UNIT - 3

SQL: QUERIES, CONSTRAINTS, TRIGGERS

What is SQL?

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce after learning about the relational model from Edgar F. Codd in the early 1970s.

It is used in programming and managing data held in relational database management systems such as MySql, MS SQL Server, oracle Sybase, etc as a medium (instructions) for accessing and interacting with data.

It enables performing several operations such as creating, deleting, modifying, and fetching entries in the database and some other advanced statistical, arithmetic, and mathematical operations.

Form of basic SQL query

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data that is stored in the form of tables).

It is also designed for **stream processing in RDBMS**. You can easily create and manipulate the database, access and modify the table rows and columns, etc. It is used in relational database management systems such as MySql, MS SQL Server, Oracle Sybase, etc as a medium (instructions) for accessing and interacting with data.

Types of SQL Commands

In this article on SQL basics, we study about following types of commands:

- **DDL (Data Definition Language):** To make/perform changes to the physical structure of any table residing inside a database, DDL is used. These commands when executed are auto-commit in nature and all the changes in the table are reflected and saved immediately.
- **DML (Data Manipulation Language):** Once the tables are created and the database is generated using DDL commands, manipulation inside those tables and databases is done using DML commands. The advantage of using DML commands is, that if in case any wrong changes or values are made, they can be changed and rolled back easily.
- **DQL (Data Query Language):** Data query language consists of only one command upon which data selection in SQL relies. The **SELECT** command in combination with other SQL clauses is used to retrieve and fetch data from databases/tables based on certain conditions applied by the user.
- **DCL (Data Control Language):** DCL commands as the name suggests manage the matters and issues related to the data controller in any database. DCL includes commands

such as **GRANT** and **REVOKE** which mainly deal with the rights, permissions, and other controls of the database system.

- **TCL (Transaction Control Language):** Transaction Control Language as the name suggests manages the issues and matters related to the transactions in any database. They are used to roll back or commit the changes in the database.

➢ Data Definition Language

In SQL DDL commands are used to create and modify the structure of a database and database objects. These commands are **CREATE**, **DROP**, **ALTER**, **TRUNCATE**, and **RENAME**. Let us discuss these commands one at a time.

CREATE

The **syntax** for create command is:

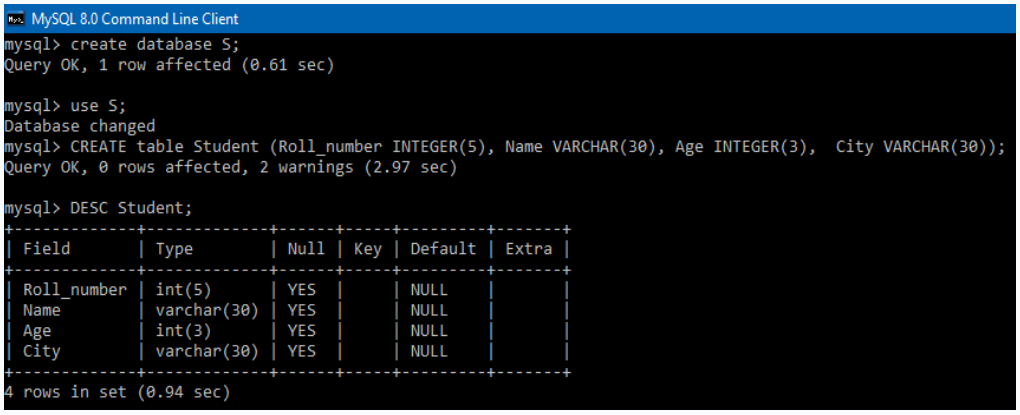
For creating a database:

```
CREATE DATABASE database_name;
```

For creating a table:

```
CREATE TABLE table_name( column1 datatype, column2 datatype, .... columnN datatype);
```

This creates a new table with the name 'table_name' in our database. It will have N columns for each of the same datatypes as mentioned adjacent to it in the create syntax.

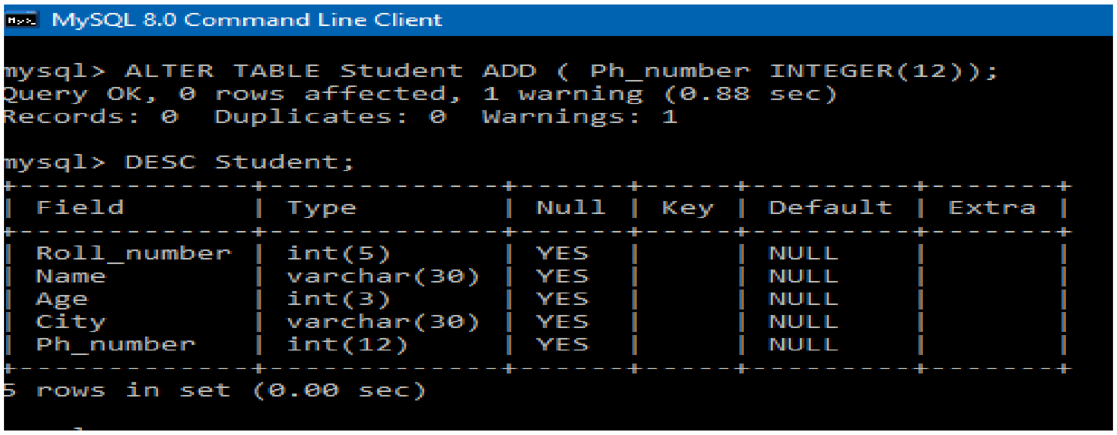


```
mysql> create database S;
Query OK, 1 row affected (0.61 sec)

mysql> use S;
Database changed
mysql> CREATE table Student (Roll_number INTEGER(5), Name VARCHAR(30), Age INTEGER(3),  City VARCHAR(30));
Query OK, 0 rows affected, 2 warnings (2.97 sec)

mysql> DESC Student;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Roll_number | int(5)      | YES  |     | NULL    |       |
| Name        | varchar(30) | YES  |     | NULL    |       |
| Age         | int(3)      | YES  |     | NULL    |       |
| City        | varchar(30) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.94 sec)
```

DESC(DESCRIBE) command is used to retrieve the structure of a table. It may confirm that the table is created.

ALTER: This command changes the structure of a table in a database. We can add a new column in a table and change the data type and size of a column.

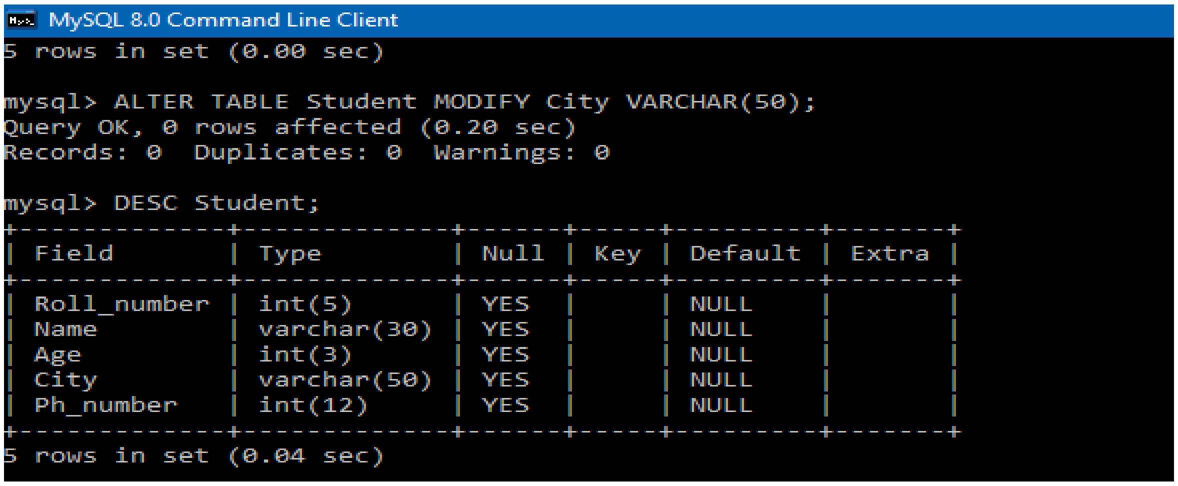
Syntax: ALTER table table_name ADD column_name COLUMN definition;
ALTER table table_name MODIFY(column_definition);
Example: ALTER TABLE Student ADD (Ph_number INTEGER(12));



```
mysql> ALTER TABLE Student ADD ( Ph_number INTEGER(12));
Query OK, 0 rows affected, 1 warning (0.88 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> DESC Student;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Roll_number | int(5)      | YES  |     | NULL    |       |
| Name        | varchar(30) | YES  |     | NULL    |       |
| Age         | int(3)      | YES  |     | NULL    |       |
| City        | varchar(30) | YES  |     | NULL    |       |
| Ph_number   | int(12)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

In the above example, we added a new column in the existing table student.

ALTER TABLE Student MODIFY City VARCHAR(50);

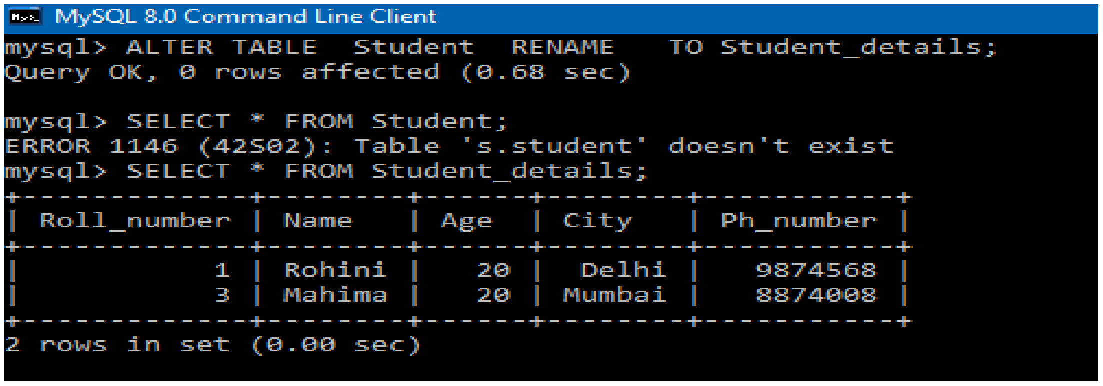


```
5 rows in set (0.00 sec)

mysql> ALTER TABLE Student MODIFY City VARCHAR(50);
Query OK, 0 rows affected (0.20 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC Student;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Roll_number | int(5)      | YES  |     | NULL    |       |
| Name        | varchar(30) | YES  |     | NULL    |       |
| Age         | int(3)      | YES  |     | NULL    |       |
| City        | varchar(50) | YES  |     | NULL    |       |
| Ph_number   | int(12)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.04 sec)
```

In the above example, we changed the size of the City from 30 to 50.

RENAME: This command changes the name of an existing table in a database. For this purpose, we use ALTER command.

Syntax: ALTER TABLE table_name RENAME TO new_table_name;
Example: ALTER TABLE Student RENAME TO Student_details;


MySQL 8.0 Command Line Client

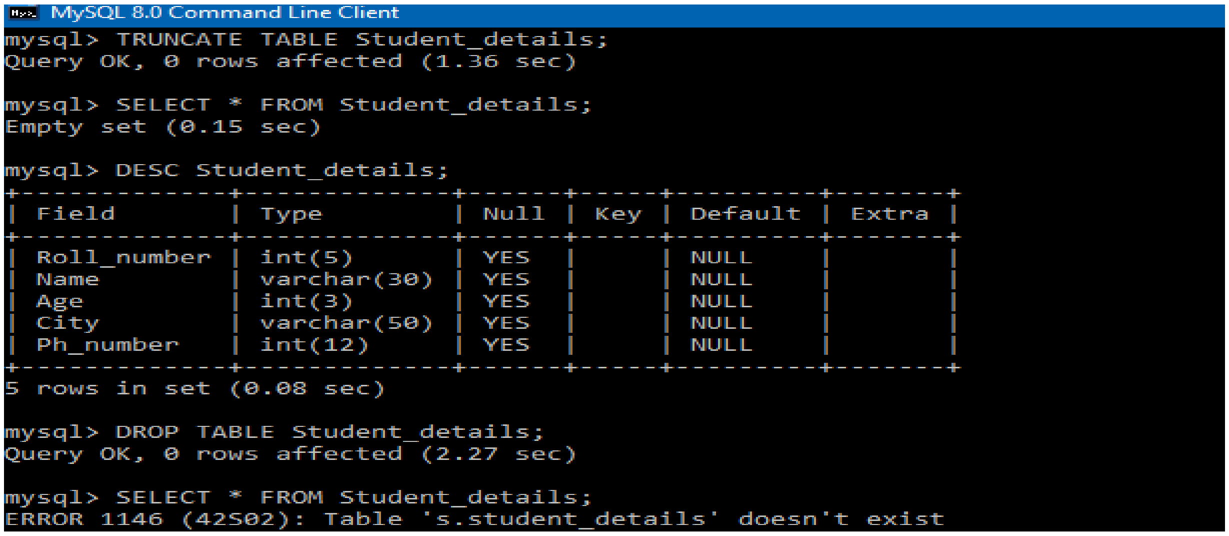
```
mysql> ALTER TABLE  Student  RENAME   TO Student_details;
Query OK, 0 rows affected (0.68 sec)

mysql> SELECT * FROM Student;
ERROR 1146 (42S02): Table 's.student' doesn't exist
mysql> SELECT * FROM Student_details;
+-------------+---------+------+--------+-----------+
| Roll_number | Name    | Age  | City   | Ph_number |
+-------------+---------+------+--------+-----------+
|           1 | Rohini  |   20 |  Delhi |   9874568 |
|           3 | Mahima  |   20 | Mumbai |   8874008 |
+-------------+---------+------+--------+-----------+
2 rows in set (0.00 sec)
```

In the above example, we changed the name of the table Student to Student_details. After that, when we use the old table name, it displays the message “Table Student doesn’t exist in databases “. In this example, S denotes the name of the database. And when we use the new table name, i.e., Student_details, it displays all details of the table.

<u>DROP:</u> The DROP command permanently removes a table from a database.

Syntax: DROP table table_name;
Example: DROP TABLE Student;
<u>TRUNCATE:</u> This command is similar to the drop table command. The only difference is that while the drop command removes the table as well as its contents, the truncate command only erases the contents of the table's contents and not the table itself.

Syntax: TRUNCATE table table_name;
Example: TRUNCATE TABLE Student;


MySQL 8.0 Command Line Client

```
mysql> TRUNCATE TABLE Student_details;
Query OK, 0 rows affected (1.36 sec)

mysql> SELECT * FROM Student_details;
Empty set (0.15 sec)

mysql> DESC Student_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Roll_number | int(5)      | YES  |     | NULL    |       |
| Name        | varchar(30) | YES  |     | NULL    |       |
| Age         | int(3)      | YES  |     | NULL    |       |
| City        | varchar(50) | YES  |     | NULL    |       |
| Ph_number   | int(12)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.08 sec)

mysql> DROP TABLE Student_details;
Query OK, 0 rows affected (2.27 sec)

mysql> SELECT * FROM Student_details;
ERROR 1146 (42S02): Table 's.student_details' doesn't exist
```

In the above example, first, we use the TRUNCATE command, which deletes all rows from the Student_details table. When we try to retrieve the table details, it shows a message ‘EMPTY set’. After that, we use the DROP command, which completely deletes the table Student_details from the database. So when we try to retrieve the table details again, it shows the message ‘Student_details’ doesn’t exist in the database.

➢ <u>**Data Manipulation Language**</u>

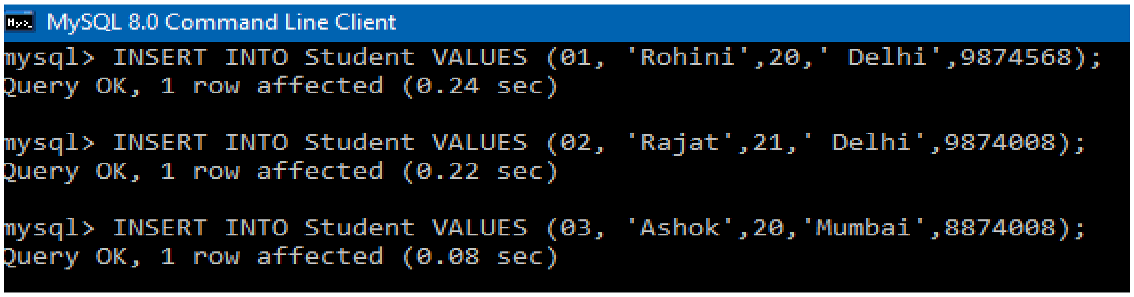
DML is used for inserting, deleting, and updating data in a database. It is used to retrieve and manipulate data in a relational database. It includes INSERT, UPDATE, and DELETE. Let's discuss these commands one at a time.

<u>INSERT</u>

Insert statement is used to insert data in a SQL table. Using the Insert query, we can add one or more rows to the table. Following is the syntax of the MySQL INSERT Statement.

```
INSERT INTO table_name (attribute1, attribute2, ...) VALUES(val1, val2, ...)
```

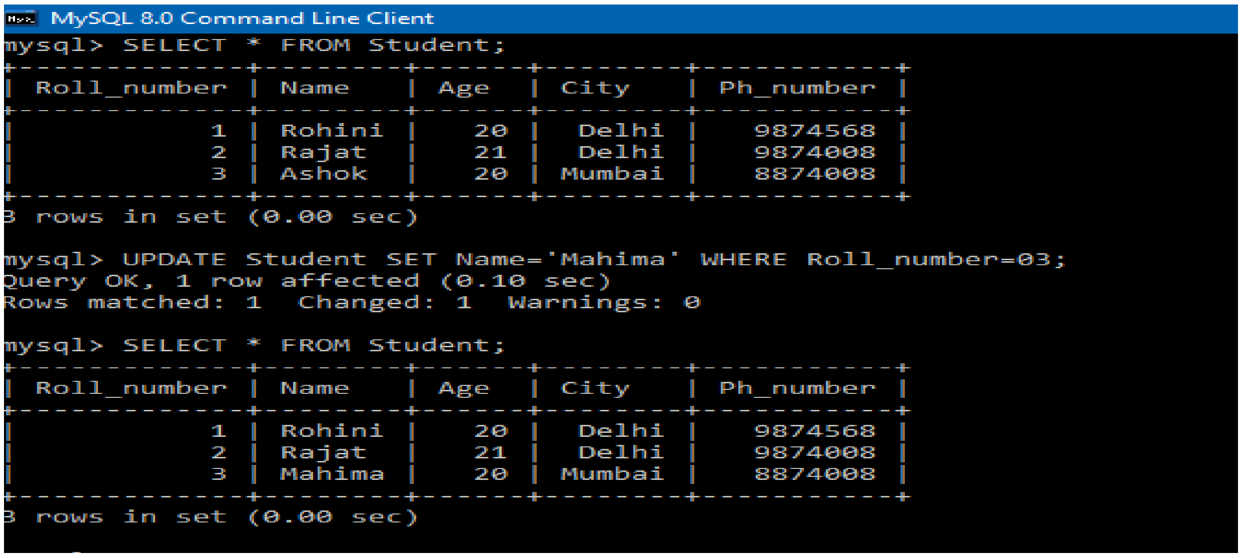
EXAMPLE: INSERT INTO Student VALUES (01, 'Rohini',20,' Delhi');



```
mysql> INSERT INTO Student VALUES (01, 'Rohini',20,' Delhi',9874568);
Query OK, 1 row affected (0.24 sec)

mysql> INSERT INTO Student VALUES (02, 'Rajat',21,' Delhi',9874008);
Query OK, 1 row affected (0.22 sec)

mysql> INSERT INTO Student VALUES (03, 'Ashok',20,'Mumbai',8874008);
Query OK, 1 row affected (0.08 sec)
```

<u>UPDATE:</u> This statement updates the values of a column in a table. To use this, the WHERE condition is necessary.

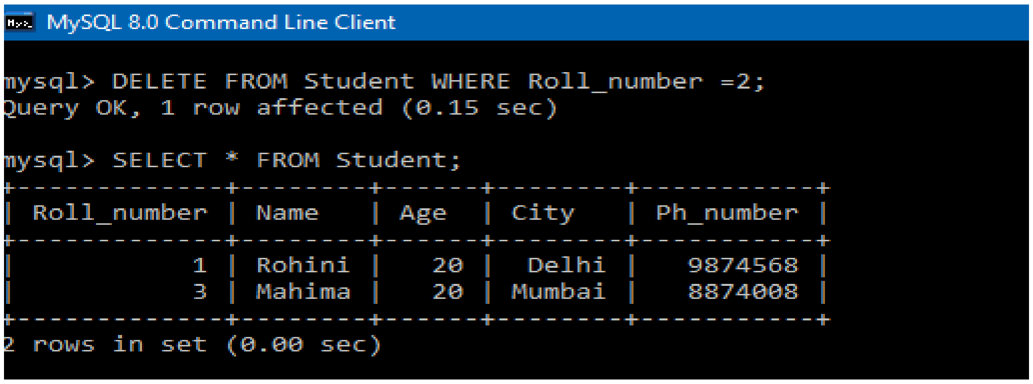
Syntax: UPDATE table_name SET [column 1= value 1 , column n=value n] WHERE condition;
Example: UPDATE Student SET Name='Mahima' WHERE Roll_number=03;



```
mysql> SELECT * FROM Student;
+-------------+--------+------+--------+-----------+
| Roll_number | Name   | Age  | City   | Ph_number |
+-------------+--------+------+--------+-----------+
|           1 | Rohini |   20 |  Delhi |   9874568 |
|           2 | Rajat  |   21 |  Delhi |   9874008 |
|           3 | Ashok  |   20 | Mumbai |   8874008 |
+-------------+--------+------+--------+-----------+
3 rows in set (0.00 sec)

mysql> UPDATE Student SET Name='Mahima' WHERE Roll_number=03;
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Student;
+-------------+--------+------+--------+-----------+
| Roll_number | Name   | Age  | City   | Ph_number |
+-------------+--------+------+--------+-----------+
|           1 | Rohini |   20 |  Delhi |   9874568 |
|           2 | Rajat  |   21 |  Delhi |   9874008 |
|           3 | Mahima |   20 | Mumbai |   8874008 |
+-------------+--------+------+--------+-----------+
3 rows in set (0.00 sec)
```

<u>DELETE:</u> This command removes a particular row from a table. Note that the **WHERE** condition is necessary to perform the delete operation.

Syntax: DELETE FROM table_name WHERE condition;
Example: DELETE FROM Student WHERE Roll_number =2;



MySQL 8.0 Command Line Client

```
mysql> DELETE FROM Student WHERE Roll_number =2;
Query OK, 1 row affected (0.15 sec)

mysql> SELECT * FROM Student;
+-------------+--------+------+--------+-----------+
| Roll_number | Name   | Age  | City   | Ph_number |
+-------------+--------+------+--------+-----------+
|           1 | Rohini |   20 |  Delhi |   9874568 |
|           3 | Mahima |   20 | Mumbai |   8874008 |
+-------------+--------+------+--------+-----------+
2 rows in set (0.00 sec)
```

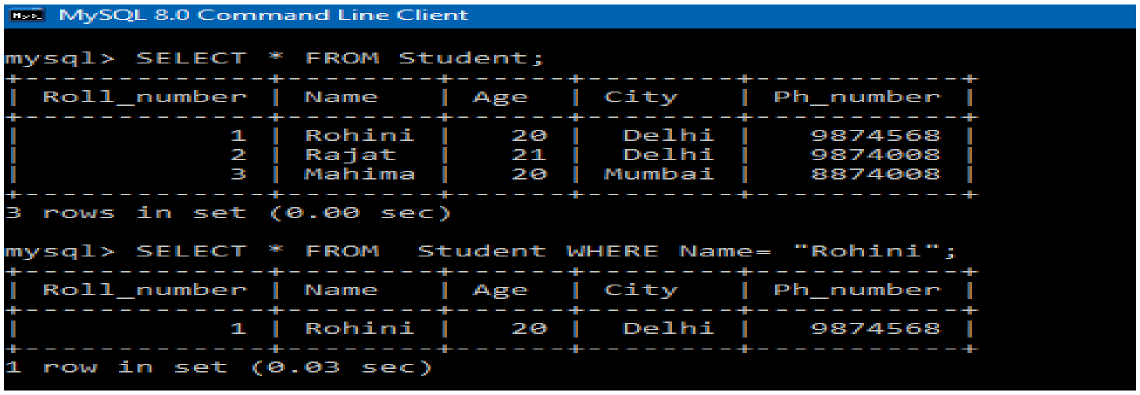
➢ <u>Data Query Language:</u>

DQL commands are used for fetching data from a relational database. They perform read-only queries of data. The only command, **'SELECT'** is equivalent to the projection operation in relational algebra. It command selects the attribute based on the condition described by the **WHERE** clause and returns them.

One can fetch either the entire table or some data according to specified rules. The data returned is stored in a result table. With the SELECT clause of a SELECT command statement, we specify the columns that we want to be displayed in the query result and, optionally, which column headings we prefer to see above the result table.

Syntax: SELECT * FROM table_name WHERE condition;
SELECT * FROM table_name;
Example: SELECT * FROM Student;

SELECT * FROM Student WHERE Name= “Rohini”;



MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM Student;
+-------------+--------+------+--------+-----------+
| Roll_number | Name   | Age  | City   | Ph_number |
+-------------+--------+------+--------+-----------+
|           1 | Rohini |   20 |  Delhi |   9874568 |
|           2 | Rajat  |   21 |  Delhi |   9874008 |
|           3 | Mahima |   20 | Mumbai |   8874008 |
+-------------+--------+------+--------+-----------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM  Student WHERE Name= "Rohini";
+-------------+--------+------+--------+-----------+
| Roll_number | Name   | Age  | City   | Ph_number |
+-------------+--------+------+--------+-----------+
|           1 | Rohini |   20 |  Delhi |   9874568 |
+-------------+--------+------+--------+-----------+
1 row in set (0.03 sec)
```

➢ Data Control Language

DCL is used to access the stored data. It is used to revoke and grant the user the required access to a database. In the database, this language does not have the feature of rollback. It is a part of the **structured query language (SQL)**.

It helps in controlling access to information stored in a database. It complements the data manipulation language and the data definition language. It is the simplest of three commands.

It provides the administrators, to remove and set database permissions to desired users as needed. These commands are employed to grant, remove and deny permissions to users for retrieving and manipulating a database. There are two relevant commands under this category: grant and revoke.

✓ GRANT

GRANT is a command used to provide access or privileges on the database objects to the users.

SYNTAX

```
GRANT PRIVILEGES ON OBJECT TO USER;
```

✓ REVOKE

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

```
REVOKE privileges ON object FROM user;
```

Object is the name of the database object that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name. Username of the user that will have these privileges revoked.

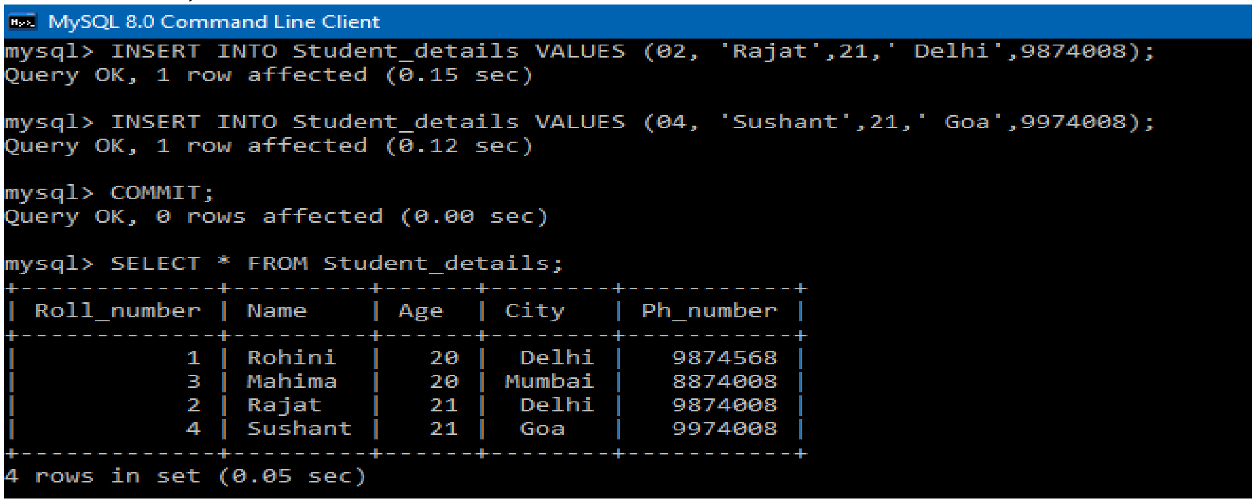
➢ Transaction Control Language

TCL includes statements that are used to manage the changes that are made from DML statements. It enhances the transactional nature of SQL. The TCL commands in SQL are:

- ✓ **COMMIT:** It's a SQL command used in the transaction tables or database to make the current transaction or database statement permanent. It shows the successful completion of a transaction. If we have successfully executed the transaction statement or a simple database query, we want to make the changes permanent. We need to perform the commit command to save the changes, and these changes become permanent for all users.

Syntax**:** COMMIT**;**
Example: INSERT INTO Student_details VALUES(02,’Rajat’,21,’Delhi’,9874008);

COMMIT;


MySQL 8.0 Command Line Client

```
mysql> INSERT INTO Student_details VALUES (02, 'Rajat',21,' Delhi',9874008);
Query OK, 1 row affected (0.15 sec)

mysql> INSERT INTO Student_details VALUES (04, 'Sushant',21,' Goa',9974008);
Query OK, 1 row affected (0.12 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Student_details;
+-------------+---------+------+--------+-----------+
| Roll_number | Name    | Age  | City   | Ph_number |
+-------------+---------+------+--------+-----------+
|           1 | Rohini  |   20 |  Delhi |   9874568 |
|           3 | Mahima  |   20 | Mumbai |   8874008 |
|           2 | Rajat   |   21 |  Delhi |   9874008 |
|           4 | Sushant |   21 |  Goa   |   9974008 |
+-------------+---------+------+--------+-----------+
4 rows in set (0.05 sec)
```

- ✓ **<u>ROLLBACK</u>:** Undoes any changes made to the database. ROLLBACK is the SQL command that is used for reverting changes performed by a transaction. When a ROLLBACK command is issued it reverts all the changes since the last COMMIT or ROLLBACK.

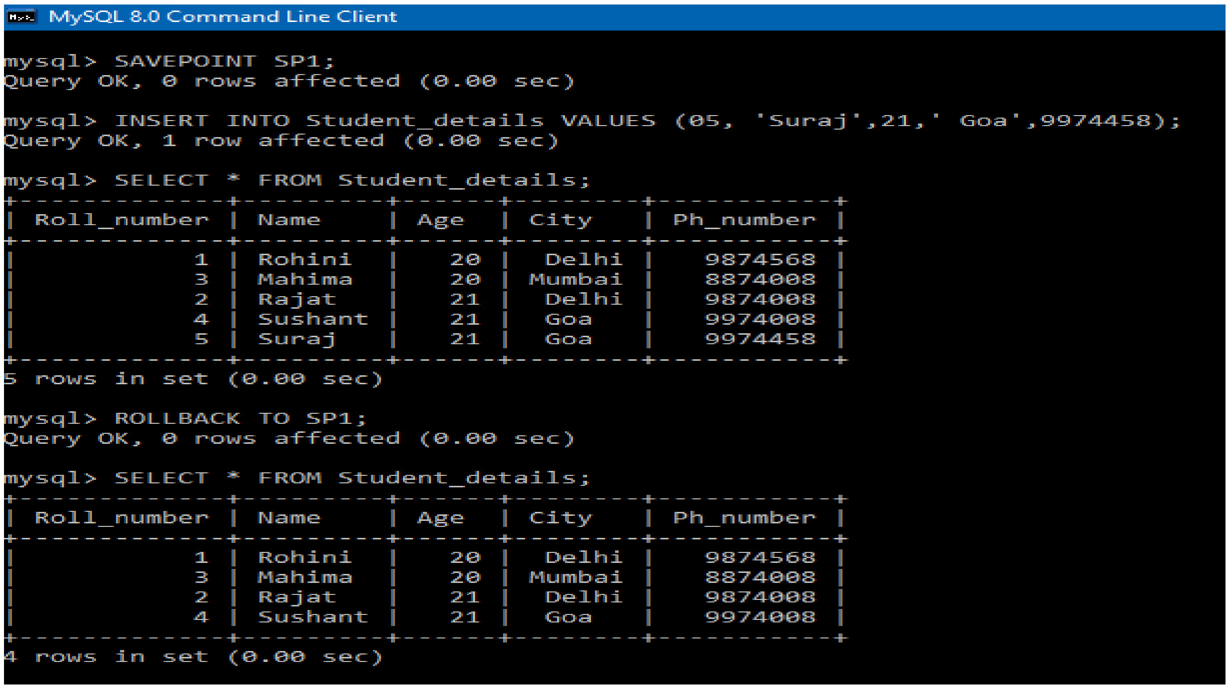
Syntax: ROLLBACK;
Example: DELETE FROM Student_details WHERE Roll_number=4;
ROLLBACK;

- ✓ **<u>SAVEPOINT</u>:** This command creates a point in your transaction to which you can roll back. It is a command in SQL that is used with the rollback command. It is a command in Transaction Control Language that is used to mark the transaction in a table.

SYNTAX

```
SAVEPOINT some_name;
```

Example: SAVEPOINT SP1;
INSERT INTO Student_details VALUES(05,'Suraj',21,'Goa',9974458); ;
//deleted
SAVEPOINT SP2;
//Savepoint created.
//Rollback completed.
//Savepoint created.
ROLLBACK TO SP1;



MySQL 8.0 Command Line Client

```
mysql> SAVEPOINT SP1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Student_details VALUES (05, 'Suraj',21,' Goa',9974458);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Student_details;
+-------------+---------+------+--------+-----------+
| Roll_number | Name    | Age  | City   | Ph_number |
+-------------+---------+------+--------+-----------+
|           1 | Rohini  |   20 |  Delhi |   9874568 |
|           3 | Mahima  |   20 | Mumbai |   8874008 |
|           2 | Rajat   |   21 |  Delhi |   9874008 |
|           4 | Sushant |   21 |  Goa   |   9974008 |
|           5 | Suraj   |   21 |  Goa   |   9974458 |
+-------------+---------+------+--------+-----------+
5 rows in set (0.00 sec)

mysql> ROLLBACK TO SP1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Student_details;
+-------------+---------+------+--------+-----------+
| Roll_number | Name    | Age  | City   | Ph_number |
+-------------+---------+------+--------+-----------+
|           1 | Rohini  |   20 |  Delhi |   9874568 |
|           3 | Mahima  |   20 | Mumbai |   8874008 |
|           2 | Rajat   |   21 |  Delhi |   9874008 |
|           4 | Sushant |   21 |  Goa   |   9974008 |
+-------------+---------+------+--------+-----------+
4 rows in set (0.00 sec)
```

✓ SQL Set Operators:

A set operator in SQL is a keyword that lets you combine the results of two queries into a single query.

The set operators work on complete rows of the queries, so the results of the queries must have the same column name, same column order and the types of columns must be compatible.

There are the following 4 set operators in SQL :

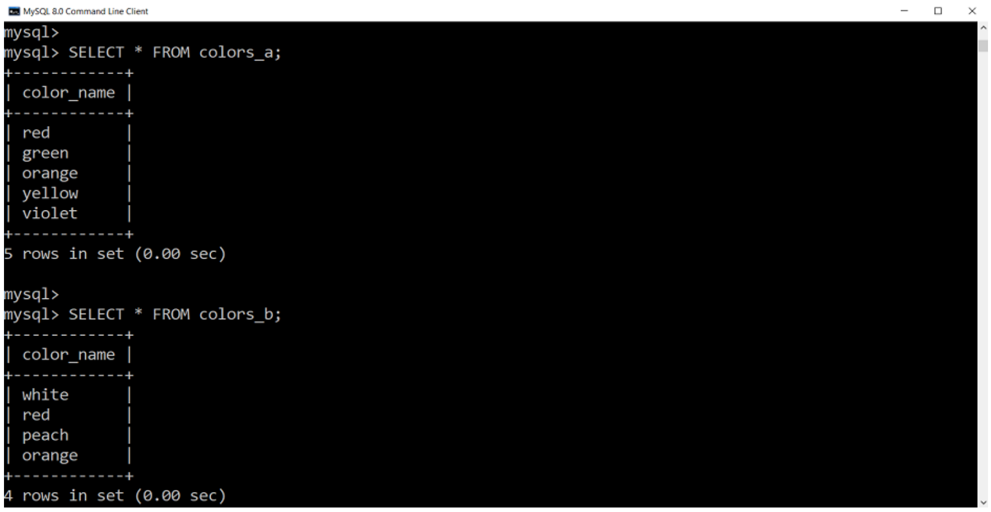
1. **UNION**: Combine two or more result sets into a single set, without duplicates.
2. **UNION ALL**: Combine two or more result sets into a single set, including all duplicates.
3. **INTERSECT**: Takes the data from both result sets which are in common.
4. **EXCEPT**: Takes the data from the first result set, but not in the second result set

UNION:

The UNION operator combines result sets of two or more SELECT statements into a single result set.Every SELECT statement within UNION must have the same number of columns.The columns must also have similar data types.The columns in every SELECT statement must also be in the same order.This operator removes any duplicates present in the results being combined.

Syntax:

- SELECT column_name(s) FROM table1
  UNION
  SELECT column_name(s) FROM table2;



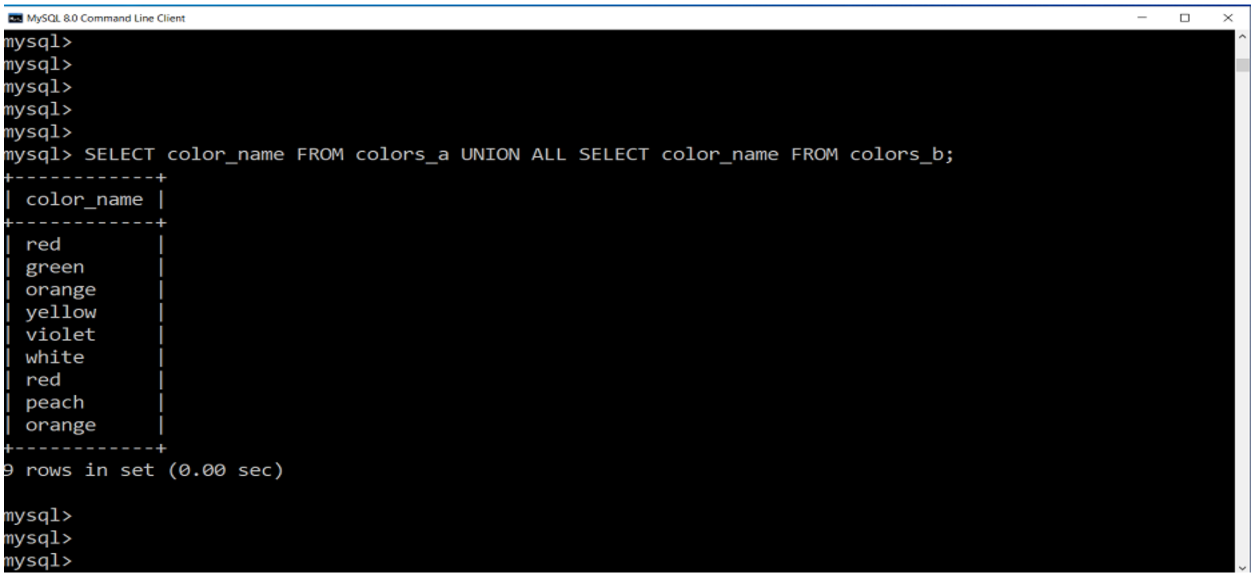
```
mysql>
mysql> SELECT * FROM colors_a;
+------------+
| color_name |
+------------+
| red        |
| green      |
| orange     |
| yellow     |
| violet     |
+------------+
5 rows in set (0.00 sec)

mysql>
mysql> SELECT * FROM colors_b;
+------------+
| color_name |
+------------+
| white      |
| red        |
| peach      |
| orange     |
+------------+
4 rows in set (0.00 sec)
```

UNION ALL:

- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

Syntax:

- SELECT column_name(s) FROM table1
  UNION ALL
  SELECT column_name(s) FROM table2



```
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> SELECT color_name FROM colors_a UNION ALL SELECT color_name FROM colors_b;
+------------+
| color_name |
+------------+
| red        |
| green      |
| orange     |
| yellow     |
| violet     |
| white      |
| red        |
| peach      |
| orange     |
+------------+
9 rows in set (0.00 sec)

mysql>
mysql>
mysql>
```

INTERSECT:

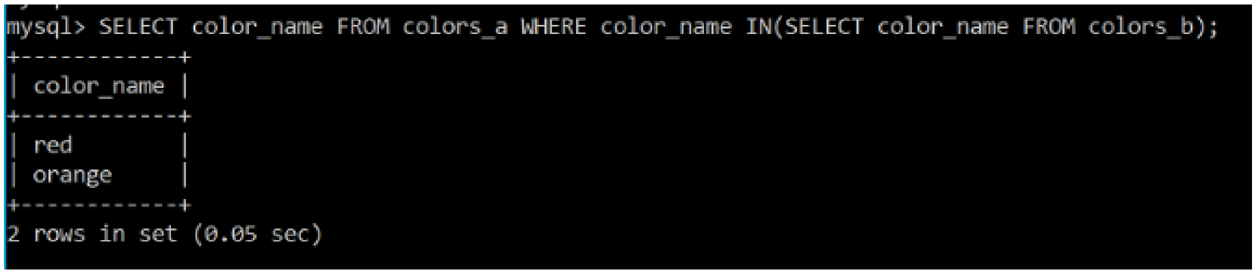
The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. The INTERSECT clause in SQL is used to combine two SELECT statements but the dataset returned by the INTERSECT statement will be the intersection of the data sets of the two SELECT statements.

Syntax:

SELECT column1 , column2 ….FROM table_names

INTERSECT

SELECT column1 , column2 ….FROM table_names



```
mysql> SELECT color_name FROM colors_a WHERE color_name IN(SELECT color_name FROM colors_b);
+------------+
| color_name |
+------------+
| red        |
| orange     |
+------------+
2 rows in set (0.05 sec)
```

EXCEPT:

- EXCEPT is a set operator in SQL that returns the distinct rows that are present in the result set of the first query but not in the result set of the second query.
- It is also known as the set difference operator. EXCEPT is used in conjunction with the SELECT statement to compare the result sets of two or more queries.
- Conditions for EXCEPT clause:
- There must be the same number of expressions in both SELECT statements.
- The data types of corresponding columns should be the same or compatible.

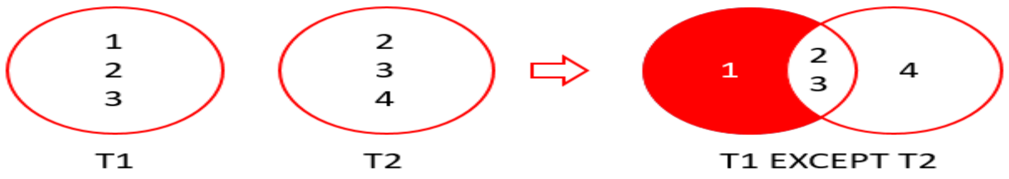


Illustration:

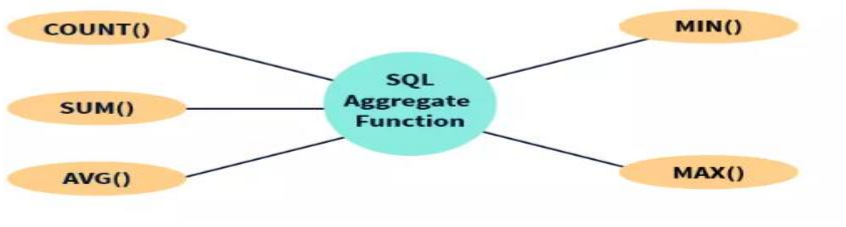
- Table T1 includes data 1, 2, and 3.
- Table T2 includes data 2, 3, and 4.
- When we execute the EXCEPT query on these tables, we will get 1, which is unique data from the T1, and it will not found in the T2.

➢ Aggregate Functions in SQL

The aggregate functions are used to perform the calculation based on multiple rows and return a single value according to the given query. All these aggregate functions are used with a SELECT statement. It is also used to summarize the data.

- **Syntax –**
  SELECT <FUNCTION NAME> (<PARAMETER>) FROM <TABLE NAME>

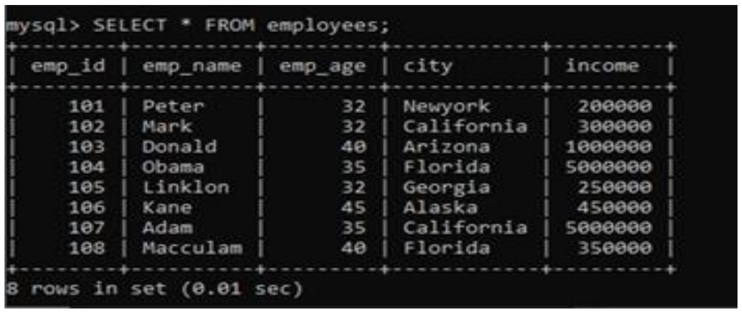
Types of Aggregate Functions in SQL



Count ():

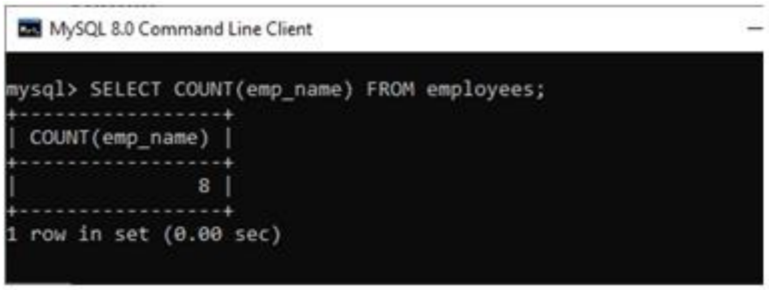
- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax: COUNT(*) or COUNT([ALL|DISTINCT] expression)

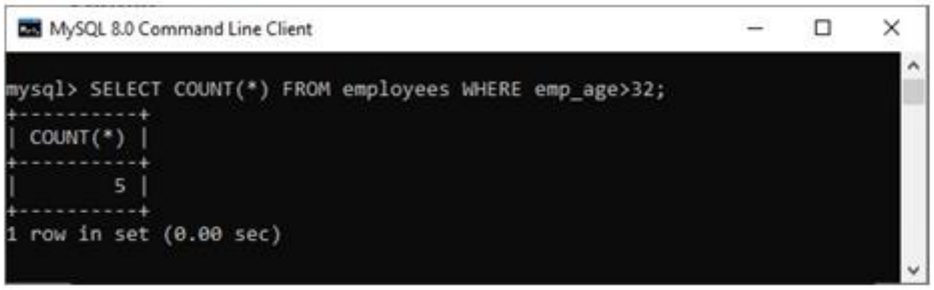


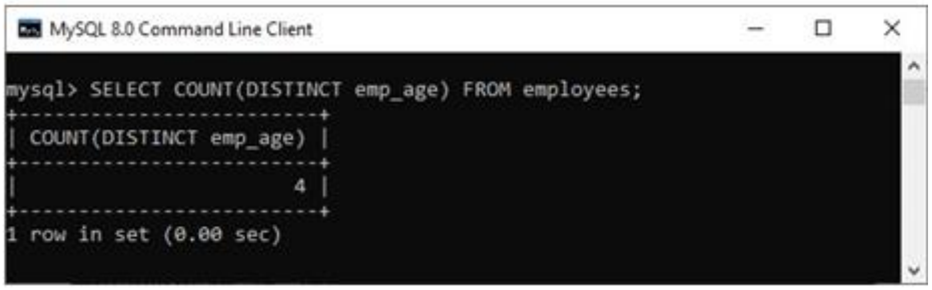
```
mysql> SELECT * FROM employees;
+--------+----------+---------+------------+---------+
| emp_id | emp_name | emp_age | city       | income  |
+--------+----------+---------+------------+---------+
|    101 | Peter    |      32 | Newyork    |  200000 |
|    102 | Mark     |      32 | California |  300000 |
|    103 | Donald   |      40 | Arizona    | 1000000 |
|    104 | Obama    |      35 | Florida    | 5000000 |
|    105 | Linklon  |      32 | Georgia    |  250000 |
|    106 | Kane     |      45 | Alaska     |  450000 |
|    107 | Adam     |      35 | California | 5000000 |
|    108 | Macculam |      40 | Florida    |  350000 |
+--------+----------+---------+------------+---------+
8 rows in set (0.01 sec)
```

Examples:



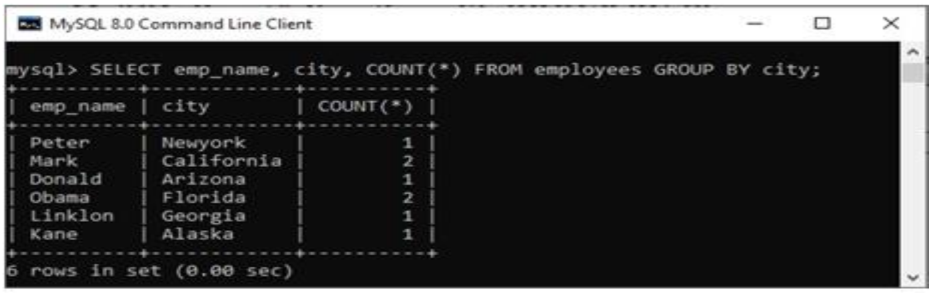
```
MySQL 8.0 Command Line Client

mysql> SELECT COUNT(emp_name) FROM employees;
+-----------------+
| COUNT(emp_name) |
+-----------------+
|               8 |
+-----------------+
1 row in set (0.00 sec)
```


```
mysql> SELECT COUNT(*) FROM employees WHERE emp_age>32;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```


```
mysql> SELECT COUNT(DISTINCT emp_age) FROM employees;
+-------------------------+
| COUNT(DISTINCT emp_age) |
+-------------------------+
|                       4 |
+-------------------------+
1 row in set (0.00 sec)
```

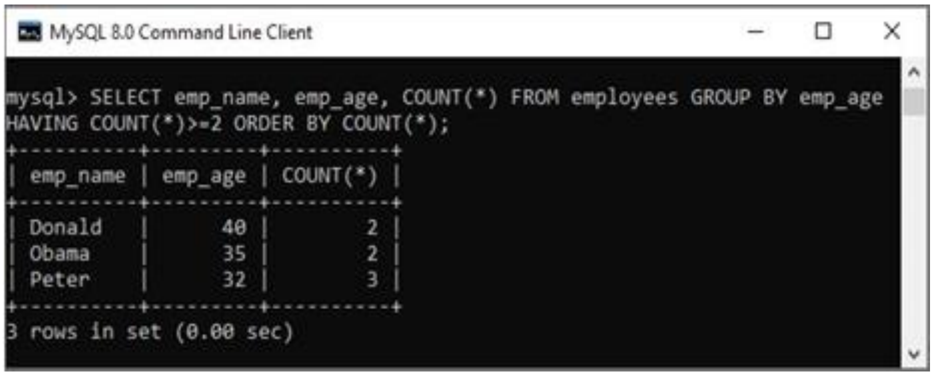
Count() Function with GROUP BY Clause

We can also use the count() function with the GROUP BY clause that returns the count of the element in each group. For example, the following statement returns the number of employee in each city:


```
mysql> SELECT emp_name, city, COUNT(*) FROM employees GROUP BY city;
+----------+------------+----------+
| emp_name | city       | COUNT(*) |
+----------+------------+----------+
| Peter    | Newyork    |        1 |
| Mark     | California |        2 |
| Donald   | Arizona    |        1 |
| Obama    | Florida    |        2 |
| Linklon  | Georgia    |        1 |
| Kane     | Alaska     |        1 |
+----------+------------+----------+
6 rows in set (0.00 sec)
```

Count() Function with HAVING and ORDER BY Clause

The following statement that gives the employee name who has at least two age same and sorts them based on the count result:

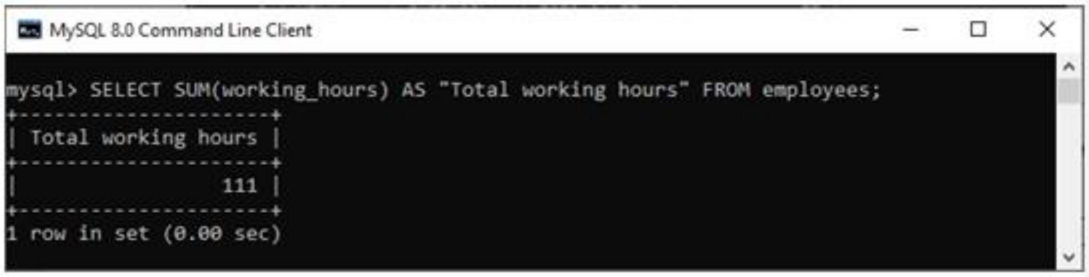

```
mysql> SELECT emp_name, emp_age, COUNT(*) FROM employees GROUP BY emp_age
HAVING COUNT(*)>=2 ORDER BY COUNT(*);
+----------+---------+----------+
| emp_name | emp_age | COUNT(*) |
+----------+---------+----------+
| Donald   |      40 |        2 |
| Obama    |      35 |        2 |
| Peter    |      32 |        3 |
+----------+---------+----------+
3 rows in set (0.00 sec)
```

SUM ( ):

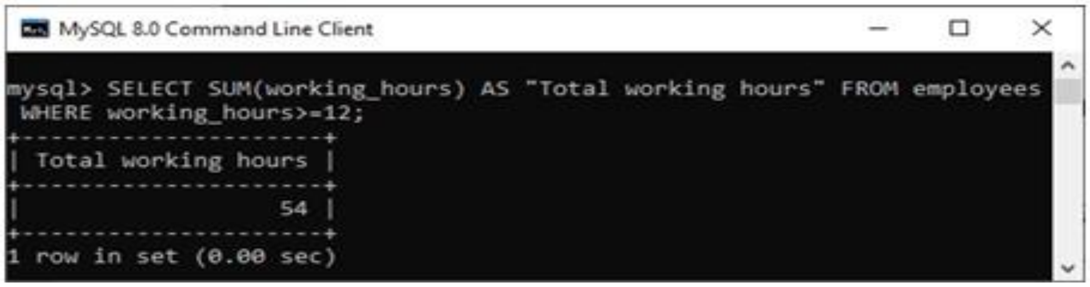
The SUM() function takes the name of the column as an argument and returns the sum of all the non NULL values in that column. It works only on numeric fields(i.e the columns contain only numeric values). When applied to columns containing both non-numeric(ex - strings) and numeric values, only numeric values are considered. If no numeric values are present, the function returns 0.

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

Examples:



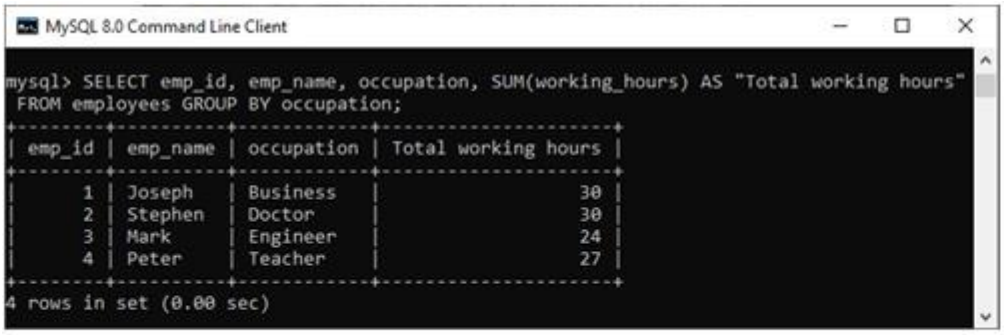
```
mysql> SELECT SUM(working_hours) AS "Total working hours" FROM employees;
+---------------------+
| Total working hours |
+---------------------+
|                 111 |
+---------------------+
1 row in set (0.00 sec)
```



```
mysql> SELECT SUM(working_hours) AS "Total working hours" FROM employees
 WHERE working_hours>=12;
+---------------------+
| Total working hours |
+---------------------+
|                  54 |
+---------------------+
1 row in set (0.00 sec)
```

MySQL sum() function with GROUP BY clause

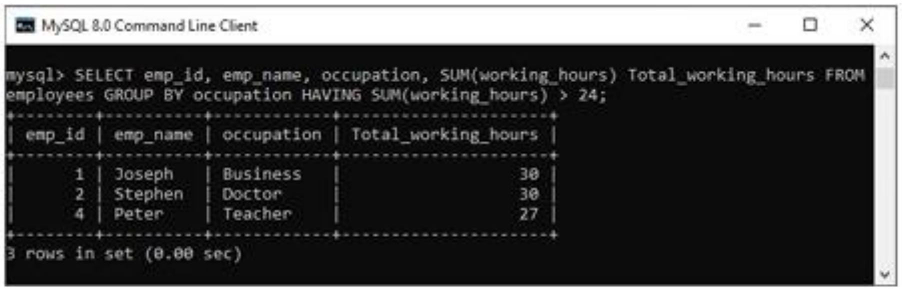
We can also use the SUM() function with the GROUP BY clause to return the total summed value for each group. For example, this statement calculates the total working hours of each employee by using the SUM() function with the GROUP BY clause, as shown in the following query:



```
mysql> SELECT emp_id, emp_name, occupation, SUM(working_hours) AS "Total working hours"
 FROM employees GROUP BY occupation;
+--------+----------+------------+---------------------+
| emp_id | emp_name | occupation | Total working hours |
+--------+----------+------------+---------------------+
|      1 | Joseph   | Business   |                  30 |
|      2 | Stephen  | Doctor     |                  30 |
|      3 | Mark     | Engineer   |                  24 |
|      4 | Peter    | Teacher    |                  27 |
+--------+----------+------------+---------------------+
4 rows in set (0.00 sec)
```

MySQL sum() function with HAVING clause

The HAVING clause is used to filter the group with the sum() function in MySQL. Execute the following statement that calculates the working hours of all employees, grouping them based on their occupation and returns the result whose Total_working_hours>24.


MySQL 8.0 Command Line Client

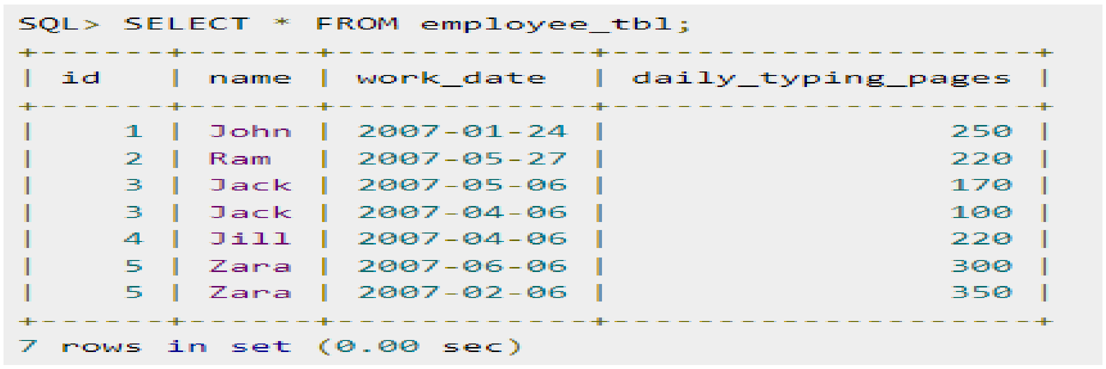
```
mysql> SELECT emp_id, emp_name, occupation, SUM(working_hours) Total_working_hours FROM
employees GROUP BY occupation HAVING SUM(working_hours) > 24;
+--------+----------+------------+---------------------+
| emp_id | emp_name | occupation | Total_working_hours |
+--------+----------+------------+---------------------+
|      1 | Joseph   | Business   |                  30 |
|      2 | Stephen  | Doctor     |                  30 |
|      4 | Peter    | Teacher    |                  27 |
+--------+----------+------------+---------------------+
3 rows in set (0.00 sec)
```

AVG ( ):

The AVG() aggregate function uses the name of the column as an argument and returns the average of all the non NULL values in that column. It works only on numeric fields.When applied to columns containing both non-numeric (ex - strings) and numeric values, only numeric values are considered. If no numeric values are present, the function returns 0.

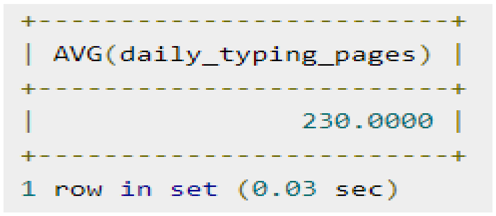
Syntax : AVG () or AVG ([ALL|DISTINCT] expression)

Example:



```
SQL> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|    1 | John | 2007-01-24 |                250 |
|    2 | Ram  | 2007-05-27 |                220 |
|    3 | Jack | 2007-05-06 |                170 |
|    3 | Jack | 2007-04-06 |                100 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
|    5 | Zara | 2007-02-06 |                350 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

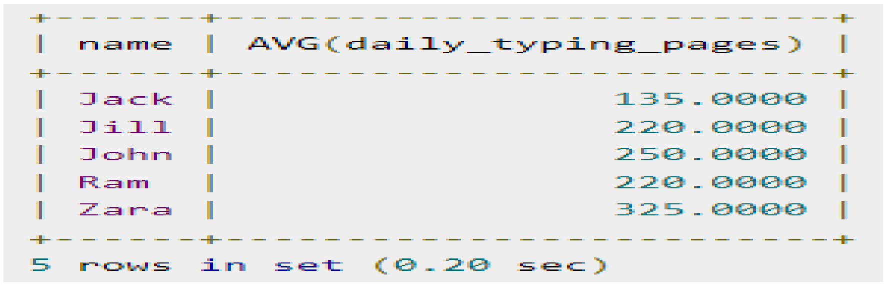
Query: SQL> SELECT AVG(daily_typing_pages) FROM employee_tbl;



```
+-------------------------+
| AVG(daily_typing_pages) |
+-------------------------+
|                230.0000 |
+-------------------------+
1 row in set (0.03 sec)
```

You can take average of various records set using GROUP BY clause. Following example will take average all the records related to a single person and you will have average typed pages by every person.

Query: SQL> SELECT name, AVG(daily_typing_pages) FROM employee_tbl GROUP BY name;



```
+------+---------------------------+
| name | AVG(daily_typing_pages)   |
+------+---------------------------+
| Jack |                  135.0000 |
| Jill |                  220.0000 |
| John |                  250.0000 |
| Ram  |                  220.0000 |
| Zara |                  325.0000 |
+------+---------------------------+
5 rows in set (0.20 sec)
```

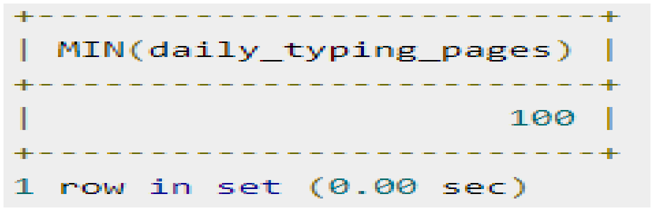
MIN ( ):

The MIN() function takes the name of the column as an argument and returns the minimum value present in the column. MIN() returns NULL when no row is selected.

Syntax : MIN() or MIN([ALL|DISTINCT] expression)

Example:

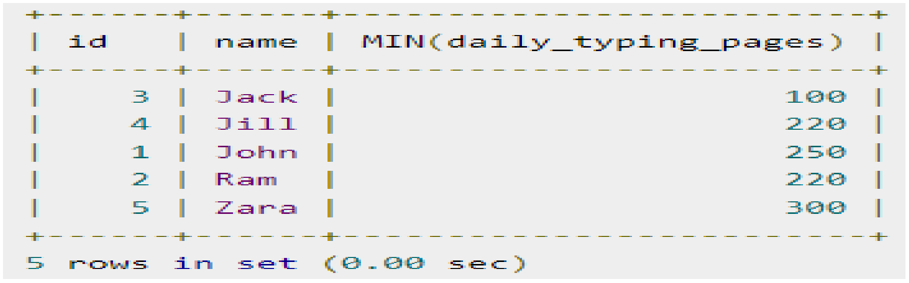
SQL> SELECT MIN(daily_typing_pages) FROM employee_tbl;



```
+-------------------------+
| MIN(daily_typing_pages) |
+-------------------------+
|                     100 |
+-------------------------+
1 row in set (0.00 sec)
```

You can find all the records with minimum value for each name using GROUP BY clause as follows −

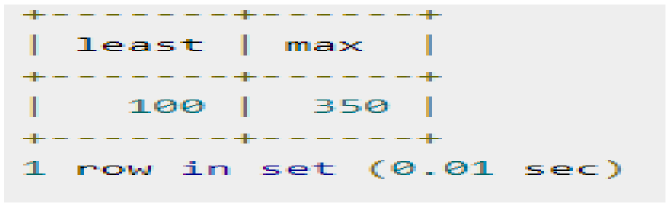
SQL> SELECT id, name, work_date, MIN(daily_typing_pages) FROM employee_tbl GROUP BY name;



```
+------+------+-------------------------+
| id   | name | MIN(daily_typing_pages) |
+------+------+-------------------------+
|    3 | Jack |                     100 |
|    4 | Jill |                     220 |
|    1 | John |                     250 |
|    2 | Ram  |                     220 |
|    5 | Zara |                     300 |
+------+------+-------------------------+
5 rows in set (0.00 sec)
```

You can use MIN Function along with MAX function to find out minimum value as well. Try out the following example −

SQL> SELECT MIN(daily_typing_pages) least,MAX(daily_typing_pages) max FROM employee_tbl;



```
+---------+-------+
| least   | max   |
+---------+-------+
|     100 |   350 |
+---------+-------+
1 row in set (0.01 sec)
```

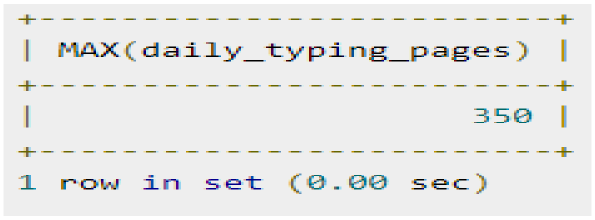
MAX ( ):

The MAX() function takes the name of the column as an argument and returns the maximum value present in the column. MAX() returns NULL when no row is selected.

Syntax: MAX() or MAX([ALL|DISTINCT] expression)

Examples:

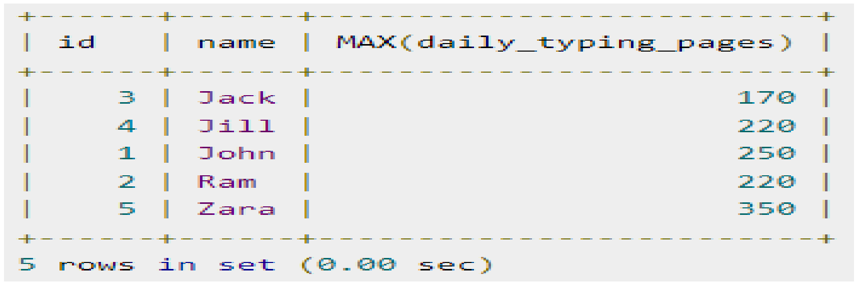
SQL> SELECT MAX(daily_typing_pages) FROM employee_tbl;



```
+-------------------------+
| MAX(daily_typing_pages) |
+-------------------------+
|                     350 |
+-------------------------+
1 row in set (0.00 sec)
```

You can find all the records with maximum value for each name using GROUP BY clause as follows −

SQL> SELECT id, name, MAX(daily_typing_pages) FROM employee_tbl GROUP BY name;



```
+------+------+-------------------------+
| id   | name | MAX(daily_typing_pages) |
+------+------+-------------------------+
|    3 | Jack |                     170 |
|    4 | Jill |                     220 |
|    1 | John |                     250 |
|    2 | Ram  |                     220 |
|    5 | Zara |                     350 |
+------+------+-------------------------+
5 rows in set (0.00 sec)
```

➢ Nested Queries or Sub Queries:

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within clauses, most commonly in the WHERE clause. It is used to return data from a table, and this data will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Types of Nested Queries in SQL:

Nested queries in SQL can be classified into two different types:

- ❖ Independent Nested Queries
- ❖ Co-related Nested Queries

✓ **<u>Independent Nested Queries</u>**

In independent nested queries, the execution order is from the innermost query to the outer query. An outer query won't be executed until its inner query completes its execution. The outer query uses the result of the inner query. Operators such as IN, NOT IN, ALL, and ANY are used to write independent nested queries.

The **IN** operator checks if a column value in the outer query's result is present in the inner query's result. The final result will have rows that satisfy the IN condition.

The **NOT IN** operator checks if a column value in the outer query's result is not present in the inner query's result. The final result will have rows that satisfy the NOT IN condition.

The **ALL** operator compares a value of the outer query's result with all the values of the inner query's result and returns the row if it matches all the values.

The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with any value.

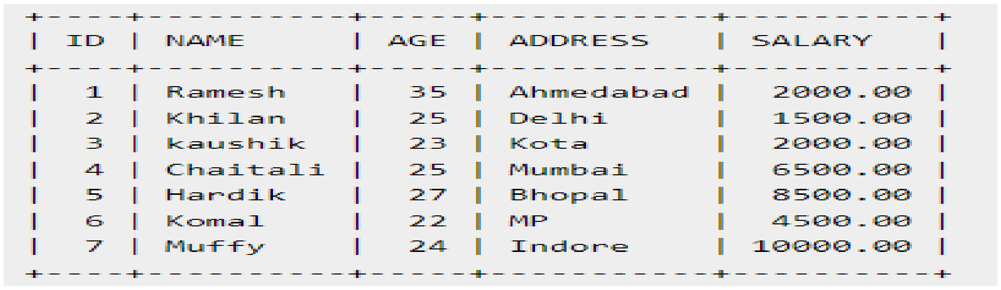
✓ **<u>Co-related Nested Queries:</u>**

In co-related nested queries, the inner query uses the values from the outer query to execute the inner query for every row processed by the outer query.

The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

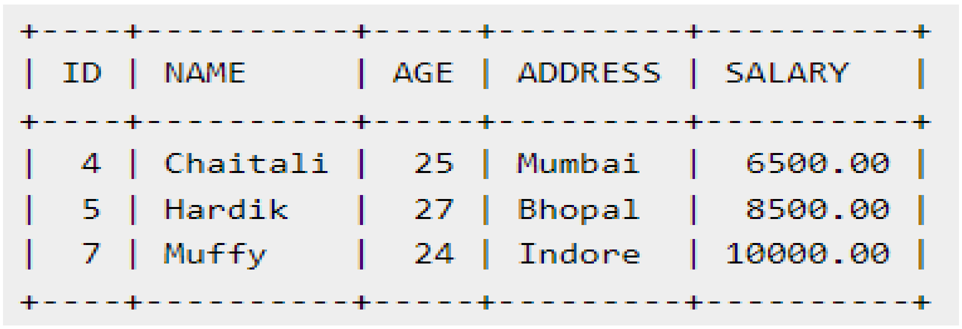
<u>Example</u>

Consider the CUSTOMERS table having the following records –



```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

Sql> select * from customers where id in (select id from customers where salary > 4500);



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

<u>Subqueries with the INSERT Statement</u>

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

<u>Example</u>

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

Sql> insert into customers_bkp select * from customers where id in (select id from customers);

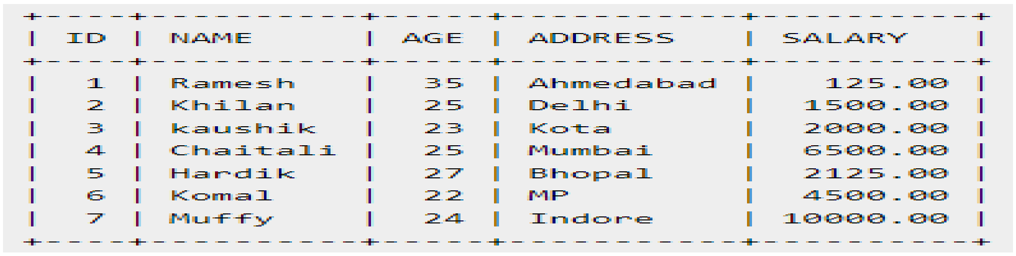
<u>Subqueries with the UPDATE Statement</u>

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

<u>Example</u>

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

SQL> update customers set salary = salary * 0.25 where age in (select age from customers_bkp where age >= 27);



```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

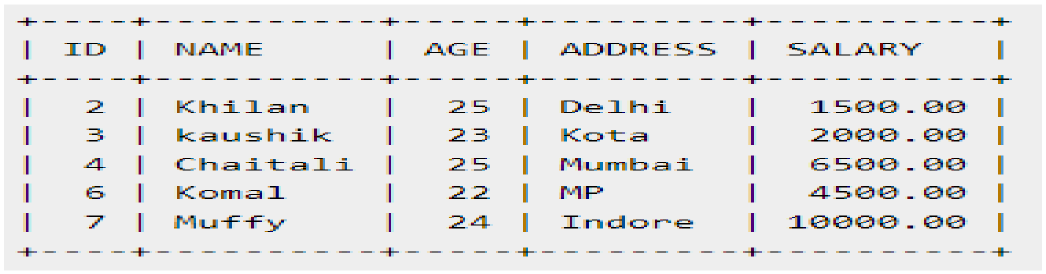
<u>Subqueries with the DELETE Statement</u>

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

Sql> delete from customers where age in (select age from customers_bkp where age >= 27);



```
+----+----------+-----+----------+----------+
| 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 |
+----+----------+-----+----------+----------+
```

➢ SQL - NULL Values:

The term **NULL** in SQL is used to specify that a data value does not exist in the database. It is not the same as an empty string or a value of zero, and it signifies the absence of a value or the unknown value of a data field.

Some common reasons why a value may be NULL −

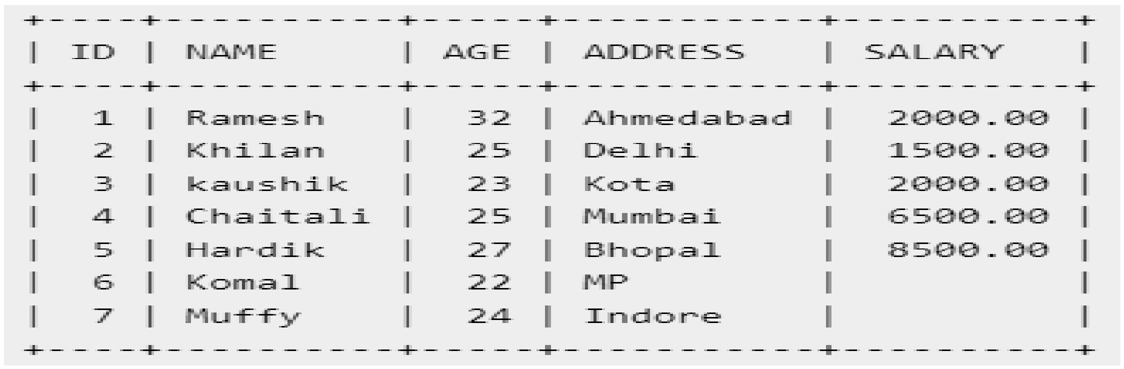
- The value may not be provided during the data entry.
- The value is not yet known.

It is important to understand that you cannot use comparison operators such as “=”, “<”, or “>” with NULL values. This is because the NULL values are unknown and could represent any value. Instead, you must use “IS NULL” or “IS NOT NULL” operators to check if a value is NULL.

Sql> create table customers(id int not null,name varchar (20) not null,age int not null,address char (25),salary decimal(18, 2),Primary key (id));

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is the one that has been left blank during the record creation.

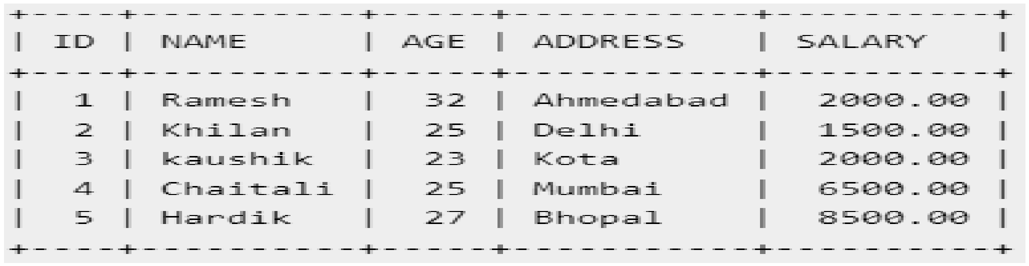


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

IS NOT NULL Query

Now, let us try to retrieve the records present in the table that are not null using the **IS NOT NULL** operator −

Sql> select id, name, age, address, salary from customers where salary is not null;

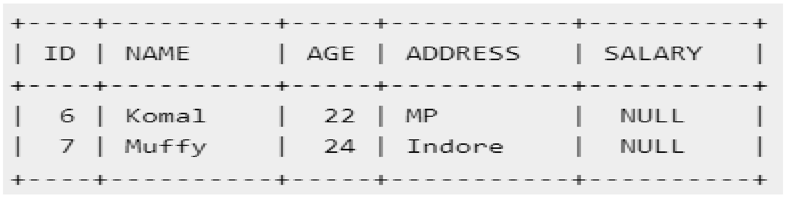


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
+----+----------+-----+-----------+----------+
```

IS NULL Query

Let us try to retrieve the records present in the table that are null using the **IS NULL** operator −

Sql> select id, name, age, address, salary from customer where salary is null;



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  6 | Komal    |  22 | MP        |  NULL    |
|  7 | Muffy    |  24 | Indore    |  NULL    |
+----+----------+-----+-----------+----------+
```

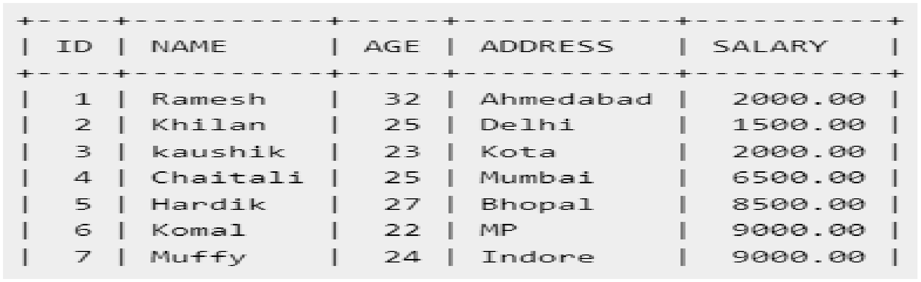
Updating NULL values in a table

You can update the NULL values present in a table using the UPDATE statement in SQL. To do so, you can use the IS NULL operator in your WHERE clause to select the rows with NULL values and then set the new value using the SET keyword.

Example

Assume the previously created table and let us try to update the NULL value(s) in the present in the table using the UPDATE statement as shown below −

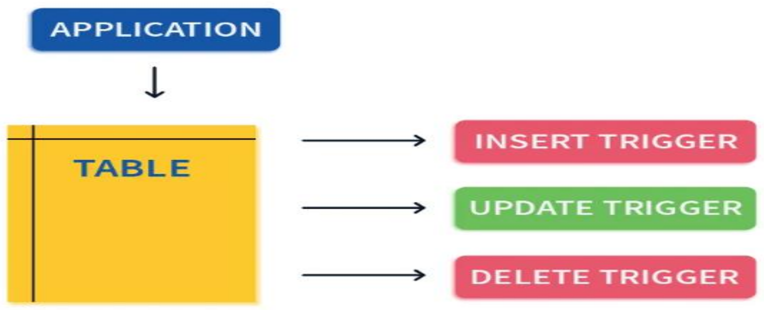
Sql> update customers set salary = 9000 where salary is null;



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | 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        |  9000.00 |
|  7 | Muffy    |  24 | Indore    |  9000.00 |
```

➢ <u>Triggers in SQL</u>

A SQL trigger is a database object which fires when an event occurs in a database. For example, a trigger can be set on a record insert in a database table.



A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA). A database with a set of associated triggers is generally called an active database.

Parts of trigger

A triggers description contains three parts, which are as follows −

- **Event** − An event is a change to the database which activates the trigger.
- **Condition** − A query that is run when the trigger is activated is called as a condition.
- **Action** −A procedure which is executed when the trigger is activated and its condition is true.

<u>Types of Triggers:</u>

The following are the different types of triggers present in SQL.

DML Triggers: These triggers fire in response to data manipulation language (DML) statements like INSERT, UPDATE, or DELETE.

After Triggers: These triggers execute after the database has processed a specified event (such as an INSERT, UPDATE, or DELETE statement). AFTER triggers are commonly used to perform additional processing or auditing tasks after a data modification has occurred.

Instead Triggers: These triggers are used for views and fire instead of the DML statement (INSERT, UPDATE, DELETE) on the view.

DDL Triggers: These triggers fire in response to data definition language (DDL) statements like CREATE, ALTER, or DROP.

LOGON Triggers: These triggers fire when a user logs into the database.

LOGOFF Triggers: These triggers fire when a user logs out of the database.

SERVERERROR Triggers: These triggers fire when a server error occurs.

Syntax of Creating Triggers in SQL

CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER | INSTEAD OF } {INSERT [OR] | UPDATE [OR] | DELETE} ON table_name [FOR EACH ROW] WHEN (condition) [trigger_body]

Query 1:

CREATE TABLE student(Id integer PRIMARY KEY, first_name varchar(50), last_name varchar(50), full_name varchar(50));

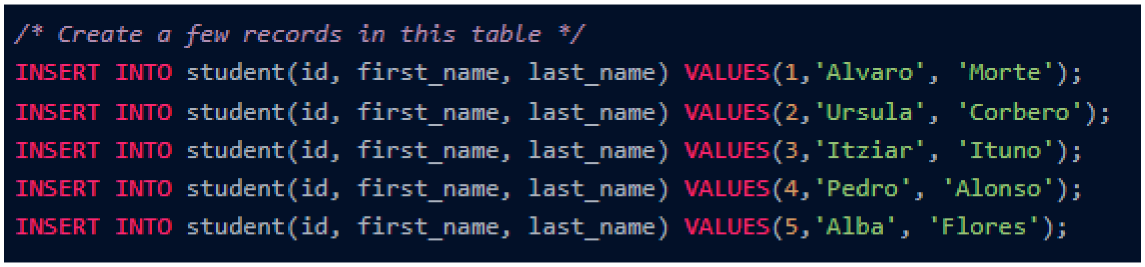
Here we will create a trigger to fill in the full name by concatenating the first and last names. So while inserting the values, we will only feed the first name and last name, and we will expect the trigger to automatically update each row with an additional column attribute bearing the full name.

Query 2:

create trigger student_name after INSERT on student for each row BEGIN UPDATE student set full_name = first_name || ' ' || last_name;

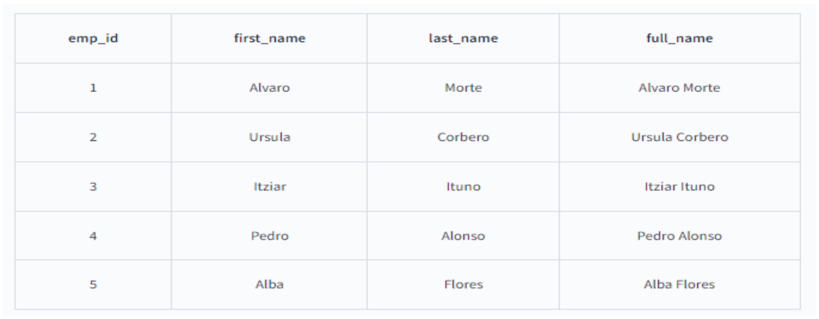
END;

Here we can understand from the trigger query we have set a trigger after an insert is made to the table student. Once the insert is done, this procedure will be fired, which will run an update command to update the students' full names.



```
/* Create a few records in this table */
INSERT INTO student(id, first_name, last_name) VALUES(1,'Alvaro', 'Morte');
INSERT INTO student(id, first_name, last_name) VALUES(2,'Ursula', 'Corbero');
INSERT INTO student(id, first_name, last_name) VALUES(3,'Itziar', 'Ituno');
INSERT INTO student(id, first_name, last_name) VALUES(4,'Pedro', 'Alonso');
INSERT INTO student(id, first_name, last_name) VALUES(5,'Alba', 'Flores');
```

Here we have inserted five students’ data, and since we have a trigger created in our system to update the full_name, we are expecting the full name to be non-empty if we run a select query on this table.



emp_id	first_name	last_name	full_name
1	Alvaro	Morte	Alvaro Morte
2	Ursula	Corbero	Ursula Corbero
3	Itziar	Ituno	Itziar Ituno
4	Pedro	Alonso	Pedro Alonso
5	Alba	Flores	Alba Flores

Here we can see since we had an update statement in the trigger procedure in query 2, the full names are automatically updated immediately after the inserts are done.

Active Databases:

An active Database is a database consisting of a set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database is activated or not, prior to executing the statement. If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement. In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as 'recursive trigger'.

Features of Active Database:

1. It possess all the concepts of a conventional database i.e. data modelling facilities, query language etc.
2. It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.
3. It supports definition and management of ECA rules.
4. It detects event occurrence.
5. It must be able to evaluate conditions and to execute actions.
6. It means that it has to implement rule execution.