|  |  |
| --- | --- |
| **Name** | Adwait S Purao |
| **UID no.** | 2021300101 |
| **Experiment No.** | 8 |

|  |  |
| --- | --- |
| **AIM:** | To implement triggers in MySQL |
| **Program 1** | |
| **PROBLEM STATEMENT :** | Implement multiple triggers in hotel database |
| **Theory :** | A trigger in MySQL is a set of SQL statements that reside in a system catalog. **It is a special type of stored procedure that is invoked automatically in response to an event**. Each trigger is associated with a table, which is activated on any DML statement such as **INSERT, UPDATE**, or **DELETE**.  A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.  Generally, **triggers are of two types** according to the [SQL](https://www.javatpoint.com/sql-tutorial) standard: row-level triggers and statement-level triggers.  **Row-Level Trigger:** It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the [insert](https://www.javatpoint.com/mysql-insert), [update](https://www.javatpoint.com/mysql-update), or [delete statement](https://www.javatpoint.com/mysql-delete).  **Statement-Level Trigger:** It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted. **Why we need/use triggers in MySQL?** We need/use triggers in MySQL due to the following features:   * Triggers help us to enforce business rules. * Triggers help us to validate data even before they are inserted or updated. * Triggers help us to keep a log of records like maintaining audit trails in tables. * SQL triggers provide an alternative way to check the integrity of data. * Triggers provide an alternative way to run the scheduled task. * Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed. * Triggers reduce the client-side code that saves time and effort. * Triggers help us to scale our application across different platforms. * Triggers are easy to maintain.  **Limitations of Using Triggers in MySQL**  * MySQL triggers do not allow to use of all validations; they only provide extended validations. **For example**, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations. * Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer. * Triggers may increase the overhead of the database server.  **Types of Triggers in MySQL?** We can define the maximum six types of actions or events in the form of triggers:   1. [**Before Insert**](https://www.javatpoint.com/mysql-before-insert-trigger)**:** It is activated before the insertion of data into the table. 2. [**After Insert**](https://www.javatpoint.com/mysql-after-insert-trigger)**:** It is activated after the insertion of data into the table. 3. [**Before Update**](https://www.javatpoint.com/mysql-before-update-trigger)**:** It is activated before the update of data in the table. 4. [**After Update**](https://www.javatpoint.com/mysql-after-update-trigger)**:** It is activated after the update of the data in the table. 5. [**Before Delete**](https://www.javatpoint.com/mysql-before-delete-trigger)**:** It is activated before the data is removed from the table. 6. [**After Delete**](https://www.javatpoint.com/mysql-after-delete-trigger)**:** It is activated after the deletion of data from the table.   When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked. **Naming Conventions** Naming conventions are the set of rules that we follow to give appropriate unique names. It saves our time to keep the work organize and understandable. Therefore, **we must use a unique name for each trigger associated with a table**. However, it is a good practice to have the same trigger name defined for different tables.  The following naming convention should be used to name the trigger in [MySQL](https://www.javatpoint.com/mysql-tutorial):   1. (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:   1. **CREATE** **TRIGGER** trigger\_name 2. (**AFTER** | BEFORE) (**INSERT** | **UPDATE** | **DELETE**) 3. **ON** table\_name **FOR** EACH ROW 4. **BEGIN** 5. --variable declarations 6. --trigger code 7. **END**; |
| **Queries** | Trigger 1: Before Insert  Statement : Make the salary 0 if a non-negative salary is inserted  Code :  DELIMITER $$  CREATE TRIGGER Discard\_Neg\_Sal  BEFORE INSERT  ON employee FOR EACH ROW  BEGIN  IF new.e\_salary<0  THEN set new.e\_salary=0;  END IF;  END $$  DELIMITER ;  insert into employee values("Jay  Nadkarni","Permanent",6,1234,"Nadkarni","Jay","Andheri","Mumbai",12346,-900);  Select \* from employee;  Output:    Trigger 2: After Insert  Statement :Add a message to the reminder column if number of employees is zero  Code :  DELIMITER $$  CREATE TRIGGER Negative\_Emp  AFTER INSERT  ON hotel\_info FOR EACH ROW  BEGIN  IF new.h\_num\_emp<0  THEN set new.reminder="Pls enter number of employees";  END IF;  END $$  DELIMITER ;  insert into hotel\_info values("Vrindavan",8766,"Andheri",-89,7,NULL);  SELECT \* FROM HOTEL\_INFO;  Output:    Trigger 3: Before Update  Statement : Give a remark message if there is abnormal hike in price  Code :  DELIMITER $$  CREATE TRIGGER Pre\_Price\_Update  BEFORE UPDATE  ON room FOR EACH ROW  BEGIN  IF new.R\_price>old.R\_price \* 5  THEN set new.remark="Abnormal price hike";  END IF;  END $$  DELIMITER ;  update room set r\_price=10000 where r\_no=12;  select \* from room;  Output:    Trigger 4: After update  Statement : Give a error message if there is an abnormal rise in salary  Code :  DELIMITER $$  CREATE TRIGGER After\_salary\_Update  AFTER UPDATE  ON employee FOR EACH ROW  BEGIN  IF new.e\_salary>old.e\_salary \* 5 Then  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT="Abnormal salary hike";  END IF;  END $$  DELIMITER ;  update employee set e\_salary=100000 where e\_id=1;  Output:    Trigger 5: Before Delete  Statement : Put records in employee archives table befored deleting an entry from employee table  Code :  DELIMITER $$  CREATE TRIGGER Before\_emp\_delete  BEFORE DELETE  ON employee FOR EACH ROW  BEGIN  INSERT INTO employee\_Archives  VALUES(OLD.e\_id,OLD.e\_name,OLD.e\_contact,OLD.city);  END $$  DELIMITER ;  delete from employee where e\_id=1;  select \* from employee\_archives;  Output:    Trigger 6: After Delete  Statement : Make the salary 0 if a non-negative salary is inserted  Code :  DELIMITER $$  CREATE TRIGGER After\_emp\_delete  AFTER DELETE  ON employee FOR EACH ROW  BEGIN  INSERT INTO Salary\_Archives  VALUES(OLD.e\_id,OLD.e\_salary);  END $$  DELIMITER ;  delete from employee where e\_id=3;  select \* from salary\_archives;  Output:    Trigger 7: Before insert(Room)  Statement : Show an error message if room price is negative  Code :  DELIMITER $$  CREATE TRIGGER Neg\_room\_price  BEFORE INSERT  ON room FOR EACH ROW  BEGIN  IF new.r\_price<0  THEN SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT="Negative room price";  END IF;  END $$  DELIMITER ;  Insert into room values(16,1,-500,"Deluxe",1234,5,NULL);  Output:    Trigger 8: After Insert  Statement : Set reminder if vacancies are negative  Code :  DELIMITER $$  CREATE TRIGGER Negative\_Vacancies\_hotel  AFTER INSERT  ON hotel\_info FOR EACH ROW  BEGIN  IF new.h\_vacancies<0  THEN set new.reminder="Vacancies cannot be negative";  END IF;  END $$  DELIMITER ;  Insert into hotel\_info values("Sawali",7564,"Kurla",88,-4,NULL);  Select \* from hotel\_info;  Output:    Trigger 9: After Update  Statement : Display error message if H\_id is changed  Code :  DELIMITER $$  CREATE TRIGGER After\_Room\_Update  AFTER UPDATE  ON room FOR EACH ROW  BEGIN  IF new.h\_id!=old.h\_id then  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT="Pls enter the correct Hotel id";  END IF;  END $$  DELIMITER ;  Update room set h\_id=2345 where r\_no=12;  Output:    Trigger 10: Before Update  Statement : Set reminder if hotel name is changed  Code :  CREATE TRIGGER Name\_update  BEFORE UPDATE  ON hotel\_info FOR EACH ROW  BEGIN  IF new.h\_name!=old.h\_name  THEN set new.reminder="Name of hotel changed";  END IF;  END $$  DELIMITER ;  update hotel\_info set h\_name= "roshan" where h\_id=7564;  Output: |
| **Conclusion**  **In this experiment we learnt about triggers and how they help us in enforcing rules and validating data before insertion , deletion and updation . We implemented triggers on MySql workbench on hotel database.** | |