

---

## **Assignment:- 07**

### **Triggers**

Please ensure you update all the details:

**Name:** ULLI VENKATA SAI KUMAR **Batch ID:** 04072024HYD10AM

**Topic:** Introduction to Database

**1) Write an SQL query to accomplish the following tasks:**

- a) Create a database named **student\_db**.  
`CREATE DATABASE STUDENT_DB;`  
`USE STUDENT_DB;`
- b) Create a table named **students\_details** with columns **id** (integer), **name** (varchar), **age** (integer), and **grade** (float). **id** should be set as the primary key.  
`CREATE TABLE STUDENT_DETAILS (ID INT PRIMARY KEY, NAME VARCHAR(20), AGE INT, GRADE FLOAT);`
- c) Insert any four records into **students\_details**.  
`INSERT INTO STUDENT_DETAILS(ID, NAME, AGE, GRADE)VALUES`  
`(1, "SAI KUMAR", 24, 94.5),(2, "SANJAY", 26, 93.5),(3, "ESHAK", 27,`  
`92.5),(4, "SURENDRA", 25, 96.5);`
- d) Create a new table named **students\_details\_copy** with the same columns as **students\_details**. **id** should also be set as the primary key.  
`CREATE TABLE STUDENT_DETAILS_COPY (ID INT PRIMARY`  
`KEY,NAME VARCHAR(20), AGE INT,GRADE FLOAT);`
- e) Create a trigger named **after\_insert\_details** that inserts a new record into **students\_details\_copy** every time a record is inserted into **students\_details**.  
`CREATE TRIGGER AFTER_INSERT_DETAILS`  
`AFTER INSERT ON STUDENT_DETAILS`  
`FOR EACH ROW`  
`BEGIN`  
`INSERT INTO STUDENT_DETAILS_COPY(ID, NAME, AGE, GRADE)`  
`VALUES (NEW.ID, NEW.NAME, NEW.AGE, NEW.GRADE);`  
`END $$`  
`DELIMITER ;`
- f) Insert a new record into **students\_details**.  
`INSERT INTO STUDENT_DETAILS(ID, NAME, AGE, GRADE) VALUES`  
`(5, "BHASKAR", 28, 91.5);`

## Assignment:- 07

### Triggers

- g) check whether a record is filling in **students\_details\_copy** as you insert value in **students\_details**.

1 • `SELECT * FROM STUDENT_DETAILS_COPY;`

| ID   | NAME      | AGE  | GRADE |
|------|-----------|------|-------|
| 1    | SAI KUMAR | 24   | 94.5  |
| 2    | SANJAY    | 26   | 93.5  |
| 3    | ESHAK     | 27   | 92.5  |
| 4    | SURENDRA  | 25   | 96.5  |
| 5    | BHASKAR   | 28   | 91.5  |
| NULL | NULL      | NULL | NULL  |

## 2) Write an SQL question that accomplishes the following tasks:

- use **student\_db**,
- Create a trigger named **update\_grade** that automatically updates the **grade** column every time a record in **students\_details** is updated based on the following criteria:  
`DELIMITER $$`  
`CREATE TRIGGER UPDATE_GRADE`  
`AFTER UPDATE ON STUDENT_DETAILS`  
`FOR EACH ROW`
- If the updated record has an age value less than 18, multiply the grade by 0.9.  
`BEGIN`  
`-- Condition for age < 18`  
`IF NEW.AGE < 18 THEN`  
`UPDATE STUDENT_DETAILS`  
`SET GRADE = NEW.GRADE * 0.9`  
`WHERE ID = NEW.ID;`
- If the updated record has an age value between 18 and 20 (inclusive), multiply the grade by 1.1.  
`-- Condition for age between 18 and 20 (inclusive)`  
`ELSEIF NEW.AGE BETWEEN 18 AND 20 THEN`  
`UPDATE STUDENT_DETAILS`  
`SET GRADE = NEW.GRADE * 1.1`  
`WHERE ID = NEW.ID;`

---

## **Assignment:- 07**

### **Triggers**

- e) If the updated record has an age value greater than 20, multiply the grade by 1.05.

-- Condition for age > 20

```
ELSEIF NEW.AGE > 20 THEN
    UPDATE STUDENT_DETAILS
    SET GRADE = NEW.GRADE * 1.05
    WHERE ID = NEW.ID;
```

END IF;

END \$\$

DELIMITER ;

- f) Update the age value of one of the records in students\_new to see the trigger in action.

```
UPDATE STUDENT_DETAILS SET AGE = 19 WHERE ID = 1
```

Error Code: 1442. Can't update table 'student\_details' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.

- 3) Explain the difference between the AFTER and INSTEAD OF trigger operators in SQL.

In SQL, AFTER and INSTEAD OF are two types of triggers that determine when and how the trigger will execute in relation to a specified event (such as INSERT, UPDATE, or DELETE). Here's a breakdown of their differences:

#### 1. AFTER Trigger

**Execution Time:** The AFTER trigger fires after the triggering event (such as INSERT, UPDATE, or DELETE) has been executed successfully.

**Use Case:** It is typically used when you want to take an action after the event has been processed by the database (e.g., logging, updating another table, or validating data).

**Applicability:** Can be used on regular tables.

**Example:**

You have an AFTER INSERT trigger on a table students. When a new student record is inserted, the trigger fires after the insert and could be used to insert a log or update another table.

```
CREATE TRIGGER after_insert_trigger
```

```
AFTER INSERT ON students
```

```
FOR EACH ROW
```

```
BEGIN
```

```
-- Action to take after the INSERT
```

```
END;
```

#### 2. INSTEAD OF Trigger

**Execution Time:** The INSTEAD OF trigger fires in place of the triggering event. The actual event (INSERT, UPDATE, or DELETE) does not happen; instead, the code within the trigger executes.

---

## **Assignment:- 07**

### **Triggers**

**Use Case:** It is typically used when you need to override the default behavior of an operation (e.g., modifying the behavior of an INSERT into a view or preventing an action).

**Applicability:** Primarily used for views where direct INSERT, UPDATE, or DELETE operations may not be allowed or desired. It can also be used on regular tables to customize the behavior of operations.

**Example:**

You have an INSTEAD OF INSERT trigger on a view student\_view, and you want to customize how data is inserted into underlying tables.

```
CREATE TRIGGER instead_of_insert_trigger
```

```
INSTEAD OF INSERT ON student_view
```

```
FOR EACH ROW
```

```
BEGIN
```

```
-- Custom action instead of the default INSERT
```

```
END;
```

- 4) What is the purpose of the INSTEAD OF DELETE trigger operator in SQL?

The INSTEAD OF DELETE trigger in SQL is used to override the default behavior of a DELETE operation on a table. Instead of executing the delete action, the trigger performs an alternative action that you define. This can be useful for a variety of reasons, such as:

1. **Preventing Deletion:** You might want to prevent rows from being deleted and instead log the delete operation or mark the rows as inactive.
2. **Cascading Actions:** You could use the trigger to handle complex cascading operations or update related tables when a delete operation is attempted.
3. **Custom Logic:** You may need to execute custom logic or maintain data integrity in a way that is not directly supported by standard delete operations.

**EXAMPLE:-**

```
CREATE TRIGGER trg_InsteadOfDelete
```

```
ON YourTable
```

```
INSTEAD OF DELETE
```

```
AS
```

```
BEGIN
```

```
-- Example: Mark rows as inactive instead of deleting them
```

```
UPDATE YourTable
```

```
SET Active = 0
```

```
WHERE ID IN (SELECT ID FROM deleted);
```

```
-- Alternatively, you could log the deletion or perform other actions here
```

```
END;
```