



College Code : SDES

SREE DATTHA INSTITUTE OF ENGINEERING & SCIENCE
(Autonomous, NAAC A+, NBA Accredited, Affiliated to JNTUH, Approved by AICTE, New Delhi)
Nagarjuna Sagar Road, Sheriguda (V), Ibrahimpatnam (M), R.R. Dist., Greater Hyderabad, Telangana-501510.
Website: www.sreedattha.ac.in Email: info@sreedattha.ac.in

Department of Computer Science and Engineering



LAB Notes

on

Database Management System Laboratory

2nd Year 2nd Semester

by

Prof. Puneet Shetteppanavar

Assistant Professor, Computer Science and Engineering Department,
Sree Dattha Institute of Engineering and Science, Sheriguda, Hyderabad - 501510

DATABASE MANAGEMENT SYSTEMS LAB

B.Tech. II Year II Sem.

| L | T | P | C |
|---|---|---|---|
| 0 | 0 | 2 | 1 |

Co-requisites: "Database Management Systems"

Course Objectives:

- Introduce ER data model, database design and normalization
- Learn SQL basics for data definition and data manipulation

Course Outcomes:

- Design database schema for a given application and apply normalization
- Acquire skills in using SQL commands for data definition and data manipulation.
- Develop solutions for database applications using procedures, cursors and triggers

List of Experiments:

1. Concept design with E-R Model
2. Relational Model
3. Normalization
4. Practicing DDL commands
5. Practicing DML commands
6. A. Querying (using ANY, ALL, UNION, INTERSECT, JOIN, Constraints etc.)
B. **Nested, Correlated subqueries**
7. Queries using Aggregate functions, GROUP BY, HAVING and Creation and dropping of Views.
8. Triggers (Creation of insert trigger, delete trigger, update trigger)
9. Procedures
10. Usage of Cursors

TEXT BOOKS:

1. Database Management Systems, Raghurama Krishnan, Johannes Gehrke, Tata Mc Graw Hill, 3rd Edition
2. Database System Concepts, Silberschatz, Korth, McGraw Hill, V edition.

REFERENCE BOOKS:

1. Database Systems design, Implementation, and Management, Peter Rob & Carlos Coronel 7th Edition.
2. Fundamentals of Database Systems, Elmasri Navrate, Pearson Education
3. Introduction to Database Systems, C.J. Date, Pearson Education
4. Oracle for Professionals, The X Team, S. Shah and V. Shah, SPD.
5. Database Systems Using Oracle: A Simplified guide to SQL and PL/SQL, Shah, PHI.
6. Fundamentals of Database Management Systems, M. L. Gillenson, Wiley Student Edition.

Note: {Previous Hand Written Notes will be Added Soon}

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 |
| DELETE | Yes | No |

MySQL Trigger Example

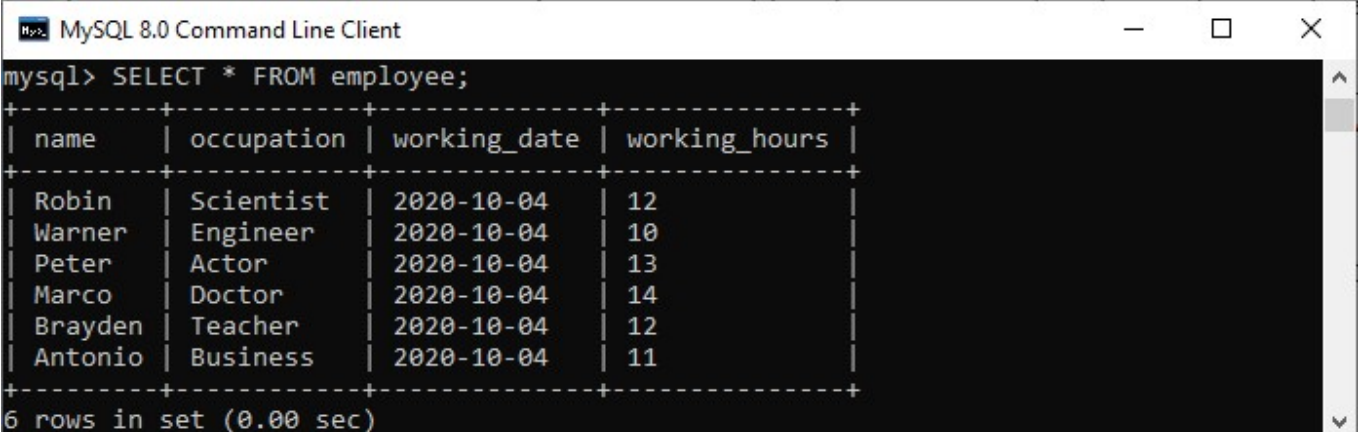
Let us start creating a trigger in [MySQL](#) that makes modifications in the employee table. First, we will create a new table named **employee** by executing the below statement:

```
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:



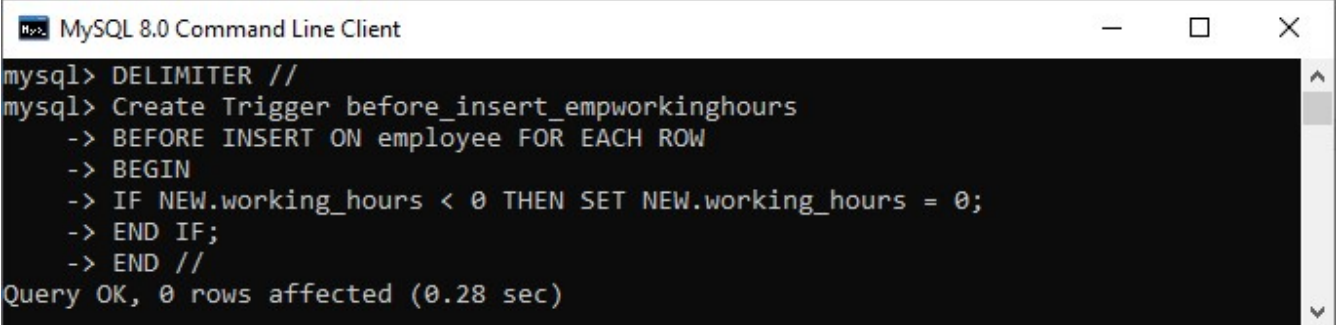
The screenshot shows a terminal window titled "MySQL 8.0 Command Line Client". The prompt is "mysql>". The command entered is "SELECT * FROM employee;". The output is a table with 4 columns: name, occupation, working_date, and working_hours. There are 6 rows of data. At the bottom, it says "6 rows in set (0.00 sec)".

| 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 |

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:



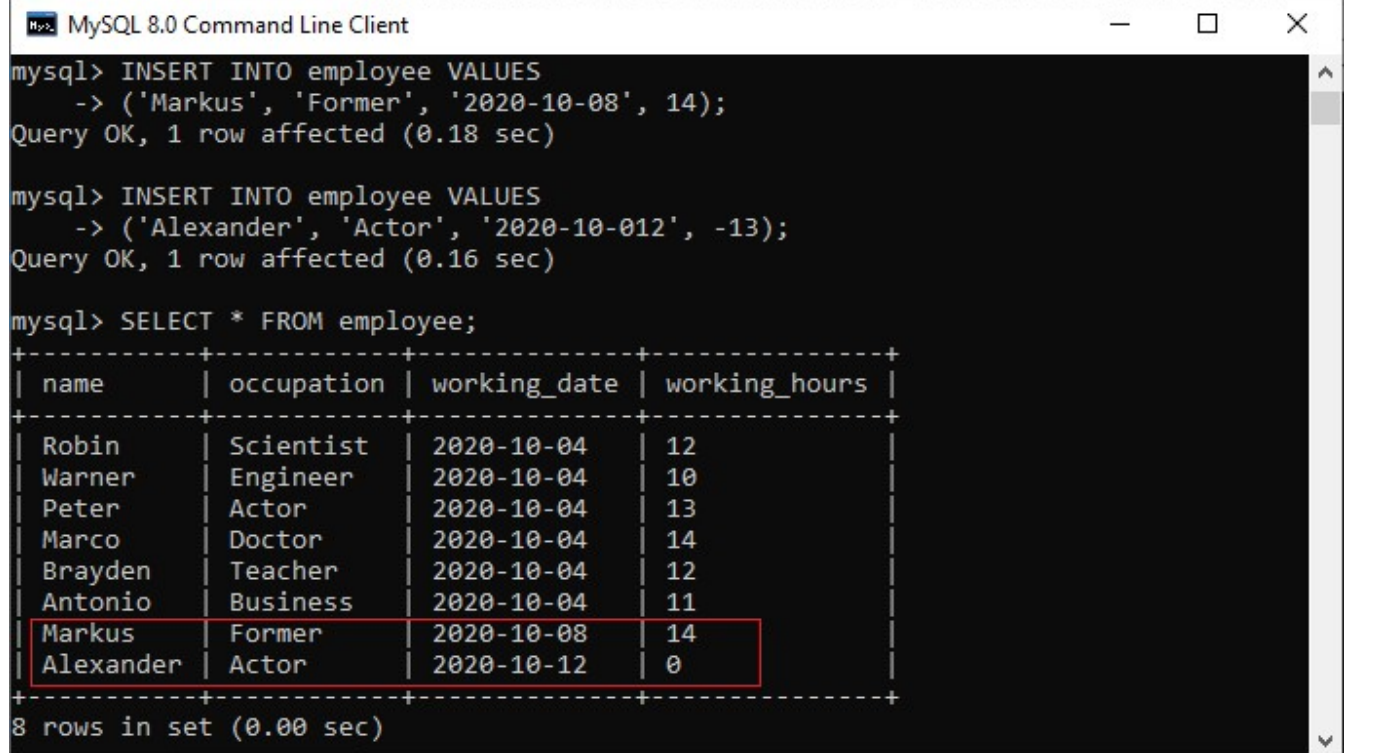
```
MySQL 8.0 Command Line Client  
mysql> DELIMITER //  
mysql> Create Trigger before_insert_empworkinghours  
-> BEFORE INSERT ON employee FOR EACH ROW  
-> BEGIN  
-> IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;  
-> END IF;  
-> END //  
Query OK, 0 rows affected (0.28 sec)
```

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.

MySQL Show/List Triggers

The show or list trigger is much needed when we have many databases that contain various tables. Sometimes we have the same trigger names in many databases; this query plays an important role in that case. We can get the trigger information in the database server using the below statement. This statement returns all triggers in all databases:

```
mysql> SHOW TRIGGERS;
```

The following steps are necessary to get the list of all triggers:

Step 1: Open the [MySQL](#) Command prompt and logged into the database server using the password that you have created during [MySQL's installation](#). After a successful connection, we can execute all the [SQL](#) statements.

Step 2: Next, choose the specific database by using the command below:

1. `mysql> USE database_name;`

Step 3: Finally, execute the SHOW TRIGGERS command.

Let us understand it with the example given below. Suppose we have a database name "**mysqltestdb**" that contains many tables. Then execute the below statement to list the [triggers](#):

1. `mysql> USE mysqltestdb;`
2. `mysql> SHOW TRIGGERS;`

The following output explains it more clearly:

```

MySQL 8.0 Command Line Client
mysql> USE mysqltestdb;
Database changed
mysql> SHOW TRIGGERS;
+-----+-----+-----+-----+
| Trigger          | Event | Table | Statement |
| Timing | Created | sql_mode |
| character_set_client | collation_connection | Database Collation |
+-----+-----+-----+-----+
| before_insert_empworkinghours | INSERT | employee | BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
END IF;
END | BEFORE | 2020-11-13 14:49:05.83 | STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION | root@loc
| cp850_general_ci | utf8mb4_0900_ai_ci |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

If we want to show or list the trigger information in a specific database from the current database without switching, MySQL allows us to use the **FROM** or **IN** clause, followed by the database name. The following statement explains it more clearly:

1. mysql> SHOW TABLES IN database_name;

The above statement can also be written as:

1. mysql> SHOW TABLES **FROM** database_name;

When we execute the above statements, we will get the same result.

MySQL DROP Trigger

We can drop/delete/remove a trigger in MySQL using the **DROP TRIGGER** statement. You must be very careful while removing a trigger from the table. Because once we have deleted the trigger, it cannot be recovered. If a trigger is not found, the DROP TRIGGER statement throws an error.

MySQL allows us to drop/delete/remove a trigger mainly in two ways:

1. MySQL Command Line Client
2. MySQL Workbench

MySQL Command Line Client

We can drop an existing trigger from the database by using the DROP TRIGGER statement with the below syntax:

DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;

Parameter Explanation

The parameters used in the drop trigger syntax are explained as follows:

| Parameter | Descriptions |
|--------------|---|
| Trigger_name | It is the name of a trigger that we want to remove from the database server. It is a required parameter. |
| Schema_name | It is the database name to which the trigger belongs. If we skip this parameter, the statement will remove the trigger from the current database. |
| IF_EXISTS | It is an optional parameter that conditionally removes triggers only if they exist on the database server. |

If we remove the trigger that does not exist, we will get an error. However, if we have specified the IF EXISTS clause, [MySQL](#) gives a **NOTE** instead of an error.

It is to note that we must have TRIGGER privileges before executing the DROP TRIGGER statement for the table associated with the trigger. Also, removing a table will automatically delete all triggers associated with the table.

MySQL DROP Trigger Example

Let us see how we can drop the trigger associated with the table through an example. So first, we will **display all triggers** available in the selected database using the below statement:

```
mysql> SHOW TRIGGERS IN employeedb;
```

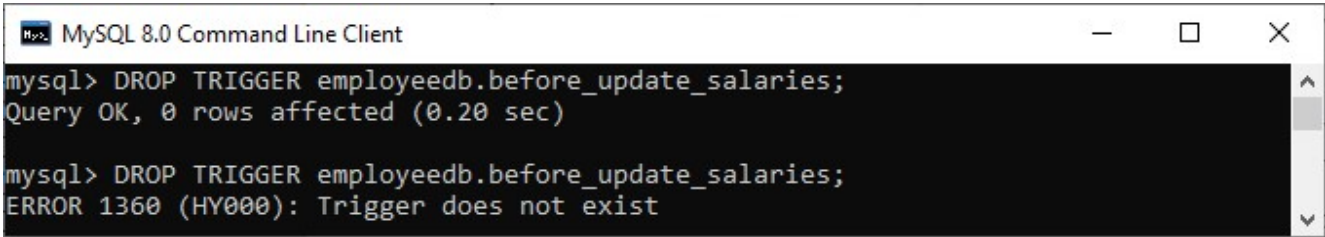
After executing the statement, we can see that there are two triggers named **before_update_salaries** and **sales_info_before_update**. See the below image:

| Trigger | Event | Table | Statement | Timing | Created | sql_mode | Definer |
|--------------------------|--------|----------------|---------------------------------|--------|--------------|---------------|--------------|
| before_update_salaries | UPDATE | salary_account | BEGIN IF new.amount < old.a... | BEFORE | 2020-11-2... | STRICT_TRA... | root@localho |
| sales_info_BEFORE_UPDATE | UPDATE | sales_info | BEGIN DECLARE error_msg VARC... | BEFORE | 2020-11-2... | STRICT_TRA... | root@localho |

If we want to remove the **before_update_salaries** trigger, execute the below statement:

```
mysql> DROP TRIGGER employeedb.before_update_salaries;
```

It will successfully delete a trigger from the database. If we execute the above statement again, it will return an error message. See the output:



```
MySQL 8.0 Command Line Client
mysql> DROP TRIGGER employeedb.before_update_salaries;
Query OK, 0 rows affected (0.20 sec)

mysql> DROP TRIGGER employeedb.before_update_salaries;
ERROR 1360 (HY000): Trigger does not exist
```

If we execute the above statement again with an **IF EXISTS** clause, it will return the warning message instead of producing an error. See the output:

```
mysql> DROP TRIGGER IF EXISTS employeedb.before_update_salaries;
```

```
MySQL 8.0 Command Line Client
mysql> DROP TRIGGER IF EXISTS employeeedb.before_update_salaries;
Query OK, 0 rows affected, 1 warning (0.12 sec)
```

We can execute the **SHOW WARNING** statement that generates a **NOTE** for a non-existent trigger when using IF EXISTS. See the output:

```
MySQL 8.0 Command Line Client
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                |
+-----+-----+-----+
| Note  | 1360 | Trigger does not exist |
+-----+-----+-----+
1 row in set (0.00 sec)
```