working in 30th dept.

select ename from emp where ename like 'A%' and deptno=30;

Q List the emps who are having R in their name OR NOT

select ename,JOB from emp where ename like '%R';

select ename,JOB from emp where ename like '%S%'

Q List the emps who are having R in their names and working as clerk

select \* from emp where ename like '%R%' AND JOB='CLERK';

Q List the emps who are having S as first char,T as last char

select ename from emp where ename like 'S%' and ename like '%T';

or

select ename from emp where ename like 'S%T';

Q List the emps who are NOT having S in THEIR NAMES

select ename from emp where ename NOT like '%S%' ;

**Like using Dates**

List the emps who are joined in the year of 81

select ename,job,hiredate from emp where hiredate like '%81';

List the emps who are joined in the month of December

dd-mon-yy

dd-dec-yy

'%DEC%'

select ename,job,hiredate from emp where hiredate like '%DEC%';

LIST THE EMPS who are not joined in may,but joined in april

select ename,job,hiredate from emp where hiredate like '%MAY%' OR HIREDATE LIKE '%APR%';

List the emps except in the month of Decemeber

select ename,job,hiredate from emp where hiredate not like '%DEC%';

**IS Operator**

to check null or not null values

null means Empty, not even 0 or ' '

Syntax

col IS null

col is not null

List the emps who are not getting commision

select \* from emp where Comm is NULL;

List the emps who are getting commision

select \* from emp where Comm is NOT NULL;

List the emp who is the boss of all

select \* from emp where MGR IS NULL;

**Joining Database Tables(Joins)**

The ability to combine (Join) tables on common attributes is perhaps the most important distinction between a relational database and other databases. A join is performed when data are retrieved from more than one table at a time.

To join tables, you simply list the tables in the FROM clause of the SELECT statement. The DBMS will create the Cartesian product of every table in the FROM clause. However, to get the correct result, you must select only the rows in which the common attribute values match. We use the WHERE clause to indicate the common attributes used to link the tables.

The join condition is generally composed of an equality comparison between the foreign key and the primary key of related tables.

**For example,** suppose you want to join the two tables EMP and DEPT. Because DEPTNO is the foreign key in the EMP table and the primary key in the DEPT table, the link is established on DEPTNO.

To join the EMP and DEPT tables, you would use the following,

SELECT EMPNO,ENAME,JOB,SAL,DNAME,LOC FROM EMP,DEPT WHERE EMP.DEPTNO=DEPT.DEPTNO;

When joining three or more tables, you need to specify a join condition for each pair of tables. The number of join conditions will always be N-1, where N represents the number of tables listed in the FROM clause.

**For example**, if you have three tables, you must have two join conditions; if you have five tables, you must have four join conditions; and so on.

**Joining Tables with an Alias**

An alias may be used to identify the source table from which the data are taken. The aliases E and D are used to label the EMP and DEPT tables in the next command sequence. Any legal table name may be used as an alias.

SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO FROM EMP E, DEPT D where E.DEPTNO=D.DEPTNO;

**Recursive Joins**

An alias is especially useful when a table must be joined to itself in a recursive query.

For example, suppose you are working with the EMP table. Using the data in the EMP table, you can generate a list of all employees with their managers' names by joining the EMP table to itself.

SELECT E.MGR, M.ENAME, E.EMPNO, E.ENAME FROM EMP E, EMP M WHERE E.MGR=M.EMPNO;

**Outer join**

An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both).

**Left outer join**

Left outer join returns all the values from the left table, plus matched values from the right table If the right table returns one row and the left table returns more than one matching row for it, the values in the right table will be repeated for each distinct row on the left table.

For example, this allows us to find an employee's department, but still shows the employee(s) even when they have not been assigned to a department

SELECT \* FROM emp, dept WHERE emp.DeptNO = dept.DeptNO (+);

**Right outer join**

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B.

A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).

For example, this allows us to find each employee and his or her department, but still show departments that have no employees. Below is shown an example of right outer join, with the additional result row italicized:

SELECT \* FROM EMP RIGHT OUTER JOIN dept ON emp.DeptNO = dept.DeptNO;

**Relational Set Operators**

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

**MINUS All** distinct rows selected by the first query but not the second.

All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle

evaluates them from the left to right if no parentheses explicitly specify another order.

**UNION** The following statement combines the results with the UNION operator, which eliminates duplicate selected rows. This statement shows how data type must match when columns do not exist in one or the other table:

SELECT part, partnum, to\_date(null) date\_in FROM orders\_list1 UNION SELECT part, to\_date (null), date\_in FROM orders\_list2;

Ex2: SELECT part FROM orders\_list1 UNION SELECT part FROM orders\_list2;

**UNION ALL** : The following statement combines the results with the UNION ALL operator, which does not eliminate duplicate selected rows:

SELECT part FROM orders\_list1 UNION ALL SELECT part FROM orders\_list2;

**Note** that the UNION operator returns only distinct rows that appear in either result, while the UNION ALL operator returns all rows. A PART value that appears multiple times in either or both queries (such as 'FUEL PUMP') is returned only once by the UNION operator, but multiple times by the UNION ALL operator.

**INTERSECT**

The following statement combines the results with the INTERSECT operator, which returns only those rows returned by both queries:

SELECT part FROM orders\_list1 INTERSECT SELECT part FROM orders\_list2;

**MINUS**

The following statement combines results with the MINUS operator, which returns only rows returned by the first query but not by the second:

SELECT part FROM orders\_list1 MINUS SELECT part FROM orders\_list2;

**Sub Queries**

A sub query is a query within a query. In Oracle, you can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

**WHERE clause**

Most often, the sub query will be found in the WHERE clause. These sub queries are also called nested sub queries.

For example:

Select \* from all\_tables tabs where tabs.table\_name IN (Select cols.table\_name from

all\_tab\_columns cols where cols.column\_name = 'SUPPLIER\_ID');

**NOTE**: Oracle allows up to 255 levels of sub queries in the WHERE clause.

FROM clause

A subquery can also be found in the FROM clause. These are called inline views.

For example:

Select suppliers.name, subquery1.total\_amt from suppliers, (select supplier\_id, Sum (orders. amount) as total\_amt from orders group by supplier\_id) subquery1, where subquery1.supplier\_id = suppliers.supplier\_id;

In this example, we've created a sub query in the FROM clause as follows:

(select supplier\_id, Sum(orders.amount) as total\_amt from orders group by supplier\_id) subquery1

This sub query has been aliased with the name subquery1. This will be the name used to reference this sub query or any of its fields.

**NOTE**: Oracle allows an unlimited number of sub queries in the FROM clause.

**SQL Functions**

SQL functions are similar to SQL operators in that both manipulate data items and both return a result. SQL functions differ from SQL operators in the format in which they appear with their arguments. The SQL function format enables functions to operate with zero, one, or more arguments.

SQL functions are used exclusively with SQL commands within SQL statements. There are two general types of SQL functions: single row (or scalar) functions and aggregate functions. These two types differ in the number of database rows on which they act. A single row function returns a value based on a single row in a query, whereas an aggregate function returns a value based on all the rows in a query.

What is a **DUAL** Table in Oracle?

This is a single row and single column dummy table provided by oracle. This is used to perform mathematical calculations without using a table.

Select \* from DUAL;

**1.Numeric Functions:**

The Oracle numeric functions take a numeric input as an expression and return numeric values. The return type for most of the numeric functions is NUMBER.

**Function** **Description**

ABS Calculates the absolute value of an expression.

ACOS Calculates the angle value (in radians) of a specified cosine.

ASIN Calculates the angle value (in radians) of a specified sine.

ATAN Calculates the angle value (in radians) of a specified tangent.

ATAN2 Returns a full-range (0 - 2 pi) numeric value indicating the arc tangent of a given ratio.

BITAND Computes an AND operation on the bits of two integers.

CEIL Returns the smallest whole number greater than or equal to a specified number.

COS Calculates the cosine of an angle expression.

COSH Calculates the hyperbolic cosine of an angle expression.

EXP Returns e raised to the nth power, where e equals 2.71828183....

FLOOR Returns the largest whole number equal to or less than a specified number.

LN Returns the natural logarithm of an expression.

LOG Computes the logarithm base 10 of an expression.

MOD Returns the modulus of a number.

POWER Returns m\_value raised to the n\_value power

REMAINDER Returns the remainder after one numeric expression is divided by another.

ROUND Returns the number rounded to the nearest multiple of a second number you specify or to the number of decimal places indicated by the second number.

SIGN Returns a value that indicates if a specified number is less than, equal to, or greater than 0 (zero).

SQRT Computes the square root of an expression.

TAN Calculates the tangent of an angle expression.

TANH Calculates the hyperbolic tangent of an angle expression.

TRUNC Truncates a number to a specified number of decimal places.

**2.Character Functions:**

The Oracle Character functions that take the input argument is CHAR or VARCHAR2.

**Function** **Description**

CONCAT The Oracle CONCAT() function returns the result (a string) of concatenating two string values. This function is equivalent to the concatenation operator (||).

INITCAP The Oracle INITCAP() function sets the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

LOWER The Oracle LOWER() function returns a specified character expression in lowercase letters.

LPAD The Oracle LPAD() function is used to pad the left side of a string with a specific set of characters. The function is useful for formatting the output of a query.

LTRIM The Oracle LTRIM() function is used to remove all specified characters from the left end side of a string. Optionally you can specify an initial character or characters to trim to, or it will default to a blank.

CONCAT The Oracle CONCAT() function returns the result (a string) of concatenating two string values. This function is equivalent to the concatenation operator (||).

INITCAP The Oracle INITCAP() function sets the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

LOWER The Oracle LOWER() function returns a specified character expression in lowercase letters.

LPAD The Oracle LPAD() function is used to pad the left side of a string with a specific set of characters. The function is useful for formatting the output of a query.

LTRIM The Oracle LTRIM() function is used to remove all specified characters from the left end side of a string. Optionally you can specify an initial character or characters to trim to, or it will default to a blank.

SUBSTR The SUBSTR function returns the specified number (substring\_length) of characters from a particular position of a given string. SUBSTRB uses bytes instead of characters. SUBSTRC uses Unicode complete characters. SUBSTR2 uses UCS2 code points. SUBSTR4 uses UCS4 code points.

TRIM The Oracle TRIM function is used to remove all leading or trailing characters (or both) from a character string. If trim\_character or trim\_source is a character literal, then it is necessary to enclose it in single quotation marks.

UPPER The Oracle UPPER() function returns a specified character expression in UPPERCASE letters.

ASCII The ASCII() function returns the decimal representation of the first character of a character expression.

INSTR The Oracle INSTR function is used to search string for substring and find the the location of the substring in the string. If a substring that is equal to substring is found, then the function returns an integer indicating the position of the first character of this substring. If no such substring is found, then the function returns zero.

LENGTH The Oracle LENGTH function is used to return the length of a given string. If the string has data type CHAR, then the length includes all trailing blanks. If the string is null, then this function returns null.

**Date/Time Functions**

Datetime functions operate on a date (DATE), timestamp (TIMESTAMP, TIMESTAMP WITH

TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE), and interval (INTERVAL DAY

TO SECOND, INTERVAL YEAR TO MONTH) values.

**SYSDATE** SYSDATE returns the current date and time.

**Function** **Description**

ADD\_MONTHS ADD\_MONTHS returns a date (date plus integer months).

CURRENT\_DATE CURRENT\_DATE returns the current date in the session time zone,

in a value in the Gregorian calendar of datatype DATE.

CURRENT\_TIMESTAMP The CURRENT\_TIMESTAMP() function returns the current date and time in the session time zone, in a value of datatype TIMESTAMP WITH TIME ZONE.

LAST\_DAY LAST\_DAY() function returns the date of the last day of the month that contains a date.

MONTHS\_BETWEEN MONTHS\_BETWEEN() function returns the number of months between dates (date1, date2).

NEW\_TIME NEW\_TIME() function converts a date from timezone1 to a date in timezone2.

NEXT\_DAY NEXT\_DAY returns the date of the first weekday that is later than the date.

SYSTIMESTAMP SYSTIMESTAMP function returns the system date, including fractional seconds and time zone.

TO\_CHAR(datetime) TO\_CHAR (datetime) function returns a datetime or interval value of DATE. TIMESTAMP, TIMESTAMP WITH TIME ZONE.

**View (Logical Table)**

A view is a virtual table that does not physically exist. It is stored in Oracle data dictionary and do not store any data. It can be executed when called. A view is created by a query joining one or more tables.

Every view has columns with data types so you can execute a query against views or manage their contents (with some restrictions) using the INSERT, UPDATE, DELETE, and MERGE statements.

Unlike a table, a view does not store any data. To be precise, a view only behaves like a table. And it is just a named query stored in the database. When you query data from a view, Oracle uses this stored query to retrieve the data from the underlying tables.

You can use views in many cases for different purposes. The most common uses of views are as

follows:

1. Simplifying data retrieval.
2. Maintaining logical data independence.
3. Implementing data security.
4. Views can act as aggregated tables, where the database engine aggregates data (sum, average etc.) and presents the calculated results as part of the data
5. Views can hide the complexity of data

For example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table

1. Depending on the SQL engine used, views can provide extra security
2. Views can limit the degree of exposure of a table or tables to the outer world

**Syntax for creating a VIEW is:**

CREATE VIEW view\_name AS SELECT columns FROM table WHERE condition;

Ex: Create View Emp\_View As Select Empno,Ename,Job,Sal From Emp Where Sal>2000 And Hiredate Like ‗%80‘;

This would create a virtual table based on the result set of the select statement. You can now query the view as follows:

SELECT \* FROM EMP\_view;

**Sequence**

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

A sequence is a user defined schema bound object that generates a sequence of numeric values. Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.

The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when exceeds max\_value.

Syntax:

CREATE SEQUENCE sequence\_name

START WITH initial\_value

INCREMENT BY increment\_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE|NOCYCLE ;

sequence\_name: Name of the sequence.

initial\_value: starting value from where the sequence starts.

Initial\_value should be greater than or equal to minimum value and less than equal to maximum value.

increment\_value: Value by which sequence will increment itself, can be positive or

negative.

minimum\_value: Minimum value of the sequence.

maximum\_value: Maximum value of the sequence.

cycle: When sequence reaches its set\_limit it starts from beginning.

nocycle: An exception will be thrown if sequence exceeds its max\_value.

ex: create sequence nos start with 50 minvalue 10 maxvalue 99

increment by 10 NOCYCLE nocache;

SQL> select nos.currval from dual;

CURRVAL

----------

1009

select nos.nextval from dual;

NEXTVAL

--------

1000