DATA TYPES:

SQL data types classify the kind of data a database column can store, ensuring consistency and preventing errors.

It is used to specify or determine the type of data that will be stored in a particular memory location.

They are of :
1. CHAR
2. VARCHAR / VARCHAR2
3. DATE
4. NUMBER
5. LARGE OBJECTS
   (i) Character Large Object.
   (ii) Binary Large Object.

NOTE: SQL is not a Case Sensitive Language.

1. CHAR : In character datatype we can store 'A-Z' , 'a-z' , '0-9', And Special Characters($, & , @ , ! …) .

Characters must always be enclosed within single quotes ' '.

Whenever we use the char datatype, we must mention the size

Size: it is used to specify the number of characters it can store.

The maximum number of characters it can store is 2000ch.

Char follows fixed-length memory allocation.

Syntax: CHAR(SIZE)

1. VARCHAR : In varchar datatype, we can store 'A-Z' , 'a-z' , '0-9', and Special Characters($, & , @ , ! …) .

- Characters must always be enclosed within single quotes ' '.
- Whenever we use char datatype, we must mention the size
- Size: it is used to specify the number of characters it can store.

The maximum number of characters it can store is 2000ch.

- VarChar follows variable-length memory allocation.

Syntax: VARCHAR(SIZE)

NOTE: VARCHAR2: it is an updated version of VARCHAR where in we can store up to

4000Ch.

Example :

STUDENT TABLE

USN	SNAME	ADDRESS	PAN_NO
CHAR(4)	VARCHAR(10)	VARCHAR(10)	CHAR(10)
QSP1	DINGA	BANGALORE	ABC123XYZ1
QSP2	DINGI	MYSORE	ABC123XYZ2

3. NUMBER: It is used to store numeric values .

SYNTAX: NUMBER (Precision, [Scale])

[ ] - Not Mandatory.

Precision: it is used to determine the number of digits used to store an integer value.

Scale: it is used to determine the number of digits used to store a decimal (floating) value within the precision.

Scale is not mandatory, and the default value of scale is zero (0).

EID	PHONE_NO	SALARY
Number(3)	Number (10)	Number (7 , 2)
101	9876543210	5000.85

4. DATE

It is used to store dates in a particular format.

It used *the Oracle-specified format*.

'DD-MON-YY' or 'DD-MON-YYYY'

SYNTAX: DATE

5. LARGE OBJECTS

1. Character large object (CLOB) :

   It is used to store characters up to 4 GB in size.

2. Binary large object (BLOB) :

   It is used to store binary values of
   images, mp3, mp4 Documents
   etc Up to 4GB in size.

CONSTRAINTS

Constraints are rules applied to columns or tables to maintain data quality.

Types of Constraints :

1. UNIQUE (no duplicates)
2. NOT NULL (mandatory values)
3. CHECK (custom conditions like age>18)
4. PRIMARY KEY (unique identifier)
5. FOREIGN KEY (enforces relationships)

1. UNIQUE: "*It is used to avoid duplicate values in the column* ".

2. NOT NULL: *"It is used to avoid Null "*.

3. CHECK: "*It is an extra validation with a condition if the condition is satisfied, then the value is accepted; else it is rejected* ".

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

Characteristics of Primary Key:

We can have only 1 PK in a table
PK cannot accept duplicate/repeated values.
PK cannot accept Null
PK is always a combination of Unique and Not Null Constraint.

5. FOREIGN KEY: "*It is used to establish a connection*
between the tables."

Characteristics of Foreign Key :

We can have only Multiple FK in a table
FK can accept duplicate / repeated values.
FK can accept Null
FK is not a combination of Unique and Not Null Constraint.
For an Attribute (column) to become an FK, it is mandatory

That it must be a PK in its own table.

Differentiate between Primary key and Foreign key.

PRIMARY KEY	**FOREIGN KEY**
It is used to uniquely identify a record from the table.	It is used to establish a connection between the tables
It cannot accept Null	It can accept Null
It cannot accept duplicate values	It can accept duplicate values
It is always a combination of Not Null and Unique constraint	It is not a combination of Not Null and Unique constraint
We can have only 1 PK in a table	We can have Multiple FK in a table

NOTE: NULL

Null is a *keyword* that is used to represent Nothing / Empty Cell.

Characteristics of Null :

Null doesn’t represent 0 or Space.
Any operations performed on a Null will result in Null itself

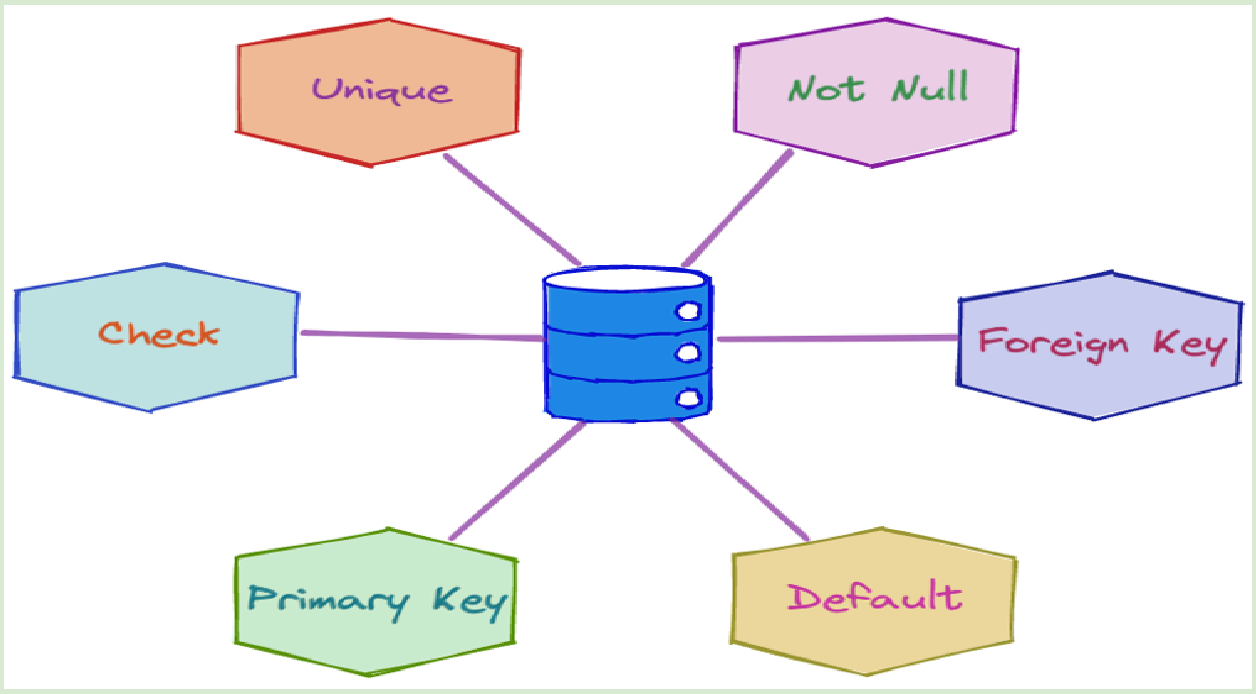
Null doesn’t occupy any Memory.
We cannot Equate Null.

Example:

Create table with constraints

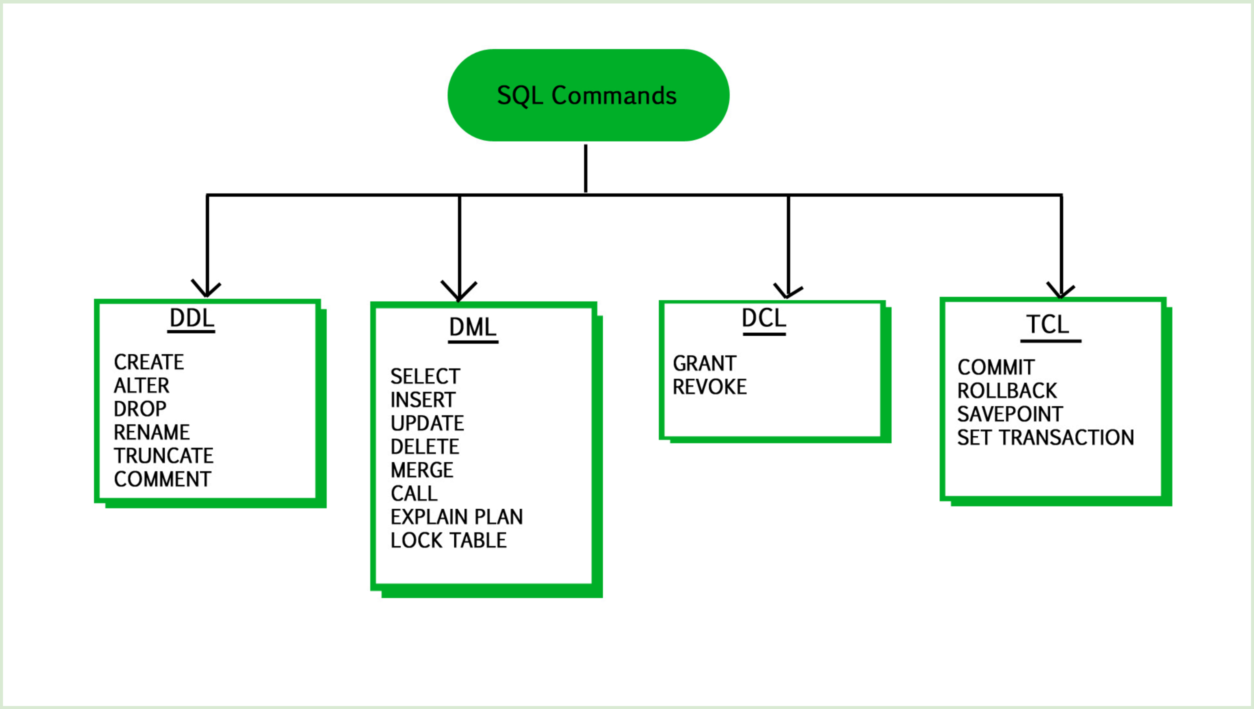
```
CREATE TABLE student(
   id INTEGER PRIMARY KEY,
email TEXT UNIQUE NOT NULL,
INTEGER CHECK (score >= 0) );
```

```
INSERT INTO raja(email, score) VALUES ('raju@gmail.com', 98);

INSERT INTO raja(email, score) VALUES ('ramesh@gmail.com', 100);

SELECT * FROM student;
```



SQL *STATEMENTS*

1. DATA DEFINITION LANGUAGE (DDL)
2. DATA MANIPULATION LANGUAGE (DML)
3. TRANSCATION CONTROL LANGUAGE (TCL)
4. DATA CONTROL LANGUAGE (DCL)
5. DATA QUERY LANGUAGE (DQL)



DATA QUERY LANGUAGE (DQL) :

" *DQL is used to retrieve the data from the*

database ".

It had 4 statements :

1. SELECT
2. PROJECTION
3. SELECTION
4. JOIN

1. SELECT: "It is used to retrieve the *data* from the table and display it.

2. PROJECTION: "It is a process of retrieving the data by *selecting only the columns* is known as Projection ".

In projection, all the records/values present in a particular column are selected by default.

3. SELECTION: "It is a process of retrieving the data by *selecting both the columns and rows* is known as Selection ".

4. JOIN: "It is a process of retrieving the data from *Multiple tables*

simultaneously, which is known as Join "

PROJECTION

"It is a process of retrieving the data by *selecting only the columns, which* is known as Projection".

In projection, all the records/values present in a particular column are selected by default.

SYNTAX :

SELECT * / [DISTINCT] Column_Name / Expression [ALIAS] FROM Table_Name ;

ORDER OF EXECUTION

1. FROM Clause
2. SELECT Clause

For example of student table

SELECT SNAME FROM STUDENT ;

NOTE :

FROM Clause starts the execution.
For FROM Clause, we can pass Table_Name as an argument.
The job of FROM Clause is to go to the Database and search for the table and put the table under execution .
SELECT Clause will execute after the execution of FROM Clause
For SELECT Clause we pass 3 arguments

◇ *

⋄ Column_Name

⋄ Expression

The job of SELECT Clause is to go the table under execution and select the columns mentioned .
SELECT Clause is responsible for preparing the result table .
Asterisk(*): It means to select all the columns from the table.
Semicolon: it means the end of the query.

DISTINCT Clause

" It is used to remove the duplicate or repeated values from the Result table."

➢ A distinct clause has to be used

As the first argument to select clause.

➢ We can use multiple columns

As an argument to distinct clause, it will remove the combination of columns in which the records are duplicated .

EXPRESSION

"A statement which gives result is known as Expression ".

Expression is a combination Operand and Operator .

Operand: These are the values that we pass .

Operator: These are the Symbols which perform some Operation on the operand.

Example: 3*5

ALIAS

"It is an alternate name given to a Column or an Expression in the result table ".

SELECTION

"It is a process of retrieving the data by selecting both the columns and rows is known as Selection " .

SYNTAX :

SELECT * / [DISTINCT] Column_Name / Expression [ALIAS]

FROM Table_Name

WHERE <Filter_Condition> ;

ORDER OF EXECUTION

FROM

WHERE

SELECT

WHERE Clause

"Where clause is used to filter the records".

OPERATORS IN SQL

1. ARITHEMATIC OPERATORS :-(+ , -, * , /)

2. CONCATENATION OPERATOR :-(||)

3. COMPARISION OPERATORS :-(= , != or <>)

4. RELATIONAL OPERATOR :-(> , < , >= , <=)

5. LOGICAL OP : (AND, OR, NOT)

6. SPECIAL OPERATOR:-

1.IN

2. NOT IN

3. BETWEEN

4 .NOT BETWEEN

5 .IS

6 .IS NOT

7. LIKE

8. NOT LIKE

IN : It is a multi-valued operator which can accept multiple values At the RHS.

NOT IN : It is a multi-valued operator which can accept multiple values At the RHS . It is similar to IN op instead of selecting it Rejects the values

BETWEEN : "It is used whenever we have range of values "

Syntax:

Column_Name BETWEEN Lower_RangeAND Higher_Range;

Between Op works including the range .

NOT BETWEEN : It is Opposite of Between .

IS : "It is used to compare only NULL "

Syntax: Column_Name ISNULL ;

IS NOT : "It is used to compare the values with NOT NULL ".

Syntax: Column_Name IS NOTNULL ;

LIKE :"It is used for Pattern Matching ".

To achieve pattern matching we use special characters .

Percentile (%)

Underscore (_)

Syntax: Column_Name LIKE 'pattern' ;

NOT LIKE :Opposite of Like .

7.SUBQUERY OPERATORS:-

1. ALL

2 .ANY

3 .EXISTS

4 .NOT EXISTS

CONCATENATION Operator :

" It is used to join the strings ".

Symbol: ||

LOGICAL OPERATORS

AND

OR

NOT

We use logical operators to write multiple conditions.

FUNCTIONS

A block of code or a list of instructions that are used to perform a specific task .

1.Function_Name

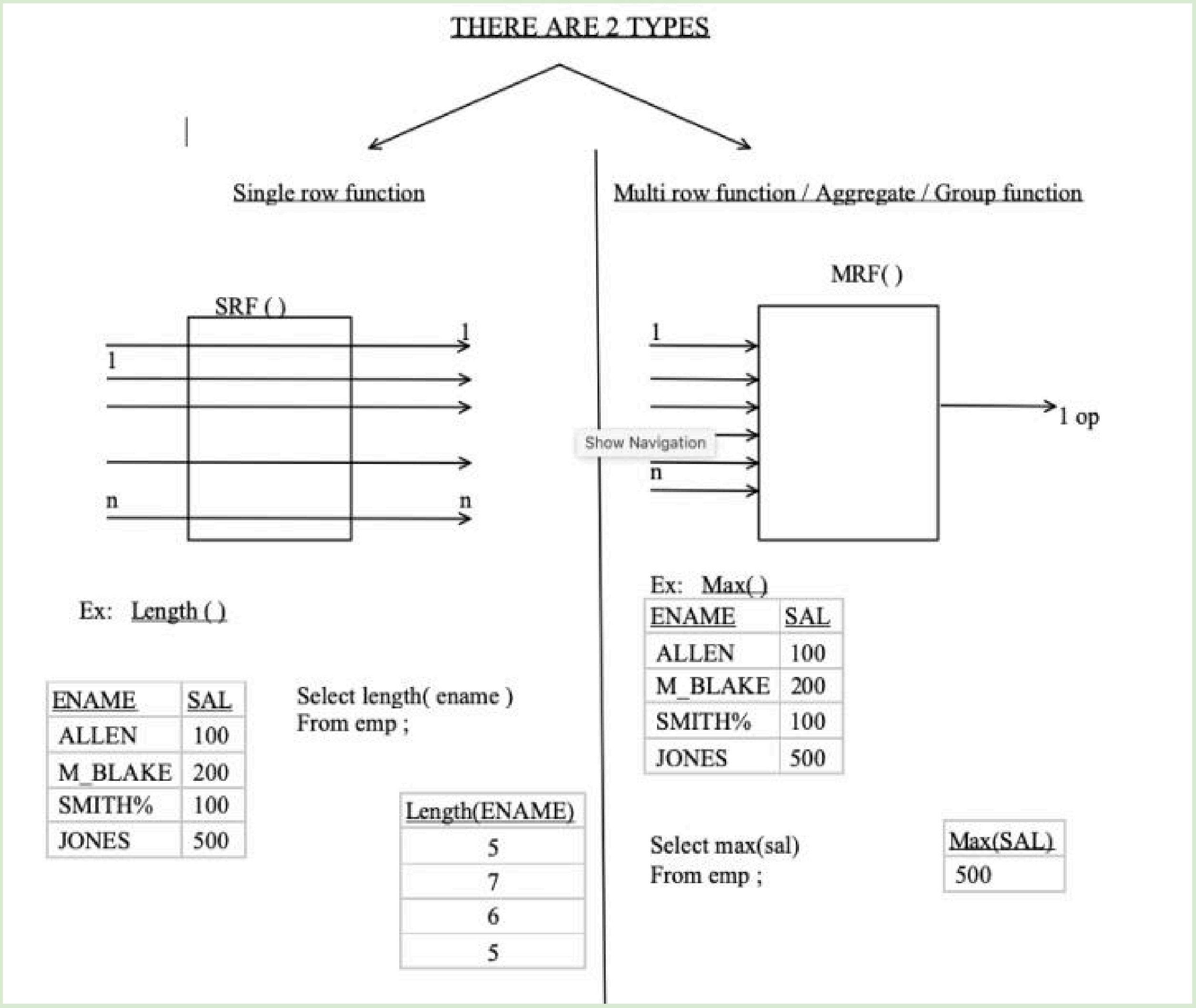
2.Number_of_arguments (no of inputs)

3. Return type

There are 3 main components of a function

Types of Functions in SQL :

1. SINGLE ROW FUNCTIONS
2. MULTI-ROW FUNCTIONS/ AGGREGATE / GROUP FUNCTIONS.



ENAME	SAL
ALLEN	100
M_BLAKE	200
SMITH%	100
JONES	500

```
Select length( ename )
From emp ;
```

Length(ENAME)
5
7
6
5

ENAME	SAL
ALLEN	100
M_BLAKE	200
SMITH%	100
JONES	500

```
Select max(sal)
From emp ;
```

Max(SAL)
500

Multi Row Functions:

It takes all the inputs at one shot and then executes and provides a single output.

If we pass 'n' number of inputs to a MRF( ) it returns '1' Output .

List of MRF ( )

MAX( ): it is used to obtain the maximum value present in the column

MIN ( ): it is used to obtain the minimum value present in the column

SUM ( ): it is used to obtain the summation of values present in the column

AVG( ): it is used to obtain the average of values present in the column

COUNT( ): it is used to obtain the number of values present in the column

NOTE :

Multi row functions can accept only one argument , i.e a Column_Name or an Expression

MRF (Column_Name / Exp)

Along with a MRF( ) we are not supposed to use any other Column_Name in the select clause .

MRF( ) ignore the Null .

We cannot use a MRF( ) in where clause .

COUNT( ) is the only MRF which can accept * as an Argument .

GROUP & FILTERING

GROUPING: GROUP BY Clause

Group by clause is used to group the records.

SYNTAX:

```
SELECT group_by_expression / group_function

FROM table_name

[WHERE <filter_condition>]

GROUP BY column_name/expression ;
```

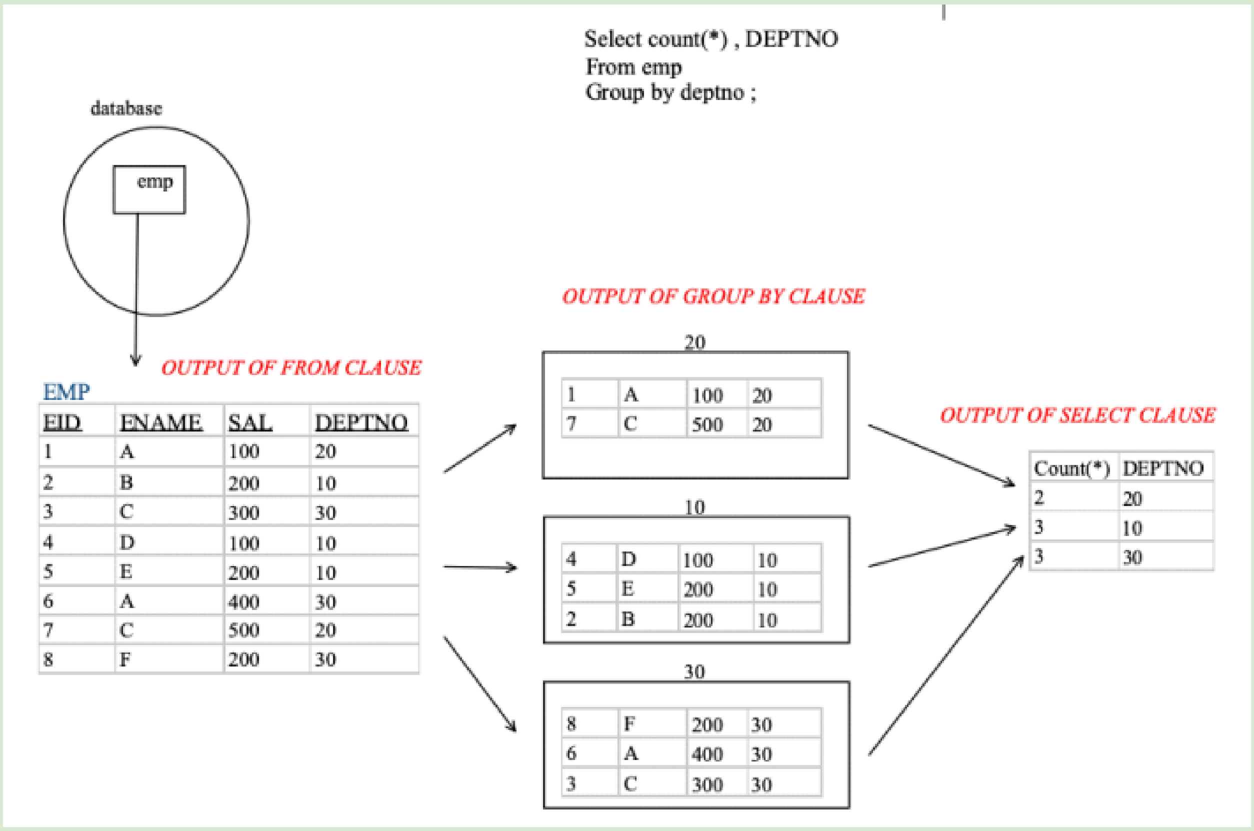
ORDER OF EXECUTION:

1-FROM

2-WHERE(if used) [ROW-BY-ROW]

3-GROUP BY [ROW-BY-ROW]

4-SELECT [GROUP-BY-GROUP]



```
Select count(*) , DEPTNO
From emp
Group by deptno ;
```

EID	ENAME	SAL	DEPTNO
1	A	100	20
2	B	200	10
3	C	300	30
4	D	100	10
5	E	200	10
6	A	400	30
7	C	500	20
8	F	200	30

1	A	100	20
7	C	500	20

4	D	100	10
5	E	200	10
2	B	200	10

8	F	200	30
6	A	400	30
3	C	300	30

Count(*)	DEPTNO
2	20
3	10
3	30

Note:

Group By clause is used to group the records.
Group By clause executes row by row.
After the execution of Group By clause, we get Groups.
Therefore any clause that executes after group by must execute Group By Group .
The Column_Name or expression used for grouping can be used in select clause .
Group By clause can be used without using Where clause .

FILTERING: HAVING Clause

" Having Clause is used to Filter the Group "

SYNTAX:

SELECT group_by_expression / group_function

FROM table_name

[WHERE <filter_condition>]

GROUP BY column_name/expression

HAVING <group_filter_condition>

ORDER OF EXECUTION:

```
1-FROM

2-WHERE(if used)        [ROW-BY-ROW]

3-GROUP BY(if used)  [GROUP-BY-GROUP]

4-HAVING (if used)      [GROUP-BY-GROUP]

5-SELECT                    [GROUP-BY-GROUP]
```

example:

➢ WAQTD to find number of employees working in each

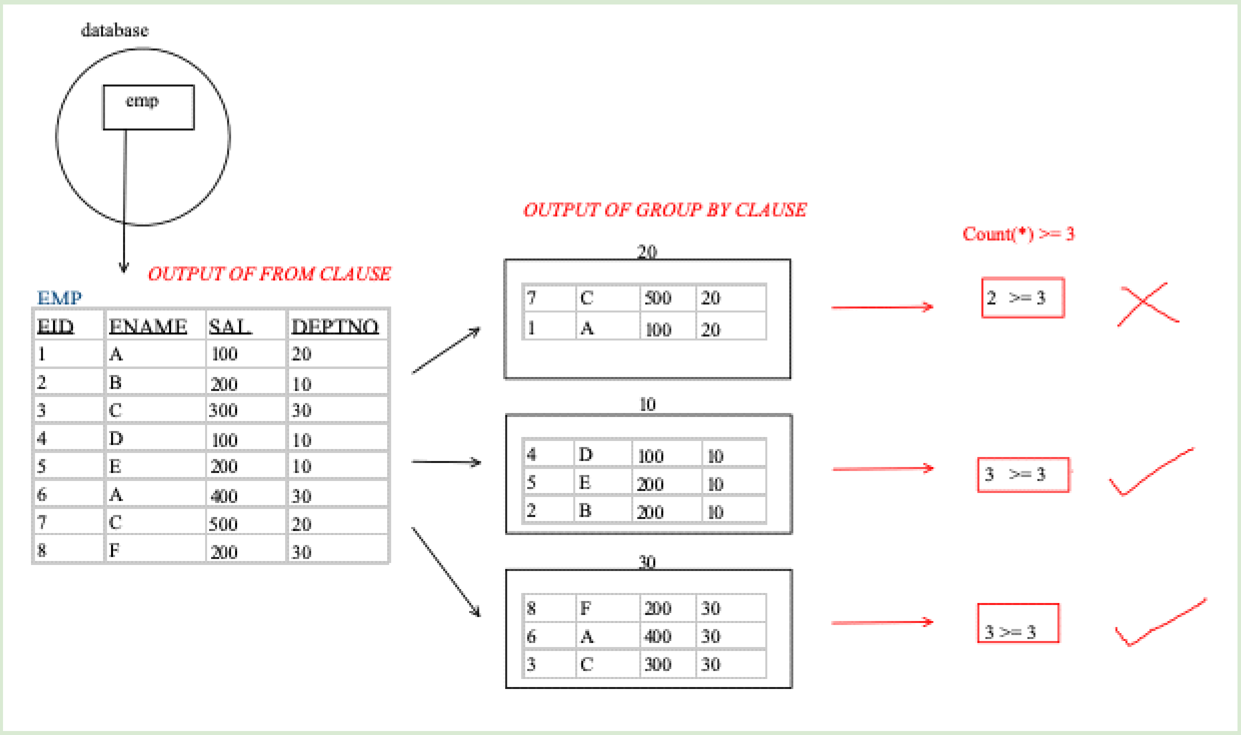
Dept if there areat least 3 employeesin each dept .

SELECT DEPTNO , COUNT(*)

FROM EMP

GROUP BY DEPTNO

HAVING COUNT(*)>=3;

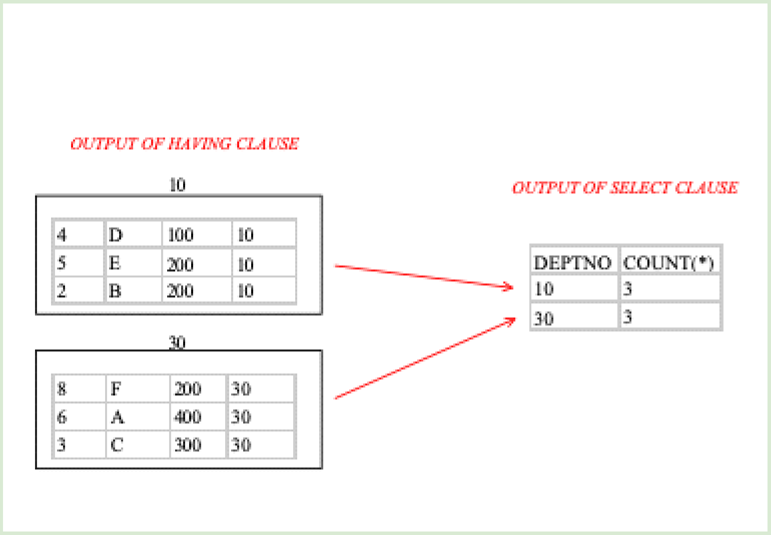


EID	ENAME	SAL	DEPTNO
1	A	100	20
2	B	200	10
3	C	300	30
4	D	100	10
5	E	200	10
6	A	400	30
7	C	500	20
8	F	200	30

7	C	500	20
1	A	100	20

4	D	100	10
5	E	200	10
2	B	200	10

8	F	200	30
6	A	400	30
3	C	300	30



Differentiate between Where and Having.

WHERE	**HAVING**
1) WHERE cannot be used with aggregate functions.	1) HAVING is just like WHERE but used with aggregate functions.
2) WHERE can be used without GROUP BY clause	2) HAVING always works with GROUPBY
3) WHERE can be used with SELECT ,UPDATE, DELETE statements.	3) HAVING can only be used with only SELECT statement.
4) Works with row-operation.	4) Works with column-operation.
5) Can work with or without GROUPBY.	5) Works only with GROUPBY.

SQL's data types provide structure, statements enable operations, constraints ensure validity, operators add logic, functions transform data, and group/filtering (GROUP BY, WHERE, HAVING) summarize with precision