What is SQL?

SQL stands for Structured Query Language. It is used for storing and managing data in RelationalDatabase Management System (RDBMS).
It is a standard language for Relational DatabaseSystem. It enables a user to create, read, updateand delete relational databases and tables.
All the RDBMS like MySQL, Informix, Oracle, MSAccess and SQL Server use SQL as their standarddatabase language.
SQL allows users to query the database in a number of ways, using English-like statements.

When an SQL command is executing for any RDBMS,then the system figure out the best way to carry out the request and the SQL engine determines that howto interpret the task.

In the process, various components are included. These components can be optimization Engine, Queryengine, Query dispatcher, classic, etc.

All the non-SQL queries are handled by the classicquery engine, but SQL query engine won't handlelogical files.

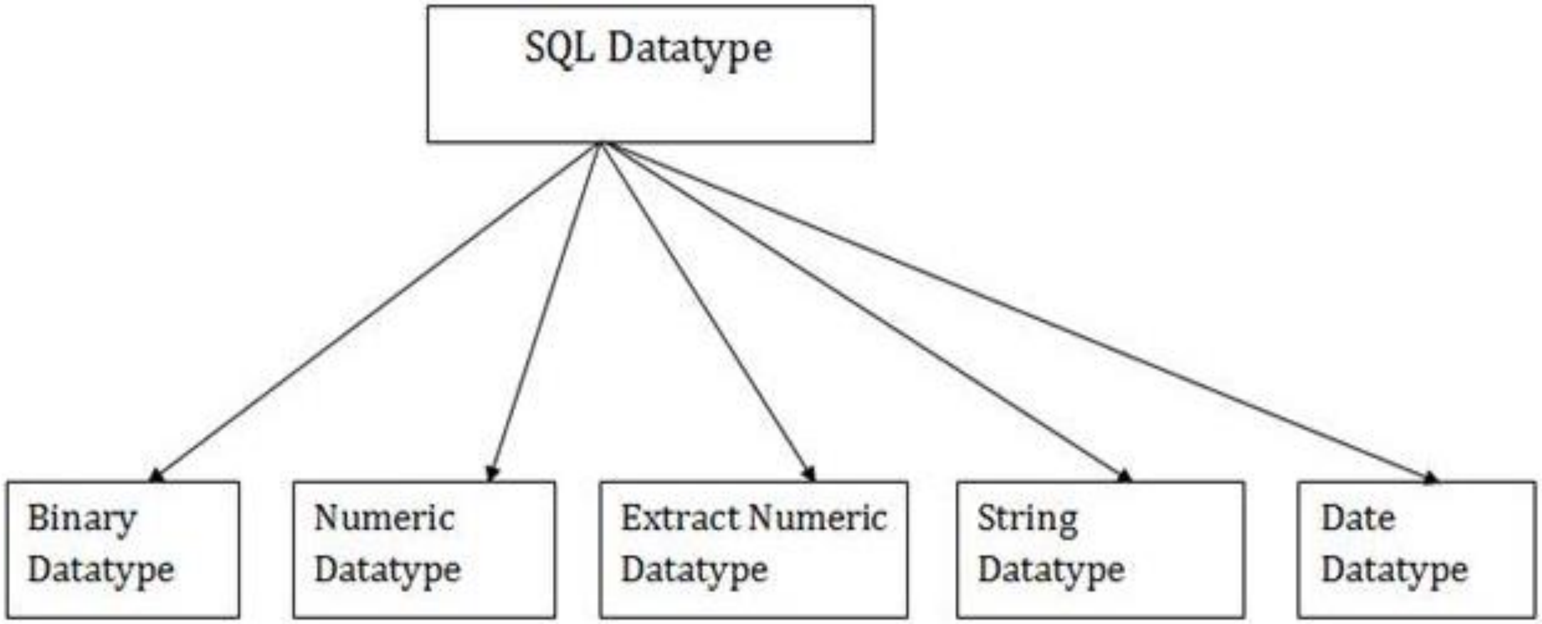
Advantages of SQL?

High speed
No coding needed
Well defined standards
Portability
Interactive language
Multiple data view

What is SQL Datatype?

SQL Datatype is used to define the values that acolumn can contain.

Every column is required to have a name and datatype in the database table.

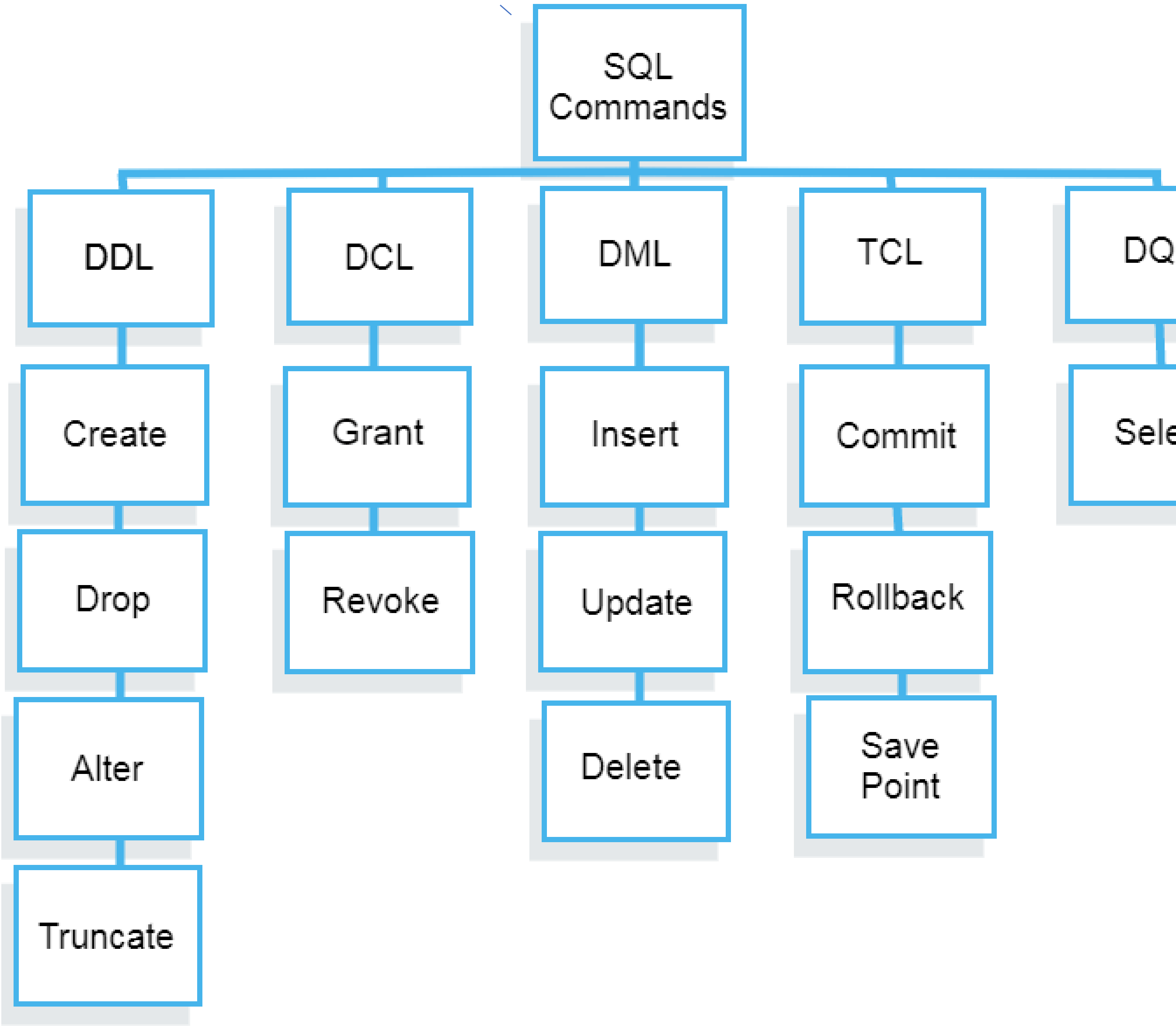


SQL Commands

SQL commands are instructions. It is used to communicate with the database. It is also used toperform 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.



What is DDL?

Data Definition Language helps you to define the database structure or schema. Let’s learn about DDL commands with syntax.

Five types of DDL commands in SQL are:

CREATE

CREATE statements is used to define the database structure schema:

Syntax:

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

For example:

Create database university;
Create table students;
Create view for_students;

DROP

Drops commands remove tables and databases from RDBMS.

Syntax

DROP TABLE ;

For example:

Drop object_type object_name;
Drop database university;
Drop table student;

ALTER

Alters command allows you to alter the structure of the database.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify an existing column in the table:

ALTER TABLE MODIFY(COLUMN DEFINITION....);

For example:

Alter table guru99 add subject varchar;

TRUNCATE:

This command used to delete all the rows from the table and free the space containing the table.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE table students;

What is Data Manipulation Language?

Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database.

There are three basic constructs which allow database program and user to enter data and information are:

Here are some important DML commands in SQL:

INSERT

UPDATE

DELETE

INSERT:

This is a statement is a SQL query. This command 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 students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');
```

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 = valueN] [WHERE CONDITION]
```

For example:

```
UPDATE students
SET FirstName = 'Jhon', LastName= 'Wick'
WHERE StudID = 3;
```

DELETE:

This command is used to remove one or more rows from a table.

Syntax:

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

For example:

```
DELETE FROM students
WHERE FirstName = 'Jhon';
```

What is DCL?

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give “rights & permissions.” Other permission controls parameters of the database system.

Examples of DCL commands:

Commands that come under DCL:

Grant

Revoke

Grant:

This command is use to give user access privileges to a database.

Syntax:

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

For example:

```
GRANT SELECT ON Users TO'Tom'@'localhost;
```

Revoke:

It is useful to back permissions from the user.

Syntax:

```
REVOKE privilege_nameON object_nameFROM {user_name |PUBLIC |role_name}
```

For example:

```
REVOKE SELECT, UPDATE ON student FROM BCA, MCA;
```

What is TCL?

Transaction control language or TCL commands deal with the transaction within the database.

Commit

This command is used to save all the transactions to the database.

Syntax:

```
Commit;
```

For example:

```
DELETE FROM Students
WHERE RollNo =25;
COMMIT;
```

Rollback

Rollback command allows you to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM Students
WHERE RollNo =25;
```

SAVEPOINT

This command helps you to sets a savepoint within a transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

Example:

```
SAVEPOINT RollNo;
```

What is DQL?

Data Query Language (DQL) is used to fetch the data from the database.

It uses only one command:

SELECT:

This command helps you to select the attribute based on the condition described by the WHERE clause.

Syntax:

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

For example:

```
SELECT FirstName
FROM Student
WHERE RollNo > 15;
```

Views in SQL

Views in SQL are considered as a virtual table. A view also contains rows and columns.
To create the view, we can select the fields from one or more tables present in the database.
A view can either have specific rows based on certain condition or all the rows of a table.

Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement. In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

DROP VIEW view_name;

Example:

If we want to delete the View **MarksView**, we can do this as:

DROP VIEW MarksView;

SQL INDEX

The Index in SQL is a special table used to speed up the searching of the data in the database tables. It also retrieves a vast amount of data from the tables frequently. The INDEX requires its own space in the hard disk.

The index concept in SQL is same as the index concept in the novel or a book.

It is the best SQL technique for improving the performance of queries. The drawback of using indexes is that they slow down the execution time of UPDATE and INSERT statements. But they have one advantage also as they speed up the execution time of SELECT and WHERE statements.

In SQL, an Index is created on the fields of the tables. We can easily build one or more indexes on a table. The creation and deletion of the Index do not affect the data of the database.

Create an INDEX

In SQL, we can easily create the Index using the following CREATE Statement:

CREATE INDEX Index_Name **ON** Table_Name (Column_Name);

Here, **Index_Name** is the name of that index that we want to create, and **Table_Name** is the name of the table on which the index is to be created. The **Column_Name** represents the name of the column on which index is to be applied.

If we want to create an index on the combination of two or more columns, then the following syntax can be used in SQL:

CREATE INDEX Index_Name **ON** Table_Name (column_name1, column_name2, ..., column_name);

Example for creating an Index in SQL:

Let's take an Employee table:

Emp_Id	Emp_Name	Emp_Salary	Emp_City	Emp
1001	Akshay	20000	Noida	U.P
1002	Ram	35000	Jaipur	Rajas
1003	Shyam	25000	Gurgaon	Harya
1004	Yatin	30000	Lucknow	U.P

The following SQL query creates an Index '**Index_state**' on **the Emp_State** column of the **Employee** table.

CREATE INDEX index_state **ON** Employee (Emp_State);

Suppose we want to create an index on the combination of the **Emp_city** and the **Emp_State** column of the above **Employee** table. For this, we have to use the following query:

CREATE INDEX index_city_State **ON** Employee (Emp_City, Emp_State);

Create UNIQUE INDEX

Unique Index is the same as the Primary key in SQL. The unique index does not allow selecting those columns which contain duplicate values.

This index is the best way to maintain the data integrity of the SQL tables.

Syntax for creating the Unique Index is as follows:

```
CREATE UNIQUE INDEX Index_Name ON Table_Name ( Column_Name);
```

Example for creating a Unique Index in SQL:

Let's take the above Employee table. The following SQL query creates the unique index **index_salary** on the **Emp_Salary** column of the **Employee** table.

```
CREATE UNIQUE INDEX index_salary ON Employee (Emp_Salary);
```

Rename an INDEX

We can easily rename the index of the table in the relational database using the ALTER command.

Syntax:

```
ALTER INDEX old_Index_Name RENAME TO new_Index_Name;
```

Example for Renaming the Index in SQL:

The following SQL query renames the index **'index_Salary'** to **'index_Employee_Salary'** of the above Employee table:

```
ALTER INDEX index_Salary RENAME TO index_Employee_Salary;
```

Remove an INDEX

An Index of the table can be easily removed from the SQL database using the DROP command. If you want to delete an index from the data dictionary, you must be the owner of the database or have the privileges for removing it.

Syntaxes for Removing an Index in MySQL database:

```
ALTER TABLE Table_Name DROP INDEX Index_Name;
```

Example for removing an Index in SQL:

Suppose we want to remove the above **'index_Salary'** from the SQL database. For this, we have to use the following SQL query:

```
DROP INDEX index_salary;
```

Alter an INDEX

An index of the table can be easily modified in the relational database using the ALTER command.

```
ALTER INDEX Index_Name ON Table_Name REBUILD;
```

What Is PL/SQL

PL/SQL is a fusion of SQL with procedural traits of programming languages. It was launched by Oracle to upgrade the features of SQL. PL SQL is considered as one of the important languages inside the Oracle database. It is primarily an extension of SQL.

This programming language was brought into the market by Oracle Corporation with the thought of extending SQL and Oracle databases. It is known as **Procedural Language extensions to the Structured Query Language**.

SQL is generally used for modifying and querying information in Relational Database Management Systems (RDBMS). PL SQL comes to plug in the shortcomings of SQL and enhances the characteristics of SQL.

Basic Syntax Of PL/SQL

PL SQL is structured in logical blocks of code. Each block has multiple subsections comprising of the following:

Declaration: This section begins with the DECLARE keyword. It is not considered as the required one and has variables, subprograms, and so on.

Executable Commands: This section begins with BEGIN and END keywords respectively. It is considered a required one and contains PL/SQL statements. It consists of at least one executable line of code.

Exception Handling: This section begins with the keyword EXCEPTION. It comprises the types of exceptions that the code will handle.

Begin: This is the keyword used for pointing to the execution block. It is required in a PL/SQL code where actual business logic is described.

End: This is the keyword used to determine the end of the block of code.

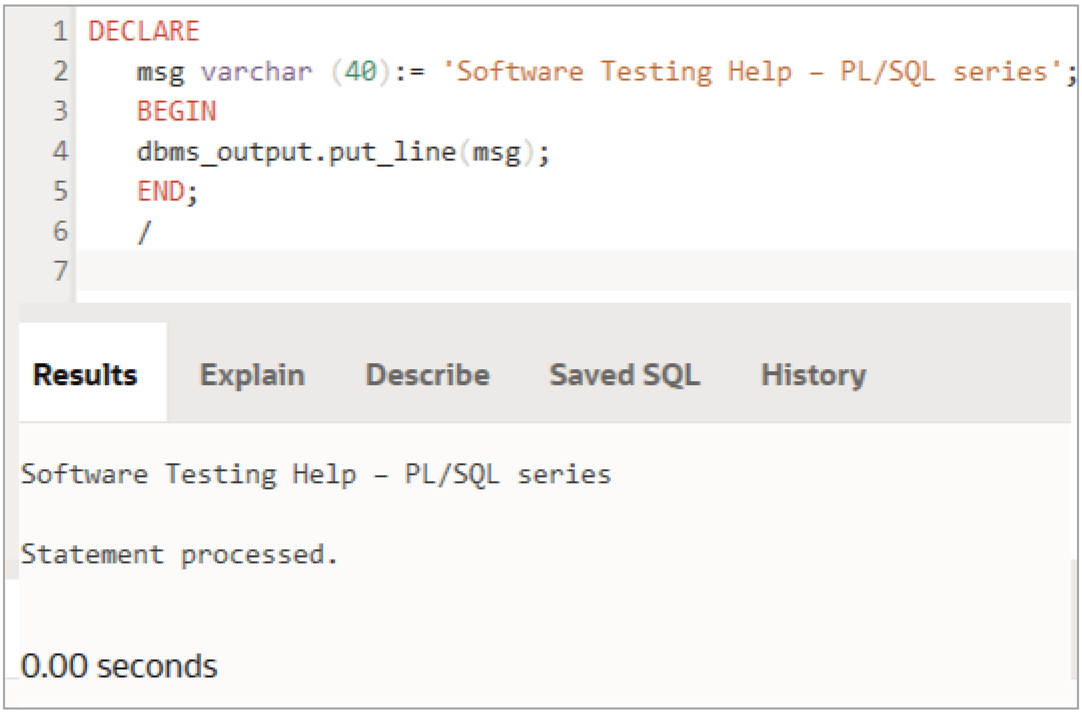
Structure of PL/SQL block:

```
[DECLARE]
<declaration statements>;
[BEGIN]
<Execution statements>;
[EXCEPTION]
<Exception statements>;
END;
```

A sample code using the above block structure is given below.

```
DECLARE
   msg varchar (40):= 'Software Testing Help – PL/SQL series';
   BEGIN
   dbms_output.put_line(msg);
   END;
/
```

Output of the above code should be.



```
DECLARE
    msg varchar (40):= 'Software Testing Help – PL/SQL series';
    BEGIN
    dbms_output.put_line(msg);
    END;
    /

```

```
Software Testing Help – PL/SQL series

Statement processed.


0.00 seconds
```

We need to add '/' at the start of the first blank line after the last code statement to execute the block of code from the SQL command line.

PL/SQL Identifiers

PL SQL identifiers include variables, constants, procedures, cursors, and so on. Their length should not be more than thirty characters and is case insensitive. A keyword in PLSQL cannot be used as an identifier.

PL/SQL Delimiters

These are basically symbols having certain characteristics. Some of the common delimiters are +, -, @, =, ||, <<>>, (,), –, <, >, <=, >=, %. **There are two types of delimiters:** simple and compound symbols.

Simple symbols are enlisted in the table below:

Sl. No.	Simple Symbols	Significance
1	.	Component selector
2	/	Operator division
3	*	Operator multiplication
4	-	Operator negation
5	+	Operator addition
6	;	End of statement
7	@	Remote access indicator
8	>	Greater than
9	<	Lesser than
10	=	Relational operator
11	"	Quoted identifier
12	,	Item separator
13	(	List delimiter
14	)	List delimiter
15	:	Host variable indicator
16	%	Attribute indicator
17	'	Delimiter for character string

Compound symbols are enlisted in the table below:

Sl. No.	Compound Symbols	Significance

Sl. No.	Compound Symbols	Significance
1	| |	Operator for concatenation
2	* *	Operator for exponentiation
3	<<	Delimiter begin
4	>>	Delimiter end
5	=>	Operator for association
6	:=	Operator for assignment
7	..	Operator for range
8	/*	multi-line comment indicator for begin
9	*/	multi-line comment indicator for end
10	<>	Not equality operator
11	>=	Greater than equal to operator
12	<=	Less than equal to operator
13	!=	Not equality operator
14	~=	Not equality operator
15	^=	Not equality operator
16	- -	Single line comment delimiter

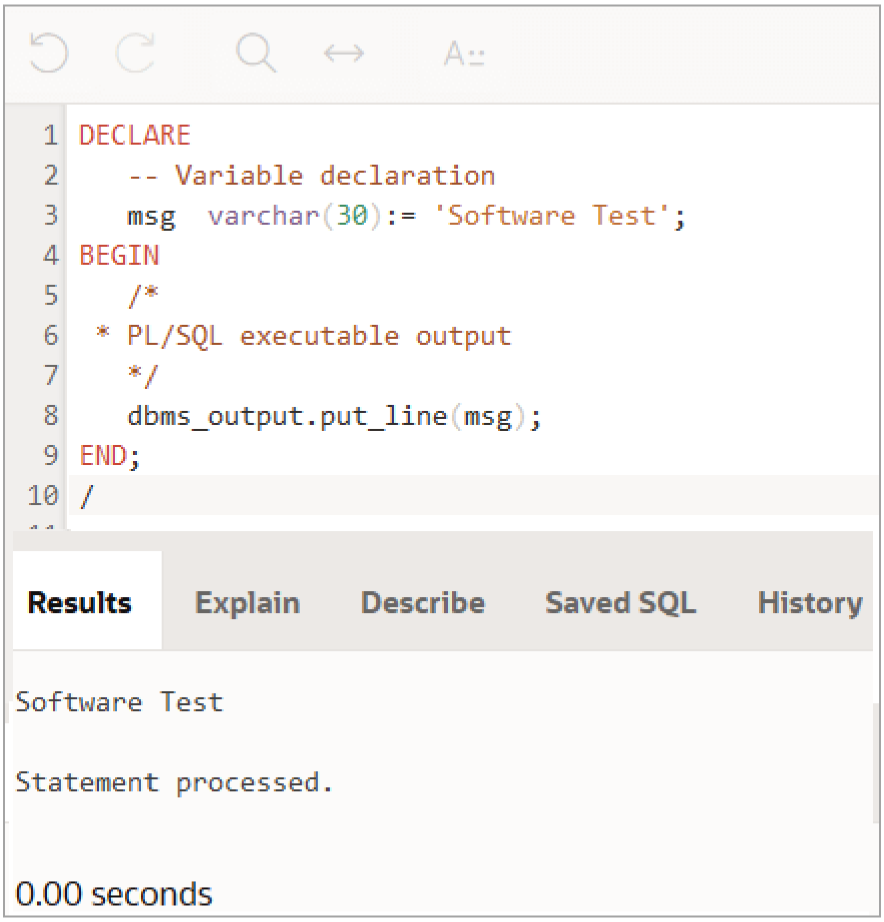
PL/SQL Comments

PLSQL code includes comments that explain the intent of the code. PL/SQL has both multiple lines and single-line comments. The single-line comments begin with delimiter double hyphen — and double line comments start with /* and end with */.

Sample Code snippet is given below:

```
DECLARE
  -- Variable declaration
  msg varchar(30):= 'Software Test';
BEGIN
  /*
  * PL/SQL executable output
  */
  dbms_output.put_line(msg);
END;
/
```

The output of the above code should be:



```
DECLARE
   -- Variable declaration
   msg  varchar(30):= 'Software Test';
BEGIN
   /*
 * PL/SQL executable output
   */
   dbms_output.put_line(msg);
END;
/
```

```
Software Test

Statement processed.
```

0.00 seconds

MySQL Cursor

In MySQL, Cursor can also be created. Following are the steps for creating a cursor.

1. Declare Cursor

A cursor is a select statement, defined in the declaration section in MySQL.

Syntax

DECLARE cursor_name **CURSOR FOR**
Select statement;

Parameter:

cursor_name: name of the cursor

select_statement: select query associated with the cursor

2. Open Cursor

After declaring the cursor the next step is to open the cursor using open statement.

Syntax

Open cursor_name;

Parameter:

cursor_name: name of the cursor which is already declared.

3. Fetch Cursor

After declaring and opening the cursor, the next step is to fetch the cursor. It is used to fetch the row or the column.

Syntax

FETCH [**NEXT** [**FROM**]] cursor_name **INTO** variable_list;

Parameter:

cursor_name: name of the cursor

variable_list: variables, comma separated, etc. is stored in a cursor for the result set

4. Close Cursor

The final step is to close the cursor.

Syntax

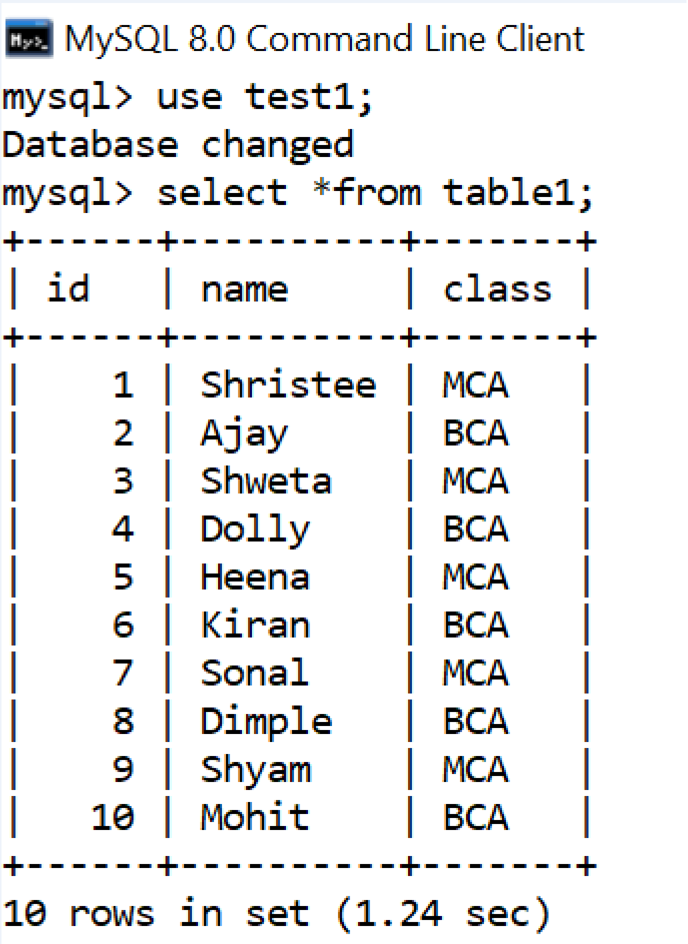
Close cursor_name;

Parameter:

Cursor_name: name of the cursor

Example for the cursor:

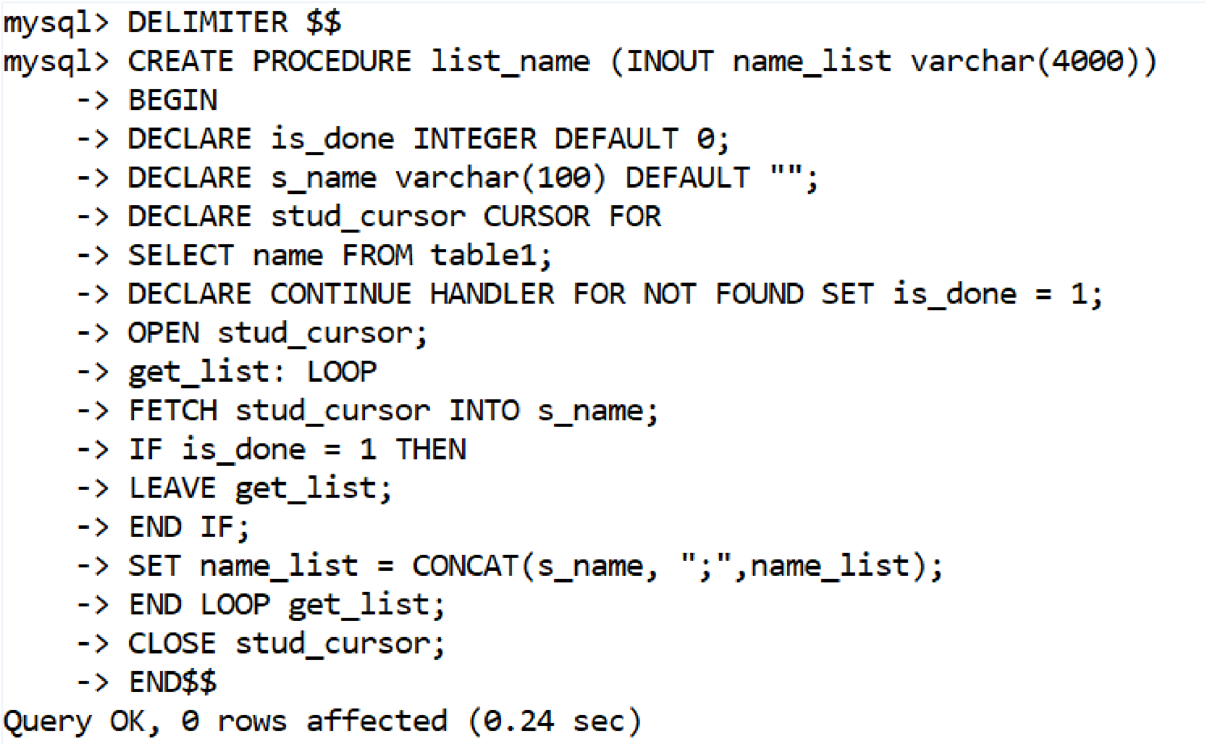
Step 1: Open the database and table.



```
mysql> use test1;
Database changed
mysql> select *from table1;
+------+----------+-------+
| id   | name     | class |
+------+----------+-------+
|    1 | Shristee | MCA   |
|    2 | Ajay     | BCA   |
|    3 | Shweta   | MCA   |
|    4 | Dolly    | BCA   |
|    5 | Heena    | MCA   |
|    6 | Kiran    | BCA   |
|    7 | Sonal    | MCA   |
|    8 | Dimple   | BCA   |
|    9 | Shyam    | MCA   |
|   10 | Mohit    | BCA   |
+------+----------+-------+
10 rows in set (1.24 sec)
```

Step 2: Now create the cursor.

Query:

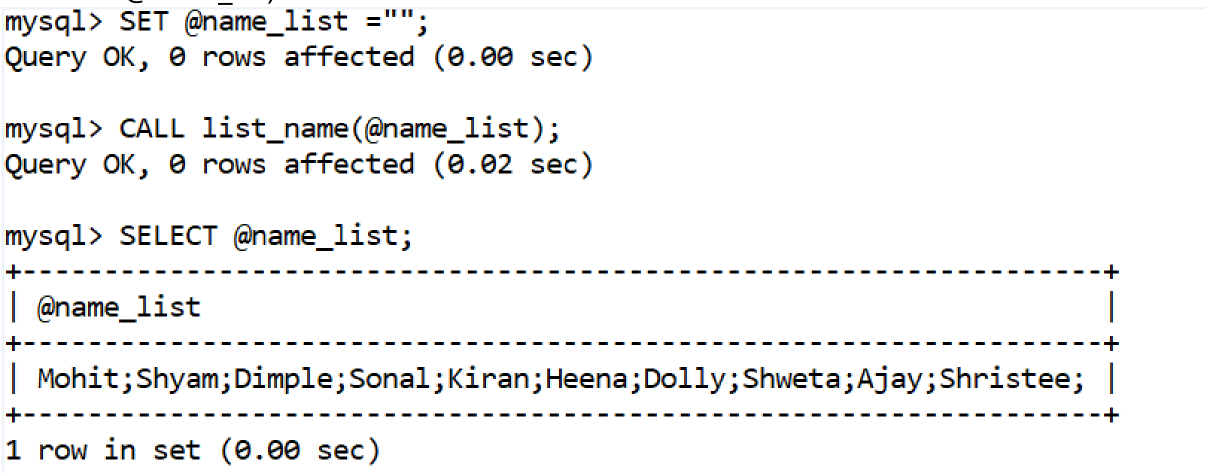


```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE list_name (INOUT name_list varchar(4000))
    -> BEGIN
    -> DECLARE is_done INTEGER DEFAULT 0;
    -> DECLARE s_name varchar(100) DEFAULT "";
    -> DECLARE stud_cursor CURSOR FOR
    -> SELECT name FROM table1;
    -> DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_done = 1;
    -> OPEN stud_cursor;
    -> get_list: LOOP
    -> FETCH stud_cursor INTO s_name;
    -> IF is_done = 1 THEN
    -> LEAVE get_list;
    -> END IF;
    -> SET name_list = CONCAT(s_name, ";",name_list);
    -> END LOOP get_list;
    -> CLOSE stud_cursor;
    -> END$$
Query OK, 0 rows affected (0.24 sec)
```

Step 3: Now call the cursor.

Query:

SET @name_list ="";
CALL list_name(@name_list);
SELECT @name_list;



```
mysql> SET @name_list ="";
Query OK, 0 rows affected (0.00 sec)

mysql> CALL list_name(@name_list);
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @name_list;
+-------------------------------------------------------------+
| @name_list                                                  |
+-------------------------------------------------------------+
| Mohit;Shyam;Dimple;Sonal;Kiran;Heena;Dolly;Shweta;Ajay;Shristee; |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

MySQL Trigger

The following naming convention should be used to name the trigger in MySQL:
(BEFOR | **AFTER**) table_name (**INSERT** | **UPDATE** | **DELETE**)
Thus,
Trigger Activation Time: BEFORE | AFTER
Trigger Event: INSERT | UPDATE | DELETE

How to create triggers in MySQL?

We can use the **CREATE TRIGGER** statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:

```
CREATE TRIGGER trigger_name
   (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
       ON table_name FOR EACH ROW
```

```
 BEGIN
--variable declarations
--trigger code
END;
```

MySQL Create Trigger

In this article, we are going to learn how to create the first trigger in MySQL. We can create a new trigger in MySQL by using the CREATE TRIGGER statement. It is to ensure that we have trigger privileges while using the CREATE TRIGGER command. The following is the basic syntax to create a trigger:

```
CREATE TRIGGER trigger_name  trigger_time trigger_event
ON table_name FOR EACH ROW
BEGIN
   --variable declarations
   --trigger code
END;
```

Parameter Explanation

The create trigger syntax contains the following parameters:

trigger_name: It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement. It is to make sure that the trigger name should be unique within the schema.

trigger_time: It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger. It indicates that the trigger will be invoked before or after each row modification occurs on the table.

trigger_event: It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation. The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.

table_name: It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

BEGIN END Block: Finally, we will specify the statement for execution when the trigger is activated. If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger.

The trigger body can access the column's values, which are affected by the DML statement.

The **NEW** and **OLD** modifiers are used to distinguish the column values **BEFORE** and **AFTER** the execution of the DML statement. We can use the column name with NEW and OLD modifiers as **OLD.col_name** and **NEW.col_name**. The OLD.column_name indicates the column of an existing row before the updation or deletion occurs. NEW.col_name indicates the column of a new row that will be inserted or an existing row after it is updated.

For example, suppose we want to update the column name **message_info** using the trigger. In the trigger body, we can access the column value before the update as **OLD.message_info** and the new value **NEW.message_info**.

We can understand the availability of OLD and NEW modifiers with the below table:

Trigger Event	OLD	NEW
INSERT	No	Yes

UPDATE	Yes	Yes
ELETE	Yes	No

MySQL Trigger Example

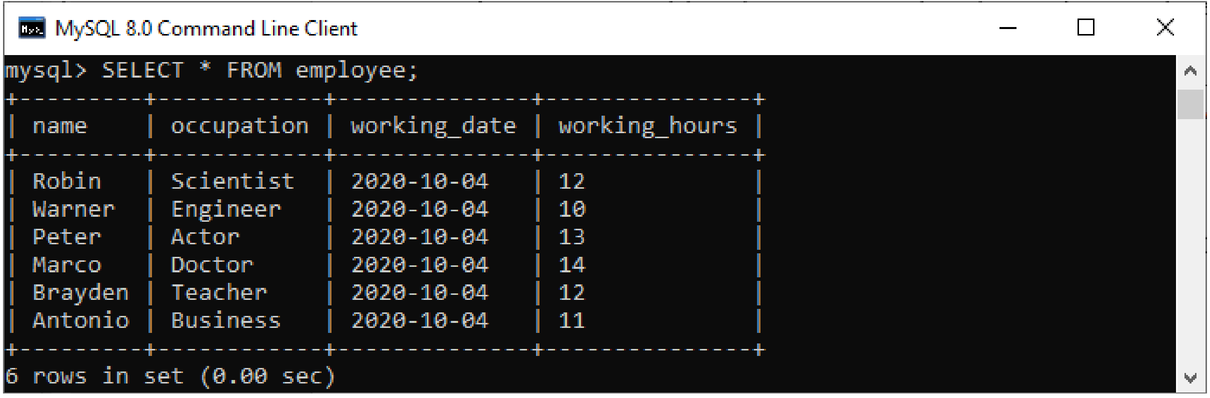
Let us start creating a trigger in MySQL that makes modifications in the employee table. First, we will create a new table named **employee** by executing the below statement:

```
CREATE TABLE employee(
  name varchar(45) NOT NULL,
  occupation varchar(35) NOT NULL,
  working_date date,
  working_hours varchar(10)
);
```

Next, execute the below statement to **fill the records** into the employee table:

```
INSERT INTO employee VALUES
('Robin', 'Scientist', '2020-10-04', 12),
('Warner', 'Engineer', '2020-10-04', 10),
('Peter', 'Actor', '2020-10-04', 13),
('Marco', 'Doctor', '2020-10-04', 14),
('Brayden', 'Teacher', '2020-10-04', 12),
('Antonio', 'Business', '2020-10-04', 11);
```

Next, execute the SELECT statement to verify the inserted record:

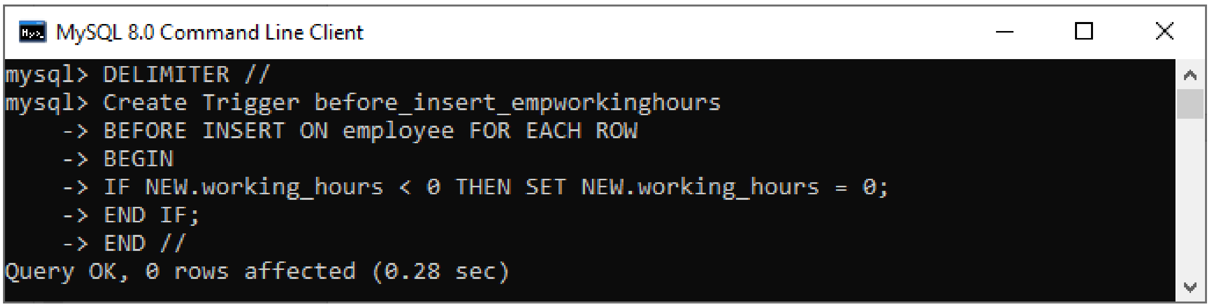


```
mysql> SELECT * FROM employee;
+---------+------------+--------------+---------------+
| name    | occupation | working_date | working_hours |
+---------+------------+--------------+---------------+
| Robin   | Scientist  | 2020-10-04   | 12            |
| Warner  | Engineer   | 2020-10-04   | 10            |
| Peter   | Actor      | 2020-10-04   | 13            |
| Marco   | Doctor     | 2020-10-04   | 14            |
| Brayden | Teacher    | 2020-10-04   | 12            |
| Antonio | Business   | 2020-10-04   | 11            |
+---------+------------+--------------+---------------+
6 rows in set (0.00 sec)
```

Next, we will create a BEFORE INSERT trigger. This trigger is invoked automatically insert the **working_hours** = **0** if someone tries to insert **working_hours** < **0**.

```
mysql> DELIMITER //
mysql> Create Trigger before_insert_empworkinghours
BEFORE INSERT ON employee FOR EACH ROW
BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
END IF;
END //
```

If the trigger is created successfully, we will get the output as follows:



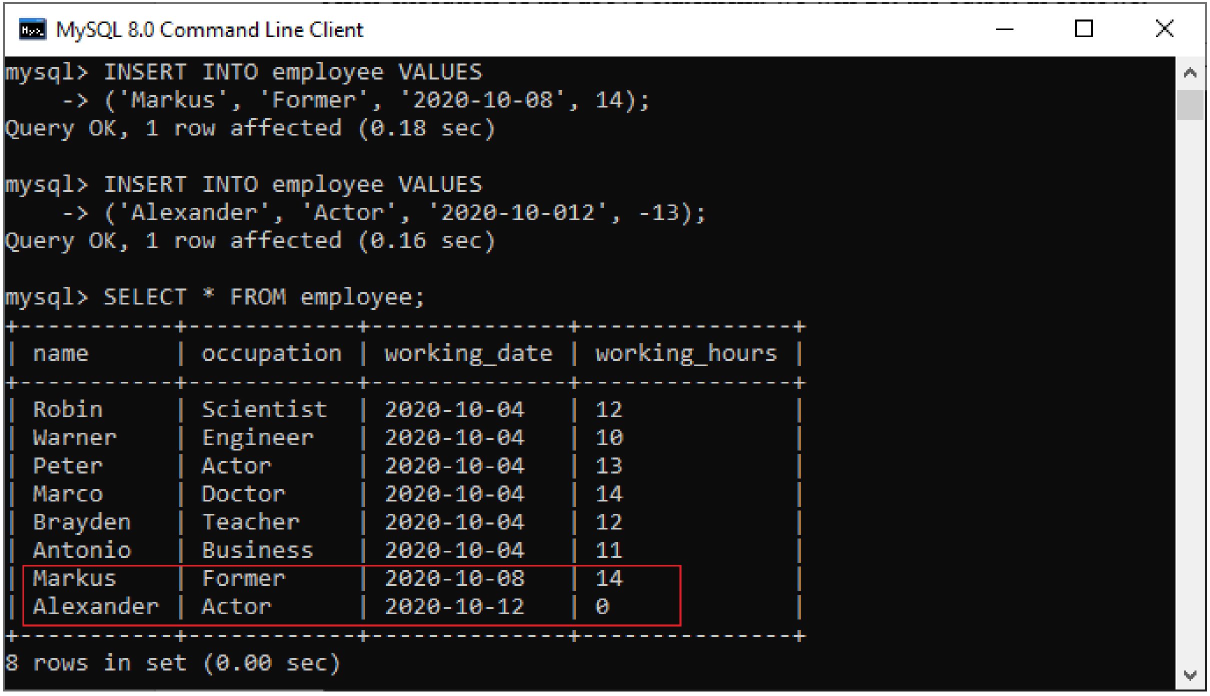
```
mysql> DELIMITER //
mysql> Create Trigger before_insert_empworkinghours
    -> BEFORE INSERT ON employee FOR EACH ROW
    -> BEGIN
    -> IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
    -> END IF;
    -> END //
Query OK, 0 rows affected (0.28 sec)
```

Now, we can use the following statements to invoke this trigger:

```
mysql> INSERT INTO employee VALUES
('Markus', 'Former', '2020-10-08', 14);

mysql> INSERT INTO employee VALUES
('Alexander', 'Actor', '2020-10-012', -13);
```

After execution of the above statement, we will get the output as follows:



```
mysql> INSERT INTO employee VALUES
    -> ('Markus', 'Former', '2020-10-08', 14);
Query OK, 1 row affected (0.18 sec)

mysql> INSERT INTO employee VALUES
    -> ('Alexander', 'Actor', '2020-10-012', -13);
Query OK, 1 row affected (0.16 sec)

mysql> SELECT * FROM employee;
+-----------+------------+--------------+---------------+
| name      | occupation | working_date | working_hours |
+-----------+------------+--------------+---------------+
| Robin     | Scientist  | 2020-10-04   | 12            |
| Warner    | Engineer   | 2020-10-04   | 10            |
| Peter     | Actor      | 2020-10-04   | 13            |
| Marco     | Doctor     | 2020-10-04   | 14            |
| Brayden   | Teacher    | 2020-10-04   | 12            |
| Antonio   | Business   | 2020-10-04   | 11            |
| Markus    | Former     | 2020-10-08   | 14            |
| Alexander | Actor      | 2020-10-12   | 0             |
+-----------+------------+--------------+---------------+
8 rows in set (0.00 sec)
```

In this output, we can see that on inserting the negative values into the working_hours column of the table will automatically fill the zero value by a trigger.