ASSIGNMENT NO :- 02

Name :- Jayant Shankar Pawar.

Class :- MCA – I

Division :- B   Batch :- B3

Roll No :- 135

Subject :- RDBMS

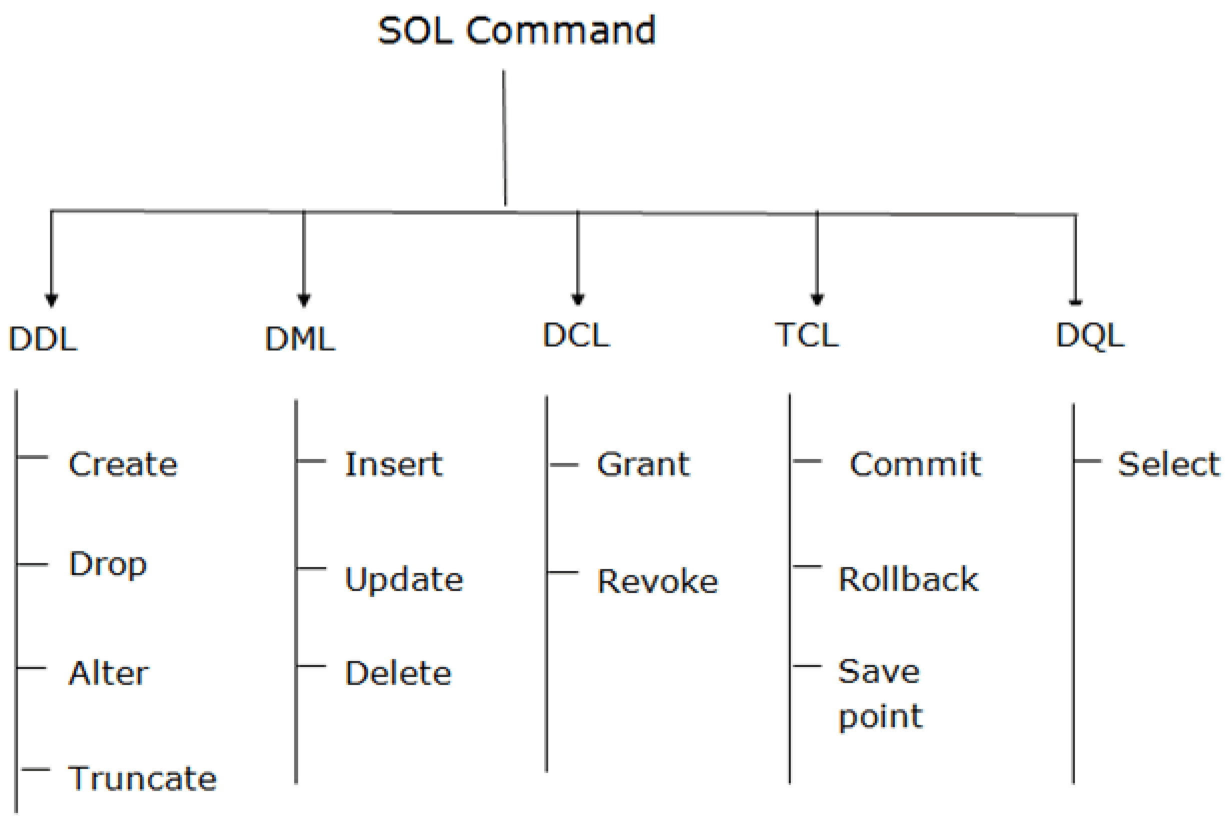
SQL Commands

Categories of SQL Commands

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

Types of SQL Commands

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



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

Example:

```
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB D
ATE);
```

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE EMPLOYEE;
```

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

ALTER TABLE table_name MODIFY(column_definitions....);

EXAMPLE

```
ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE EMPLOYEE;

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

INSERT INTO TABLE_NAME

```
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
```

Or

```
INSERT INTO TABLE_NAME    VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = val
ueN] [WHERE CONDITION]
```

For example:

```
UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM javatpoint
WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

b. Revoke: It is used to take back permissions from the user.

Example

```
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
COMMIT;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
SELECT expressions
FROM TABLES
WHERE conditions;
```

For example:

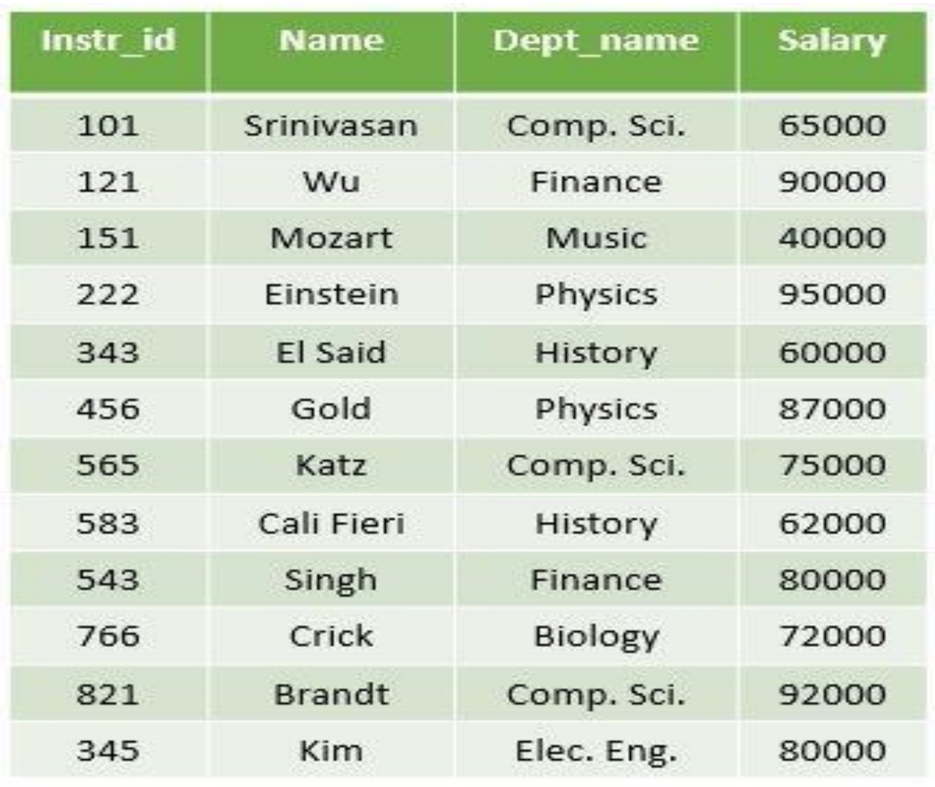
```
SELECT emp_name
```

```
FROM employee
WHERE age > 20;
```

Basic Structure of SQL Queries

- In the select clause, you have to specify the attributes that you want to see in the result relation
- In the from clause, you have to specify the list of relations that has to be accessed for evaluating the query.
- In the where clause involves a predicate that includes attributes of the relations that we have listed in the from clause.

Queries on Single Relation



Instr_id	Name	Dept_name	Salary
101	Srinivasan	Comp. Sci.	65000
121	Wu	Finance	90000
151	Mozart	Music	40000
222	Einstein	Physics	95000
343	El Said	History	60000
456	Gold	Physics	87000
565	Katz	Comp. Sci.	75000
583	Cali Fieri	History	62000
543	Singh	Finance	80000
766	Crick	Biology	72000
821	Brandt	Comp. Sci.	92000
345	Kim	Elec. Eng.	80000

Figure 1. Instructor Relation

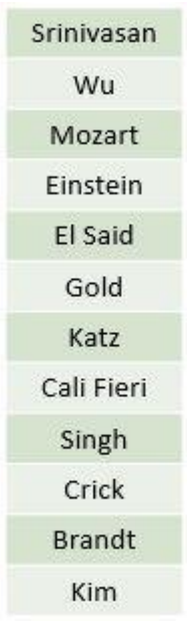
1.Select Clause & From Clause

Statement :- Lets consider statement find the names of all instructor.

select name,

from instructor;

O/P :-



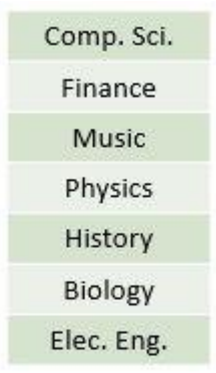
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Cali Fieri
Singh
Crick
Brandt
Kim

select distinct dept_name

from instructor;

To eliminate the duplicates you can make use of the **distinct** keyword.

O/P :-



Comp. Sci.
Finance
Music
Physics
History
Biology
Elec. Eng.

include the arithmetic expression in the select clause using operators such as +, -, *, and /. In case, you want the result relation to display instructor name along with their salary which reduced by 10%. Then the SQL query you will impose on the data set is:

select instr_name, salary*0.9

from instructor;

2. **Where Clause**

Where clause allows us to select only those rows in the result relation of from clause that satisfy a specified condition.

Statement :- Find the names of all instructor in the physics who have salary greater than 90000.

select name **from** instructor

where dept_name = ‘physics’ and salary > 90000;

O/P :- Name

Einstein

SQL allows logical connectives & ,OR,NOT, in where clause also it allows comparision operators <,>,>=,<=.

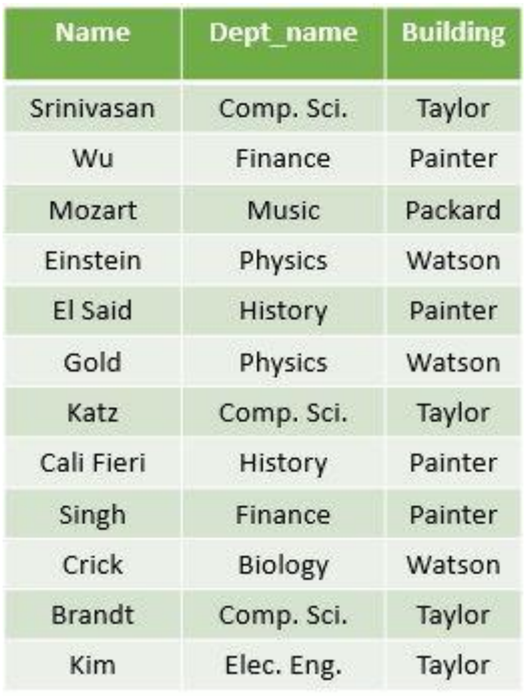
Queries on Multiple Relation

select name, instructor.dept_name, building

from instructor, department

where instructor.dept_name= department.dept_name;

O/P :-



Name	Dept_name	Building
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Cali Fieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function

These functions are used to do operations from the values of the column and a single value is returned.

1. AVG()
2. COUNT()
3. FIRST()

4. LAST()
5. MAX()
6. MIN()
7. SUM()

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

1. COUNT() FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

```
COUNT(*)
or
COUNT( [ALL|DISTINCT] expression )
```

Example: COUNT()

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
```

Output:

```
10
```

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output:

```
3
```

Example: COUNT() with GROUP BY

SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;

Output:

```
Com1     5
Com2     3
Com3     2
```

Example: COUNT() with HAVING

SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;

Output:

```
Com1     5
Com2     3
```

2. SUM() Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM()
or
SUM([ALL|DISTINCT] expression)

Example: SUM()

SELECT SUM(COST)
FROM PRODUCT_MAST;

Output:

```
670
```

Example: SUM() with WHERE

SELECT SUM(COST)

FROM PRODUCT_MAST

WHERE QTY>3;

Output:

```
320
```

Example: SUM() with GROUP BY

SELECT SUM(COST)

FROM PRODUCT_MAST

WHERE QTY>3

GROUP BY COMPANY;

Output:

```
Com1    150
Com2    170
```

Example: SUM() with HAVING

SELECT COMPANY, SUM(COST)

FROM PRODUCT_MAST

GROUP BY COMPANY

HAVING SUM(COST)>=170;

Output:

```
Com1    335
Com3    170
```

3. AVG() function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

or

AVG([ALL|DISTINCT] expression)

Example:

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

Output:

```
67.00
```

4. MAX() Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

```
MAX()
or
MAX( [ALL|DISTINCT] expression )
```

Example:

```
SELECT MAX(RATE)
FROM PRODUCT_MAST;
```

```
30
```

5. MIN() Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

```
MIN()
or
MIN( [ALL|DISTINCT] expression )
```

Example:

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```

Output:

```
10
```

6. LAST() Function

The LAST() function returns the last value of the selected column. It can be used only in MS ACCESS.

Syntax:

```
SELECT LAST(column_name) FROM table_name;
```

Example:

```
SELECT LAST(Cost) FROM PRODUCT_MAST;
```

Output:

```
120
```

7. FIRST() Function

The FIRST() function returns the first value of the selected column.

Syntax:

```
SELECT LAST(column_name) FROM table_name;
```

Example:

```
SELECT LAST(Cost) FROM PRODUCT_MAST;
```

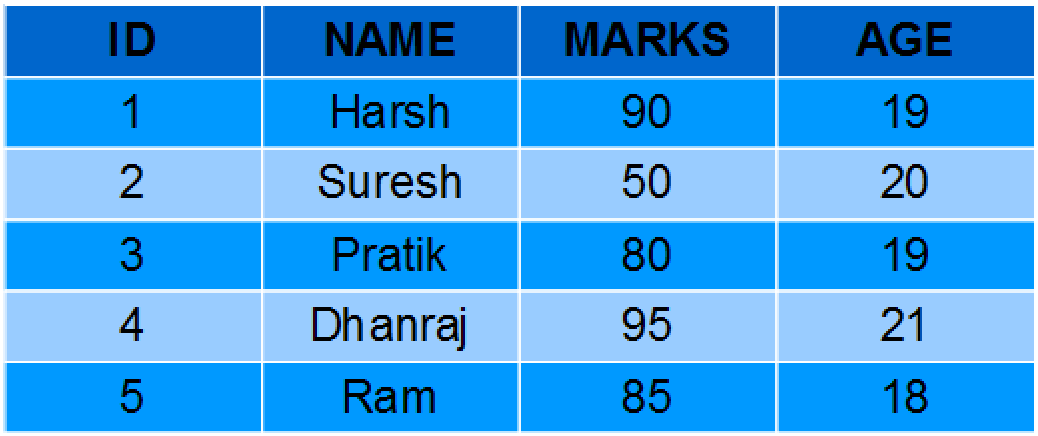
Output:

```
20
```

SQL Scalar Functions

1. These functions are based on user input, these too returns single value.
    1. UCASE()
    2. LCASE()
    3. MID()
    4. LEN()
    5. ROUND()
    6. NOW()
    7. FORMAT()

Students-Table



ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

UCASE(): It converts the value of a field to uppercase.

Syntax:

```
SELECT UCASE(column_name) FROM table_name;
```

Example:

```
SELECT UCASE(NAME) FROM Students;
```

Output:

```
NAME

HARSH

SURESH

PRATIK

DHANRAJ

RAM
```

LCASE(): It converts the value of a field to lowercase.

Syntax:

```
SELECT LCASE(column_name) FROM table_name;
```

Example:

```
SELECT LCASE(NAME) FROM Students;
```

Output:

NAME
Harsh
Suresh
Pratik
Dhanraj
Ram

MID(): The MID() function extracts texts from the text field.

Syntax:

```
SELECT MID(column_name,start,length) AS some_name FROM table_name;


specifying length is optional here, and start signifies start
position ( starting from 1 )
```

Example:

```
SELECT MID(NAME,1,4) FROM Students;
```

Output:

NAME
HARS
SURE
PRAT
DHAN

RAM

LEN(): The LEN() function returns the length of the value in a text field.

Syntax:

```
SELECT LENGTH(column_name) FROM table_name;
```

Example:

```
SELECT LENGTH(NAME) FROM Students;
```

Output:

NAME
5
6
6
7
3

ROUND(): The ROUND() function is used to round a numeric field to the number of decimals specified.NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.

Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;


decimals- number of decimals to be fetched.
```

Example:

```
SELECT ROUND(MARKS,0) FROM table_name;
```

Output:

MARKS

90
50
80
95
85

NOW(): The NOW() function returns the current system date and time.

Syntax:

```
SELECT NOW() FROM table_name;
```

Example:

```
SELECT NAME, NOW() AS DateTime FROM Students;
```

Output:

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM
SURESH	1/13/2017 1:30:11 PM
PRATIK	1/13/2017 1:30:11 PM
DHANRAJ	1/13/2017 1:30:11 PM
RAM	1/13/2017 1:30:11 PM

FORMAT(): The FORMAT() function is used to format how a field is to be displayed.

Syntax:

```
SELECT FORMAT(column_name,format) FROM table_name;
```

Example:

```
SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;
```

Output:

NAME	**Date**
HARSH	2017-01-13
SURESH	2017-01-13
PRATIK	2017-01-13
DHANRAJ	2017-01-13
RAM	2017-01-13

SQL Clauses

1.GROUP BY

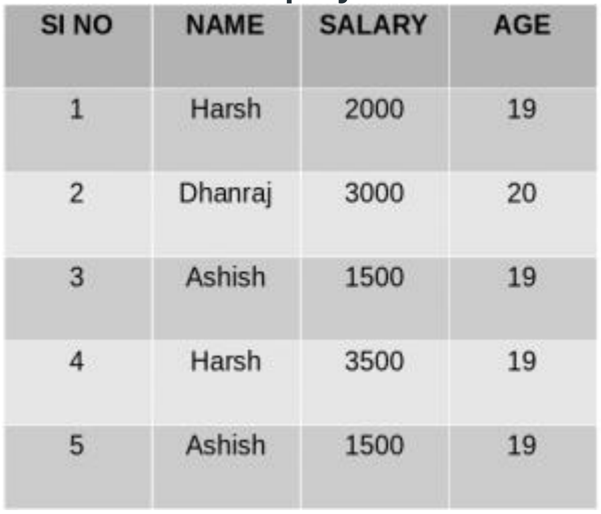
The GROUP BY Statement in SQL is used to arrange identical data into groups used in select statement.

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.
- In the query , Group BY clause is placed before Having clause .

Syntax:

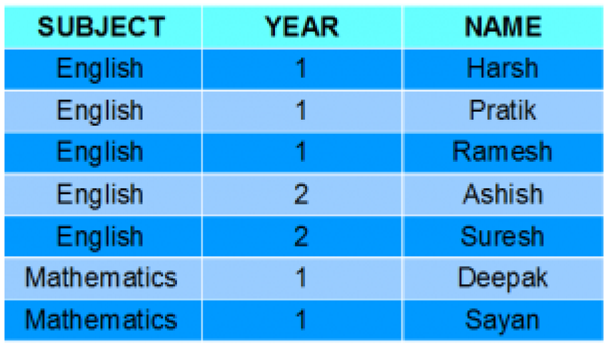
```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
ORDER BY column1, column2;
```

Employee



SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

Student



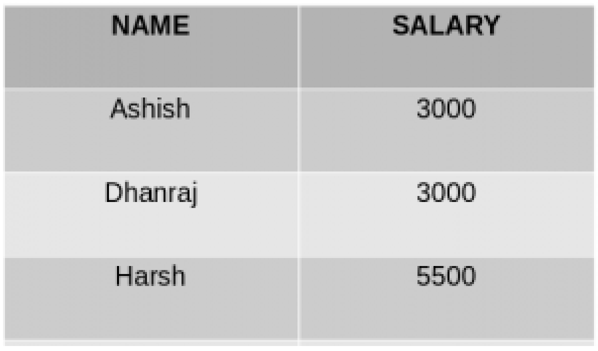
SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

Example:

Group By single column:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME;
```

Output:

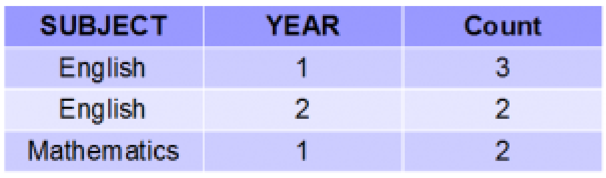


NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

Group By multiple columns:

```
SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;
```

Output:



SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

2.HAVING CLAUSE

Place condition on groups.

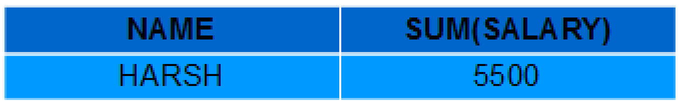
Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING condition
ORDER BY column1, column2;
```

Example:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME
HAVING SUM(SALARY)>3000;
```

Output:

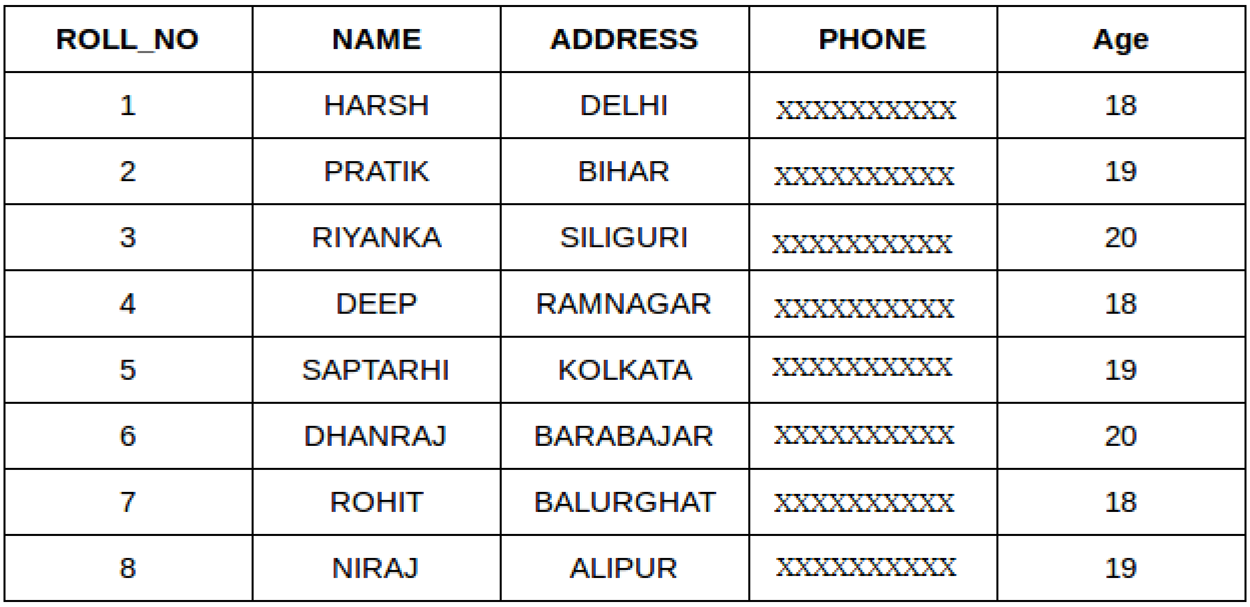


NAME	SUM(SALARY)
HARSH	5500

3.ORDER BY

sort the fetched data in either ascending or descending order.

- By default ORDER BY sorts the data in ascending order.
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.



ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Sort according to a single column:

Syntax:

```
SELECT * FROM table_name ORDER BY column_name ASC|DESC
```

Example:

```
SELECT * FROM Student ORDER BY ROLL_NO DESC;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
2	PRATIK	BIHAR	XXXXXXXXXX	19
1	HARSH	DELHI	XXXXXXXXXX	18

Sort according to multiple columns:

```
SELECT * FROM Student ORDER BY Age ASC , ROLL_NO DESC;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
1	HARSH	DELHI	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
2	PRATIK	BIHAR	XXXXXXXXXX	19

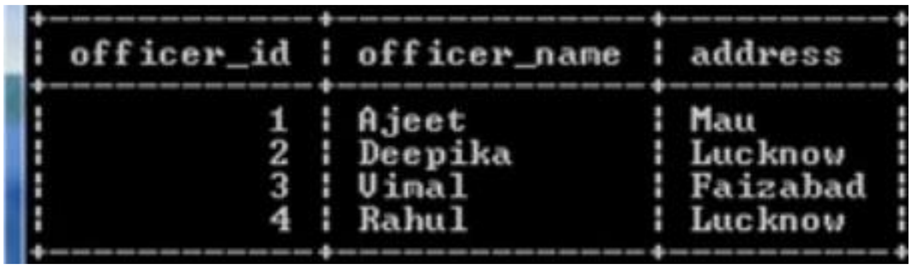
ROLL_NO	NAME	ADDRESS	PHONE	Age
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
3	RIYANKA	SILIGURI	XXXXXXXXXX	20

MySQL JOINS

JOINS are used with SELECT statement. It is used to retrieve data from multiple tables.

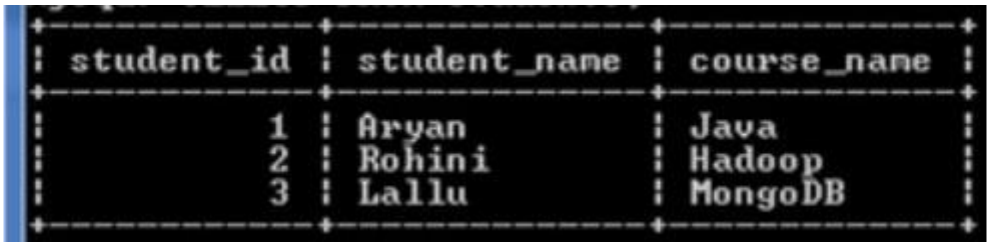
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

OFFICER TABLE



```
+------------+--------------+----------+
| officer_id | officer_name | address  |
+------------+--------------+----------+
|          1 | Ajeet        | Mau      |
|          2 | Deepika      | Lucknow  |
|          3 | Vimal        | Faizabad |
|          4 | Rahul        | Lucknow  |
+------------+--------------+----------+
```

STUDENT TABLE



```
+------------+--------------+-------------+
| student_id | student_name | course_name |
+------------+--------------+-------------+
|          1 | Aryan        | Java        |
|          2 | Rohini       | Hadoop      |
|          3 | Lallu        | MongoDB     |
+------------+--------------+-------------+
```

1.Inner JOIN

Inner Join is used to return all rows from multiple tables where the join condition is satisfied.

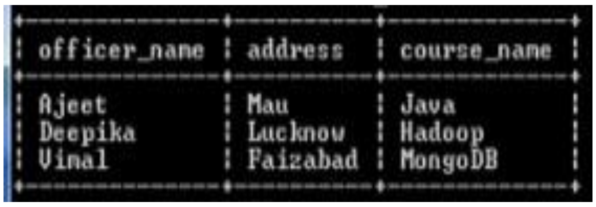
Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Example :-

SELECT officers.officer_name, officers.address, students.course_name
FROM officers
INNER JOIN students
ON officers.officer_id = students.student_id;

Output:



```
+--------------+----------+-------------+
| officer_name | address  | course_name |
+--------------+----------+-------------+
| Ajeet        | Mau      | Java        |
| Deepika      | Lucknow  | Hadoop      |
| Vimal        | Faizabad | MongoDB     |
+--------------+----------+-------------+
```

2.LEFT OUTER JOIN

It return all rows from left hand table where condition satisfy.

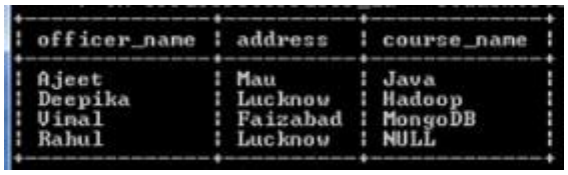
Syntax:

SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.**column** = table2.**column**;

Example :-

SELECT officers.officer_name, officers.address, students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;

Output:



```
+--------------+----------+-------------+
| officer_name | address  | course_name |
+--------------+----------+-------------+
| Ajeet        | Mau      | Java        |
| Deepika      | Lucknow  | Hadoop      |
| Vimal        | Faizabad | MongoDB     |
| Rahul        | Lucknow  | NULL        |
+--------------+----------+-------------+
```

3.RIGHT OUTER JOIN

It returns all rows right hand table where condition satisfy.

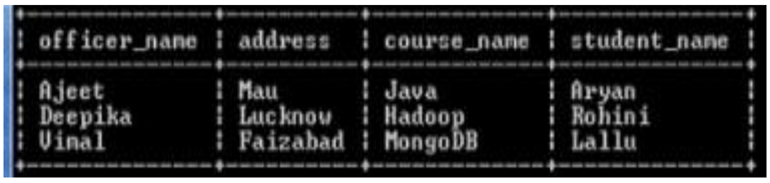
Syntax:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Example :-

```
SELECT officers.officer_name, officers.address, students.course_name, students.stude
nt_name
FROM officers
RIGHT JOIN students
ON officers.officer_id = students.student_id;
```

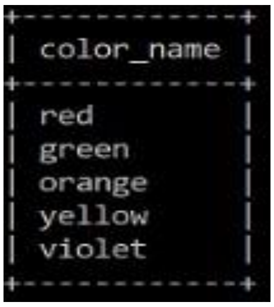
Output:



```
+--------------+----------+-------------+--------------+
| officer_name | address  | course_name | student_name |
+--------------+----------+-------------+--------------+
| Ajeet        | Mau      | Java        | Aryan        |
| Deepika      | Lucknow  | Hadoop      | Rohini       |
| Vimal        | Faizabad | MongoDB     | Lallu        |
+--------------+----------+-------------+--------------+
```

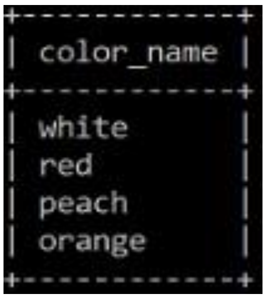
BASIC OPERATIONS LIKE UNION, MINUS, INTERSECTION

Colors_a



```
+------------+
| color_name |
+------------+
| red        |
| green      |
| orange     |
| yellow     |
| violet     |
+------------+
```

Colors_b



```
+------------+
| color_name |
+------------+
| white      |
| red        |
| peach      |
| orange     |
+------------+
```

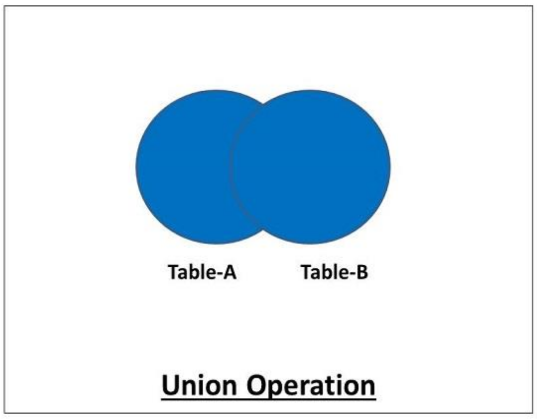
1.UNION

The Union is a binary operator, It is used to combine the result set of two select queries.

Conditions :-

1. Both SELECT statements should have an equal number of fields in the same order.
2. The data types of these fields should either be the same or compatible with each other.

The Union operation can be demonstrated as follows:



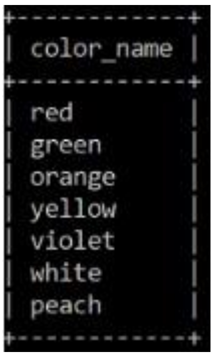
Syntax :-

```
SELECT (coloumn_names) from table1
[WHERE condition]
UNION SELECT (coloumn_names) from table2
[WHERE condition];
```

Example 1:-

```
SELECT color_name FROM colors_a
UNION SELECT color_name FROM colors_b;
```

OUTPUT :-

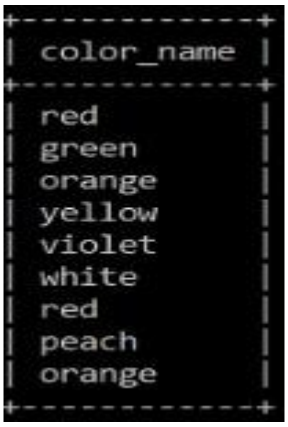


```
+------------+
| color_name |
+------------+
| red        |
| green      |
| orange     |
| yellow     |
| violet     |
| white      |
| peach      |
+------------+
```

Example 2:-

```
SELECT color_name FROM colors_a
UNION ALL SELECT color_name FROM colors_b;
```

OUTPUT :-



```
+------------+
| color_name |
+------------+
| red        |
| green      |
| orange     |
| yellow     |
| violet     |
| white      |
| red        |
| peach      |
| orange     |
+------------+
```

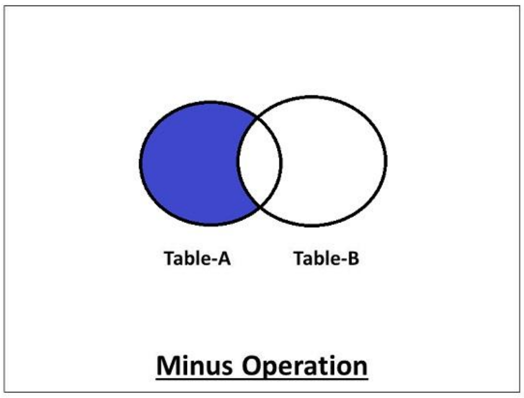
2.MINUS

Minus is a binary operator , The minus operation between two selections returns the rows that are present in the first selection but not in the second selection.

Conditions :-

1. Both SELECT statements should have an equal number of fields in the same order.
2. The data types of these fields should either be the same or compatible with each other.

The minus operation can be demonstrated as follows:



Note :-

Minus not present in mysql, use either ‘Not in’ or ‘join’ for performing a minus operation.

Syntax :-

```
SELECT (coloumn_names) from table1
[WHERE condition]
MINUS SELECT (coloumn_names) from table2
[WHERE condition];
```

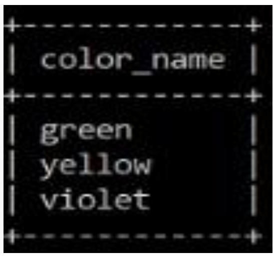
Example 1:-

```
SELECT color_name FROM colors_a
WHERE color_name NOT IN(SELECT color_name FROM
colors_b);
```

Example 2:-

```
SELECT color_name FROM colors_a
LEFT JOIN colors_b USING (color_name) WHERE
colors_b.color_name IS NULL;
```

OUTPUT :-



```
+------------+
| color_name |
+------------+
| green      |
| yellow     |
| violet     |
+------------+
```

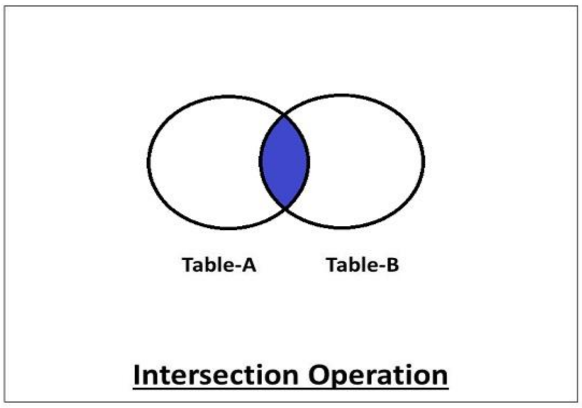
3.INTERSECTION

Intersect is a binary operator , The intersection operation between two selections returns only the common data sets or rows between them.

Conditions :-

1. Both SELECT statements should have an equal number of fields in the same order.
2. The data types of these fields should either be the same or compatible with each other.

The intersection operation can be demonstrated as follows:



NOTE :-

Intersect not present in MySQL use either 'IN' or 'Exists for performing a intersect operation in MySQL.

SYNTAX :-

```
SELECT (coloumn_names) from table1
[WHERE condition]
INTERSECT SELECT (coloumn_names) from table2
[WHERE condition];
```

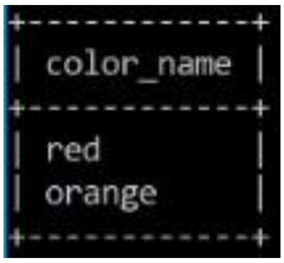
Example 1:-

```
SELECT color_name FROM colors_a WHERE color_name
IN(SELECT color_name FROM colors_b);
```

Example 2:-

```
SELECT color_name FROM colors_a WHERE color_name
Exists(SELECT color_name FROM colors_b);
```

Output :-

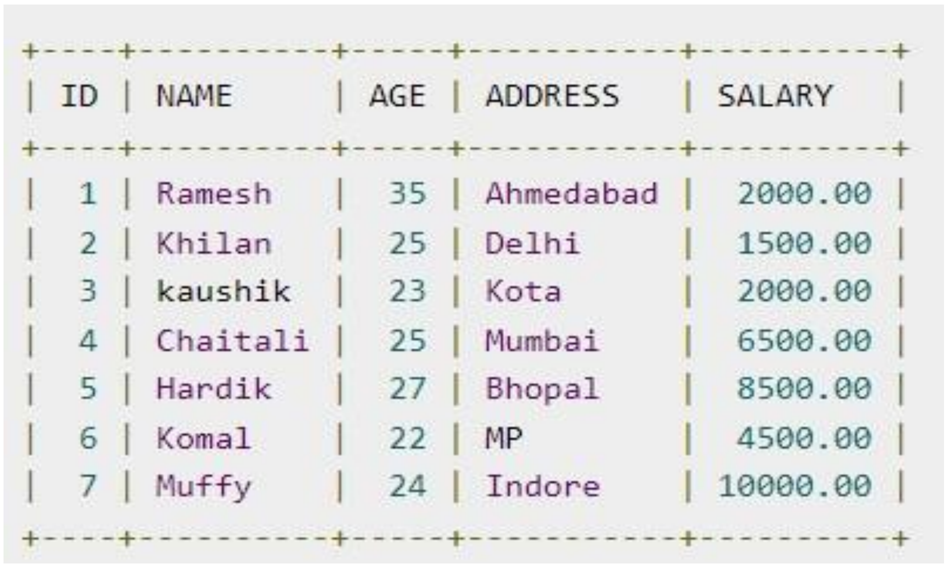


```
+------------+
| color_name |
+------------+
| red        |
| orange     |
+------------+
```

SQL - Sub Queries

- Quries nested into another subquery.
- Used in select, insert, update & delete statement.
- Used in where clause, from clause & having clause.
- Used with comparison operator & logical operator.

CUSTOMER_TABLE



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Subqueries with the SELECT Statement

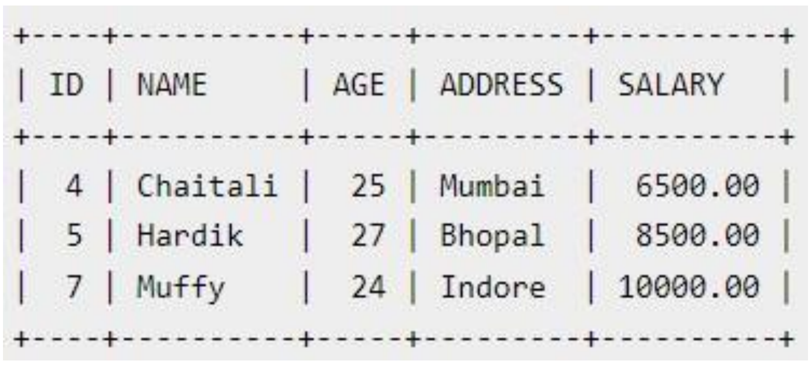
Syntax :-

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

Example :-

```
SELECT *
   FROM CUSTOMERS
   WHERE ID IN (SELECT ID
         FROM CUSTOMERS
         WHERE SALARY > 4500)
```

Output :-



```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

Subqueries with the INSERT Statement

Syntax :-

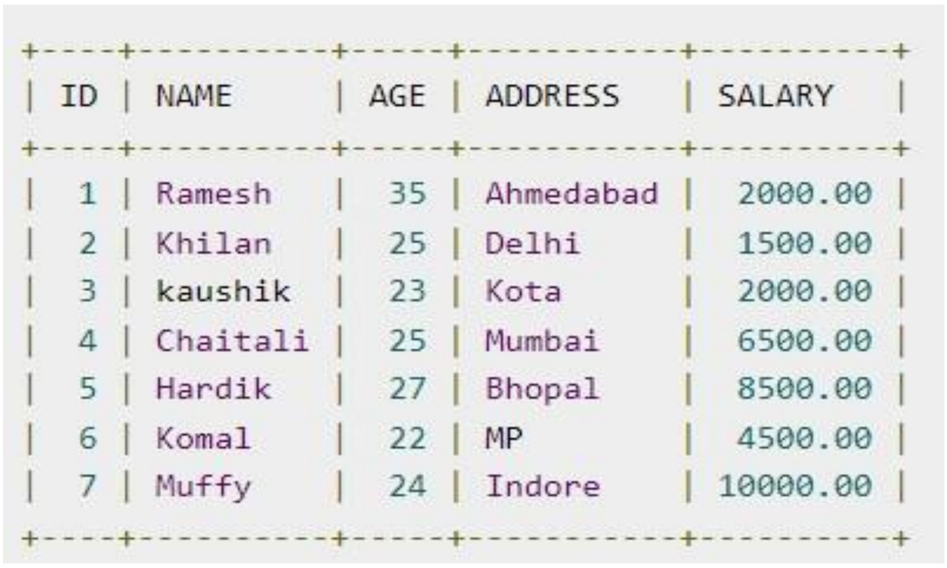
```
INSERT INTO table_name [ (column1 [, column2 ]) ]
   SELECT [ *|column1 [, column2 ]
   FROM table1 [, table2 ]
   [ WHERE VALUE OPERATOR ]
```

Example :-

```
INSERT INTO CUSTOMERS_1
   SELECT * FROM CUSTOMERS
   WHERE ID IN (SELECT ID
   FROM CUSTOMERS) ;
```

Output :-

CUSTOMER_TABLE_1



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

- Copy the customer into customer_table_1.

Subqueries with the UPDATE Statement

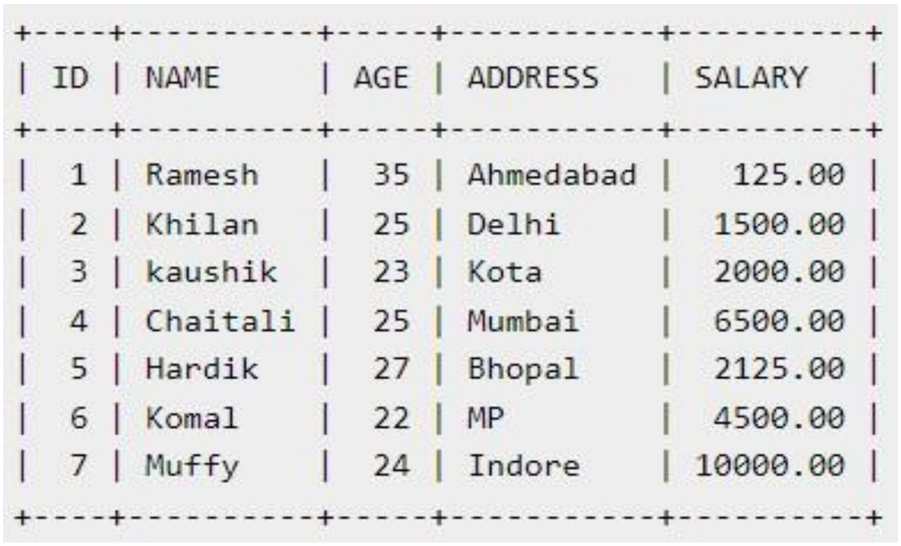
Syntax :-

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example :-

```
UPDATE CUSTOMERS
  SET SALARY = SALARY * 0.25
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 27 );
```

Output :-



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |   125.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  2125.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Subqueries with the DELETE Statement

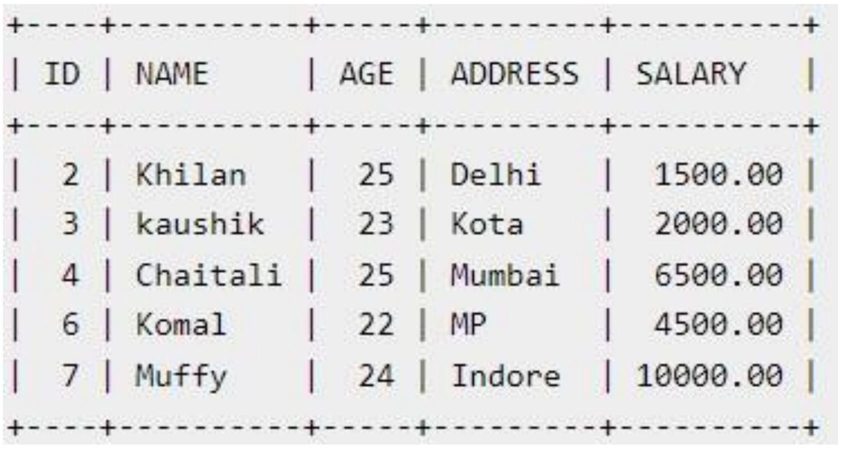
Syntax :-

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
  (SELECT COLUMN_NAME
  FROM TABLE_NAME)
  [ WHERE) ]
```

Example :-

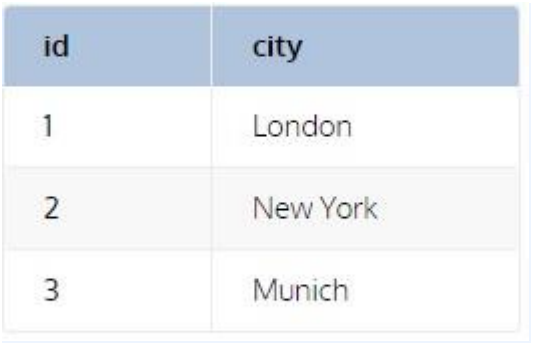
```
DELETE FROM CUSTOMERS
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 27 );
```

Output :-



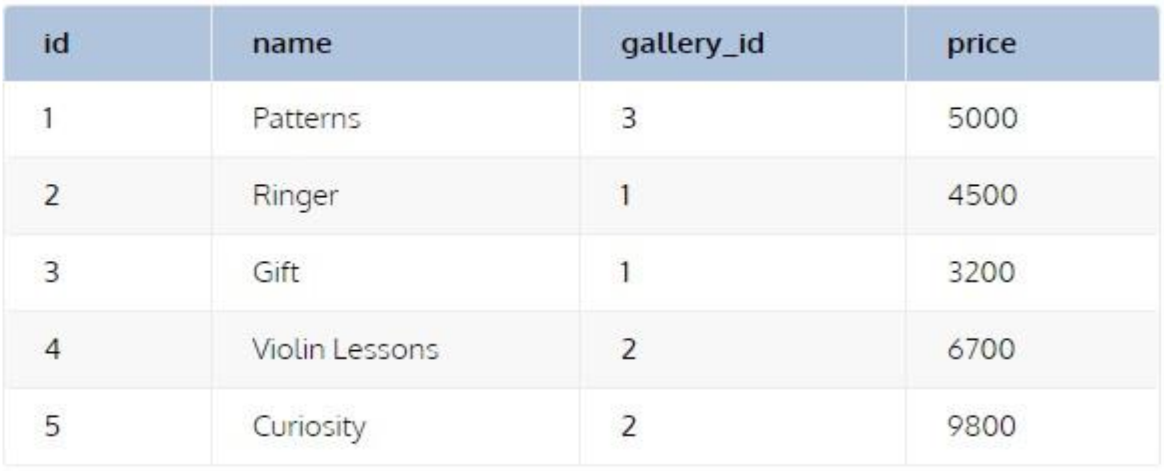
```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

GALARIES_TABLE



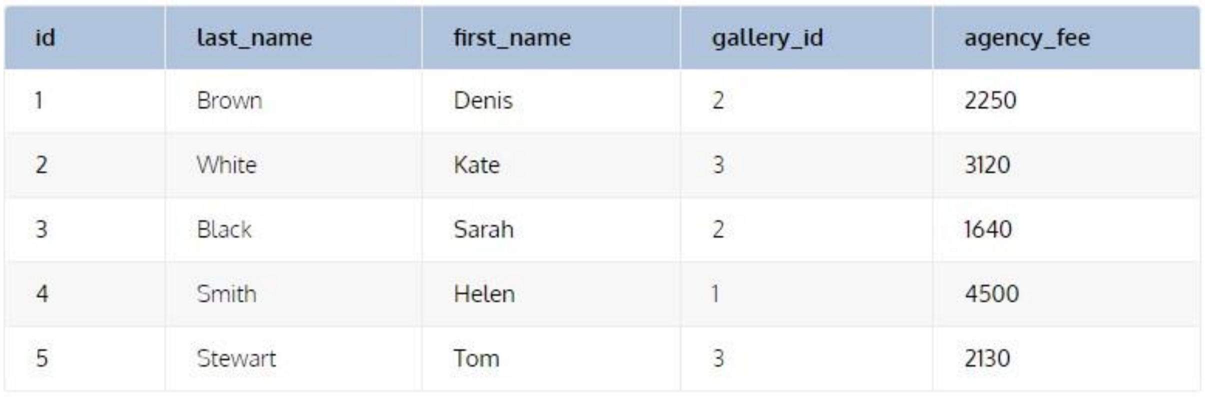
id	city
1	London
2	New York
3	Munich

PAINTINGS_TABLE



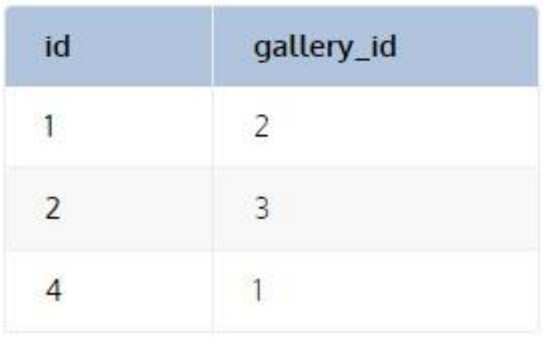
id	name	gallery_id	price
1	Patterns	3	5000
2	Ringer	1	4500
3	Gift	1	3200
4	Violin Lessons	2	6700
5	Curiosity	2	9800

SALES_AGENTS_TABLE



id	last_name	first_name	gallery_id	agency_fee
1	Brown	Denis	2	2250
2	White	Kate	3	3120
3	Black	Sarah	2	1640
4	Smith	Helen	1	4500
5	Stewart	Tom	3	2130

MANAGERS_TABLE



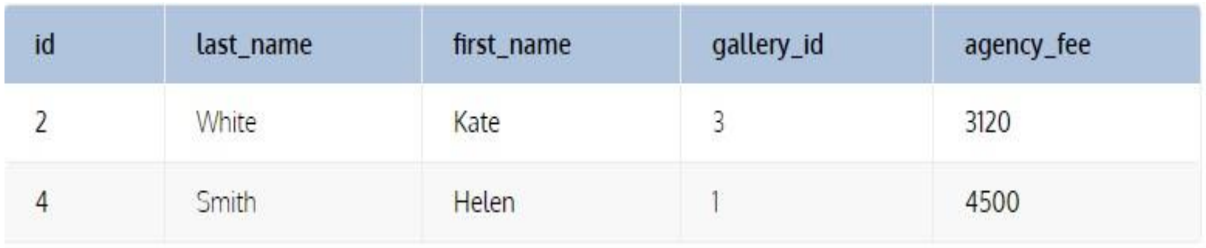
id	gallery_id
1	2
2	3
4	1

Single row sub – query

Example 1:-

```
SELECT * FROM sales_agents
WHERE agency_fee >
(SELECT AVG(agency_fee)
 FROM sales_agents);
```

Output :-

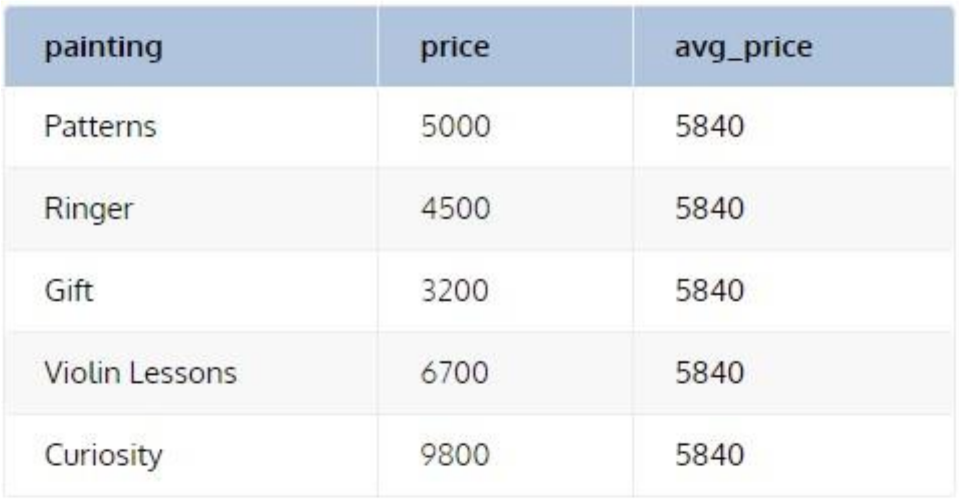


id	last_name	first_name	gallery_id	agency_fee
2	White	Kate	3	3120
4	Smith	Helen	1	4500

Example 2:-

```
SELECT name AS painting, price,
    (SELECT AVG(price)
  FROM paintings) AS avg_price
FROM paintings;
```

Output :-



painting	price	avg_price
Patterns	5000	5840
Ringer	4500	5840
Gift	3200	5840
Violin Lessons	6700	5840
Curiosity	9800	5840

Multiple-Row Subqueries

Example :-

```
SELECT AVG(agency_fee)
FROM sales_agents
WHERE id NOT IN (SELECT id FROM managers);
```

Output :-