SQL

Data : It is a **RAWFACT** which describes the **ATTRIBUTES** of an **ENTITY.**

Database : It is a place or a medium in which the data is stored in **SYSTEMATIC** and **ORGANISED** manner.

e.g. :- Gmail db, Oracle db, IBM db2, etc.

Basic operations performed on a databases are :-

- Create (Insert)
- Read (Retrieve)
- Update (Modify)
- Delete (Drop)

DBMS (Data Base Management System)

- DBMS is a software which is used to maintain and manage the database.
- Security and Authorization are the main features provided by DBMS.
- We use Query Language to communicate with DBMS.

RDBMS (Relational Data Base Management System)

- RDBMS is a type of DBMS software in which we store the data in form of tables.
- We use **STRUCTURED QUERY LANGUAGE (SQL)** to communicate with RDBMS.

Tables : Table is a logical organisation of data which consists of rows and columns.

Columns : It is known as **Attributes** or **Fields.** It is used to represent all the properties of all the entities.

Rows : It is known as **Records** or **Tuples**. It is used to represent all the properties of a single entity.

Cell : It is the **intersection** of **rows** and **columns**. It is smallest unit in a table in which we store data.

Rules of "E.F Codd"

- According to "E.F Codd" the data entered into the cell should be single valued or atomic data.
- "E.F Codd" says we can store the data in multiple tables, if needed we can establish connection between two tables using key attributes.
- Data entering the table can be validated in two steps :-
  - (i) By assigning data types.
  - **(ii)** By assigning constrains.

Note: Data types are mandatory whereas constrains are optional.

Data Types : It defines the type of data that is being stored in the memory location.

Data Types in SQL

- CHAR
- VARCHAR/VARCHAR2
- NUMBER
- DATE
- LARGE OBJECTS { (I) Binary LOB, (ii) Character LOB}

CHAR

- Character data type can accept characters such as 'A-Z', 'a-z', '0-9' in string format & special characters.
- Whenever we use char datatype, we have to mention the size for it. The max size it can take is 2000 characters.
- It is a type of *fixed length memory allocation*.

Syntax : **CHAR(size)**

Example : **CHAR(10)**

VARCHAR

- Varchar data type can accept characters such as 'A-Z', 'a-z', '0-9' in string format & special characters.
- Whenever we use this datatype, we have to mention the size for it. The max size it can take is 2000 characters.
- It is a type of *variable length memory allocation*.

Syntax : **VARCHAR(size)**

Example : **VARCHAR(10)**

VARCHAR2

- It is an upgraded version of varchar where the max size is 4000 characters.

NUMBER

- Number datatype can accept any numeric value

Syntax : **NUMBER(Precision, [Scale])**

- ***Precision*** : It is used to determine the number of digits to store the value.

Example : **NUMBER(5)** ±

9	9	9	9	9

- ***Scale*** : It is used to determine the number of digits required to store decimal numbers in precision.

Example : **NUMBER(5,2)** ±

9	9	9

9	9

Note: (i) If the scale >= precision, the range starts with '0 . '

NUMBER(2,5)   0. $\boxed{}\boxed{}\boxed{}\boxed{}\boxed{}$

(ii) The max precision can be "38", and scale can be "127".

DATE

- The format provided by Oracle is **'DD-MON-YY'** or **'DD-MON-YYYY'**

  Syntax : **Date .**

LARGE OBJECT

- Character Large Object : It is used to store characters upto 4GB of size.

  Syntax: CLOB

- Binary Large Object : It is used to store binary values of Mp3, Mp4, images, docs, etc. upto 4GB of size.

  Syntax: BLOB

Constraints

- **UNIQUE :** This constraint is used to avoid **DUPLICACY** of values into the column.

- **NOT NULL :** This constraint is used to specify that the particular is **MANDATORY**.

- **CHECK :** It is an extra validation given to a column with a condition. If the condition is satisfied then the value is accepted else discarded.

  Example : **CHECK(Sal > 0).**

  CHECK(Length(Ph.No.) =10).

PRIMARY KEY : It is a constraint which is used to *identify a record uniquely* from the table.

- We can have **ONLY ONE** primary in a table.
- Primary Key cannot accept **DUPLICATE** value.
- Primary Key is always a combination of **UNIQUE & NOT NULL.**
- Primary Key is not **MANDATORY** but it is **HIGHLY RECCOMENDED**.

FOREIGN KEY : It is used to establish a connection between two tables.

- We can have **ANY NUMBER** of **FOREIGN KEY.**
- It can accept **DUPLICATE VALUES** and can be **NULL**.
- It is present in CHILD table but actually belongs to the PARENT table.
- An attribute defined as Primary Key in its table can only become Foreign Key in other table.

STUDENT TABLE

Stu_Id [P.K]	First_Name	Last_Name	Specialization
2019001	Mike	Hannigun	CSE
2019002	Robb	Walsh	ECE
2019003	Michael	Pompae	IS

$$\begin{tikzcd} \text{Stu\_Id [P.K]} \arrow[r] & \text{Stu\_Id [F.K]} \end{tikzcd}$$

SUBJECTS TABLE

Sub_Id [P.K]	Sub_Name	Stu_Id [F.K]
EA1140	Basics Of Electronics	2019002
FA2530	C++	2019002
CA1356	Engg. Mathematics	2019001

- *Assignment: List out the differences between (i) CHAR & VARCHAR,*

  (ii) PRIMAY KEY & FOREIGN KEY

Statements:

- **Data Definition Language {DDL}**
- **Data Manipulation Language{DML}**
- **Transaction Control Language{TCL}**
- **Data Control Language{DCL}**
- **Data Query Language{DQL}**

Data Query Language

i. **SELECT**
ii. **PROJECTION**
iii. **SELECTION**
iv. **JOIN**

SELECT :- This statement is used to **RETRIVE** the data from the database and **DISPLAY**.

PROJECTION :- The retrieval of data by selecting only the **COLUMN** name.

SELECTION :- The retrieval of data by selecting both **COLUMN** and **ROW** .

JOIN :- The retrieval of data from **MULTIPLE TABLE SIMULTANEOUSLY**.

- **PROJECTION**

  The retrieval of data by selecting only the **COLUMN** name.

*Syntax : SELECT * / [DISTINCT] COLUMN_NAME / EXPRESSION [ALIAS]*

*FROM TABLE_NAME **;***

- **From** clause executes FIRST.
- For **From** clause we can pass table_name as an argument.
- **From** clause is used to search the table in the database and put it under execution.
- **Select** clause executes after **From** clause.
- For **Select** clause we can pass **Asterisk (*), Column_Name, Expression** as an argument.
- **Select** clause is used to **Retrieve** the column from the table which is under execution and **Display** it.
- **Select** clause is responsible for preparing the **Result Table.**
- **Asterisk (*),** it means to select all the column present in the table.

Note: If **Asterisk (*)** *is used as an argument, then we have to use it individually.*

*'**TAB :** It is a TABLE in which all the table names are present.'*

- **<u>Expression</u>**

  I. Anything which yields results is known as Expression.
  II. Expression consists of **operands** and **operators**.
  III. Operands are of two types : (a) Column Name.
  (b) Direct/Indirect Value.

- **<u>Alias</u>**

  I. It is an alternative name given to a column present in the result table.
  II. We can assign alias name with or without using the Keyword "**AS".**
  III. We can assign alias name for multiple Column or Expression.
  IV. The alias name should be a **Single Word** OR a **String** enclosed within (**" "**).

*Example : (i)SELECT SAL*12 AS "Annual Salary"   (ii)SELECT SAL*12 ANNUAL_SALARY*

FROM EMP ;   FROM EMP ;

<u>Description of Table :-</u> Syntax :- *DESC TABLE_NAME;*

- **DISTINCT**

I. **Distinct** clause is used to eliminate the duplicate records present in the result table.
II. **Distinct** clause has to be written as the first **argument** for **Select** clause.
III. **Distinct** clause can have multiple column names as argument. Distinct clause will eliminate the **combination of columns** which are duplicate.

➢ **SELECTION**

The retrieval of data by selecting both **COLUMN** and **ROW** .

*Syntax : SELECT * / [DISTINCT] COLUMN_NAME / EXPRESSION [ALIAS]*

FROM TABLE_NAME

*WHERE < FILTER CONDITION > **;***

ORDER OF EXECUTION

I. FROM
II. WHERE
III. SELECT

- **WHERE**

I. **Where** clause is used to **filter** the records.
II. **Where** clause executes **row by row.**
III. For where clause we have to provide **condition as an argument**.
IV. We can write multiple condition as well.
V. We cannot use **alias** name in Where clause.

Questions.

- **<u>OPERATORS</u>**

1. Arithmetic Operators (**‘+’ , ‘ - ’ , ‘ * ’ , ‘/ ’**)
2. Concatenation (‘||’)
3. Logical Operators (‘AND’ , ‘OR’ , ‘NOT’)
4. Relational Operators (‘>’ , ‘<’ , ‘>=’ , ‘<=’ , ‘=’ , ‘!= or <>’)
5. Special Operators { (IN, NOT IN); (BETWEEN, NOT BETWEEN), (IS ,IS NOT); (LIKE, NOT LIKE) }
6. Sub Query Operators (All , Any , Exists , Not Exists)

<u>Special Operators</u>

I. **IN** :- It is a multivalued operator which can accept one value at the L.H.S and multiple value at it’s R.H.S . It returns true if any of the conditions is satisfied.

Syntax: Col/Exp IN (V1, V2, …….Vn) ;

II. **NOT IN** :- It is a multivalued operator which can accept one value at the L.H.S and multiple value at it’s R.H.S . It returns false in any of the conditions is satisfied.

Syntax: Col/Exp NOT IN (V1,V2,…..Vn) ;

III. **Between :-** It is used whenever we have Range of Values. **Between** operator works including the ranges.

Syntax: Col/Exp Between lower_range AND higher_range ;

IV. **NOT BETWEEN :-** It is used whenever we have Range of Values. It works excluding the ranges.

Syntax: Col/Exp NOT Between lower_range AND higher_range ;

V. **IS :-** It is used to verify whether the value present at L.H.S is NULL.

Syntax: Col/Exp IS NULL ;

VI. **IS NOT :-** It is used to verify whether the value at L.H.S is NOT NULL.

Syntax: Col/Exp IS NOT NULL ;

VII. **LIKE :-** It is used to perform pattern matching.
To perform pattern matching we use special characters **' % ' and ' _ ' .**

Percentile (%) :- It is used to match any number of characters any number of times.

Underscore (_) :- It can exactly accept one character.

Syntax: Col/Exp LIKE (' % ') ; Col/Exp LIKE ('_ ') ;

Note: **Escape** :- This character is used to remove the special behaviour given to a special character.

Syntax: 'pattern_to_match' ESCAPE 'char' ;

Q. WAQTD name of the employee having _ in there name.

```
Ans. SELECT ENAME
     FROM EMP
     WHERE ENAME LIKE (' % !_ %') ESCAPE '!' ;
```

FUNCTIONS

- Functions are block of codes or set of instructions which performs a particular task.

- Major Components of Function are :-
  I. Function Name.
  II. Number of Args. / Type of Args.
  III. Return Type.

I. Type of Functions in SQL
   I. Single Row Functions.
   II. Multi Row / Aggregate / Group Functions.

Single Row Functions :- (i) It takes an input, processes it, executes it and gives output; and then takes the second input.

(ii) If we pass 'n' number of inputs to single row function it returns 'n' number of outputs.

Multi Row Functions :- (i) These function will aggregate (combine) all the inputs at once and executes, returning single output.

(ii) If we pass 'n' number of inputs to multi row function it returns one output.

(iii) Types of Multi Row Functions :-

1) MAX ( )
2) MIN ( )
3) AVG ( )
4) SUM ( )
5) COUNT ( )

NOTE:

1. *Multi Row Functions can accept only one argument at a time.*
2. *We cannot write column name along with a Multi Row Function.*
3. *Any number of Multi Row Functions can be used together.*
4. *We cannot use Multi Row Functions in Where clause.*
5. *Multi Row Functions ignore the NULL values.*
6. *Multi Row Functions cannot be NESTED.*

GROUP BY

This clause is used to **GROUP** the records.

Syntax :
```
SELECT  * /  [DISTINCT] COLUMN_NAME / EXPRESSION  [ALIAS]
  FROM TABLE_NAME
 [ WHERE < FILTER CONDITION > ]
GROUP BY COL_NAME/EXP;
```

ORDER OF EXECUTION

I. FROM
II. WHERE (if used)
III. GROUP BY
IV. SELECT

NATURE OF EXECUTION

I. FROM
II. WHERE => ROW BY ROW
III. GROUP BY => ROW BY ROW
IV. SELECT => GROUP BY GROUP

1) Group By clause executes Row By Row.
2) After execution of Group By clause we get Groups.
3) Any clause which executes after Group By clause will execute Group By Group.
4) Column and Expression used in Group By clause can be used in Select clause.

HAVING CLAUSE

This clause is used to **FILTER** the **GROUPS**.

```
Syntax :    SELECT  * /  [DISTINCT] COLUMN_NAME / EXPRESSION  [ALIAS]
              FROM TABLE_NAME
             [ WHERE < FILTER CONDITION > ]
            GROUP BY COL_NAME/EXP
          HAVING < GROUP FILTER CONDITION > ;
```

ORDER OF EXECUTION

I. FROM
II. WHERE (if used)
III. GROUP BY
IV. HAVING
V. SELECT

NATURE OF EXECUTION

I. FROM
II. WHERE => ROW BY ROW
III. GROUP BY => ROW BY ROW
IV. HAVING => GROUP BY GROUP
V. SELECT => GROUP BY GROUP

1) HAVING clause executes Group by Group.
2) HAVING clause executes after Group By clause.
3) HAVING clause executes after Group By and hence we can use multi row functions in having clause.
4) In Having clause we write Group Filter Condition which includes multi row functions.

Q. Differentiate between HAVING CLAUSE & WHERE CLAUSE.

ORDER BY

This clause is used to sort the results in either Ascending or Descending Order.

```
Syntax :   SELECT  * /  [DISTINCT] COLUMN_NAME / EXPRESSION  [ALIAS]
             FROM TABLE_NAME
            [ WHERE < FILTER CONDITION > ]
           [ GROUP BY COL_NAME/EXP ]
         [ HAVING < GROUP FILTER CONDITION > ]
         ORDER BY COL_NAME/EXP ' [ASC]/ DESC ' ;
```

ORDER OF EXECUTION

I. FROM
II. WHERE (if used)
III. GROUP BY
IV. HAVING
V. SELECT
VI. ORDER BY

NATURE OF EXECUTION

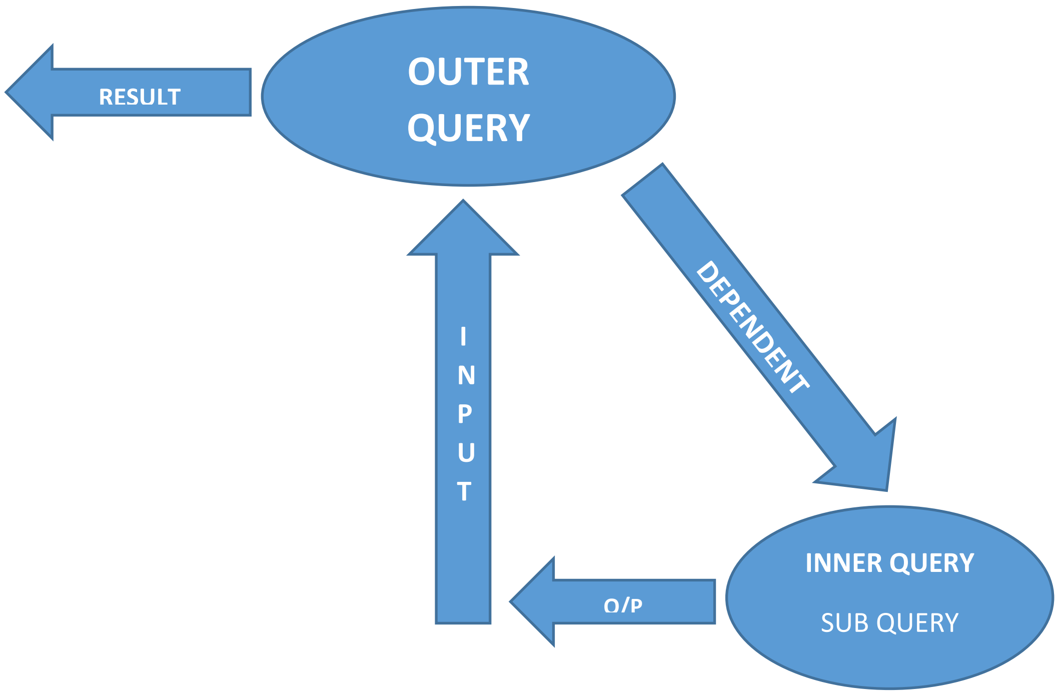
I. FROM
II. WHERE => ROW BY ROW
III. GROUP BY => ROW BY ROW
IV. HAVING => GROUP BY GROUP
V. SELECT => GROUP BY GROUP
VI. ORDER BY => ROW BY ROW

1) ORDER BY clause executes after SELECT clause.
2) ORDER BY clause has to written as the last statement in the Query.
3) We can use alias name in ORDER BY clause as an Argument.
4) We can sort the result table using Multiple column names.
5) All the tables present in the database will be sorted on the basis of primary key.
6) ORDER BY clause sorts the record in Ascending order by default.

<u>SUB QUERY</u>

A query written inside another is known as **SUB QUERY**.

Working Principle :



1. Inner query executes first.
2. The output of the inner query is given as an input the outer query.
3. Outer query generates the results.
4. Therefore, the outer query is dependent upon the inner query.

Q. Why do we use Sub Query?

Ans. (i) Whenever the data to be selected and the condition to be executed are in different tables we use sub-queries.

(ii) Whenever we have unknowns we write sub-queries to find them.

e.g. WAQTD details of the employees, who are earning more than SCOTT.

```
SELECT *
FROM EMP
WHERE SAL > (SELECT SAL
             FROM EMP
             WHERE ENAME = 'SCOTT' ) ;
```

In the above query, the salary of SCOTT is unknown and hence we use sub-query to find it.

JOINS

The retrieval of data form multiple tables simultaneously is known as **JOINS.**

TYPES OF JOINS :-

1. CARTESIAN JOIN / CROSS JOIN
2. INNER JOIN / EQUI JOIN
3. OUTER JOIN
   (i) LEFT OUTER JOIN
   (ii) RIGTH OUTER JOIN
   (iii) FULL OUTER JOIN
4. SELF JOIN
5. NATURAL JOIN

<u>CARTESIAN JOIN</u> :- Record from table 1 will be merged with all the records of table 2.

```
Syntax :-   ANSI :   SELECT  *
                   FROM table_name 1 CROSS JOIN table_name 2 ;
        ORACLE :  SELECT *
                  FROM table_name 1 , table_name 2 ;
```

NOTE : (i) Number of Columns = col_of_tab1 + col_of_tab2.

*(ii) Number of Rows = row_of_tab1 * row_of_tab2.*

<u>INNER JOIN / EQUI JOIN</u> :- Inner join is used to obtain only matching records from different tables.

OR

Inner join is used to obtain records which are present pairs from different tables.

```
Syntax :-   ANSI :   SELECT  *
                   FROM table_name 1 INNER  JOIN table_name 2
                  ON < join condition >  ;
```

Join Condition : table_name 1 . col_name = table_name 2 . col_name.

```
ORACLE :   SELECT *
           FROM table_name 1 , table_name 2
          WHERE < join condition >  ;
```

<u>NATURAL JOIN</u> :- Natural join is similar to Inner join wherein we don’t specify any join condition.

Natural join provides Inner join output if there are common columns present, else it returns Cartesian join output.

*Syntax :- ANSI : SELECT **

```
                  FROM table_name 1 NATURAL JOIN table_name 2 ;
```

<u>OUTER JOIN</u> :- Outer joins are used to obtain unmatched records.

LEFT OUTER JOIN :- It is used to obtain unmatched records from left table.

OR

It is used to obtain records of left table which do not have pair in right table.

*Syntax :- ANSI : SELECT **

```
                  FROM table_name 1 LEFT [OUTER] JOIN table_name 2
                  ON < join condition > ;
```

*ORACLE : SELECT **

```
                  FROM table_name 1 , table_name 2
                  WHERE < join condition > ( + ) ;
```

RIGHT OUTER JOIN :- It is used to obtain unmatched records from right table.

OR

It is used to obtain records of right table which do not have pair in left table.

*Syntax :- ANSI : SELECT **

```
FROM table_name 1 RIGHT [OUTER] JOIN table_name 2
ON < join condition > ;
```

*ORACLE : SELECT **

```
FROM table_name 1 , table_name 2
WHERE (+) < join condition > <filter condition> ;
```

FULL OUTER JOIN :- It is used to obtain unmatched records from both the tables.

*Syntax :- ANSI : SELECT **

```
FROM table_name 1 FULL [OUTER] JOIN table_name 2
ON < join condition > ;
```

*NOTE :- There is no **full outer join** syntax in Oracle.*

SELF JOIN :- Joining of two same tables is known as Self Join.

We use self join whenever the data to be selected is in the same table but present in two different records.

Syntax :-   *ANSI :*

```
SELECT  *
FROM table_name T1  JOIN  table_name T2
ON < join condition >  ;
```

ORACLE :

```
SELECT *
FROM table_name T1 , table_name T2
WHERE  < join condition >  ;
```

SINGLE ROW FUNCTIONS

CASE MANIPULATION FUNCTION :

	Syntax :-
a) UPPER ( )	*UPPER (‘string’)*
b) LOWER ( )	*LOWER (‘string’)*
c) INITCAP ( )	*INITCAP (‘string’)*

CHARACTER MANIPULATION FUNCTION :

a) LENGTH ( )	*LENGTH (‘string’)*
b) REVERSE ( )	*REVERSE (‘string’)*
c) SUBSTR ( )	*SUBSTR (‘original string’ , ‘position’ , [length])*
d) INSTR ( )	*INSTR (‘original string ’, ’sub string’ , ‘position’ , [nth occurrence])*
e) REPLACE ( )	*REPALCE (‘original string’ , ‘old substr’ , ‘new substr’)*
f) CONCAT ( )	*CONCAT (string1 , string2)*
g) TRIM ( )	*TRIM (leading/trailing/both ‘char’ FROM ‘string’)*

i. Length ( ) is used to obtain number of characters present in a given string.

```
e.g.  SELECT LENGTH (‘ length’)
      FROM DUAL ;
```

ii. Reverse ( ) is used to reverse the given string.

```
e.g. SELECT REVRSE (‘reverse’)
     FROM DUAL ;
```

iii. Substr ( ) is used to obtain a substring from a given original string.

```
   SUBSTR (‘original string’ , ‘position’ , [length])
```

Args (i) Original String :- It the string from which substring is extracted.

Args (ii) Position :- It specifies from where to begin the extraction. Position can be –ve or +ve. (**THE EXTRACTION IS TOWARDS RIGHT SIDE)**.

Args (iii) Length :- It specifies the number of characters to be extracted.
It is not mandatory. Where the length is not mentioned it assumes the complete length.

```
e.g. SELECT SUBSTR(‘bangalore’ , 2 , 4),
            SUBSTR('bangalore', -5, 3),
            SUBSTR('bangalore', 11, 4)
            FROM DUAL ;
```

iv. Instr ( ) is used to obtain index value if the given substring present in the original string.

```
e.g. SELECT INSTR (‘BANANA’ , ‘NA’ , 1 , 2)
     FORM DUAL ;
```

v. Replace ( ) is used to replace substring in a given string.

```
e.g. SELECT REPLACE ( ‘BANGALORE’ , ‘ B ‘ , ‘ M ‘ )
            REPLACE ( ‘BANGALORE’ , ‘ B ‘ )
            FROM DUAL ;
```

vi. Concat ( ) is used to merge the given two strings.

```
e.g. SELECT CONCAT( ENAME , CONCAT (‘    ‘ , SAL))
       FROM EMP ;
```

vii. Trim ( ) is used to remove the repeated character form the beginning or from the end or from both sides in the given string.

```
e.g. SELECT TRIM( BOTH ‘E’ FROM ‘EEEESQLEEE’)
      FROM DUAL ;
```

<u>DATE SINGLE ROW FUNCTION</u> :-

NOTE :-

1. *SYSDATE : It is used to obtain current date of system in which the RDBMS is installed.*

   ```
   e.g.  SELECT SYSDATE
           FROM DUAL ;
   ```

2. *SYSTIMESTAMP : It is used to obtain current date , time and time zone of the system.*

I. ADD_MONTHS ( ) : This function is used to add number of months to a given date.

Syntax :- ADD_MONTHS ('DATE' , NUM_OF_MONTHS)

e.g.
```
SELECT ADD_MONTHS ( SYSDATE , 4 )
FROM DUAL ;
```

II. MONTHS_BETWEEN ( ) : This function is used to obtain number of between given two dates.

Syntax : MONTHS_BETWEEN ('DATE 1' , 'DATE 2')

e.g.
```
SELECT MONTHS_BETWEEN ( SYSDATE , '31-DEC-19')
FROM DUAL ;
```

III. LAST_DAY ( ) :- This function is used to obtain the last day of the given month in the given date.

Syntax : LAST_DAY ('DATE ')

e.g.
```
SELECT LAST_DAY ( SYSDATE)
FROM DUAL ;
```

IV. EXTRACT ( ) :- This function is used to obtain day, month and year in numeric format.

Syntax : EXTRACT (DAY/MONTH/YEAR FROM TO_DATE ('DATE'))

e.g.
```
SELECT EXTRACT  (MONTH FROM  SYSDATE) ,
       EXTRACT (DAY FROM TO_DATE ( 'DATE') )
       FROM DUAL ;
```

NOTE :

- *We can subtract two given dates to find the number of days between them.*
- *We can add number of days to a given date.*
- *We can subtract number of days from a given date.*
- ***We cannot add two given dates.***

e.g. SELECT TO_DATE('31-DEC-19) – TO_DATE('01-DEC-19)

FROM DUAL ;

<u>CONVERSION SINGLE ROW FUNCTION</u>

(i) TO_DATE ( )
(ii) TO_CHAR ( )

To_Date :- This function is used to convert date to string format.

Syntax : TO_DATE ('date string')

To_Char :- This function is used to convert given date to char string format

Syntax : TO_CHAR (to_date('date') , 'format model')

TO_CHAR (DATE, 'format model')

FORMAT MODEL :

- YEAR – TWENTY NINTEEN
- YYYY – 2019
- YY – 19
- MONTH – MARCH
- MON – MAR
- MM – 03
- DAY – MONDAY
- DY – MON
- DD – 06
- D – 3(DAY OF WEEK)

<u>NUMBER SINGLE ROW FUNCTION</u>

- MOD ( )
- POWER ( )
- ABS ( )
- SQRT ( )
- ROUND ( )
- TRUNC ( )

(i) MOD ( ) is used to find the modulus of the given number.

Syntax : MOD (m , n)

(ii) POWER ( ) is used to obtain the exponential value of the given number.

Syntax : POWER (n , e) $\{ n^e \}$

(iii) ABS ( ) [absolute] is used to obtain only the magnitude of a given number.

Syntax : ABS (± number)

(iv) SQRT ( ) is used to obtain the sqr-root of the given number

Syntax : SQRT (number)

(v) ROUND ( ) is used to round off the given number to the nearest value.

Syntax : ROUND (number, ± scale)

-ve scale means before the decimal point, and count starts from (-) 1.

+ve scale means after the decimal point, and the count starts from 0.

(vi) TRUNC ( ) is used to round off the value always to the lower value.

Syntax : TRUNC (number)

General Single Row Function

- NVL ( ) [Null Value Logic]

  - This function is used to overcome the drawbacks of Null.
  - This function can accept two arguments

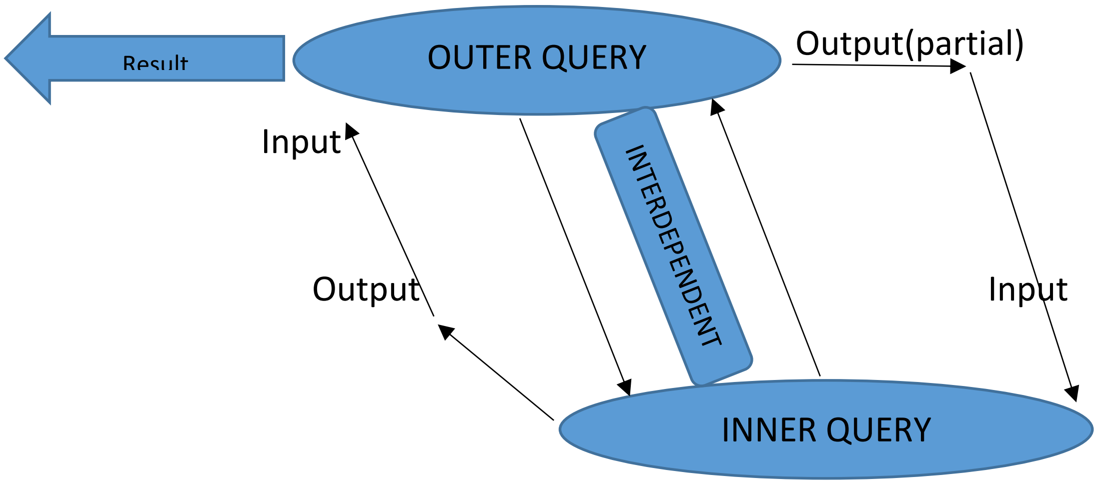
  NVL (arg1 , arg2)

  - Arg 1 : In this argument(arg1) we write the col_name in which values can be Null. If the value of this argument is null, then the value present in the second argument(arg2) will be returned, else value of argument(arg1) will be retained.

  - Arg 2 : In this argument we write the value that has to be substituted if the value of arg1 is null.

  e.g. SELECT ENAME , SAL+NVL(COMM,0)
  FROM EMP ;

Co-Related Sub Query



- In co-related sub-queries the outer query is dependent on the inner query, and the inner query is dependent on the outer query
- In co-related sub-queries the outer query executes first, but partially.
- The output of the outer query is given as input to inner query and the inner query executes completely.
- The output of the inner query is given as input the outer query, and outer query executes completely and gives out the Result.
- Therefore, in co-related sub-queries inner query and outer query are interdependent.

NOTE :

a) Co-related sub-queries work with the principle of both sub-queries and joins.
b) In co-related sub-queries the join condition has to be written in inner query, such that one of the columns in the join condition should belong to the outer query.

Statements

1) Data Definition Language

a) Create : To create a table

```
Syntax :   CREATE TABLE   tab_name
             ( col_nm1  datatype(size)  null / not null ,
              col_nm2  datatype(size)  null / not null ,
                .
                .
                .
             col_nmn  datatype(size)  null / not null ,
            constraint constraint_ref_name  constraint(col_nm), ) ;
```

NOTE : DDL statements are AUTO-COMMIT statements.

To Create Database

Syntax : Create Database database_name ;

To Create View

```
Syntax : Create View view_name
          As
         Select *
        From table_name;
```

View is virtual table based on the result set of an SQL STATEMENT.

If we change any thing in the original table of view those changes will be reflected in the view.

---

b) Rename :

```
Syntax : RENAME current_name
         TO  new_name ;
```

---

c) Alter :

To Add Column

```
Syntax : Alter Table tab_name
         Add col_name datatype(size) [null / not null ] ;
```

To Drop Column

```
Syntax : Alter Table tab_name
         Drop Column col_name ;
```

To Modify Column

```
Syntax : Alter Table table_name
        Modify col_name     new_datatype ;
```

To Change NOT NULL Constraint :

Syntax : Alter Table table_name
Modify col_name   existing_datatype NULL / NOT NULL ;

To Rename a Column :

Syntax : Alter Table table_name
Rename Column current_name   To   new_name ;

To Modify Constraints :

Syntax :

i) Alter Table table_name
Add Constarint const_ref_name UNIQUE (col_name) ;

ii) Alter Table table_name
Add Constarint const_ref_name CHECK (condition);

iii) Alter Table table_name
Add Constarint const_ref_name PRIMARY KEY (col_name);

iv) Alter Table table_name
Add Constarint const_ref_name FOREIGN KEY (col_name)
References parent_table_name (col_name) ;

---

d) Truncate : It is used to remove all the records from the table permanently.

Syntax : Truncate Table table_name ;

---

e) Drop :

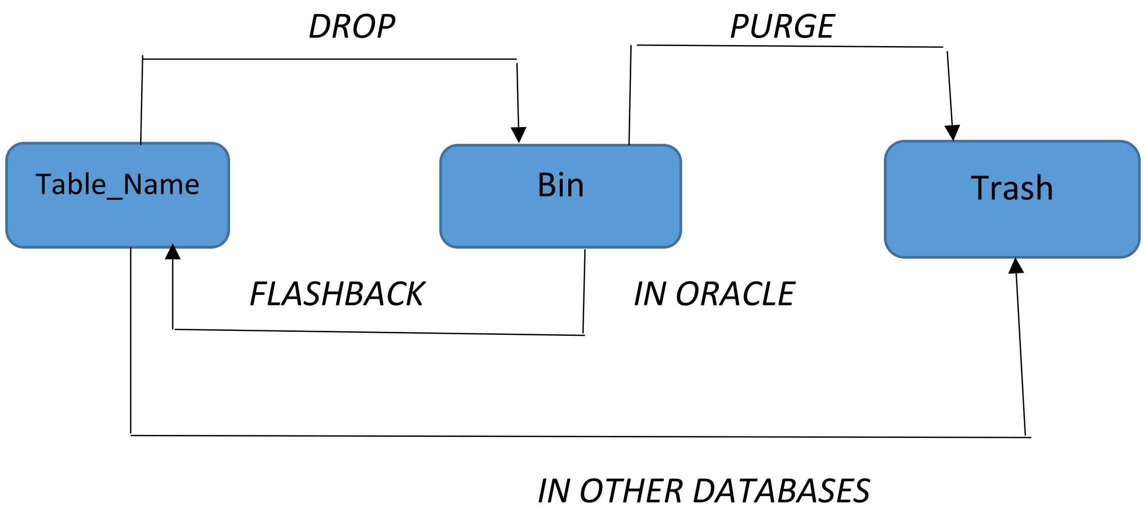
Syntax : Drop Table table_name ;

To Recover the Table (only in oracle)

Syntax : Flashback Table table_name To Before Drop [Rename to new name]

To Drop the table from Bin.

Syntax : Purge Table table_name ;



---

2) Data Manipulation Language

a) Insert : It is used to create records into a table.

Syntax : Insert Into table_name Values (V1, V2, ……………. V_n)

Insert Into table_name (col1, col2, ……………………. col_n)
Values (V1, V2, ……………. V_n)

Insert Into table_name (col1, col2, ……………………. col_n)
Values (&col1, &col2, ……………………. &col_n)

b) Update : It is used to update records into a table.

Syntax : Update table_name
Set Col1=V1, Col2=V2, ………… Col_n=V_n
[Where < filter condition >] ;

c) Delete : It is used to delete records into a table.

   Syntax : Delete From table_name
   [Where < filter condition >] ;

---

3) <u>Transaction Control Language</u>

a) Commit : It is used to save changes invoked by a transaction to the database.
   Syntax : Commit ;

b) Savepoint : It is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
   Syntax : Savepoint   savepoint_name ;

c) Rollback : This command is used to restore the database to the last committed state. It can also be used with savepoint command to jump to a savepoint in a transaction.
   Syntax : Rollback ;
   Rollback To savepoint_name ;

---

4) <u>Data Control Language</u>

a) Grant : It is used to provide any user access privileges or other privileges for the database.
   Syntax : Grant sql_stmt On table_name To user_name ;

b) Revoke : It is used to take back privileges from any user.

   Syntax : Revoke sql_stmt On table_name From user_name ;

Attributes

1) Key Attribute : The attribute with the help of which we can identify a record uniquely is known as Key Attribute.

   e.g. Aadhar Card No. , Passport No., D.L. No. , etc.

2) Non-Key Attribute : All the attributes except the key attribute is known as non-key attribute.

   e.g. Name, D.O.B, Job, Sal, etc.

3) Prime Key Attribute : Among the key attributes, an attribute is chosen to identify the records uniquely from the table. This key attribute is known as Prime Key Attribute.

4) Non-Prime Key Attribute : All the key attributes except the prime key attribute is known as non-prime key attribute.

5) Composite Key Attribute : The combination of two or more non-key attributes which is used to identify a record uniquely from the is known as composite key.

6) Super Key Attribute : The set of all the key attributes is known as Super key attribute.

7) Foreign Key Attribute : It behaves as an attribute of another table to represent the relationship.

Functional Dependency

As the name suggests it is a relationship between attributes of a table, dependent on each other. It helps in preventing data Redundancy and Anomalies.

Let us consider a relation 'R' with some attributes in which an attribute is determining another attribute or an attribute is dependent on another attribute, then there exists a functional dependency.

R $\longrightarrow$ *{ X , Y }*	*{ Empid , Ename }*
X is determining Y	*Empid is determining Ename*
Y is dependent on X	*Ename is dependent on Empid.*

Types of Functional Dependency :

1) Total Functional Dependency : In a relation all the attributes are determined by a single attribute which is a key attribute.

   e.g. Let us consider a relation 'R' with attributes A,B,C,D & A is key attribute.

A $\longrightarrow$ *{ B }*

A $\longrightarrow$ *{ C }*

A $\longrightarrow$ *{ D }*

A $\longrightarrow$ *{ A,B,C }* *(Total Functional Dependency)*

2) Partial Functional Dependency : A relation is said to have partial functional dependency if it consists of a composite key attribute.

   If there exist a dependency such that an attribute is determined by another attribute which is a part of a composite key attribute.

   (A , B) $\longrightarrow$ *{ D }*

   B $\longrightarrow$ *{ D }* *(Partial Functional Dependency)*

3) Transitive Functional Dependency : There exists a dependency such that a non-key attribute determines an attribute which is intern determined by a key attribute.

$R \longrightarrow \{ A, B,C,D \}$
$A \longrightarrow B$
$A \longrightarrow C$
$A \longrightarrow D$
$C \longrightarrow D$

- Redundancy : Repetition of unwanted data is called as redundancy.
- Anomaly : These are the side-effects that occur due to Data Manipulation Language .

<u>Normalization</u>

The process of decomposing a large table into smaller table in order to remove the redundancies and anomalies by identifying the dependency.

OR

The process of reducing the table towards normal form is normalization.

Normal Form : A table without redundancies and anomalies.

Levels of Normal Form : (I) First Normal Form (1 NF)
(II) Second Normal Form (2 NF)
(III) Third Normal From (3 NF)
(IV) Boyce_Codd NF (BNCF)

Note : A table is said to be normalised if the given table is reduced to third normal form.

- First Normal Form : A table is said to be in first normal form if it satisfies the following condition :-

    a) The table should not contain duplicate records.
    b) Cell should consist of single valued data or multivalued data should not be present.

- Second Normal From : A table is said to be in first normal form if it satisfies the following condition :-

    a) The table should be in 1st normal form.
    b) The table should not contain partial function dependency.

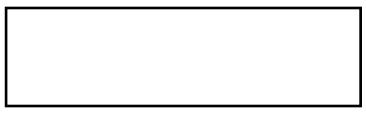
Note : If the table consist of partial f d then the attribute responsible are removed form the table.

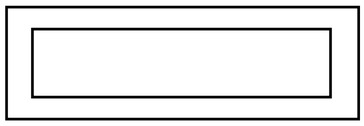
- Third Normal Form : A table is said to be in 3rd normal from if it satisfies following conditions :-

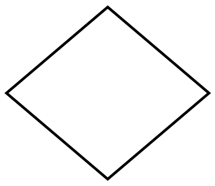
    a) The table should be in 2nd normal form
    b) The table should not contain transitive functional dependency.

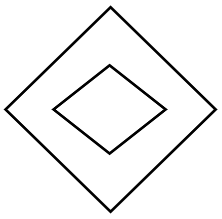
Note : If the table contains transitive fd then the attributes responsible are removed from the table.

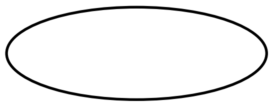
Entity Relationship Diagram (Schema) a.k.a E R Diagram

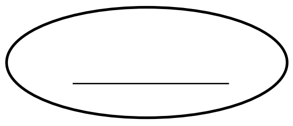

Entity (Strong Entity)

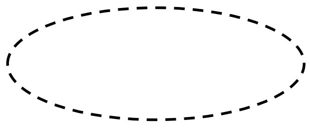

Weak Entity

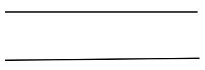

Strong Relationship

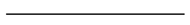

Weak Relationship


Attribute


Key Attribute


Derived Attribute


Total Participation


Partial Participation

Union & Union All

Union : This set operator returns results from all the participating queries after eliminating duplications.

Union All : This operator returns results from all the participating queries, including the duplications.

Intersect : This operator returns rows that are common to both queries.

Minus : This operator returns rows in the first query that are not present in the second query.

Note : Number of columns and datatype returned by the query must match, however the column name can be different.