SQL

Structured Query Language (SQL) is the set of statements with which all programs and users access data in an Oracle database. The language, Structured English Query Language ("SEQUEL") was developed by IBM Corporation, Inc. SEQUEL later became SQL (still pronounced "sequel". All major relational database management systems support SQL, so you can transfer all skills you have gained with SQL from one database to another. In addition, all programs written in SQL are portable. They can often be moved from one database to another with very little modification.

SQL has the following advantages:

- Efficient
- Easy to learn and use
- With SQL, you can define, retrieve, and manipulate data in the tables

CHARACTERISTICS:

1. It is a non procedural query language.
2. SQL is common language for most RDBMS.

SQL Standards:-

Oracle SQL complies with industry-accepted standards. . Industry-accepted committees are the American National Standards Institute (ANSI). And the International Standards Organization (ISO) . Both ANSI and ISO have accepted SQL as the standard language for relational databases.

Writing SQL Statements

Using the following simple rules and guidelines, you can construct valid statements that are both easy to read and easy to edit:

- SQL statements are not case sensitive.
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.
- Keywords typically are entered in uppercase; all other words, such as table names and columns, are entered in lowercase.

SQL Statements

SELECT	Data retrieval
INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction control
GRANT REVOKE	Data control language (DCL)

Data types:-

When you create a table or cluster, you must specify a data type for each of its columns. When you create a procedure or stored function, you must specify a data type for each of its arguments. These data types define the domain of values that each column can contain or each argument can have. For example, DATE columns cannot accept the value February 29 (except for a leap year. or the values 2 or 'SHOE'. Each value subsequently placed in a column assumes the column's data type. For example, if you insert '01-JAN-98' into a DATE column, Oracle treats the '01-JAN-98' character string as a DATE value after verifying that it translates to a valid date.

CHAR(size) :-

Fixed-length character data of length size bytes. Maximum size is 2000 bytes. Default and minimum size is 1 byte.

VARCHAR2(size) :-

Variable-length character string having maximum length size bytes or characters. Maximum size is 4000 bytes, and minimum is 1 byte or 1 character. You must specify size for VARCHAR2.

NCHAR(size):-

Fixed-length character data of length size characters or bytes, depending on the choice of national character set. Maximum size is determined by the number of bytes required to store each character, with an upper limit of 2000 bytes. Default and minimum size is 1 character or 1 byte, depending on the character set.

NVARCHAR2(size) :-

Variable-length character string having maximum length size characters or bytes, depending on the choice of national character set. Maximum size is determined by the number of bytes required to store each character, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2.

NUMBER(p,s) :-

Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127

LONG :-

Character data of variable length up to 2 gigabytes, or 231 -1 bytes.

DATE :-

Allows date & time but Time is optional if not entered by user then oracle inserts 12:00AM. Valid date range from January 1, 4712 BC to December 31, 9999 AD. a Date field occupies 7 bytes of memory

RAW(size) :-

Raw binary data of length size bytes. Maximum size is 2000 bytes. You must specify size for a RAW value.

LONG RAW :-

Raw binary data of variable length up to 2 gigabytes.

ROWID :-

Hexadecimal string representing the unique address of a row in its table. This datatype is primarily for values returned by the ROWID pseudocolumn.

UROWID [(size)] :-

Hexadecimal string representing the logical address of a row of an index-organized table. The optional size is the size of a column of type UROWID. The maximum size and default is 4000 bytes.

CLOB :-

A character large object containing single-byte characters. Both fixed-width and variable-width character sets are supported, both using the CHAR database character set. Maximum size is 4 gigabytes.

NCLOB :-

A character large object containing unicode characters. Both fixed-width and variable-width character sets are supported, both using the NCHAR database character set. Maximum size is 4 gigabytes. Stores national character set data.

BLOB :-

A binary large object. Maximum size is 4 gigabytes.

BFILE :-

Contains a locator to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. Maximum size is 4 gigabytes.

BINARY_FLOAT :-

32-bit single precision floating point number datatype. Binary float equires 5 bytes including a length byte.

BINARY_DOUBLE:-

64-bit double precision floating point number datatype. Binary double requires 9 bytes including a length byte.

OPERATORS IN SQL :-

Operators in ORACLE categorized into following categories

ARTHMETIC OPERATORS :-

+ - * /

Operator precedence:-

- ➔ Operators * , / having higher precedence than operators + , -
- ➔ Operators of the same priority are evaluated from left to right.
- ➔ Use parenthesis to control the precedence.

RELATIONAL OPERATORS :-

Used for comparision , different relational operators supported by oracle

Operator	Description
>	Greater than
>=	greater than or equal
<	Less than
<=	less than or equals
=	equal
< >	not equal

LOGICAL OPERATORS :-

AND used to combine two conditions
OR used to combine two conditions
NOT negate condition

SPECIAL OPERATORS :-

||
BETWEEN
IN
LIKE
IS NULL
ANY
ALL
EXISTS

Creating table:-

Different types of tables can be created in ORACLE.

- ➔ Standard tables
- ➔ Partitioned tables
- ➔ Clustered tables
- ➔ Index organized tables
- ➔ External tables
- ➔ Global temporary tables

Standard Table:-

Syntax:-

SQL> CREATE TABLE <Table Name>
(Colname datatype (size),
Colname datatype (size),
----------.) ;

Rules for creating a table :-

- tablename should start with alphabet
- tablename should not contain spaces or special symbols , but allows _ , $, #
- tablename should not be a oracle reserved word
- tablename can contain max 30 chars
- a table can contain max of 1000 columns

Example:-

```
SQL> CREATE TABLE emp
        (empno NUMBER(4) , ename VARCHAR2(20) ,
         job    VARCHAR2(10) ,  hiredate DATE,
        sal     NUMBER(6,2) , comm NUMBER(6,2)  ,
        deptno  NUMBER(2) ) ;
```

Inserting Data into a Table:-

INSERT command is used to insert record into a table.

Syntax:-

INSERT INTO <table name> VALUES(list of values)

Note :- Strings and Dates must be enclosed in single quotes.

Example :-

SQL>INSERT INTO emp VALUES(1000,'BLAKE','MANAGER', '10-JAN-10',5000,500,10);

NOTE:-

Order of values in the INSERT command should match with order of columns declared in table. to insert values in different order then we need to specify the order.

Inserting NULL values:-

→NULL values are inserted when value is

- Absent
- Unknown
- Not Applicable

→NULL is not equal to 0 and not equal to space

→NULL values can be inserted in two ways.

- ➔ EXPLICITLY
- ➔ IMPLICITLY

Inserting NULL values EXPLICITLY:

- to insert Null values into Numeric columns use NULL keyword.
- To insert Null values into character & date columns use '' .

Example :-

SQL>INSERT INTO emp VALUES(1002,'JAMES','',5000,NULL,10);

Inserting NULL values IMPLICITLY :-

Example :-

SQL> INSERT INTO emp(EMPNO,ENAME,SAL,DEPTNO.
VALUES(1005,'SMITH',2000,10) ;

Remaining columns are automatically filled with NULL values.

Inserting MULTIPLE records :-

The same INSERT command can be executed number of times with different values by using substitution variables. Substitution variables can be declared by using

Single ampersand (&.
Double ampersand (&&.

These variables stores data temporarily

Using Single ampersand :-

These variables are prefixed with &. Values assigned to these variables exists upto the command , once command execution is completed values assigned to these variables are erased.

Example:-

SQL>INSERT INTO emp VALUES(&empno,'&ename','&job',
'&hiredate',&sal,&comm,&deptno) ;

Using Double Ampersand :-

These variables are prefixed with &&. Values assigned to these variables even after execution of INSERT command upto the end of session.

Example :-

SQL>INSERT INTO emp VALUES
(&empno,'&ename','&&job',&&sal,'&&hiredate',&deptno);

SQL>SELECT * FROM employees;

ID	FIRST_NAME	LAST_NAME	SALARY	COMM	HRA
1	JOHN	DOE	1000	500	300

Data Retrieval :-

SELECT statement can be used to retrieve data from database.

Capabilities of SELECT Statement:-

Using a **SELECT** statement, you can do the following :

- **Projection** :- You can use the projection capability in SQL to choose the columns in a table that you want . You can choose as few or as many columns of the table as you require.
- **Selection:**- You can use the selection capability in SQL to choose the rows in a table that you . You can use various criteria to restrict the rows that you see.
- **Joining:** You can use the join capability in SQL to bring together data that is stored in different tables by creating a link between them.

Syntax :-

SELECT * / {*column*|*expression* [*alias*],. . . } FROM *table;*

In the syntax :-

- A SELECT clause, which specifies the columns to be displayed
- A FROM clause, which specifies the table containing the columns listed in the SELECT clause

Selecting All Columns :-

SQL>SELECT * FROM dept;

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

Selecting Specific Columns :-

Display only empno,ename,job,sal from emp table ?

SQL>SELECT empno, ename, job, sal, FROM emp;

NOTE:-

→Date and Character data aligned to LEFT

→Numeric data aligned to RIGHT

Arthmetic Expressions :-

an arithmetic expression contain column names,constant numeric values and athmetic operator.

Example :-

Display ename ,sal, annual salaries ?

SQL>SELECT ename, sal, sal*12 FROM emp;

Operator precedence :-

• Multiplication and division take priority over addition and subtraction.

• Operators of the same priority are evaluated from left to right.

• Parentheses are used to force prioritized evaluation and to clarify statements

Example :-

SQL>SELECT ename, sal, 12*sal+100 FROM emp;

The above example displays the ename, sal, and annual sal of employees. It calculates the annual sal as 12 multiplied by the monthly salary, plus a one-time bonus of 100. Notice that multiplication is performed before addition.

You can override the rules of precedence by using parentheses.

SQL>SELECT ename, sal, 12*(sal+100) FROM emp;

Concatenation Operator:-

This operator concatenates two strings represented by two vertical bars || .

Example :-

SQL>SELECT ename||' working as '||job FROM emp;

SQL>SELECT ename||' joined on '||hiredate FROM emp;

Literals in ORACLE:-

A Literal is a Constant

Typesof Literals :-

- String constant
- Numeric constant

- Date constant

NOTE :- String constant and Date constants must be enclosed in ‘ ‘.

Example :-

SQL>SELECT ename || ‘ EARNS ‘|| sal*12 ||’ PER YEAR’ FROM emp;

Declaring Alias:-

An Alias is an another name or alternative name, aliases in Oracle are of two types.

- Column Alias
- Table Alias

Column Alias :-

Alias declared for column is called column alias.

Syntax :-

COLNAME / EXPR [AS] ALIAS

→If alias contains spaces or special characters then alias must be enclosed in “ “

→The scope of the alias is upto that query.

Example :-

Display ename,sal,comm and in report display sal as basic and comm as bonus ?

SQL>SELECT ename,sal AS basic,comm AS bonus FROM emp;

Display ename , annual salary ?

SQL>SELECT ename,sal*12 AS “ANNUAL SALARY” FROM emp;

Display ename,sal,hra,da,tax,totsal ?

SQL>SELECT ename,sal,sal*0) 3 AS hra,sal*0) 2 AS da, sal*0) 1 AS tax ,
sal+(sal*0) 3) +(sal*0) 2) -(sal*0) 1) AS totsal FROM emp;

Clauses in ORACLE :-

- WHERE
- ORDER BY
- DISTINCT
- GROUP BY
- HAVING
- ON
- USING
- START WITH
- CONNECT BY
- WITH
- RETURNING
- FOLLOWS

- MODEL

Data Filtering using WHERE clause:-

You can restrict the rows returned from the query by using the WHERE clause) A WHERE clause contains a condition that must be met, and it directly follows the FROM clause) If the condition is true, the row meeting the condition is returned)

syntax:

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

WHERE restricts the rows that meet a condition)

condition is composed of column names, expressions,constants, and a comparison operator

It consists of three elements:

- Column name
- Comparison operator
- Column name, constant, or list of values

Examples :-

Display employee record whose empno=7844 ?

SQL>SELECT * FROM emp WHERE empno=7844 ;

Display employee records whose job='CLERK' ?

SQL>SELECT * FROM emp WHERE job='CLERK' ;

Display employee records working for 10 dept and working as CLERK ?

SQL>SELECT * FROM emp WHERE deptno=10 AND job='CLERK';

Display employee records working as CLERK OR MANAGER ?

SQL>SELECT * FROM emp WHERE job='CLERK' OR job='MANAGER' ;

Display employee records earning between 2000 and 5000 ?

SQL>SELECT * FROM emp WHERE sal>=2000 AND sal<=5000;

Display employee records joined after 1981 ?

SQL>SELECT * FROM emp WHERE hiredate > '31-DEC-1981' ;

Expect the output of the following Query ?

SQL>SELECT * FROM emp

WHERE job='CLERK' OR job='MANAGER' AND sal>2000 ;

BETWEEN operator:-

You can display rows based on a range of values using the BETWEEN operator) The range that you specify contains a lowerlimit and an upperlimit) Values specified with the BETWEEN condition are inclusive) You must specify the lower limit first)

Syntax:- BETWEEN value1 and value2

Example :-

Display employee records earning between 2000 and 5000 ?

SQL>SELECT * FROM emp WHERE sal BETWEEN 2000 AND 5000;

Note:-

BETWEEN … AND … is actually translated by Oracle server to a pair of AND conditions: (a >=lower limit) AND (a <= higher limit)) So using BETWEEN … AND … has no performance benefits, and it is used for logical simplicity)

Example :-

Display employee records who are joined between 1981 year?

```
SQL>SELECT * FROM emp
        WHERE hiredate BETWEEN '01-JAN-1981' AND '31-DEC-1981' ;
```

Display employee records who are not joined in 2000 year ?

```
SQL>SELECT * FROM emp
    WHERE hiredate NOT BETWEEN '01-JAN-2000' AND '31-DEC-2000' ;
```

OCA question :-

Expect the output of the following query ?

SQL>SELECT * FROM emp WHERE sal BETWEEN 5000 AND 2000 ;

A error B returns records C returns no rows D none

IN operator :-

To test for values in a specified list of values, use IN operator) The IN operator can be used with any data type) If characters or dates are used in the list, they must be enclosed in single quotation marks (''))

Syntax:-

IN (V1,V2,V3----------) ;

Example :-

Display employee records working as CLERK OR MANAGER ?

SQL>SELECT * FROM emp WHERE job IN ('CLERK','MANAGER') ;

Display employee records not working for dept 10 or 20 ?

SQL>SELECT * FROM emp WHERE deptno NOT IN (10,20)

Note :-

IN ( ) is actually translated by Oracle server to a set of OR conditions: a =value1 OR a = value2 OR a = value3) so using IN ( ) has no performance benefits, and it is used for logical simplicity)

LIKE operator:-

You may not always know the exact value to search for) You can select rows that match a character pattern by using the LIKE operator) The character pattern-matching operation is referred as *wildcard* search)

Syntax:-

LIKE 'pattern'
NOT LIKE 'pattern'

Pattern consists of alphabets,digits and metacharacters) The different meta characters in ORACLE

% denotes zero or many characters)
_ denotes one character)

Display employee records whose name starts with S ?
SQL>SELECT * FROM emp WHERE ename LIKE 'S%';

Display employee records whose name ends with S ?
SQL>SELECT * FROM emp WHERE ename LIKE '%S';
Display employee records whose name doesn't contain S ?
SQL>SELECT * FROM emp WHERE ename NOT LIKE '%S%';
Display employee records where A is the second char in their name?
SQL>SELECT * FROM emp WHERE ename LIKE '_A%';
Display employee records who are joined in JAN month?
SQL>SELECT * FROM emp WHERE hiredate LIKE '%JAN%' ;
Display employee records who are joined in 1981 year ?
SQL>SELECT * FROM emp WHERE hiredate LIKE '%81';
Display employee records who are joined in 1st 9 days ?
SQL>SELECT * FROM emp WHERE hiredate LIKE '0%' ;
Display employee records who are earning 5 digits salary ?
SQL>SELECT * FROM emp WHERE sal LIKE '_____' ;
Display employee records whose name contains _ ?
SQL>SELECT * FROM EMP WHERE ENAME LIKE '%_%' ESCAPE '\' ;
Expect the output of the following query
SQL>SELECT * FROM EMP WHERE JOB IN ('CLERK','%MAN%') ;

OCA question :-

You need to extract details of those products in the SALES table where the PROD_ID column contains the string '_D123') ?

IS operator :

The IS operator tests for nulls. A null value means the value is unavailable, unassigned,unknown, or inapplicable. Therefore, you cannot test with = because a null cannot be equal or unequal to any value.

Syntax :-

IS NULL
IS NOT NULL

Example:-

Display employee records whose comm) Is null ?

SQL>SELECT * FROM emp WHERE comm IS NULL ;

Display employee records whose comm) Is not null ?

SQL>SELECT * FROM emp WHERE comm IS NOT NULL ;

Operator Precedence :-

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

NOTE:- we can override rules of precedence by using parentheses).

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. If you use the ORDER BY clause, it must be the last clause of the SQL statement. You can specify an expression, or an alias, or column position in ORDER BY clause.

Syntax:-

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

Examples :-

Arrange employee records in ascending order of their sal ?

SQL>SELECT * FROM emp ORDER BY sal ;

Arrange employee records in descending order of their sal ?

SQL>SELECT * FROM emp ORDER BY sal DESC;

Display employee records working for 10^{th} dept and arrange the result in ascending order of their sal ?

SQL>SELECT * FROM emp WHERE deptno=10 ORDER BY sal ;

Arrange employee records in ascending of their deptno and with in dept arrange records in descending order of their sal ?

SQL>SELECT * FROM emp ORDER BY deptno,sal DESC ;

In ORDER BY clause we can use column name or column position , for example

SQL>SELECT * FROM emp ORDER BY 5 DESC ;

In the above example records are sorted based on the fifth column in EMP table) Arrange employee records in descending order of their comm) If comm) Is null then arrange those records last ?

SQL>SELECT * FROM emp ORDER BY comm DESC NULLS LAST ;

DML commands :-

- ➔ INSERT
- ➔ UPDATE
- ➔ DELETE
- ➔ INSERT ALL
- ➔ MERGE

Copying Data from one table to another table :-

Syntax:-

INSERT INTO <TARGETTABLE>
SELECT <COLLIST> FROM <SOURCE TABLE>

Example :-

SQL>INSERT INTO emp_temp
SELECT * FROM emp;

In the above example first SELECT statement gets data from EMP table and inserts data into EMP_TEMP table and command will be successful only if both tables structure is same.

UPDATE command:-

Update command is used to modify data in a table)

Syntax:-

UPDATE table SET column = value[, column = value,……] [WHERE condition];

Examples :-

Update all employees commission to 500 ?

SQL>UPDATE EMP SET comm=500 ;

Update employee comm to 500 whose comm) Is null ?

SQL>UPDATE EMP SET comm=500 WHERE comm IS NULL ;

Increment employee salary by 10% and comm) By 20% Those who are working as SALESMAN ?

SQL>UPDATE EMP SET sal=sal*1) 1 , comm=comm*1) 2 WHERE job='SALESMAN' ;

Update different employees comm) With different values ?

SQL>UPDATE EMP SET comm = &comm WHERE empno=&empno;

Update the column value with DEFAULT value ?

SQL>UPDATE EMP SET hiredate=DEFAULT WHERE empno=7844;

Returning Clause:-

→returning clause is used to return values into variables after update)

→To use returning clause declare bind variable (session-level variables)

→Bind variables are declared at SQL prompt , and accessed using : operator)

SQL>variable sumsal number ;

SQL>UPDATE emp SET sal=sal*1) 2 Where deptno=10
RETURNING SUM(sal) INTO :sumsal;

SQL> print :sumsal

DELETE command :-

DELETE command is used to delete record or records from a table)

Syntax:-

DELETE FROM <TABNAME> [WHERE <cond> ----] ;

Delete all employee records ?

SQL>DELETE FROM emp ;

Delete employee records whose empno=7844 ?

SQL>DELETE FROM emp WHERE empno=7844 ;

Delete employee records having more than 30 yrs expr ?

SQL>DELETE FROM emp WHERE (SYSDATE-hiredate) /365 >= 3 ;

DDL commands :-

- ➔ CREATE
- ➔ ALTER
- ➔ DROP
- ➔ TRUNCATE
- ➔ RENAME

Creating a table from another table:-

Syntax :-

CREATE TABLE <TABNAME>
AS SELECT STATEMENT [WHERE <cond>];

Example :-

Create table emp11 from table emp ?

```
SQL>CREATE TABLE emp11
      AS
    SELECT * FROM emp;
```

After executing above command a new table is created called emp11 from the result of SELECT Statement.

Copying only structure:-

Create new table emp12 from emp and into the new table copy only structure but do not copy data?

```
SQL>CREATE TABLE emp12
AS
SELECT * FROM emp WHERE 1=2;
```

Because no record in emp table satisfies condition 1=2 , so no record is copied to EMP12 only the structure is copied.

ALTER command:-

ALTER command is used to modify data definition of a table. ALTER command is used to do following operations.

- ➔ ADD A COLUMN(S)
- ➔ DROP A COLUMN(S)
- ➔ TO RENAME A COLUMN
- ➔ MODIFY A COLUMN
  - INCR/DECR FIELD SIZE
  - CHANGING DATATYPE
  - CHANGING FROM NULL TO NOT NULL
  - CHANGING FROM NOT NULL TO NULL)
- ➔ TO MAKE TABLE READ ONLY

Adding a Column:-

Syntax :-

ALTER TABLE <tabname> ADD (colname DATATYPE(SIZE) [, colname -------])

Example:-

SQL>ALTER TABLE emp ADD (dob DATE) ;

Droping a Column:-

Syntax :-

ALTER TABLE <TABNAME> DROP COLUMN COLNAME ;

Example :-

SQL>ALTER TABLE emp DROP COLUMN dob;

SQL>ALTER TABLE emp DROP (ename,sal) ;

NOTE :- all columns in a table cannot be dropped , because the table should contain atleast one column)

Renaming a Column :-

Syntax:-

ALTER TABLE <tabname> RENAME COLUMN <oldname> to <newname> ;

SQL>ALTER TABLE emp RENAME COLUMN sal TO salary ;

Modifying a Column:-

Syntax :-

ALTER TABLE <TABNAME>

MODIFY(COLNAME DATATYPE(SIZE) ,----------)

Increasing / Decreasing Field Size:-

Increase size of ENAME field to 20 ?

SQL> ALTER TABLE emp MODIFY (ename VARCHAR2(20)) ;

NOTE :- 1 char field size can be decremented upto max length)

2 to decrement precision or scale of a numeric field , field must be empty)

Changing Datatype:-

SQL>ALTER TABLE emp MODIFY (ename CHAR(20)) ;

NOTE :-

To change datatype of a column the column should be empty)

Changing Column from NULL to NOT NULL

SQL>ALTER TABLE emp MODIFY (ename NOT NULL) ;

Changing column from NOT NULL to NULL:-

SQL>ALTER TABLE emp MODIFY(ename NULL) ;

Read only Tables :-

From **ORACLE 11g** we can make the table as read only , prior to ORACLE 11g we can do this through view) A read only table doesn't allow DML operations)

SQL>ALTER TABLE emp READ ONLY ;

To make table read , write

SQL>ALTER TABLE emp READ WRITE ;

DROP command :-

DROP command drops a table from database)

Syntax :-

DROP TABLE <TABNAME> ;

Example :-

SQL>DROP TABLE customer;

TRUNCATE command :-

- TRUNCATE command releases memory allocated for a table)
- TRUNCATE deletes all the data from a table)

Syntax :-

TRUNCATE TABLE <TABNAME>

Example :-

SQL>TRUNCATE TABLE EMP ;

Difference between DELETE and TRUNCATE :-

DELETE	**TRUNCATE**
DML command	DDL command
Deletes all or particular records	deletes only all records
Data can be restored	Data cannot be restored
Deletes row by row	doesn't read record before deleting
Used by developer	used by DBA
Triggers can be created	triggers cannot be created

Note :- TRUNCATE is faster than DELETE

RENAME command :-

Used to change name of the table)

Syntax :-

RENAME <OLDNAME> TO <NEWNAME> ;

Example :-

SQL>RENAME emp TO employee;

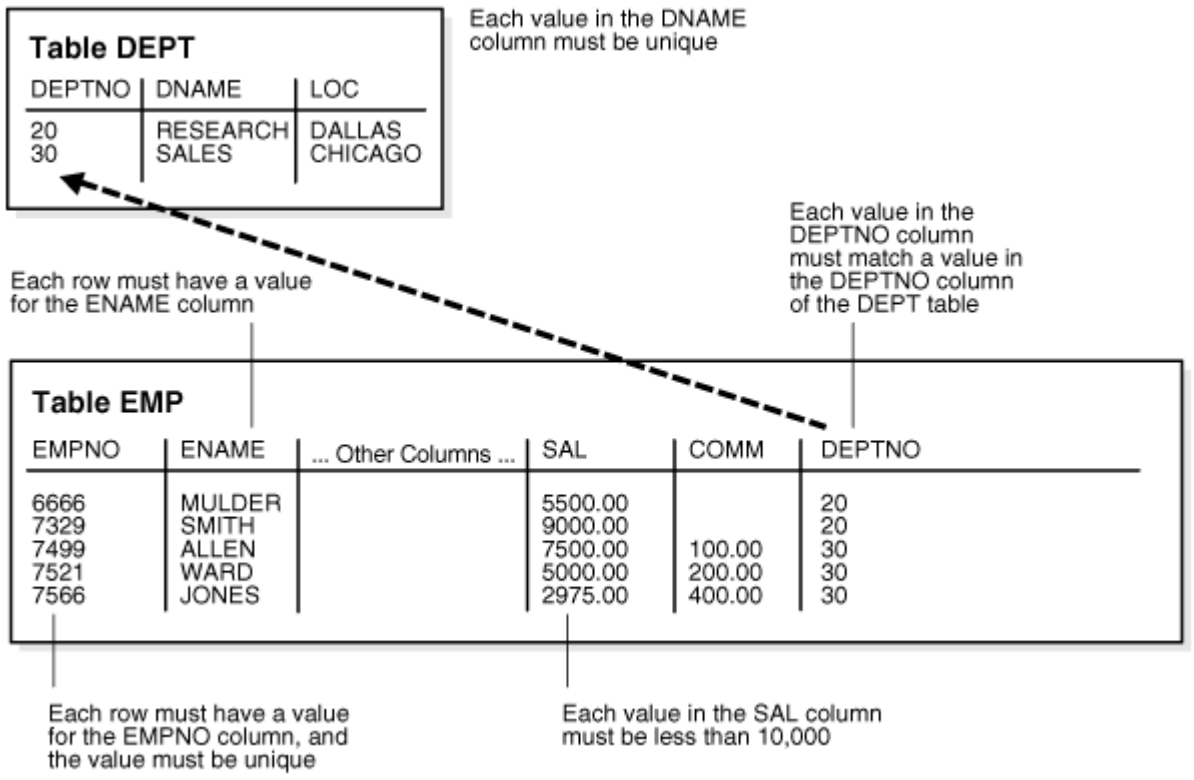
Integrity Constraints

Integrity constraints are the rules in real life, which are to be imposed on the data. If the data is not satisfying the constraints then it is considered as inconsistent. These rules are to be enforced on data because of the presence of these rules in real life. These rules are called integrity constraints. Every DBMS software must enforce integrity constraints, otherwise inconsistent data is generated.

You can use constraints to do the following:

- to prevent invalid data entry into tables.
- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a record from a table if there are dependencies.

Example for Integrity Constraints :-



Types of Integrity Constraints:-

Entity Integrity:-

Entity Integrity constraints are two types

→**Unique Constraint**

→**Primary Constraint**

Refrential Integrity:-

→A refrential integrity constraint states that the values of the foreign key value should match with values of primary key/unique Column of another or same table. Foreign key constraint establishes relationship between tables.

→ The table holding primary key is called parent /master table.

→ The table holding foreign key is called child /detail table.

Self Refrential Integrity :-

If a foreign key in one table refers primary key/unique column of the same table then it is called self refrential Integrity.

Domain constraints:-

A domain means a set of values assigned to a column. Domain constraints are handled by

→defining proper data type

→specifying not null constraint

→specifying check constraint.

Types of Constraints in ORACLE:-

The above said constraints are implemented in oracle with the help of

→**NOT NULL**

→**UNIQUE**

→**PRIMARY KEY**

→**CHECK**

→**FOREIGN KEY**

The above constraints can be declared at

→Column level

→ Table level

Column level :-

→ Constraint is declared immediately declaring column.

→ Use column level to declare constraint for single column.

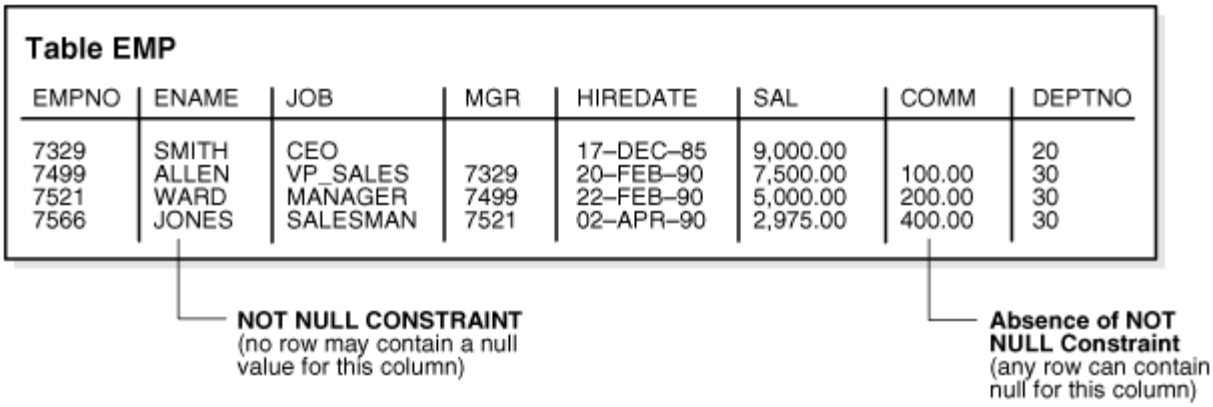
Table level :-

→ use table level to declare constraint for combination of columns.

→ constraint is declared after declaring all columns.

NOT NULL constraint :-

→ It ensures that a table column cannot be left empty.
→ Column declared with NOT NULL is a mandatory column.
→ The NOT NULL constraint can only be applied at column level.



Syntax :-

Columnname Datatype(size) NOT NULL

Example :-

```
SQL> CREATE TABLE emp(
                    Empno      NUMBER(4) ,
                    Ename      VARCHAR2(20)  NOT NULL,
                    Job        VARCHAR2(20)  ,
                    Mgr        NUMBER(4) ,
                    Hiredate   DATE,
                    Sal        NUMBER(7,2) ,
                    Comm       NUMBER(7,2) ,
                    Deptno     NUMBER(2)  ) ;
```

SQL>INSERT INTO emp VALUES(7329,'SMITH','CEO',NULL,'17-DEC-85',9000,NULL,20) ;

1 row created

SQL>INSERT INTO emp VALUES(7499,'', 'VP_SALES',7329,'20-FEB-90',7,500,100,30) ;

ERROR ORA-1400 :- cannot insert null into (scott) dept) dname)

UNIQUE constraint :-

→A column declared with UNIQUE constraint does not accept duplicate values.
→One table can have a number of unique keys.
→By default UNIQUE columns accept null values unless declared with NOT NULL constraint

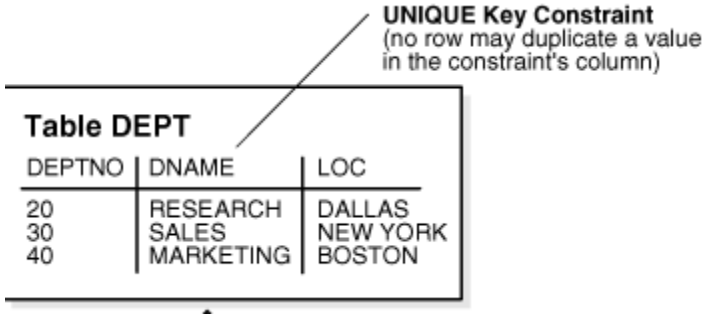
→Oracle automatically creates UNIQUE index on the column declared with UNIQUE constraint

→UNIQUE constraint can be declared at column level and table level.

Declaring UNIQUE constraint at Column Level :-

Syntax :-

Columnname Datatype(size) UNIQUE



Example :-

```
SQL> CREATE TABLE dept
        (deptno NUMBER(4)
         dname VARCHAR2(20)   CONSTRAINT uq_dname_dept UNIQUE ,
         loc VARCHAR2(20)  ) ;
```

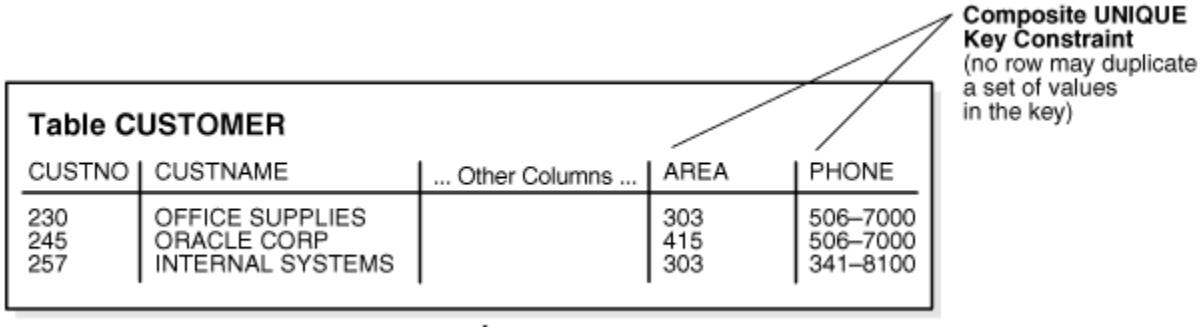
SQL>INSERT INTO dept VALUES(10,'ACCOUNTING','HYDERABAD') ;

1 row created

SQL>INSERT INTO dept VALUES(20,'ACCOUNTING','MUMBAI') ;

ERROR ORA-00001 :- unique constraint (uq_dname_dept) violated

Declaring UNIQUE constraint Table Level :-




```
SQL>CREATE TABLE customer(custno      NUMBER(4) ,
                          custname    VARCHAR2(20) ,
                          area        NUMBER(3) ,
                          phone       VARCHAR2(8)  ,
```

CONSTRAINT uq_area_ph_cust

UNIQUE(area,phone)) ;

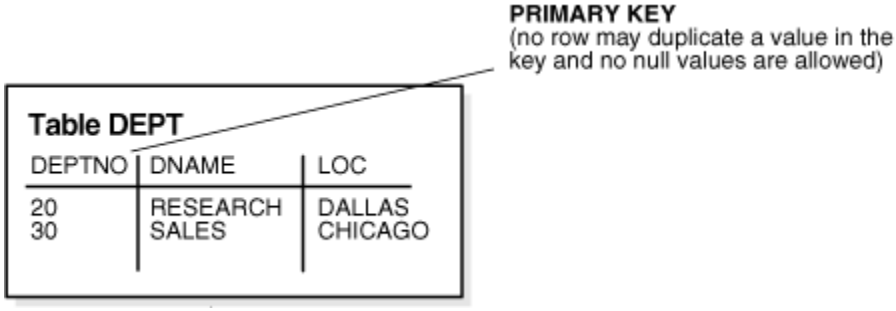
PRIMARY KEY constraint :-

PRIMARY KEY is the candidate key which uniquely identifies a record in a table)

characterstics of PRIMARY KEY :-

- ➔ There should be at the most one PK per table.
- ➔ PK column do not accept null values.
- ➔ PK coumn do not accept duplicate values.
- ➔ RAW,LONG RAW,VARRAY,NESTED TABLE,BFILE columns cannot be declared with PK
- ➔ If PK is composite then uniqueness is determined by the combination of columns.
- ➔ A composite primary key cannot have more than 32 columns
- ➔ It is recommended that PK column should be short and numeric.
- ➔ Oracle automatically creates Unique Index on PK column

Declaring PRIMARY KEY at Column Level :-



Syntax :-

Colname Datatype(size) PRIMARY KEY

Example :-

SQL> CREATE TABLE dept(deptno NUMBER(4) CONSTRAINT pk_dept PRIMARY KEY, dname VARCHAR2(20) , loc VARCHAR2(20)) ;

Declaring PRIMARY KEY at Table Level :-

Example :-

consider the following ORDER_DETAILS table

OrderId	ProdId	Quantity
1000	10	100
1000	11	50
1001	10	20
1001	11	50

In the above example values of **OrderId** are repeated, so it cannot be taken as primary key. And the values of **ProdId** are also repeated , so it cannot be taken as primary key .

when it is not possible with single column to uniquely identify the records then take combination of columns. In the above example combination or **OrdId & ProdId** is not repeated so this combination can be taken as **PRIMARY KEY.** if combination uniquely identifies the records then it is called **composite primary key.**

```
SQL>CREATE TABLE order_details
                (ordid    NUMBER(4) ,
                 prodid  NUMBER(4) ,
                  qty      NUMBER(2) ,
                 CONSTRAINT pk_ordid_prodid  PRIMARY KEY(ordid,prodid) )
;
```

CHECK Constraint :-

- Check constraint validates data based on a condition .
- Value entered in the column should not violate the condition.
- Check constraint allows null values.
- Check constraint can be declared at table level or column level.

Limitations :-

- Conditions should not contain pseudo columns like ROWNUM,SYSDATE etc.
- Condition should not access columns of another table

Declaring Check Constraint Column level :-

Syntax :-

```
COLNAME DATATYPE(SIZE)      [CONSTRAINT <NAME>]
CHECK(CONDITION)
```

Example :-

```
SQL>CREATE TABLE accounts_master(
              accno NUMBER(4)  PRIMARY KEY,
              acname VARCHAR2(20)  NOT NULL ,
              balance NUMER(11,2)    CONSTRAINT
                      ck_bal_accts  CHECK(bal>1000) )  ;
```

```
SQL>INSERT INTO accounts_master VALUES(1,'A',500) ;
```

ERROR ORA-02293 :- cannot validate (SCOTT) CK_BAL_ACCTS) check constraint violated

Declaring CHECK constraint at Table level :-

Table :- Managers

Mgrno	Mgrname	Start_date	End_date

Rule :- End_date should be greater than Start_date

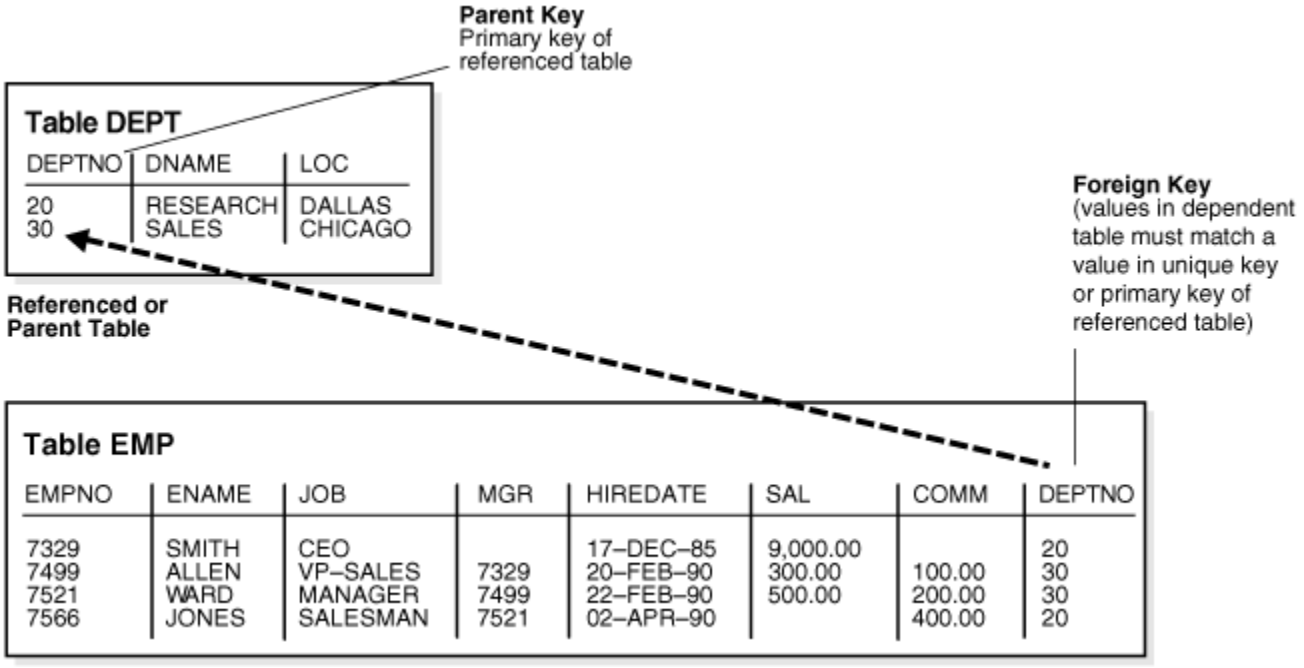
```
SQL>CREATE TABLE managers
          (mgrno        NUMBER(4)  PRIMARY KEY,
           mname        VARCHAR2(20)  NOT NULL,
           start_date   DATE,
           end_date     DATE ,
          CONSTRAINT ck_mgr  CHECK(end_date > start_date) ) ;
```

SQL>INSERT INTO manager VALUES(1,'A','01-JAN-2011','01-JAN-2010') ;
ERROR :- ORA-02290 :- check constraint violated

FOREIGN KEY Constraint:-

- Foreign key is used to establish relationship between tables.
- Foreign key is a column in one table that refers primary key/unique columns of another or same table.
- Values of foreign key should match with values of primary key/unique or foreign key can be null.
- Foreign key column allows null values unless it is declared with NOT NULL.
- Foreign key column allows duplicates unless it is declared with UNIQUE
- By default oracle establish 1: M relationship between two tables.
- To establish 1:1 relationship between two tables declare foreign key with unique constraint
- Foreign key can be declared at column level or table level.
- Composite foreign key must refer composite primary key or Composite unique key.

Declaring foreign key at column level :-



Syntax :-

Colname datatype(size) [constraint <name>] REFERENCES tabname(colname)

Example :-

Creating Parent table :-

```
SQL> CREATE TABLE dept
        (deptno NUMBER(2)    CONSTRAINT pk_dept PRIMARY KEY,
         dname VARCHAR2(20) ,
         loc  VARCHAR2(20) ) ;
```

insert records into DEPT table as follows

Deptno	Dname	Loc
10	Accounting	Hyderabad
20	Research	Mumbai

Creating child table :-

```
SQL> CREATE TABLE emp
   (empno       NUMBER(4)       CONSTRAINT pk_emp PRIMARY KEY,
    ename       VARCHAR2(20)    NOT NULL ,
    sal           NUMBER(7,2)    CONSTRAINT ck_sal_emp CHECK(sal>3000) ,
    deptno      NUMBER(2)        CONSTRAINT fk_deptno_emp    REFERENCES
dept(deptno) ) ;
```

insert records into EMP table as follows

Empno	Ename	Salary	Deptno	Result
1	Smith	5000	10	**Record is inserted because fk value is matching with pk value**
2	Allen	4000	Null	**Record is inserted because fk allows NULL values**
3	Blake	6000	90	**Oracle returns error because fk value is not matching with pk value**
4	King	7000	10	**Record is inserted because fk allows duplicates**

Declaring Foreign Key constraint at Table Level :-

```
SQL>CREATE TABLE stud_course
        (sid NUMBER(2) ,
         cid NUMBER(2) ,
         doc DATE ,
         CONSTRAINT pk_stud_course  PRIMARY KEY(sid,cid) ) ;

SQL>CREATE TABLE certificates
          (certno NUMBER(4)  PRIMARY KEY,
            doi  DATE ,
             sid NUMBER(2) ,
            cid NUMBER(2) ,
             CONSTRAINT fk_sid_cid  FOREIGN KEY(sid,cid)
                                    REFERENCES  stud_course(sid,cid) ) ;
```

DEFAULT Option :-

→If column Declared with DEFAULT option then oracle inserts DEFAULT value when value is not provided.

→DEFAULT option prevents entering NULL values into the column.

Example :-

```
SQL>CREATE TABLE emp
        (empno NUMBER(4) ,
```

ename VARCHAR2(20) ,
hiredate DATE DEFAULT SYSDATE) ;

SQL> INSERT INTO emp(empno,ename) VALUES(1,'x') ;

After executing the above command oracle inserts sysdate into Hiredate column.

Adding constraints to an existing table :-

Constraints can be also be added to an existing table with the help of ALTER command

Syntax :-

ALTER TABLE <TABNAME> ADD [CONSTRAINT <NAME>]
CONSTRAINT_TYPE(COL1 [,COL2])

Example :-

Create a table without constraints later add constraints

```
SQL>CREATE TABLE emp55
          (empno NUMBER(4) ,
           ename VARCHAR2(20) ,
           sal  NUMBER(7,2) ,
           dno NUMBER(2) ) ;
```

Adding PRIMARY KEY :-

```
SQL>ALTER TABLE emp55
              ADD CONSTRAINT  pk_emp55  PRIMARY KEY(empno) ;
```

Note:- primary key constraint cannot be added to a column that already contains duplicates or NULL values.

Adding FOREIGN KEY :-

```
SQL>ALTER TABLE emp55
            ADD CONSTRAINT fk_dno_emp55
                FOREIGN KEY(dno)  REFERENCES  dept(deptno) ;
```

Adding CHECK constraint :-

```
SQL> ALTER TABLE emp55
      ADD  CONSTRAINT ck_sal_emp55 CHECK(sal>3000)    NOVALIDATE ;
```

NOVALIDATE option :-

If constraint added with NOVALIDATE option then oracle doesn't validate existing data and validates only future DML operations.

Dropping Constraints:-

Syntax :-

ALTER TABLE <TABNAME> DROP CONSTRAINT <NAME>

Example :-

SQL>ALTER TABLE emp55 DROP CONSTRAINT pk_emp55;

SQL>ALTER TABLE emp55 DROP CONSTRAINT ck_sal_emp55

Note :-

→ PRIMARY KEY cannot be dropped if it referenced by any FOREIGN KEY constraint.

→If PRIMARY KEY is dropped with CASCADE option then along with PRIMARY KEY referencing FOREING KEY is also dropped.

→PRIMARY KEY column cannot be dropped if it is referenced by some FOREIGN KEY.

→PRIMARY KEY table cannot be dropped if it is referenced by some FOREIGN KEY.

→PRIMARY KEY table cannot be truncated if it is referenced by some FOREIGN KEY.

Enabling/Disabling a Constraint:

If the constraints are present, then for each DML operation constraints are checked by executing certain codes internally. It may slow down the DML operation marginally. For massive DML operations, such as transferring data from one table to another because of the presence of constraint, the speed will be considered slower. To improve the speed in such cases, the following methods are adopted:

→Disable constraint

→Performing the DML operation DML operation

→Enable constraint

Disabling Constraint:-

Syntax :-

ALTER TABLE <tabname> DISABLE CONSTRAINT <constraint_name> ;

Example :-

SQL>ALTER TABLE emp DISABLE CONSTRAINT ck_sal_emp ;

SQL>ALTER TABLE dept DISABLE PRIMARY KEY CASCADE;

NOTE:-

If constraint is disabled with CASCADE then PK is disabled with FK.

Enabling Constraint :-

Syntax :-

ALTER TABLE <TABNAME> ENABLE CONSTRAINT <NAME>

Example :-

SQL>ALTER TABLE emp ENABLE CONSTRAINT ck_sal_emp;

ON DELETE NO ACTION :-

If foreign key declared with ON DELETE NO ACTION then parent record cannot be deleted if any child records exists.

ON DELETE CASCADE :-

If foreign key declared with ON DELETE CASCADE then if any parent record is deleted then dependent child records also deleted automatically.

```
SQL>CREATE TABLE dept
        (deptno NUMBER(2)   PRIMARY KEY,
         dname  VARCHAR2(20)  NOT NULL ,
         loc  VARCHAR2(20)  ) ;

SQL>CREATE TABLE emp (
      empno NUMBER(4)  PRIMARY KEY,
      ename VARCHAR2(20)  NOT NULL,
      sal NUMBER(7,2)  CHECK(sal>3000)  ,
      dno NUMBER(2)  REFERENCES  dept(deptno)  ON DELETE CASCADE) ;
```

ON DELETE SET NULL :-

if foreign key declared with ON DELETE SET NULL then foreign key value in child table is set to NULL if user deletes record from parent table.

```
SQL>CREATE TABLE dept
        (deptno NUMBER(2)   PRIMARY KEY,
         dname  VARCHAR2(20)  NOT NULL ,
         loc  VARCHAR2(20)  ) ;
SQL>CREATE TABLE emp (
      empno NUMBER(4)  PRIMARY KEY,
      ename VARCHAR2(20)  NOT NULL,
      sal NUMBER(7,2)  CHECK(sal>3000)  ,
      dno NUMBER(2)  REFERENCES  dept(deptno)  ON DELETE SET NULL) ;
```

Example :-

Display list of constraints declared in EMP table ?

SQL>SELECT constraint_name,constraint_type FROM user_constraints WHERE table_name='EMP';

CONSTRAINT_NAME	CONSTRAINT_TYPE
PK_EMP	P
SYS_C004455	C
CK_SAL_EMP	C
FK_DNO_EMP	R

Oracle gives same code for CHECK and NOT NULL constraint , to know whether constraint is CHECK or NOT NULL use SEARCH_CONDITION as given below.

SQL>SELECT constraint_name ,constraint_type ,search_condition
FROM user_constraints
WHERE table_name='EMP' ;

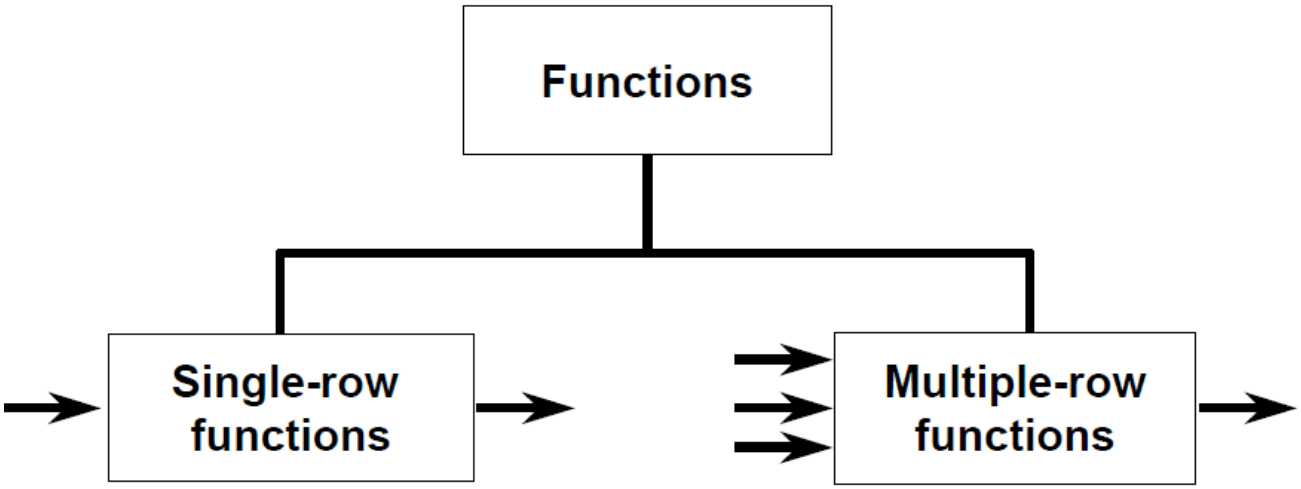
Display which columns are declared with what constraints in EMP table ?

SQL>SELECT constraint_name , column_name FROM user_constraints WHERE table_name='EMP';

SQL Functions

Functions are a very powerful feature of SQL and can be used to do the following:-

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



SINGLE ROW FUNCTIONS :-

These functions operate on single rows only and return one result per row. The single row functions are categorized as follows.

- **Character functions**
- **Date functions**
- **Mathematical functions**
- **Conversion functions**
- **Special functions**
- **OLAP functions**

Character functions:-

These functions mainly operate on character data)

UPPER :- converts string to uppercase

Syntax:- UPPER(string)

Example:-

SQL>SELECT UPPER('hello') FROM DUAL;

HELLO

LOWER :- converts string to lower case

Syntax:- **LOWER(string)**

Example:-

SQL>SELECT LOWER('HELLO') FROM DUAL;

hello

Display ename,salaries and display names in lower case ?

SQL>SELECT LOWER(ename) ,sal FROM emp;

Convert all ename from uppercase to lowercase in table ?

SQL>UPDATE emp SET ename=LOWER(ename) ;

INITCAP:- first character is capitalized

Syntax:- **INITCAP(string)**

Example :-

SQL>SELECT INITCAP('hello welcome') FROM DUAL ;

Hello Welcome

LENGTH :- returns string length

Syntax :- **LENGTH(string)**

Example :-

SQL> SELECT LENGTH('hello') FROM DUAL;

5

Display employee records whose name contains 5 characters ?

SQL>SELECT * FROM emp WHERE LENGTH(ename) =5;

SUBSTR:- used to extract part of the string

Syntax:- **SUBSTR(string1,start [, length])**

Example:-

SQL>SELECT SUBSTR('hello',2,4) FROM DUAL;

ello

SQL>SELECT SUBSTR('hello welcome',-5,4) FROM DUAL;

lcom

Display employee records whose name starts with and ends with same character ?

SQL>SELECT * FROM emp WHERE SUBSTR(ename,1,1) =SUBSTR(ename,-1,1) ;

Display employee records whose name starts between 'A' AND 'P' ?

SQL>SELECT * FROM emp WHERE SUBSTR(ename,1,1) BETWEEN 'A' AND 'P' ;

INSTR :- returns occurrence of one string in another string

Syntax:- **INSTR(str1,str2 [,start , occurrence])**

If str2 exists in str1 returns position

If not exists returns 0.

Example:-

SQL> SELECT INSTR('HELLO WELCOME','O') FROM DUAL:

5

SQL> SELECT INSTR('HELLO WELCOME','O',1,2) FROM DUAL:
11
SQL>SELECT INSTR('HELLO WELCOME','O',-1,2) FROM DUAL ;
5
Display employee records whose name contains 'S' ?
SQL>SELECT * FROM EMP WHERE INSTR(ENAME,'S') <> 0 ;
Scenario :-
1 CUSTOMER TABLE :-

email
sachin@gmail) com
sourav@gmail) com

from the above email addresses display only the first part ?
SQL>SELECT SUBSTR(EMAIL,1,INSTR(EMAIL,'@') -1) FROM customer;
2 CUSTOMER_TABLE :-

CNAME
Rahuld dravid
Virendra sehwag
Sachin ramesh tendulkar
Sourav ganguly
Mahindra singh dhoni

From the above customer names display only those names that contains 3 parts ?
SQL>SELECT * FROM customer WHERE INSTR(cname,' ',1,2) > 0;
LTRIM :- trims white spaces and unwanted characters on left side
Syntax:- LTRIM(string1 [, string2])
Example:-
SQL>SELECT LTRIM(' HELLO') FROM DUAL;
HELLO
SQL>SELECT LTRIM('XXXXXHELLO','X') FROM DUAL;
HELLO
RTRIM :- trims whitespaces and unwanted characters on right side)
Syntax:- RTRIM(string1 [,string2])
Example:-
SQL>SELECT RTRIM('HELLO ') FROM DUAL;
HELLO
SQL>SELECT RTRIM('HELLOXXXX','X') FROM DUAL;
HELLO
TRIM :- trims whitespaces and unwanted characters on both left and right side
SQL>SELECT TRIM(' HELLO ') FROM DUAL;
HELLO

SQL>SELECT TRIM(LEADING 'X' FROM 'XXXXHELLO') FROM DUAL;
HELLO
SQL>SELECT TRIM(TRAILING 'X' FROM 'HELLOXXXXX') FROM DUAL;
HELLO
SQL>SELECT TRIM(BOTH 'X' FROM 'XXXXHELLOXXXX') FROM DUAL;
HELLO
LPAD :- one string is filled with another string on left side
Syntax :- LPAD(string1,length,string2)
SQL>SELECT LPAD('hello',10,'*') FROM DUAL;
*****hello
RPAD :- fills one string with another string on right side
Syntax:- RPAD(string1,length,string2)
Example :-
SQL>SELECT RPAD('HELLO',10,'*') FROM DUAL;
Display ename , salaries and in salary column display **** instead of actual values , for example if salary is 4000 display **** ?
SQL> SELECT ename,RPAD('*',sal/1000,'*') as salary FROM emp ;
REPLACE :- to replace one string with another string
Syntax:- REPLACE(string1,string2,string3)
Example:-
SQL>SELECT REPLACE('UTI BANK','UTI','AXIS') FROM DUAL;
AXIS BANK
Display employee records whose name contains exactly one 'A' ?
SQL> SELECT * FROM emp
WHERE LENGTH(ename) – LENGTH(REPLACE(ename,'A',''))
=1 ;
Scenario :-
Examine the data in the ENAME and HIREDATE columns of the EMPLOYEES table:

ENAME	HIREDATE
SMITH	17-DEC-80
ALLEN	20-FEB-81
WARD	22-FEB-81

SQL>SELECT SUBSTR(INITCAP(ename) ,1,3) || REPLACE(hiredate,'-','') "USERID" FROM emp;
TRANSLATE:- translates one char to another character
Syntax:- TRANSLATE(string1,string2,string3)
Example:-

SQL> SELECT TRANSLATE('HELLO','ELL','ABC') FROM DUAL;
HABBO

SQL>SELECT ename, TRANSLATE(sal,'0123456789','$qT*K#PjH@') FROM emp;

CONCAT :- concatenates two strings
Syntax :- CONCAT(str1,str2)
Example :-
SQL> SELECT CONCAT('HELLO ', ' WELCOME') FROM DUAL;
HELLO WELCOME
SOUNDEX:- A character value representing the sound Of a word, using this we can find strings that sounds same)
Syntax:- SOUNDEX(string)
Example :-
SQL>SELECT * FROM EMP
WHERE SOUNDEX('SMITH') =SOUNDEX('SMYTH') ;

ASCII :- returns ASCII value of first character
Syntax: ASCII(string)
Example :-
SQL>SELECT ASCII('A') FROM DUAL ;
65
CHR :- returns character for a given ASCII value
Syntax :- CHR(ascii value)
Example :-
SQL>SELECT CHR(65) FROM DUAL ;
A
Date Functions:-
EXTRACT :- used to extract part of the date)
Syntax:- EXTRACT(FMT FROM DATE)
Extracting year from date:-
SQL>SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL;
2012
Extracting month from date:-
SQL>SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;
5
Extracting day from date :-
SQL> SELECT EXTRACT(DAY FROM SYSDATE) FROM DUAL;
23

Display employee records joined in first 15 days in the month APR,DEC in the year between
1980 and 1987 ?

```
SQL> SELECT * FROM emp
        WHERE  EXTRACT(DAY FROM hiredate)  BETWEEN 1 AND 15
                    AND
                EXTRACT(MONTH FROM hiredate)  IN (4,12)
                    AND
              EXTRACT(YEAR FROM hiredate)  BETWEEN 1980 AND 1987;
```

ADD_MONTHS:- adds no of months to a date)
Syntax:- ADD_MONTHS(DATE,MONTHS)
Example:-
SQL>SELECT ADD_MONTHS(SYSDATE,2) FROM DUAL;
23-JUN-12
SQL>SELECT ADD_MONTHS(SYSDATE,-2) FROM DUAL;
23-MAR-12
Display ename,sal,hiredate and date of retirement , assume that date of retiment is 30 years after date of join ?
SQL>SELECT ename,sal,hiredate,ADD_MONTHS(hiredate,30*12) AS DOR FROM emp ;
LAST_DAY:- returns last day of the month
Example:-
SQL>SELECT LAST_DAY(sysdate) FROM DUAL;
31-MAY-12
Display first day of the current month ?
SQL>SELECT ADD_MONTHS(LAST_DAY(SYSDATE) +1,-1) FROM DUAL ?
MONTHS_BETWEEN :- returns no of months between two dates)
Syntax:- MONTHS_BETWEEN(date1,date2)
Example:-
SQL>SELECT MONTHS_BETWEEN(Sysdate,'20-APR-11') FROM DUAL
12

NEXT_DAY :- returns next specified day starting from given date)
Syntax:- NEXT_DAY(DATE ,DAY)
Example :-
SQL>SELECT NEXT_DAY(SYSDATE,'SUNDAY') FROM DUAL;
27-MAY-12
Mathematical Functions:-
ABS:- returns absolute value

Syntax:- **ABS(number)**

Example:-

SQL>SELECT ABS(-10) FROM DUAL;

10

SIGN :-

Syntax :- SIGN(expr)

If expr >0 then returns 1

If expr <0 then returns -1

If expr=0 then returns 0

Example :-

SQL>SELECT SIGN(100) FROM DUAL ;

1

POWER:- returns power

Syntax :- POWER(M,N)

Example :-

SQL>SELECT POWER(3,2) FROM DUAL;

9

SQRT:- returns square root)

Syntax :- SQRT(N)

Example:-

SQL>SELECT SQRT(25) FROM DUAL ;

5

MOD:- returns remainder

Syntax:- MOD(m,n)

Example:-

SQL>SELECT MOD(10,2) FROM DUAL;

0

Display employee records earning multiple of 50)

SQL>SELECT * FROM emp WHERE MOD(sal,50) =0;

CEIL:- returns integer greater than or equal to given number)

Syntax:- CEIL (number)

Example:-

SQL>SELECT CEIL(9) 5) FROM DUAL

10

FLOOR:- returns integer less than or equal to given number)

Syntax:- FLOOR(number)

Example:-

SQL>SELECT FLOOR(9) 5) FROM DUAL

9

ROUND:- rounds number to given number of decimal places)

Syntax:- **ROUND(number [,decimal places])**

Example:-

SQL>SELECT ROUND(3) 456,2) FROM DUAL ;

3) 46

SQL> SELECT ROUND(3) 453,2) FROM DUAL;

3) 45

SQL>SELECT ROUND(3) 456) FROM DUAL ;

3

SQL>SELECT ROUND(3) 65) FROM DUAL ;

4

SQL>SELECT ROUND(383) 456,-2) FROM DUAL ;

400

SQL>SELECT ROUND(383) 456,-1) FROM DUAL

380

SQL>SELECT ROUND(383) 456,-3) FROM DUAL ;

0

Note :- ROUND function can also be used to round dates) Date can be rounded to YEAR / MONTH/DAY part)

Assume SYSDATE = 20-apr-2012

SQL>SELECT ROUND(SYSDATE,'YEAR') FROM DUAL;

01-JAN-2012

SQL>SELECT ROUND(SYSDATE,'MONTH') FROM DUAL;

01-may-2012

SQL>SELECT ROUND(SYSDATE,'DAY') FROM DUAL;

22-APR-2012

TRUNC :- truncated the number to specified number of decimal places

Syntax:- TRUN(m,n)

Example :-

SQL>SELECT TRUNC(3) 456,2) FROM DUAL ;

3) 45

SQL>SELECT TRUN(SYSDATE,'YEAR') FROM DUAL;

01-JAN-2012

Conversion Functions :-

These functions are used to convert from one datatype to another datatype

Conversion of two types :-

→**implicit conversion**

→**explicit conversion**

Implicit Conversion:-

if conversion is performed by ORACLE then it is called implicit conversion.

For assignments, the oracle server can automatically convert the following.

FROM	TO
VARCHAR2	NUMBER
VARCHAR2	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

For expression evaluation , the oracle server can automatically convert the following .

FROM	TO
VARCHAR2	NUMBER
VARCHAR2	DATE

Example for implicit conversion :-

SQL>SELECT 1000 + '1000' FROM DUAL ;

2000

Explicit Conversion:-

if conversion is performed by user then it is called explicit conversion. The following functions are used to do explicit conversion

1 TO_CHAR
2 TO_DATE
3 TO_NUMBER

TO_CHAR :-

This function is used to convert DATE / NUMBER to CHAR type

Converting DATE to CHAR type :-

DATEs are converted to CHAR type to display DATEs in different format.

Syntax:- TO_CHAR(DATE [,FORMAT])

The different formats supported by ORACLE listed below

Century formats :-

CC Two Digits Century

21

Scc Two Digits Century with a negative sign for Bc

-10

Year Formats :-

YYYY All four Digits of the Year

2012

IYYY	All four Digits of the ISO year	2012
SYYYY	All four Digits of the Year with a negative sign for Bc	-1001
YY	Last Two Digits of the Year	12
YEAR	Name of the Year	Two Thousand Twelve

Example :-

Display employee records joined between JANUARY and APRIL ?

SQL>SELECT * FROM emp WHERE TO_CHAR(hiredate,'mm') BETWEEN 1 AND 4 ;

Day :-

DD	Day of the Month	26
DDD	Day of the Year	103
DAY	Name of the Week Day	SATURDAY
DY	First Three letter from Week Day	SAT
D	Day of the Week	7

Example :-

Display employee records joined on SUNDAY ?

SQL>SELECT * FROM emp WHERE TO_CHAR(hiredate,'DAY') = 'SUNDAY';

Display on which day employee joined ?

SQL>SELECT ENAME || ' joined on ' || TO_CHAR(hiredate,'DAY') FROM emp;

Week :-

WW	week of the year	24
W	week of the month	4

Time :-

HH	hour in 12-format	12
HH24	hour in 24-format	23
MI	minute	20
SS	second	30
AM/PM	AM/PM as appropriate	

Example :-

Display sysdate as follows ?

25 january 2012 , Monday 10:00:00 AM

SQL>SELECT TO_CHAR(SYSDATE,'DD month YYYY , Day HH:MI:SS PM') FROM DUAL;

Other Formats :-

AD/BC	AD/BC date as appropriate
TH	th,rd,nd,st
SP	Number is spelled out.
J	Date is displayed in Julian format

Example :-

SQL>SELECT TO_CHAR(SYSDATE,'J') FROM DUAL;

2439892

The above number representes number of days passed since 01 JAN 4712BC to SYSDATE)

SQL>SELECT TO_CHAR(SYSDATE,'DDSPTH MON YYYY') FROM DUAL;

To change default DATE format during the session execute following command

SQL>ALTER SESSION SET NLS_DATE_FORMAT='MM/DD/YY' ;

then execute the following command

SQL>SELECT ENAME , HIREDATE FROM EMP ;

When above query is executed then HIREDATEs are displayed in MM/DD/YY format)

OCA question :-

You need to display the date 11-oct-2007 in words as 'Eleventh of October, Two Thousand Seven')

Which SQL statement would give the required result?

A. SELECT TO_CHAR('11-oct-2007', 'fmDdspth "of" Month, Year') FROM DUAL;

B. SELECT TO_CHAR(TO_DATE('11-oct-2007') , 'fmDdspth of month, year') FROM DUAL;

C. SELECT TO_CHAR(TO_DATE('11-oct-2007') , 'fmDdthsp "of" Month, Year') FROM DUAL;

D) SELECT TO_DATE(TO_CHAR('11-oct-2007','fmDdspth "of" Month, Year')) FROM DUAL;

Converting number to character type :-

Syntax :- TO_CHAR(NUMBER [,FORMAT])

Format	Description
sS999	Returns Digit with a leading - sign for negative number
0999	returns number with a leading zeros
9900	returns number with trailing zeros.
999. 99	returns decimal point in the specified position.
9,999	returns comma in the specified position.
$999	returns a leading Dollar Sign.
C999	returns ISO currency symbol in the specified position
9. 99EE	returns number in scientific notation.
RN	returns number in roman format.
L999	retuns number with local currency symbol.

Example :-

SQL>SELECT ename, TO_CHAR(sal,'L9,999') AS sal FROM emp ;

To set local currency symbol execute the following command)

SQL>ALTER SESSION SET NLS_TERRITORY=America;

SQL>ALTER SESSION SET NLS_TERRITORY=Germany;

TO_DATE :-

Used to convert string to datetime) You can provide an optional format to indicate the format of string) if you omit format , the date must be in the default format usually (DD-MON-YYYY ,DD-MON-YY))

Syntax:- TO_DATE(string [,format])

Example :-

SQL>SELECT '26-AUG-2012' + 10 FROM DUAL ;

The above statement returns ORACLE error INVALID NUMBER , because 26-AUG-2012 is treated as string , so to do the calculation conversion is required)

SQL>SELECT TO_DATE('26-AUG-2012') + 10 FROM DUAL ;

SQL>SELECT TO_DATE('08/26/12','MM/DD/YY') + 10 FROM DUAL;

Display on which day india has got independenc ?

SQL>SELECT TO_CHAR(TO_DATE('15-AUG-1947') ,'DAY') FROM DUAL;

Display employee names , salaries and display salaries in words ?

SQL>SELECT ename , TO_CHAR(TO_DATE(sal,'J') ,'JSP') AS SAL FROM emp;

Example :-

SQL>CREATE TABLE emp (empno NUMBER(4) , dob DATE) ;

You need to insert date & time into dob column , but by default DATE data type accepts only DATE but not time. To insert date along with time conversion is required.

SQL>INSERT INTO emp VALUES (1, TO_DATE('26-AUG-2012 10:20:30','DD-MON-YYYY HH:MI:SS')) ;

But TIMESTAMP datatype allows both date and time without conversion)

SQL>CREATE TABLE emp (empno NUMBER(4) , dob TIMESTAMP) ;

SQL>INSERT INTO emp VALUES(1,'26-AUG-2012 10:20:30') ;

Multi-Row functions:-

→These functions will process group of rows and Returns one value from that group.

→These functions are also called AGGREGATE functions or GROUP functions

MAX :-

Returns maximum value of a given expression

Syntax:- MAX(expr)

Example :-

SQL>SELECT MAX(sal) FROM emp;

Display maximum salary of 30th DEPT ?

SQL>SELECT MAX(sal) FROM EMP WHERE deptno=20;

MIN:-

Returns minimum value of a given expression)

Syntax :- MIN(EXPR)

Example:-

SQL>SELECT MIN(sal) FROM emp;

SUM :-

→Returns sum of a given expression.

→This function cannot be applied on strings and dates.

Syntax:- SUM(expr)

Example:-

SQL>SELECT SUM(sal) FROM emp;

Display total salary paid to MANAGERS ?

SQL>SELECT SUM(sal) FROM emp WHERE job = 'MANAGER' ;

Scenario :-

Calculate total salaries paid to each dept as follows ?

DEPT_10 DEPT_20 DEPT_30

? ? ?

```
SQL>SELECT  SUM(DECODE(deptno,10,sal) )  as DEPT_10 ,
            SUM(DECODE(deptno,20,sal) )  as DEPT_20,
            SUM(DECODE(deptno,30,sal) )  as DEPT_30
```

FROM emp;

AVG :-

Returns avg value of a given expression.

Syntax:- AVG(expr)

Example:-

SQL>SELECT AVG(sal) FROM emp;

COUNT :-

→Returns no of values present in a column.

→COUNT function ignores NULL values.

Syntax :- COUNT(expr)

Example:-

SQL>SELECT COUNT(empno) FROM emp;

SQL>SELECT COUNT(DISTINCT deptno) FROM emp;

COUNT(*) :-

Returns no of records

Example :-

SQL>SELECT COUNT(*) FROM emp;

Display number of employees joined in 1981 year ?

SQL>SELECT COUNT(*) FROM emp WHER TO_CHAR(hiredate,'yyyy') =1981;

Display number of employees joined as follows ?

1981 1982 1983

? ? ?

```
SQL>SELECT  COUNT(DECODE(TO_CHAR(hiredate,'YYYY') ,1981,empno) )
AS Y1981 ,
                COUNT(DECODE(TO_CHAR(hiredate,'YYYY') ,1982,empno) )
AS Y1982,
                COUNT(DECODE(TO_CHAR(hiredate,'YYYY') ,1983,empno) )
AS Y1983  FROM emp;
```

CASE Statement :-

→The CASE expression performs if-then –else logic .

→introduced in ORACLE 9i

→The CASE expression works in a similar manner to DECODE, but use CASE because it is ANSI-compliant .

→the CASE expression is easier to read.

There are two types of CASE Statements

→**Simple case.**
→**Searched case** .

Simple CASE Statement :-
Simple CASE expressions use expressions to determine the value to return)
Syntax :-

```
CASE search_expression
WHEN expression1 THEN result1
WHEN expression2 THEN result2
……)
WHEN expression THEN result
ELSE default_result
END ;
```

→Search_expression is the expression to be evaluated.
→expression1, expression2, ……,expression are the expressions to be evaluated against search_expression.
→result1, result2,……. . , result are the returned results(one for each possible expression. . If expression1 evaluates to search_expression, results is returned, and similarly for the other expressions.
→default_result is returned when no matching expression is found.
Example :-

```
SQL>SELECT ename,sal,    CASE job
                  WHEN 'CLERK' THEN 'WORKER'
                              WHEN 'MANAGER' THEN 'BOSS'
                  WHEN 'PRESIDENT' THEN 'BIG BOSS'
                  ELSE
                       'EMPLOYEE'
              END  AS JOB
        FROM emp ;
```

Searched CASE Statement :
Searched CASE expressions use conditions to determine the returned value.
Syntax :-

```
CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        ……) )
        WHEN condition THEN result
```

ELSE
default_result
END;

Where,
→condition1, condition2,……. . conditionN are expressions to be evaluated.
→result1, result2,……resultN are the returned results(one for each possible condition. . If condition is true, result1 is returned, and similarly for the other expressions.
→default_result is returned when there is no condition returns true

Example :-

```
SQL>SELECT ename,sal,   CASE
                    WHEN sal>3000 THEN 'HISAL'
                  WHEN sal<3000 THEN 'LOSAL'
              ELSE
                    'MODERATE SAL'
                  END AS SALRANGE
FROM emp ;
```

GROUP BY clause:-

You can use GROUP BY clause to divide the rows in a table into smaller groups. You can then use the group functions to return summary information for each group.

Syntax :-

SELECT *column, group_function(column.*
FROM *table*
[WHERE *condition*]
[GROUP BY *group_by_expression*]
[HAVING condition] [ORDER BY *column*];

Guidelines :-

- only GROUP BY columns and AGGREGATE functions should appear in SELECT list other than these two if any column appears then oracle returns error.
- Using WHERE clause, you can exclude rows before dividing them into groups.
- You cannot use a column alias in the GROUP BY clause.
- By default, rows are sorted by ascending order of the columns included in the GROUP BY list. You can override this by using the ORDER BY clause.

Examples :-

Display total salaries paid to each department ?

SQL>SELECT deptno,SUM(sal) FROM emp GROUP BY deptno ;

DEPTNO	SUM(SAL)
10	8750
20	10875
30	9400

Display no of employees joined each year ?

SQL>SELECT
EXTRACT(YEAR FROM hiredate) AS YEAR, COUNT(*) AS EMPS
FROM emp
GROUP BY EXTRACT(YEAR FROM hiredate) ;

Display total salaries paid to each department where deptno in (10,20) ?

SQL>SELECT deptno,SUM(sal) FROM emp
WHERE deptno IN (10,20)
GROUP BY deptno ;

DEPTNO	SUM(SAL)
10	8750
20	10875

HAVING clause :-

In the same way that you use the WHERE clause to restrict the rows that you select, you can use the HAVING clause to restrict groups.

find the maximum salary of each department, but show only the depts. that have a maximum salary more than 10,000, you need to do the following:

1 Find the maximum salary for each department by grouping by deptno

2. Restrict the groups to those departments with a maximum salary greater than 10,000.

The Oracle server performs the following steps when you use the HAVING clause:

1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the HAVING clause are displayed.

Example :-

SQL>SELECT deptno, SUM(sal)
FROM emp
GROUP BY deptno
HAVING SUM(sal) >10000;

Deptno SUM(SAL)

20 10875

WHERE Vs HAVING :-

WHERE	**HAVING**
Filter rows	filter groups
Filter data before group by	filter data after group by

NOTE:- in condition if there is no group function then use WHERE clause , if condition contains group function use HAVING clause.

Using WHERE , GROUP BY ,HAVING clauses Together :-

You can use WHERE,GROUP BY, and HAVING clauses together in the same query. When you do this the WHERE clause first filters the rows, the GROUP BY clause then groups the remaining rows and finally HAVING clause filters the groups.

Example :-

```
SQL>SELECT   deptno,sum(sal)   FROM emp
       WHERE deptno IN (10,20)
        GROUP BY deptno
        HAVING SUM(sal)  > 10000 ;
```

```
 DEPTNO  SUM(SAL)
------- ---------
     20     10875
```

Grouping Rows Based on more than one Column :-

You can GROUP rows based on more than one column.

Calcuate total salaries department wise and within department job wise ?

Example :-

```
SQL>SELECT deptno,job,SUM(sal)
       FROM emp
        GROUP BY deptno,job;
```

```
 DEPTNO JOB         SUM(SAL)
------- --------- ---------
     10 CLERK          1300
     10 MANAGER        2450
     10 PRESIDENT      5000
     20 CLERK          1900
     20 ANALYST        6000
     20 MANAGER        2975
     30 CLERK           950
     30 MANAGER        2850
     30 SALESMAN       5600
```

```
SQL>BREAK ON deptno
```

SQL> /

```
 DEPTNO JOB         SUM(SAL)
------- --------- ---------
     10 CLERK           1300
        MANAGER         2450
        PRESIDENT       5000
     20 CLERK           1900
        ANALYST         6000
        MANAGER         2975
     30 CLERK            950
        MANAGER         2850
        SALESMAN        5600
```

```
SQL>SELECT  TO_CHAR(hiredate,'YYYY') AS Year ,
                TO_CHAR(hiredate,'Mon')  AS Month,
                TO_CHAR(hiredate,'Dy')  AS Day , COUNT(*.  AS Emps
    FROM emp
    GROUP BY TO_CHAR (hiredate,'YYYY'),
                TO_CHAR (hiredate,'Mon') ,
                TO_CHAR (hiredate,'Dy')
   ORDER BY Year, Month, Day;
```

Cross Tabulation:-

An example of cross tabulation shown below :-

DEPTNO	CLERK	MANAGER	SALESMAN
10	1300	2450	
20	1900	2975	
30	95	2850	5600

To produce the above result the following query should be run

```
SQL>SELECT    deptno,    SUM( DECODE(job,'CLERK',sal. .  AS CLERK ,
                          SUM(DECODE(job,'MANAGER',sal. .  AS
MANAGER,
                                SUM(DECODE(job,'SALESMAN',sal.
        ) AS SALESMAN
        FROM emp
        GROUP BY deptno;
```

Using PIVOT operator :-

Cross tabulation is simplified in ORACLE 11g with the help of PIVOT operator.

```
SQL>SELECT * FROM
  (SELECT DEPTNO,SAL,JOB FROM EMP.
PIVOT
```

(SUM(SAL. FOR JOB IN ('CLERK','MANAGER','SALESMAN')

ORDER BY DEPTNO;

UNPIVOT operator :-

The UNPIVOT operator converts column-based data into separate rows. To see the UNPIVOT operator in action we need to create a test table.

```
SQL>CREATE TABLE unpivot_test (
                        id            NUMBER,
                        customer_id   NUMBER,
                        product_code_a  NUMBER,
                        product_code_b  NUMBER,
                        product_code_c  NUMBER,
                        product_code_d  NUMBER. ;

SQL>INSERT INTO unpivot_test VALUES (1, 101, 10, 20, 30, NULL. ;
SQL>INSERT INTO unpivot_test VALUES (2, 102, 40, NULL, 50, NULL. ;
SQL>INSERT INTO unpivot_test VALUES (3, 103, 60, 70, 80, 90. ;
SQL>INSERT INTO unpivot_test VALUES (4, 104, 100, NULL, NULL, NULL. ;
SQL>COMMIT;
```

So our test data starts off looking like this.

```
SQL>SELECT * FROM unpivot_test;
```

ID	CUSTOMER_ID	PRODUCT_CODE_A	PRODUCT_CODE_B	PRODUCT_CODE_C	PRODUCT_CODE_D
1	101	10	20	30	
2	102	40	50	80	
3	103	60	70	90	
4	104	100	70	90	

The UNPIVOT operator converts this column-based data into individual rows.

```
SQL>SELECT *
FROM   unpivot_test
UNPIVOT (quantity FOR product_code IN (product_code_a AS 'A',
product_code_b AS 'B', product_code_c AS 'C', product_code_d AS 'D'. . ;
```

ID	CUSTOMER_ID	P	QUANTITY

1	101	A	10
1	101	B	20
1	101	C	30
2	102	A	40
2	102	C	50
3	103	A	60
3	103	B	70
3	103	C	80
3	103	D	90
4	104	A	100

Convert rows to columns :-

SQL> desc t1

Name	**Null?**	**Type**
NAME		VARCHAR2(10.
YEAR		NUMBER(4.
VALUE		NUMBER(4.

SQL> select * from t1;

NAME	**YEAR**	**VALUE**
john	1991	1000
john	1992	2000
john	1993	3000
jack	1991	1500
jack	1992	1200
jack	1993	1340
mary	1991	1250
mary	1992	2323
mary	1993	8700

perform a sql query to return results like this:

year, john, Jack, mary

1991, 1000, 1500 1250

1992, 2000, 1200, 2323

1993, 3000, 1340, 8700

Joins

In OLTP db tables are normalized and data organized in more than one table. For example sales DB is organized in customer, product, and supplier tables etc. JOIN is an operation that combines rows from two or more tables or view. ORACLE performs JOIN operation when more than one table is listed in FROM clause. Tables participated in JOIN operation must share a meaningful relationship.

Types of JOINS :-

- Inner join or Equi Join
- Non-Equi Join
- Self Join
- Outer Join
- Cross Join

Inner Join :-

- In INNER JOIN join operation is performed based on common columns.
- To perform INNER JOIN there should be a common column in joining tables and name of the common column need not to be same.
- To perform INNER JOIN parent/child relationship between the tables is not mandatory.
- INNER join is most commonly used join in realtime.

Syntax :-

SQL> SELECT <collist> FROM <tab11> , <tab2>
WHERE <join cond
[AND <join cond> AND <cond>-------]

Join Condition :-

Child. fk = parent. pk (if relationship exists.
Tab1. commoncolumn = Tab2. commoncolumn (if there is no relationship.

- Oracle performs INNER JOIN by comparing fk value with pk value by using = operator.
- INNER JOIN is also called EQUI JOIN because join cond is based on = operator.
- INNER JOIN returns all rows from both tables that satisfies the JOIN CONDITION.
- No of JOIN CONDS depends on number of tables to be joined .
- To join N tables , min N-1 JOIN CONDS are required.

Guidelines:-

When writing a SELECT statement that joins tables, precede the column name with the table name or table alias for faster access and to avoid ambiguity.

Example:-

Display EMPNO,ENAME,DEPTNO,DNAME,LOC ?

```
SQL> SELECT    e. empno,  e.  ename , e. sal, d.  deptno ,  d.  dname , d.  loc
          FROM emp e,dept  d
          WHERE e. deptno = d. deptno;
```

Display ENAME of the employees working at NEW YORK location ?

```
SQL>SELECT e. ename
      FROM  emp e, dept d
     WHERE      e. deptno = d. deptno
                  AND
                  d. loc='NEW YORK' ;
```

Display ENAME of the employees working at NEW YORK location and earning more than 2000 ?

```
SQL>SELECT e. ename
      FROM emp e,dept d
      WHERE   e. deptno=d. deptno
                AND
                d. loc='NEW YORK'  and e. sal > 2000;
```

Using ON clause :-

```
SQL>SELECT e. empno,e. ename,e. sal,d. dname,d. loc
         FROM emp e JOIN dept d
         ON (e. deptno = d. deptno.  ;
```

Using USING clause :-

```
SQL>SELECT  e. empno,e. ename,e. sal,d. dname,d. loc
         FROM emp e JOIN  dept d
         USING (DEPTNO)  ;
```

HINT :- In USING clause common column name should not be prefixed with table alias.

NOTE :- A join order is the order in which tables are accessed and joined together. For example, in a join order of table1, table2, and table3, table table1 is accessed first. Next,

table2 is accessed, and its data is joined to table1. Finally, table3 is accessed, and its data is joined to the result of the join between table1 and table2.

Non Equi Join :-

When the Join Cond is based on equality operator, the join is said to be an equi join. When the join condition based on otherthan equality operator , the join is said to be a non-equi join.

Syntax:-

Select col1,col2,…….
From <table 1>,<table 2>
Where <join cond> [AND <join cond> AND <cond> ----]

→In NON-EQUI JOIN JOIN COND is not based on = operator. It is based on other than = operator usually BETWEEN or > or < operators.

Example:-

Display EMPNO,ENAME,SAL,GRADE ?

```
SQL>  SELECT e. empno,e. ename,e. sal,s,grade
      FROM emp e, salgrade s
      WHERE e. sal BETWEEN s. losal AND s. hisal;
```

Display EMPNO,ENAME,SAL,DNAME,LOC,GRADE ?

```
SQL>SELECT e. empno,e. ename,e. sal,d. dname,d. loc,s. grade
      FROM emp e,dept d,salgrade s
      WHERE e. deptno = d. deptno
                  AND
          e. sal between g. losal AND g. hisal ;
```

Self Join :-

- Joining a table to itself is called Self Join.
- Self Join is performed when tables having self-referential integrity.
- To perform Self Join same table must be listed twice with different alias.
- Self Join is Equi Join within the table.

Syntax :-

```
SQl>SELECT <collist>
    From Table1 T1, Table1 T2
     Where T1. Column1=T2. Column2;
```

Example:-

Display EMPNO,ENAME,SAL,MGRNAME ?

```
SQL>SELECT e. empno,e. ename,e. sal,m. ename
        FROM emp e, emp m
        WHERE e. mgr = m. empno ;
```

```
SQL>SELECT e. empno,e. ename,e. sal,d. dname,d. loc,,s. grade,m. ename
        FROM emp e JOIN dept d
        USING(deptno.
        JOIN salgrade s
        ON (e. sal BETWEEN g. losal  AND g. hisal.
        JOIN emp m
        ON ( e. mgr = m. empno) ;
```

Outer Join:-

Equi join returns only matching records from both the tables but not unmatched record, an outer join retrieves a row even when one of the column in the join contains a null value. For example there are two tables one is CUSTOMER that stores customer information and another ORDERS table that stores orders placed by customers , INNER JOIN returns only the list of customer who placed orders,but OUTER JOIN also returns customer who did not placed any order. Outer join is 3 types.

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

To perform OUTER JOIN use Oracle Proprietary operator (+) .

Left Outer Join:-

LEFT OUTER JOIN returns all rows(matched and unmatched. from LEFT SIDE table and matching records from RIGHT SIDE table. To perform LEFT OUTER JOIN (+) should be on RIGHT SIDE.

Syntax :-

```
SELECT <collist> FROM <tablist>
        WHERE  t1. commoncolumn  =  t2. commoncolumn (+)
```

Example :-

Display EMPNO,ENAME,DNAME,LOC and also display employee list who are not assigned to any dept?

```
SQL>  SELECT e. empno,e. ename,d. dname,d. loc
```

FROM emp e, dept d
WHERE e. deptno = d. deptno (+) ;

ANSI Style :-

In SQL/92 standard use keyword LEFT OUTER JOIN instead of using operator (+. .
Display EMPNO,ENAME,DNAME,LOC and also display employee list who are not assigned to any dept?

SQL> SELECT e. empno,e. ename,d. dname,d. loc
FROM emp e LEFT OUTER JOIN dept d
USING(Deptno) ;

Right Outer Join:-

RIGHT OUTER JOIN returns all rows(matched and unmatched. from RIGHT SIDE table and matching records from LEFT SIDE table. To perform RIGHT OUTER JOIN use (+. on LEFT SIDE.

Syntax :-

SELECT <collist> FROM <tablist>
WHERE t1. commoncolumn(+) = t2. commoncolumn

Example :-

Display EMPNO,ENAME,DNAME,LOC and also display department which are empty ?

SQL> SELECT e. empno,e. ename,d. dname,d. loc
FROM emp e, dept d
WHERE e. deptno(+) = d. deptno ;

ANSI Style :-

In SQL/92 standard use keyword RIGHT OUTER JOIN instead of using operator (+. .
Display EMPNO,ENAME,DNAME,LOC and also display departments which are empty ?

SQL> SELECT e. empno,e. ename,d. dname,d. loc
FROM emp e RIGHT OUTER JOIN dept d
USING(Deptno);

Full Outer Join:-

→Returns all rows (matched and unmatched. from both tables.
→Prior to oracle 9i doesn't support FULL OUTER JOIN.
→To perform FULL OUTER JOIN in prior to ORACLE 9i.

```
SQL> SELECT e. empno,e. ename,d. dname,d. loc
     FROM emp e, dept d
     WHERE e. deptno  = d. deptno (+) ;
     UNION
     SELECT e. empno,e. ename,d. dname,d. loc
     FROM emp e, dept d
     WHERE e. deptno(+)  = d. deptno ;
```

HINT :-

(+. should be either left side or right side but cannot be on both sides.

CROSS JOIN :-

- CROSS JOIN returns cross product of two tables.
- Each record of one table is joined to each and every record of another table.
- If table1 contains 10 records and table2 contains 5 records then CROSS JOIN between table1 and table2 returns 50 records.
- ORACLE performs CROSS JOIN when we submit query without JOIN COND.

Syntax :-

```
SQL>SELECT  col1 ,col2  FROM  tab1 , tab2 ;
```

Example:-

TABLE		**TABLE**
ORDERS		**DISCOUNT**
ORDAMT		**DIS**
100000		5
	7	
	12	

Display ORDAMT for each and every DISCOUNT percentage ?

```
SQL>SELECT o. ordamt,d. dis, (o. ordamt*d. dis. /100  AS  amount
     FROM orders o,discounts d ;
```

ANSI Style :-

```
SQL>SELECT o. ordamt,d. dis, (o. ordamt*d. dis. /100  AS  amount
     FROM orders o CROSS JOIN discounts d ;
```

Natural Join :-

- NATURAL JOIN is possible in ANSI SQL/92 standard.

- NATURAL JOIN is similar to EQUI JOIN.
- NATURAL JOIN is performed only when common column name is same.
- in NATURAL JOIN no need to specify join condition explicitly , ORACLE automatically performs join operation on the column with same name. .

Example :-

SQL>SELECT e. empno,e. ename,e. sal,d. dname,d. loc
FROM emp e NATURAL JOIN dept d ;

Above query performs JOIN operation on DEPTNO.

Set Operators:-

- **UNION**
- **UNION ALL**
- **INTERSECT**
- **MINUS**

Syntax :-

SELECT statement 1
UNION / UNION ALL / INTERSECT / MINUS
SELECT statement 2 ;

Rules :-

1. No of columns returned by first query must be equal to no of columns returned by second query
2. Corrosponding columns datatype type must be same.

UNION:-

- UNION operator combines data returned by two SELECT statement.
- eliminates duplicates.
- Sorts result.

Example :-

1
```
SQL>SELECT job FROM emp WHERE deptno=10
UNION
SELECT job FROM emp WHERE deptno=20 ;
```

2
```
SQL>SELECT job,sal FROM emp WHERE deptno=10
UNION
SELECT job,sal FROM emp WHERE deptno=20
ORDER BY sal ;
```

NOTE:- ORDER BY clause must be used with last query.

UNION ALL:-

→UNION ALL is similar to UNION but it includes duplicates

Example :-

```
SQL>SELECT job FROM emp WHERE deptno=10
    UNION ALL
```

SELECT job FROM emp WHERE deptno=20 ;

Scenario :-

EMP1

EMPNO	ENAME	DNO
1	A	10
2	B	20

EMP2

EMPNO	ENAME	DNO
100	X	10
101	Y	20

DEPT:-

DNO	DNAME	LOC
10	ACCT	HYD
20	SALES	HYD

Display all employee list along with department names and locations ?

Solution :- (EMP1 union EMP2. Join DEPT

INTERSECT:-

INTERSECT operator returns common values from the result of two SELECT statements.

Example:-

Display common jobs belongs to 10^{th} and 20^{th} departments ?

SQL>SELECT job FROM emp WHERE deptno=10
INTERSECT
SELECT job FROM emp WHERE deptno=20;

MINUS:-

MINUS operator returns values present in the result of first SELECT statement and not present in the result of second SELECT statement.

Example:-

Display jobs in 10^{th} dept and not in 20^{th} dept ?

SQL>SELECT job FROM emp WHERE deptno=10
MINUS
SELECT job FROM emp WHERE deptno=20;

Sub queries

Sub query:-

- Query embedded in another query is called sub query.
- One query is called inner/child/sub query.
- Another query is called outer/parent/main query.
- The result of inner query acts as an input to outer query.
- Outer query can be INSERT,UPDATE,DELETE,SELECT
- Inner query must be always SELECT
- Sub queries can appear in
  WHERE CLAUSE
  HAVING CLAUSE
  FROM CLAUSE
  SELECT CLAUSE

Types of SUBQUERIES :-

- ➔ Single Row Subqueries
- ➔ Multi Row Subqueries
- ➔ Nested Queries
- ➔ Muliti Column Subqueries
- ➔ Co-related Subqueries

SINGLE ROW SUBQUERIES:-

If inner query returns only one row then it is called single row subquery.

Syntax :-

SELECT <collist> FROM <tabname>
WHERE colname OP (SELECT statement.

OP can be < > <= >= = <>

Example :-

Subqueries in WHERE clause :-

Display employee records whose job equals to job of SMITH?

```
SQL>SELECT * FROM emp
      WHERE job = (SELECT job FROM emp WHERE  ename='SMITH'.  ;
```

Display employee name earning maximum salary ?

```
SQL>SELECT ename FROM emp
  WHERE sal = (SELECT MAX(sal.  FROM emp.  ;
```

Display all records except last record ?

```
SQL>SELECT * FROM emp
 WHERE ROWID <(SELECT MAX(ROWID.  FROM emp.  ;
```

Subqueries with BETWEEN operator:-

Display employee reocrds earning salary between min sal of 10 dept and max sal of 30 dept ?

```
SQL>SELECT * FROM emp
      WHERE sal BETWEEN  (SELECT MIN(sal.  FROM emp WHERE
deptno=10.
                          AND
                          (SELECT MAX(sal.  FROM emp WHERE deptno=30.
;
```

Subqueries in HAVING clause:-

Display departmentss whose avg(sal. geater than avg(sal. of 10 dept?

```
SQL>SELECT deptno FROM emp
  GROUP BY deptno
   HAVING AVG(sal.  > (SELECT AVG(sal.  FROM emp
                              WHERE deptno=10.  ;
```

Subqueries in UPDATE command :-

Update employee salary to maximum salary whose empno=7369 ?

```
SQL>UPDATE emp SET sal = (SELECT MAX(sal.  FROM emp.   WHERE
EMPNO=7369 ;
```

Swap employee salaries whose empno in (7369,7499. ?

```
SQL>UPDATE emp SET sal=DECODE(empno,7369,(SELECT sal FROM emp
                                          WHERE empno=7499. ,
                                    7499,(SELECT sal FROM emp
                                          WHERE empno=7369.  . ;
```

Subqueries in DELETE command:-

Delete employee record whose job equals to job of SMITH ?

```
SQL>DELETE FROM emp
WHERE job= (SELECT job FROM emp WHERE
ename='SMITH'. ;
```

Multi Row Subqueries:-

if inner query returns more than one row then it is called multi row subquery.

Syntax :-

```
SQL>SELECT <collist> FROM <tabname>
   WHERE colname OP (SELECT statement.  ;
```

OP must be IN , NOT IN, ANY, ALL

Example :-

Displaye employee records whose job equals to job of SMITH or job of BLAKE ?

```
SQL>SELECT * FROM emp
```

WHERE job IN (SELECT job FROM emp WHERE ename IN ('SMITH','BLAKE'. . ;

Displaye employee records who are earning minimum and maximum salaries ?

```
SQL>SELECT * FROM emp WHERE sal IN (SELECT MIN(sal. FROM emp
                                    UNION
                                    SELECT MAX(sal. FROM emp);
```

Display 4^{th},7^{th},11^{th} record in EMP table ?

```
SQL>SELECT * FROM emp
 WHERE ROWID IN (SELECT DECODE(ROWNUM,4,ROWID,
                               7,ROWID,
                               11,ROWID)
                 FROM emp) ;
```

ANY operator:-

Compares a value to each value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=. Evaluates to FALSE if the query returns no rows.

Example:-

Select employees whose salary is greater than any salesman's salary ?

```
SQL>SELECT ename FROM emp
        WHERE SAL > ANY ( SELECT sal FROM emp WHERE job =
'SALESMAN') ;
```

ALL operator :-

Compares a value to every value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=. evaluates to TRUE if the query returns no rows.

Example:-

Select employees whose salary is greater than every salesman's salary ?

```
SQL>SELECT ename FROM emp
        WHERE SAL > ALL ( SELECT sal FROM emp WHERE job =
'SALESMAN') ;
```

Nested Queries:-

→A subquery embedded in another subquery is called NESTED QUERY.

→Queries can be nested upto 255 level.

Example :-

Display employee name earning second maximum salary ?

```
SQL>SELECT ename FROM emp
   WHERE sal = (SELECT MAX(sal.  FROM EMP
         WHERE sal < (SELECT MAX(sal.  FROM emp)  ;
```

Update the employee salary to maximum salary of SALES dept ?

```
SQL>UPDATE emp
      SET sal = (SELECT MAX(sal.  FROM emp
 WHERE deptno = (SELECT deptno FROM dept
                        WHERE dname='SALES')  ;
```

Multi Column Subqueries:-

If inner query returns more than one column value then it is called MULTI COLUMN subquery.

Example :-

Display employee names earning maximum salaries in their dept ?

```
SQL>SELECT ename FROM emp
      WHERE (deptno,sal.  IN (SELECT deptno,MAX(sal.
                              FROM emp
                              GROUP BY Deptno) ;
```

Co-related Subqueries:-

If a subquery references one or more columns of parent query is called CO-RELATED subquery because it is related to outer query. This subquery executes once for each and every row of main query.

Example :-

→Display employee names earning more than avg(sal. of their dept ?

```
SQL>SELECT ename FROM emp x
 WHERE sal > (SELECT AVG(sal.  FROM emp
                        WHERE deptno=x. Deptno) ;
```

→Display employee names earning more than their manager ?

```
SQL>SELECT ename FROM emp x
      WHERE sal > (SELECT sal FROM emp
                           WHERE empno=x. mgr) ;
```

→Delete duplicate records in a table ?

```
SQL>DELETE FROM emp X
```

WHERE ROWID > (SELECT MIN(ROWID. FROM emp
WHERE empno=x. empno
AND
ename=x. ename
AND
sal=x. sal) ;

Display top 3 maximum salaries in emp table ?

SQL>SELECT DISTINCT sal FROM emp a
WHERE 3 > (SELECT COUNT(DISTINCT sal.
FROM emp b
WHERE a. sal < b. sal) ;

Using EXISTS operator :-

→EXISTS operator returns TRUE or FALSE.

→If inner query returns at least one record then EXISTS returns TRUE otherwise returns FALSE.

→ORACLE recommends EXISTS and NOT EXISTS operators instead of IN and NOT IN.

Display dept which not empty ?

SQL>SELECT * FROM dept d
WHERE EXISTS (SELECT * FROM emp WHERE deptno =d. Deptno) ;
SQL>SELECT * FROM dept d
WHERE NOT EXISTS (SELECT * FROM emp
WHERE deptno = d. Deptno) ;

VIEWS

Data abstraction is usually required after a table is created and populated with data. Data held by some tables might require restricted access to prevent all users from accessing all columns of a table, for data security reasons. Such a security issue can be solved by creating several tables with appropriate columns and assigning specific users to each such table, as required. This answers data security requirements very well but gives rise to a great deal of redundant data being resident in tables, in the database. To reduce redundant data to the minimum possible, Oracle provides Virtual tables which are Views.

View Definition :-

→A View is a virtual table based on the result returned by a SELECT query.

→The most basic purpose of a view is restricting access to specific column/rows from a table thus allowing different users to see only certain rows or columns of a table.

Composition Of View:-

→A view is composed of rows and columns, very similar to table. The fields in a view are fields from one or more database tables in the database.

→SQL functions, WHERE clauses and JOIN statements can be applied to a view in the same manner as they are applied to a table.

View storage:-

→Oracle does not store the view data. It recreates the data, using the view's SELECT statement, every time a user queries a view.

→A view is stored only as a definition in Oracle's system catalog.

→When a reference is made to a view, its definition is scanned, the base table is opened and the view is created on top of the base table. This, therefore, means that a view never holds data, until a specific call to the view is made. This reduces redundant data on the HDD to a very large extent.

Advantages Of View:-

Security:- Each user can be given permission to access only a set of views that contain specific data.

Query simplicity:- A view can drawn from several different tables and present it as a single table turning multiple table queries into single table queries against the view.

Data Integrity:- If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraints.

Disadvantage of View:-

Performance:- Views only create the appearance of the table but the RDBMS must still translate queries against the views into the queries against the underlined source tables. If the view is defined on a complex multiple table query then even a simple query against the view becomes a complicated join and takes a long time to execute.

Types of Views :-

- **Simple Views**
- **Complex Views**

Simple Views :-

a View based on single table is called simple view.

Syntax:-

```
CREATE VIEW <View Name>
AS
SELECT<ColumnName1>,<ColumnName2>
FROM  <TableName>
[WHERE <COND>]
[WITH CHECK OPTION]
[WITH READ ONLY]
```

Example :-

```
SQL>CREATE VIEW emp_v
      AS
      SELECT empno,ename,sal FROM emp ;
```

→Views can also be used for manipulating the data that is available in the base tables[i. e. the user can perform the Insert, Update and Delete operations through view.

→Views on which data manipulation can be done are called Updateable Views.
→If an Insert, Update or Delete SQL statement is fired on a view, modifications to data in the view are passed to the underlying base table.
→For a view to be updatable,it should meet the following criteria:
→Views defined from Single table.
→If the user wants to INSERT records with the help of a view, then the PRIMARY KEY column(s. and all the NOT NULL columns must be included in the view.

Inserting record through view :-

```
SQL>INSERT INTO emp_v VALUES(1,'A',5000,200.  ;
```

Updating record throught view :-

```
SQL>UPDATE emp_v SET sal=2000 WHERE empno=1;
```

Deleting record throught view :-

SQL>DELETE FROM emp_v WHERE empno=1;

With Check Option :-

If VIEW created with WITH CHECK OPTION then any DML operation through that view violates where condition then that DML operation returns error.

Example :-

```
SQL>CREATE VIEW V2
        AS
        SELECT empno,ename,sal,deptno FROM emp
        WHERE deptno=10
        WITH CHECK OPTION ;
```

Then insert the record into emp table through view V2

SQL>INSERT INTO V2 VALUES(2323,'RAJU',4000,20. ;

The above INSERT returns error because DML operation violating WHERE clause.

Complex Views :-

A view is said to complex view

→If it based on more than one table

→Query contains

AGGREGATE functions

DISTINCT clause

GROUP BY clause

HAVING clause

Sub-queries

Constants

Strings or Values Expressions

UNION,INTERSECT,MINUS operators.

Example 1 :-

```
SQL>CREATE VIEW V3
AS
SELECT E. empno,E. ename,E. sal,D. dname,D. loc
FROM emp E JOIN dept D
USING(deptno. ;
```

Complex views are not updatable i. e. we cannot perform insert or update or delete operations on base table through complex views.

Example 2 :-

SQL>CREATE VIEW V2
AS
SELECT deptno,SUM(sal. AS sumsal
FROM EMP
GROUP BY deptno;

Destroying a View:-

The DROP VIEW command is used to destroy a view from the database.

Syntax:-

DROP VIEW<viewName>

Example :-

SQL>DROP VIEW emp_v;

Querying VIEWS information :-

USER_VIEWS
ALL_VIEWS
DBA_VIEWS

OCA questions :-

1. Which two statements are true regarding views? (Choose two. .

A. A subquery that defines a view cannot include the GROUP BY clause.
B. A view that is created with the subquery having the DISTINCT keyword can be updated.
C. A view that is created with the subquery having the pseudo column ROWNUM keyword cannot be
updated.
D. A data manipulation language (DML. operation can be performed on a view that is created with the
subquery having all the NOT NULL columns of a table.

2 You want to create a SALE_PROD view by executing the following SQL statement:

CREATE VIEW sale_prod
AS SELECT p. prod_id, cust_id, SUM(quantity_sold. "Quantity",
SUM(prod_list_price. "Price"
FROM products p, sales s
WHERE p. prod_id=s. prod_id
GROUP BY p. prod_id, cust_id;

Which statement is true regarding the execution of the above statement?

A. The view will be created and you can perform DML operations on the view.
B. The view will be created but no DML operations will be allowed on the view.

C. The view will not be created because the join statements are not allowed for creating a view.
D. The view will not be created because the GROUP BY clause is not allowed for creating a view.

3 Evaluate the following command:
CREATE TABLE employees
(employee_id NUMBER(2. PRIMARY KEY,
last_name VARCHAR2(25. NOT NULL,
department_id NUMBER(2. NOT NULL,
job_id VARCHAR2(8. ,salary NUMBER(10,2. . ;

You issue the following command to create a view that displays the IDs and last names of the sales staff in the organization:

CREATE OR REPLACE VIEW sales_staff_vu
AS
SELECT employee_id,last_name,job_id
FROM employees
WHERE job_id LIKE 'SA_%'
WITH CHECK OPTION;

Which two statements are true regarding the above view? (Choose two. .
A. It allows you to insert rows into the EMPLOYEES table.
B. It allows you to delete details of the existing sales staff from the EMPLOYEES table.
C. It allows you to update job IDs of the existing sales staff to any other job ID in the EMPLOYEES table.
D. It allows you to insert IDs, last names, and job IDs of the sales staff from the view if it is used in multitable INSERT statements.

Triggers

DESCRIPTION :

A trigger is a statement (action. that is executed automatically by the system (DBMS. in a side effect of the modification in the database.

A trigger is a procedure that is executed automatically whenever an event occurs in a database.

A trigger describes 3 parts.

1. **An Event:** A change in database that activates a trigger.
2. **A Condition**: It is a query of test, if true, then the trigger is activated.
3. **An Action :** It is the procedure that is executed when trigger is activated and its condition is true.

Triggers are useful mechanisms alternating or performing certain events automatically in database and some conditions are met.

A trigger can form the following events

1. Insert
2. Delete
3. Update A row form a relation.

SYNTAX:

Create TRIGGER <trigger_name>
BEFORE | INSERT |
AFTER | UPDATE| ON <TABLE _NAME>
|DELETE|
for each row
DECLARE
BEGIN

```
   DBMS_OUTPUT. PUT_LINE(‘Trigger Message’. ;
   <SQL STATEMENT>
 END;
```

LAB:

- - This trigger is used to raise an error when all the seats are reserved in a Bus.

```
-- Trigger DDL Statements
DELIMITER $$
USE `rwt`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `rwt`. `upchk`
AFTER INSERT ON `rwt`. `ticket`
FOR EACH ROW
Begin
DECLARE
TNSEATS INT(3. ;
DECLARE
CPCT INT(3. ;
  SELECT SUM(TICKET. NOOFSEATS. ,BUS. CAPACITY INTO TNSEATS,CPCT
FROM TICKET GROUP BY TICKET. BUSNO;
  IF TNSEATS > CPCT THEN
    CALL RAISE_APPLICATION_ERROR(-30000,'SEATS   ARE   NOT
AVAILABLE'. ;
  END IF;
 END$$
```

Procedures

This procedure is used to display the bus type, passenger name, source and destination when a passenger id is given.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `rwtdet` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `rwtdet`(pid int(11. .
BEGIN
DECLARE
BTYPE VARCHAR(45. ;
DECLARE
PNM VARCHAR(45. ;
DECLARE
SRC VARCHAR(45. ;
DECLARE
DEST VARCHAR(45. ;
SELECT BUS. BUSTYPE,PASSENGER. PNAME,TICKET. SOURCE,TICKET.
DESTINATION INTO BTYPE,PNM,SRC,DEST FROM BUS,TICKET,PASSENGER
WHERE TICKET. BUSNO=BUS. BUSNO AND PASSENGER. MOBNO=TICKET.
MOBNO;
 select BTYPE AS BUS_TYPE,PNM AS PASS_NAME,SRC AS PASS_SRC,DEST AS
PASS_DEST;
END $$

DELIMITER ;
```

OutPut:

```
mysql>  CALL rwt. rwtdet(213657. ;
```

BUS_TYPE	PASS_NAME	PASS_SRC	PASS_DEST
SUPER LUXURY	SREENIVAS	HYD	BGLR

Cursors

What is cursors?

The Oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is private to SQL's operations and is called a cursor.

The data is stored in the cursor is called the active data set.

Every SQL statement executed by the Oracle server has an individual cursor associated with it and are called implicit cursors. There are two types of cursors.

Implicit cursors: Declared for all DML and PL/SQL SELECT statements.
Explicit cursors: Declared and names by the programmer.

Explicit Cursors:

- Individually process each row returned by a multiple row select statement.
- A PL/SQL program opens a cursor, processes rows returned by a query, and then closes the cursor. The cursor marks the current position in the active set.

    - Can process beyond the first row returned by the query, row by row.
    - Keep track of which row is currently being processed.
    - Allow the programmer to manually control explicit cursors in the PL/QL block.

Controlling Explicit Cursors:

- Declare the cursor by naming it and defining the structure of the query to be performed. Within it.
- Open the cursor: The OPEN statement executes the query and binds the variables that are referenced. Rows identified by the query are called the active set and are now available for fetching.
- Fetch data from the cursor: After each fetch, you test the cursor for any existing row. If there are no more rows to process, then you must close the cursor.
- Close the cursor: The CLOSE statement releases the active set of rows. It is now possible to reopen the cursor to establish a fresh active set.

Syntax:

Declaring a cursor:

```
CURSOR cursor_name IS
    Select_statement;
```

Opening a cursor:

```
OPEN cursor_name;
```

Fetch data from a cursor:

```
FETCH cursor_name INTO [variable1, variable2,…. ]| record_name];
```

Closing a cursor:

```
Close cursor_name;
```

Attributes of an Explicit Cursor:

- %ISOPEN [is cursor open]
- %NOTFOUND [is row not found]
- %FOUND [is row found]
- %ROWCOUNT [rows returned so far]

Cursors can be passed parameters. Cursors also have FOR UPDATE option which allows more fine grained control of locking at a table level. WHERE CURRENT OF can be used to apply the update or delete operation to current row in the cursor.

This procedure includes a cursor which is used to display particular passenger details based onhis mobile number who have reserved the seats in different buses.

```
-- ----------------------------------------------------------------------------
-- Routine DDL
```

```
-- Note: comments before and after the routine body will not be stored by the server
-- ------------------------------------------------------------------------
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `pass_mob_tct`(mno bigint(11. .
BEGIN
declare
no_more_rows int default false;
declare
pid int(11. ;
declare
tid int(11. ;
declare
jdt date;
DECLARE
COUNT INT(3. ;
declare
C1 cursor for select passenger. PPNO,TICKETID,JDATE from passenger,TICKET
WHERE mno=ticket. mobno and mno=passenger. mobno AND TICKET.
MOBNO=PASSENGER. MOBNO;
DECLARE
CONTINUE HANDLER FOR NOT FOUND SET no_more_rows =TRUE;
OPEN C1;
TICK:LOOP
  FETCH C1 INTO PID,TID,JDT;
  IF no_more_rows THEN
   LEAVE TICK;
  END IF;

   SELECT PID AS PASS_ID,MNO AS MOBILE_NO,TID AS TCT_NO,JDT AS
JOURNEY_DATE;
  SET COUNT=(SELECT FOUND_ROWS(. . ;

END LOOP TICK;

  SELECT COUNT AS NO_OF_ROWS_FETCHED;
CLOSE C1;
END
```

OutPut:

mysql> CALL rwt. pass_mob_tct(7766445533. ;

PASS_ID	MOBILE_NO	TCT_NO	JOURNEY_DATE
213657	7766445533	10001	2012-05-01

NO_OF_ROWS_FETCHED
1

REFERENCE BOOKS:

1. SQL, PL/SQL The Programming Language of Oracle --Ivan Bayross
2. Introduction to SQL, Rick F. Vander Lans, Pearson education.
3. . Oracle PL/SQL, B. Rosenzweig and E. Silvestrova, Pearson education.