Triggers

SCS2209-Database II
Mr. Rangana Jayashanka

Triggers

- Triggers are stored programs, which are automatically executed or fired when some events occur.
- Triggers are, in fact, written to be executed in response to any of the following events:

  - A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
  - A database definition (DDL) statement (CREATE, ALTER, or DROP)
  - A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- Triggers can be defined on the table, view, schema, or database with which the event is associated.

Syntax for specifying a Trigger

```
<trigger>: CREATE TRIGGER <trigger name>
           (AFTER/BEFORE) <triggering events> ON <table name>
           [FOR EACH ROW]
           BEGIN
           [WHEN <condition>]<trigger actions>;
           END

<triggering events>: INSERT|UPDATE|DELETE[OF<column name>]
<triggering actions>:<SQL block>
```


Example 1

Suppose *student_age* table is included in a School DB.
Create the table using following command.

```
Create table Student_age (age INT, Name Varchar(35));
```

Student_age

Age	Name

If we want to prevent inserting negative numbers for the age column, we can write a trigger. Insert 0 automatically, for any number < 0.

Example 01

```
DELIMITER //
Create Trigger before_insert_studentage BEFORE INSERT ON student_age
FOR EACH ROW
BEGIN
IF NEW.age < 0 THEN SET NEW.age = 0;
END IF;
END //
```

1. `INSERT INTO Student_age (age, Name) values (30, 'Rahul');`
2. `INSERT INTO Student_age (age, Name) values (-10, 'Harshit');`

Example 01

```
Create Trigger before_insert_studentage BEFORE INSERT ON student_age
FOR EACH ROW
BEGIN
IF NEW.age < 0 THEN SET NEW.age = 0;
END IF;

Create Trigger before_insert_studentage AFTER INSERT ON student_age
FOR EACH ROW
BEGIN
IF NEW.age < 0 THEN SET NEW.age = 0;
END IF;

INSERT INTO Student_age (age, Name) values (, 'Harshit')
```

Example 1

```
Create Trigger before_insert_studentage AFTER INSERT ON student_age
FOR EACH ROW
BEGIN
WHEN NEW.age < 0 THEN SET NEW.age = 0;
END IF;
```


Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Syntax

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.

Syntax

- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example 02

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

```
Select * from customers;
```


Example 2

Create a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger should display the salary difference between the old values and new values.

Example 2

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```


Example 2

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Example 2

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00);

Old salary:
New salary: 7500
Salary difference:

UPDATE customers
SET salary = salary + 500
WHERE id = 2;

Old salary: 1500
New salary: 2000
Salary difference: 500

Exam 2015

Consider the following schema to answer the given questions.

```
Employee (eno, ename, hireDate, Salary)
Project (projectID, projName, budget)
Emp_Proj (eno, ProjID, assignedDate)
```

Write a trigger to calculate and print the different between the old and new salary when the salary is updated.

Exam 2015

```
CREATE trigger R1
After update ON Employee
FOR EACH ROW
BEGIN
      declare sal.dif int;
       if (new.Salary >= old.Salary) then
       set sal.dif = new.Salary - old.Salary;
       else
       set sal.dif = old.Salary - new.Salary;
        end if;
       dbms_output.put_line(sal.dif);
END//
```