Aim: SQL Triggers.

Objectives:

The objectives for learning MySQL triggers are to understand how triggers automatically perform actions when data changes, such as during INSERT, UPDATE, or DELETE events. Triggers are used for tasks like logging changes or updating related tables. The goal is to learn how to use triggers effectively without causing performance issues and ensure data consistency.

Tools Used:

- MySQL Workbench

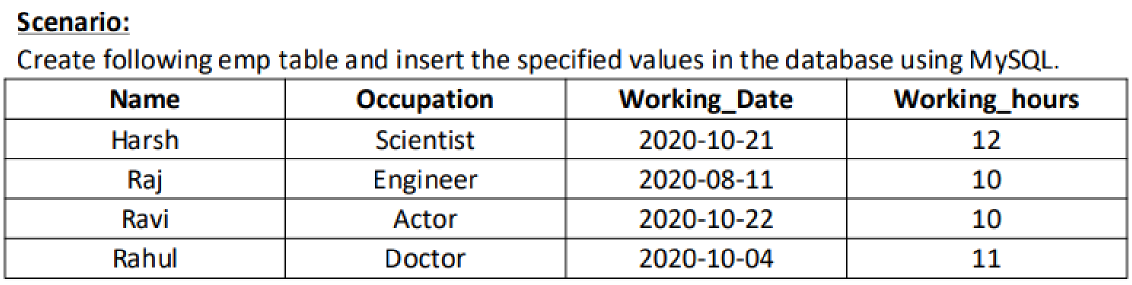
Concept:

Triggers in SQL: Triggers are special types of stored procedures that automatically execute predefined actions when certain events (like INSERT, UPDATE, or DELETE) occur in a table. They help automate tasks such as data validation, updating related tables, or logging changes, ensuring data consistency and reducing the need for manual interventions.

Example:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `emp_BEFORE_INSERT` BEFORE INSERT
ON `emp` FOR EACH ROW BEGIN

if new.Working_hours < 0

then

set new.Working_hours = 0;

end if;

END
```

Problem Statement:



Scenario:

Create following emp table and insert the specified values in the database using MySQL.

Name	Occupation	Working_Date	Working_hours
Harsh	Scientist	2020-10-21	12
Raj	Engineer	2020-08-11	10
Ravi	Actor	2020-10-22	10
Rahul	Doctor	2020-10-04	11

1) Question on Before Insert Trigger:

Write a trigger which ensures that if user enters negative value in Working_hours the value is set to 0. 2) Question on After Insert Trigger:

Create a table emp_audit(name, audit_description)

Create a trigger to make sure If any employee information is inserted in emp table then trigger is inserting the row in emp_audit table automatically. Output should look like:

Name	Audit_description
Arti	A row has been inserted in emp table at 2020-01-23 at 11:23:45 PM

3) Question on Before Update Trigger:

Create a trigger if a new working date is greater than today's date to raise an error message.

4) Question on After Update Trigger:

Create a table EmpChanges(Name, New Occupation, Old Occupation,Updatedate as shown in following output. Create a trigger that will keep history of changes in the EmpChange table when you change data in Emp table. Output should look like:

Name	New Occupation	Old Occupation	Updatedat
Harsh	Professor	Scientist	2020-01-23 at 11:23:45 PM

5) Question on Before Delete Trigger:

Create a table Emp_archeives (Name,Occupation,Working_date,WorkingHours, Deletedat)

Create trigger to ensure before removing data from Emp table, the record should be entered in Emp_archieves table.

6) Question on After Delete Trigger:

Consider you have two tables Emp Table(Original Table) and Total_working_hours_table which looks like

Total
43

Create a trigger that changes the Total of above table when Emp leaves the company.

Solution:

1)

CREATE DEFINER=`root`@`localhost` TRIGGER `emp_BEFORE_INSERT` BEFORE INSERT ON `emp` FOR EACH ROW BEGIN

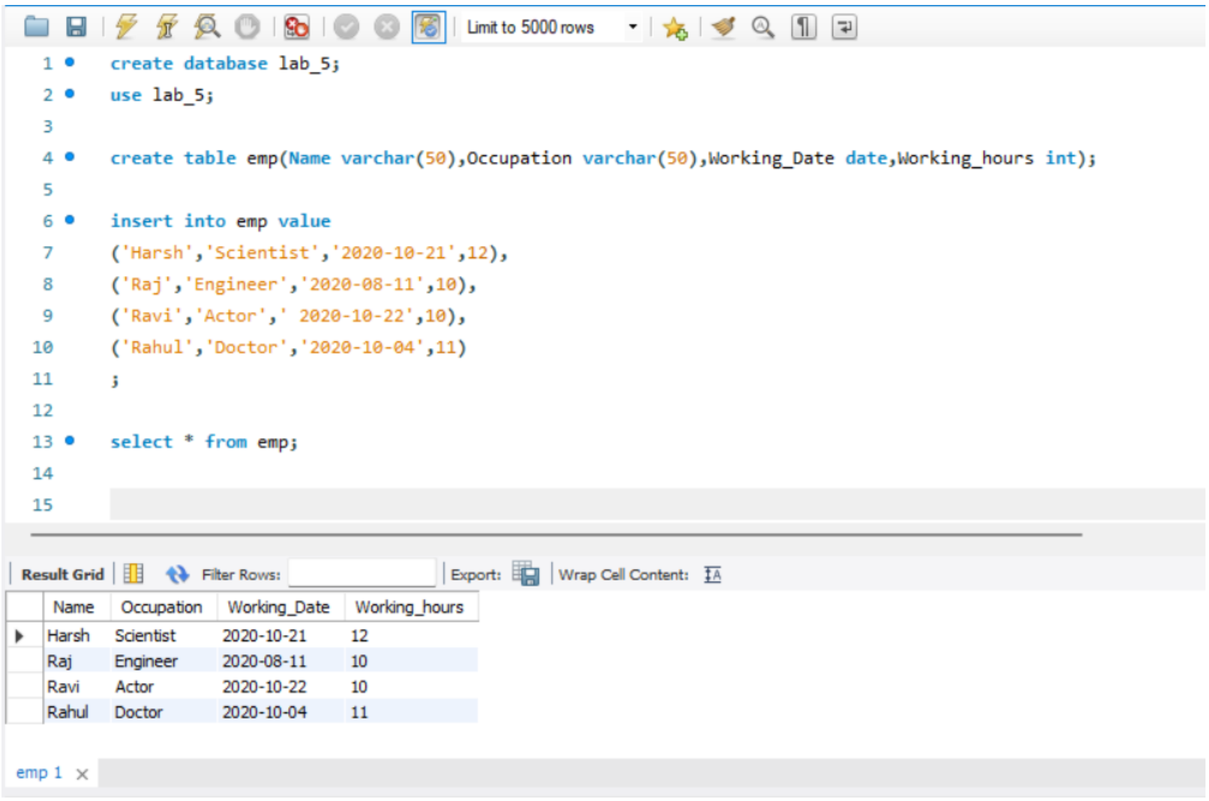
if new.Working_hours < 0

then

set new.Working_hours = 0;

end if;

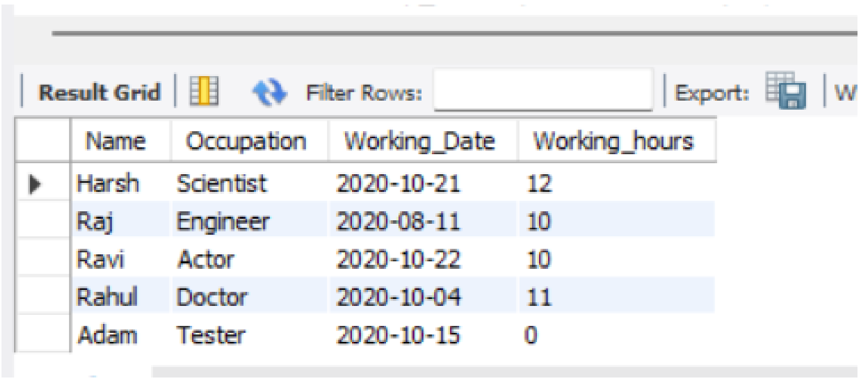
END



```
create database lab_5;
use lab_5;

create table emp(Name varchar(50),Occupation varchar(50),Working_Date date,Working_hours int);

insert into emp value
('Harsh','Scientist','2020-10-21',12),
('Raj','Engineer','2020-08-11',10),
('Ravi','Actor',' 2020-10-22',10),
('Rahul','Doctor','2020-10-04',11)
;

select * from emp;
```

Name	Occupation	Working_Date	Working_hours
Harsh	Scientist	2020-10-21	12
Raj	Engineer	2020-08-11	10
Ravi	Actor	2020-10-22	10
Rahul	Doctor	2020-10-04	11

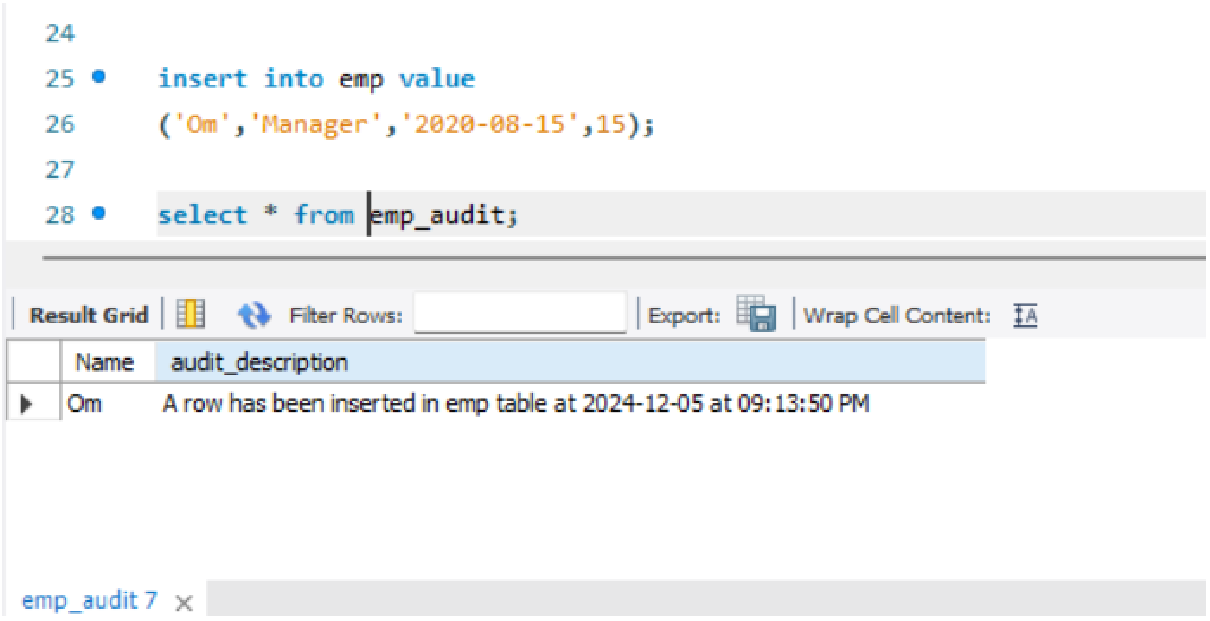
insert into emp value('Adam','Tester','2020-10-15',-20);



Name	Occupation	Working_Date	Working_hours
Harsh	Scientist	2020-10-21	12
Raj	Engineer	2020-08-11	10
Ravi	Actor	2020-10-22	10
Rahul	Doctor	2020-10-04	11
Adam	Tester	2020-10-15	0

2)

```
CREATE DEFINER=`root`@`localhost` TRIGGER `emp_AFTER_INSERT` AFTER INSERT ON `emp` FOR
EACH ROW BEGIN

insert into emp_audit values

(new.Name, concat(

'A row has been inserted in emp table at ',

Date_Format(NOW(),'%Y-%m-%d'),

' at ',

Date_Format(NOW(),'%h:%i:%s %p')

));

END
```

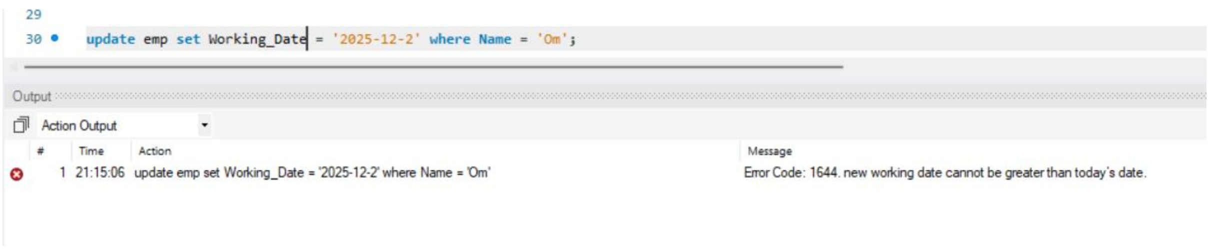


```
insert into emp value
('Om','Manager','2020-08-15',15);

select * from emp_audit;
```

Name	audit_description
Om	A row has been inserted in emp table at 2024-12-05 at 09:13:50 PM

3)

```
CREATE DEFINER=`root`@`localhost` TRIGGER `emp_BEFORE_UPDATE` BEFORE UPDATE ON `emp`
FOR EACH ROW BEGIN


  if new.working_date > date_format(now(),'%y-%m-%d') then
    signal sqlstate '45000'
    set message_text = 'new working date cannot be greater than today’s date.';
  end if;
END
```



```
update emp set Working_Date = '2025-12-2' where Name = 'Om';
```

Output

Action Output

```
#  Time      Action                                                     Message
1  21:15:06  update emp set Working_Date = '2025-12-2' where Name = 'Om'  Error Code: 1644. new working date cannot be greater than today’s date.
```

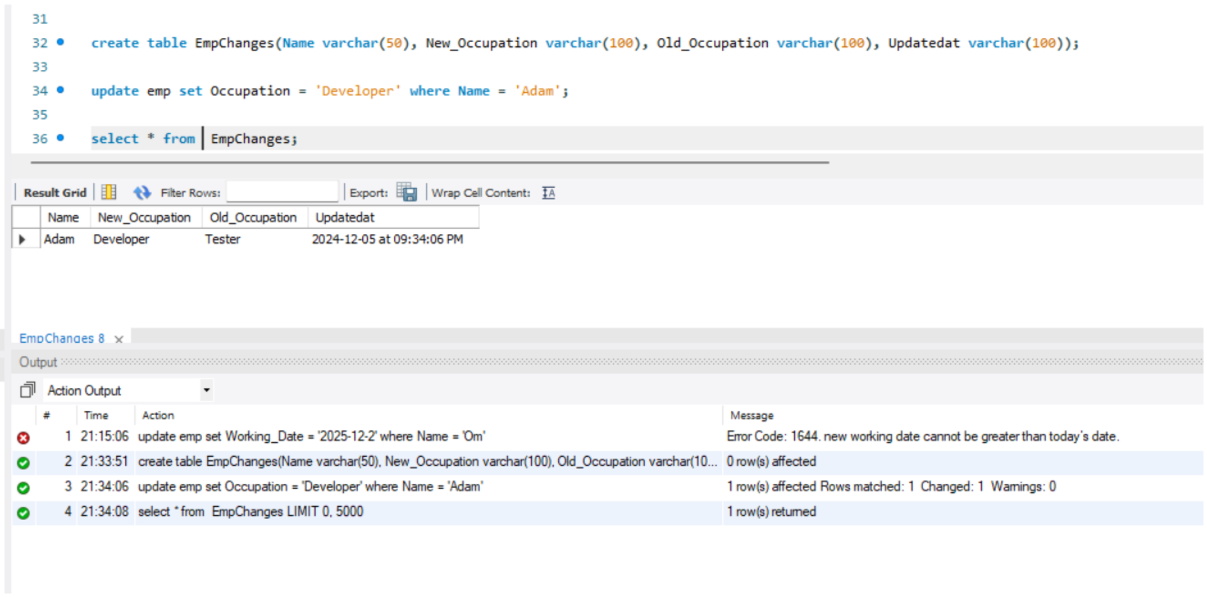
4)

CREATE DEFINER=`root`@`localhost` TRIGGER `emp_AFTER_UPDATE` AFTER UPDATE ON `emp` FOR EACH ROW BEGIN

insert into EmpChanges values

(new.Name,new.Occupation, old.Occupation, concat(Date_Format(now(),'%Y-%m-%d'), ' at ',Date_Format(now(),'%h:%i:%s %p')));

END



```
create table EmpChanges(Name varchar(50), New_Occupation varchar(100), Old_Occupation varchar(100), Updatedat varchar(100));

update emp set Occupation = 'Developer' where Name = 'Adam';

select * from EmpChanges;
```

Name	New_Occupation	Old_Occupation	Updatedat
Adam	Developer	Tester	2024-12-05 at 09:34:06 PM

```
#  Time      Action                                                                        Message
1  21:15:06  update emp set Working_Date = '2025-12-2' where Name = 'Om'                   Error Code: 1644. new working date cannot be greater than today's date.
2  21:33:51  create table EmpChanges(Name varchar(50), New_Occupation varchar(100), Old_Occupation varchar(10...  0 row(s) affected
3  21:34:06  update emp set Occupation = 'Developer' where Name = 'Adam'                   1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
4  21:34:08  select * from  EmpChanges LIMIT 0, 5000                                      1 row(s) returned
```

5)

CREATE DEFINER=`root`@`localhost` TRIGGER `emp_BEFORE_DELETE` BEFORE DELETE ON `emp` FOR EACH ROW BEGIN

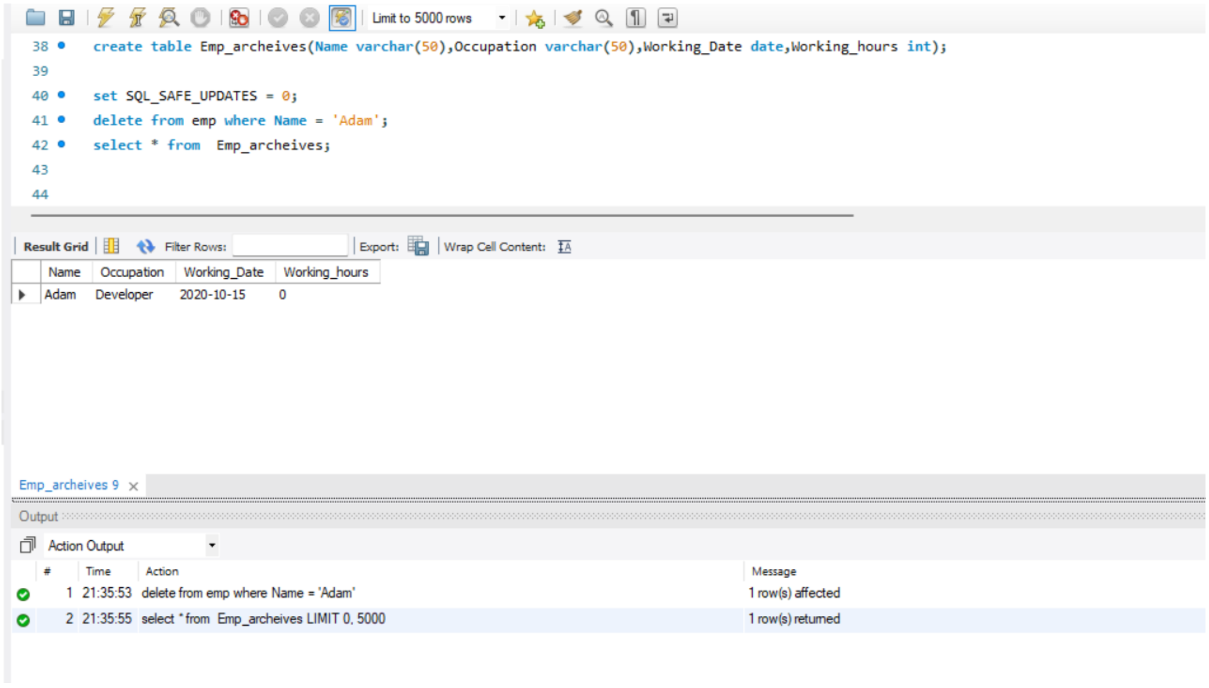
insert into Emp_archeives values

(

old.Name,old.Occupation,old.Working_Date,old.Working_hours

);

END



```
create table Emp_archeives(Name varchar(50),Occupation varchar(50),Working_Date date,Working_hours int);

set SQL_SAFE_UPDATES = 0;
delete from emp where Name = 'Adam';
select * from  Emp_archeives;
```

Name	Occupation	Working_Date	Working_hours
Adam	Developer	2020-10-15	0

Emp_archeives 9

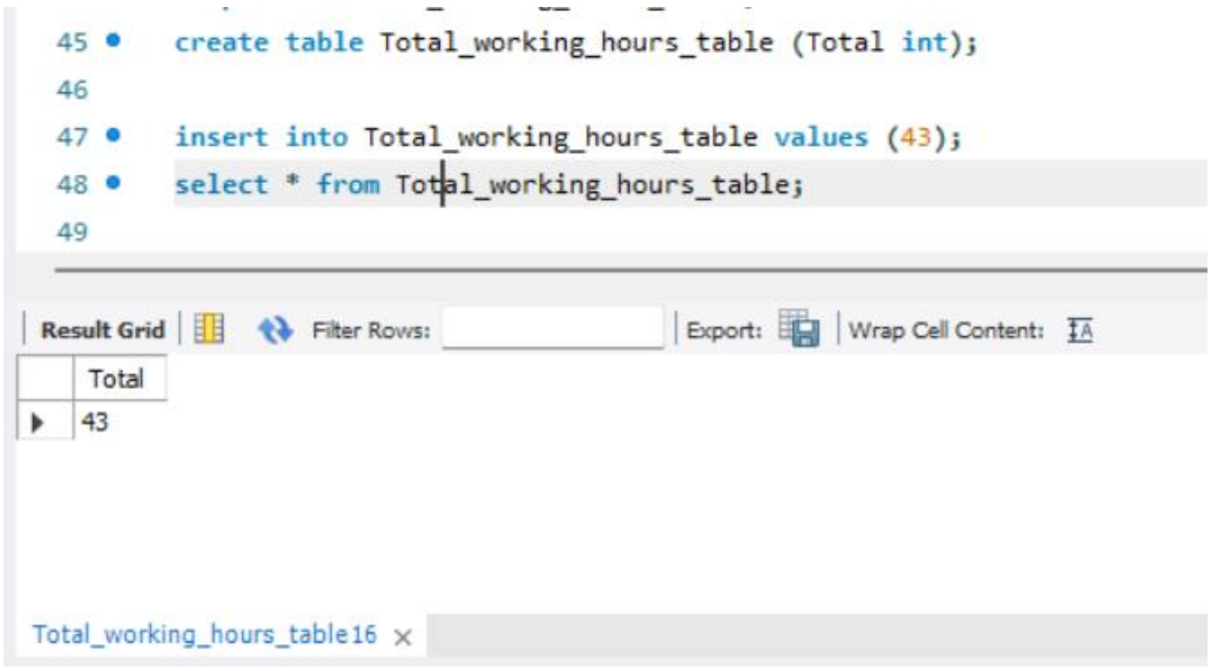
Output

Action Output

#	Time	Action	Message
1	21:35:53	delete from emp where Name = 'Adam'	1 row(s) affected
2	21:35:55	select * from Emp_archeives LIMIT 0, 5000	1 row(s) returned

6)

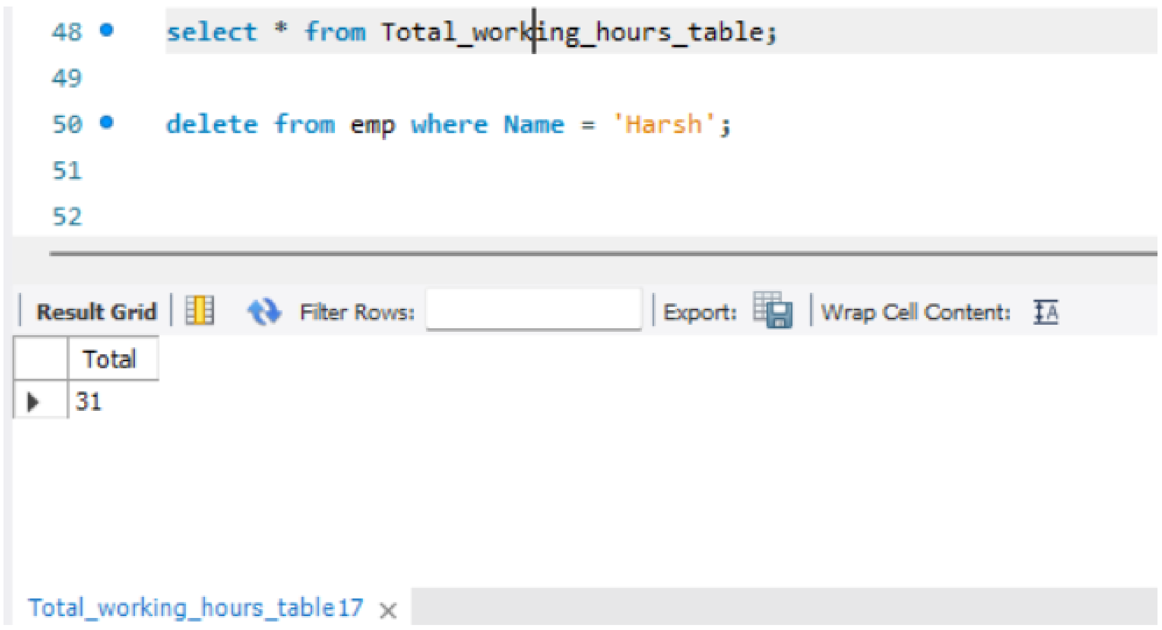
```
CREATE DEFINER=`root`@`localhost` TRIGGER `emp_AFTER_DELETE` AFTER DELETE ON `emp` FOR EACH ROW BEGIN

 update total_working_hours_table

    set total = total - old.working_hours;

END
```



```
create table Total_working_hours_table (Total int);

insert into Total_working_hours_table values (43);
select * from Total_working_hours_table;
```

Total
43

Total_working_hours_table16



```
select * from Total_working_hours_table;

delete from emp where Name = 'Harsh';
```

Total
31

Total_working_hours_table17

Observation:

Triggers are useful tools in SQL that automatically perform actions in response to certain events, such as INSERT, UPDATE, or DELETE. They help automate tasks like updating related tables, enforcing data integrity, and logging changes, without requiring manual intervention.