SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)
Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

Department of MCA

DBMS Triggers in DBMS

Course Name : 23CAT603 - DATA BASE MANAGEMENT SYSTEM

Class : I Year / I Semester

Unit IV – PL/SQL - Triggers

PL/SQL - 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.

Benefits of Triggers

Triggers can be written for the following purposes

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is

```
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;
```

Creating Triggers

- **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.
- **[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.

Creating Triggers

Example

To start with, we will be using the CUSTOMERS table

Select * from customers;

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

The following program creates 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 will display the salary difference between the old values and new values

Creating Triggers

```
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;
/
```

When the above code is executed at the SQL prompt, it produces the following result.

Trigger created.

Creating Triggers

The following points need to be considered here

- 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.

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

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

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result

Old salary:
New salary: 7500
Salary difference:

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table

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

Triggering a Trigger

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result

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

Trigger in SQL

A **Trigger** in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

In SQL, this concept is the same as the trigger in real life. For example, when we pull the gun trigger, the bullet is fired.

To understand the concept of trigger in SQL, let's take the below hypothetical situation:

Suppose Rishabh is the human resource manager in a multinational company. When the record of a new employee is entered into the database, he has to send the 'Congrats' message to each new employee. If there are four or five employees, Rishabh can do it manually, but if the number of new Employees is more than the thousand, then in such condition, he has to use the trigger in the database.

Thus, now Rishabh has to create the trigger in the table, which will automatically send a 'Congrats' message to the new employees once their record is inserted into the database.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

Trigger in SQL

A **Trigger** in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

In SQL, this concept is the same as the trigger in real life. For example, when we pull the gun trigger, the bullet is fired.

To understand the concept of trigger in SQL, let's take the below hypothetical situation:

Suppose Rishabh is the human resource manager in a multinational company. When the record of a new employee is entered into the database, he has to send the 'Congrats' message to each new employee. If there are four or five employees, Rishabh can do it manually, but if the number of new Employees is more than the thousand, then in such condition, he has to use the trigger in the database.

Thus, now Rishabh has to create the trigger in the table, which will automatically send a 'Congrats' message to the new employees once their record is inserted into the database.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

Trigger in SQL

A **Trigger** in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

In SQL, this concept is the same as the trigger in real life. For example, when we pull the gun trigger, the bullet is fired.

To understand the concept of trigger in SQL, let's take the below hypothetical situation:

Suppose Rishabh is the human resource manager in a multinational company. When the record of a new employee is entered into the database, he has to send the 'Congrats' message to each new employee. If there are four or five employees, Rishabh can do it manually, but if the number of new Employees is more than the thousand, then in such condition, he has to use the trigger in the database.

Thus, now Rishabh has to create the trigger in the table, which will automatically send a 'Congrats' message to the new employees once their record is inserted into the database.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

Trigger in SQL

In Structured Query Language, triggers are called only either before or after the below events:

INSERT Event: This event is called when the new row is entered in the table.

UPDATE Event: This event is called when the existing record is changed or modified in the table.

DELETE Event: This event is called when the existing record is removed from the table.

Types of Triggers in SQL

Following are the six types of triggers in SQL:

AFTER INSERT Trigger

This trigger is invoked after the insertion of data in the table.

AFTER UPDATE Trigger

This trigger is invoked in SQL after the modification of the data in the table.

AFTER DELETE Trigger

This trigger is invoked after deleting the data from the table.

BEFORE INSERT Trigger

This trigger is invoked before the inserting the record in the table.

BEFORE UPDATE Trigger

This trigger is invoked before the updating the record in the table.

BEFORE DELETE Trigger

This trigger is invoked before deleting the record from the table.

Trigger in SQL

Syntax of Trigger in SQL
CREATE TRIGGER Trigger_Name
[BEFORE | **AFTER**] [**Insert** | **Update** | **Delete**]
ON [Table_Name]
[**FOR** EACH ROW | **FOR** EACH **COLUMN**]
AS
Set of SQL Statement

In the trigger syntax, firstly, we have to define the name of the trigger after the CREATE TRIGGER keyword. After that, we have to define the BEFORE or AFTER keyword with anyone event.

Then, we define the name of that table on which trigger is to occur.
After the table name, we have to define the row-level or statement-level trigger.
And, at last, we have to write the SQL statements which perform actions on the occurring of event.

Trigger in SQL

Example of Trigger in SQL

To understand the concept of trigger in SQL, first, we have to create the table on which trigger is to be executed. The following query creates the **Student_Trigger** table in the SQL database:

```
CREATE TABLE Student_Trigger
(
Student_RollNo INT NOT NULL PRIMARY KEY,
Student_FirstName Varchar (100),
Student_EnglishMarks INT,
Student_PhysicsMarks INT,
Student_ChemistryMarks INT,
Student_MathsMarks INT,
Student_TotalMarks INT,
Student_Percentage  );
```

The following query shows the structure of the **Student_Trigger** table:

```
DESC Student_Trigger;
```

Trigger in SQL

Output:

Field	Type	NULL	Key	Default
Student_RollNo	INT	NO	PRI	NULL
Student_FirstName	Varchar(100)	YES		NULL
Student_EnglishMarks	INT	YES		NULL
Student_PhysicsMarks	INT	YES		NULL
Student_ChemistryMarks	INT	YES		NULL
Student_MathsMarks	INT	YES		NULL
Student_TotalMarks	INT	YES		NULL
Student_Percentage	INT	YES		NULL

Trigger in SQL

The following query fires a trigger before the insertion of the student record in the table:

```
CREATE TRIGGER Student_Table_Marks
BEFORE INSERT
ON
Student_Trigger
FOR EACH ROW
SET new.Student_TotalMarks = new.Student_EnglishMarks + new.Student_PhysicsMarks + new.Student_Chemistry
Marks + new.Student_MathsMarks,
new.Student_Percentage = ( new.Student_TotalMarks / 400) * 100;
```

The following query inserts the record into Student_Trigger table:

```
INSERT INTO Student_Trigger (Student_RollNo, Student_FirstName, Student_EnglishMarks, Student_PhysicsMarks
, Student_ChemistryMarks, Student_MathsMarks, Student_TotalMarks, Student_Percentage) VALUES ( 201, Sorya,
 88, 75, 69, 92, 0, 0);
```

Trigger in SQL

Output:

To check the output of the above INSERT statement, you have to type the following SELECT statement:
SELECT * **FROM** Student_Trigger;

Student_RollNo	Student_FirstName	Student_EnglishMarks	Student_PhysicsMarks	Student_chemistryMarks	Student_MathsMarks	Student_TotalMarks	Student_Percentage
201	Sorya	88	75	69	92	324	81

Trigger in SQL

Advantages of Triggers in SQL

Following are the three main advantages of triggers in Structured Query Language:

- SQL provides an alternate way for maintaining the data and referential integrity in the tables.
- Triggers helps in executing the scheduled tasks because they are called automatically.
- They catch the errors in the database layer of various businesses.
- They allow the database users to validate values before inserting and updating.

Disadvantages of Triggers in SQL

Following are the main disadvantages of triggers in Structured Query Language:

- They are not compiled.
- It is not possible to find and debug the errors in triggers.
- If we use the complex code in the trigger, it makes the application run slower.
- Trigger increases the high load on the database system.

Trigger in SQL

SQL triggers are essential tools in **database management systems (DBMS)** that allow automatic execution of a set of SQL statements when specific database events, such as **INSERT**, **UPDATE**, or **DELETE** operations, occur.

Triggers are commonly used to maintain **data integrity**, **track changes**, and **enforce business rules** automatically, without needing manual input. By using **SQL triggers**, developers can **automate tasks**, ensure **data consistency**, and keep accurate records of **database activities**.

In this article, we will explain **SQL triggers**, explaining their types, syntax, and practical use cases. We will explain different types of triggers, such as **DML triggers**, **DDL triggers**, and **logon triggers**, and discuss how SQL triggers work in various **database systems** like **SQL Server**, **MySQL**, and **Oracle**

What is an SQL Trigger?

A trigger is a stored procedure in a **database** that automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated.

In simple words, a **trigger** is a collection of SQL statements with particular names that are stored in system memory. It belongs to a specific class of **stored procedures** that are automatically invoked in response to database server events. Every **trigger** has a table attached to it.

Trigger in SQL

Key Features of SQL Triggers:

Automatic Execution: Triggers fire automatically when the defined event occurs (e.g., INSERT, UPDATE, DELETE).

Event-Driven: Triggers are tied to specific events that take place within the database.

Table Association: A trigger is linked to a specific table or view, and operates whenever changes are made to the table's data.

Syntax

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Trigger in SQL

Key Terms

Create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
[before | after]: This specifies when the trigger will be executed.
{insert | update | delete}: This specifies the DML operation.
On [table_name]: This specifies the name of the table associated with the trigger.
[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each affected row.
[trigger_body]: This provides the operation to be performed as the trigger is fired

Why do We Use Triggers in SQL?

When we need to carry out some actions automatically in certain **desirable scenarios**, triggers will be useful. For instance, we need to be aware of the frequency and timing of changes to a table that is constantly changing. In such cases, we could create a **trigger** to insert the required data into a different table if the **primary table** underwent any changes.

Types of SQL Triggers

Triggers can be categorized into **different types** based on the action they are associated with:

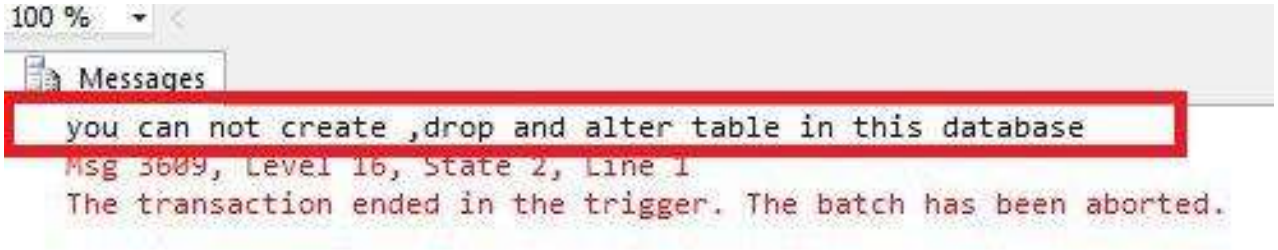
1. DDL Triggers

The **Data Definition Language** (DDL) command events such as **Create_table**, **Create_view**, **drop_table**, **Drop_view**, and **Alter_table** cause the DDL triggers to be activated.

Trigger in SQL

Query:

```
CREATE TRIGGER prevent_table_creation
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
PRINT 'you can not create, drop and alter table in this database';
ROLLBACK;
END;
```



Trigger in SQL

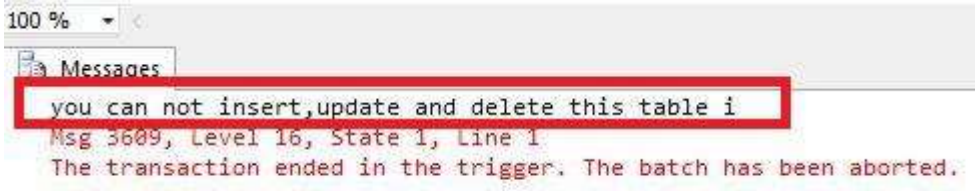
2. DML Triggers

The **Data manipulation Language** (DML) command events that begin with **Insert**, **Update**, and **Delete** set off the DML triggers. corresponding to insert_table, **update_view**, and **delete_table**.

Query:

```
CREATE TRIGGER prevent_update
ON students
FOR UPDATE
AS
BEGIN
PRINT 'You can not insert, update and delete this table i';
ROLLBACK;
END;
```

Output



Trigger in SQL

3. Logon Triggers

These triggers are fired in response to **logon events**. Logon triggers are useful for **monitoring user sessions** or restricting user access to the database. As a result, the **PRINT statement** messages and any errors generated by the trigger will all be visible in the **SQL Server** error log. **Authentication errors** prevent logon triggers from being used. These triggers can be used to **track login activity** or set a limit on the number of sessions that a given login can have in order to **audit** and **manage server sessions**.

Query:

```
CREATE TRIGGER track_logon
ON LOGON
AS
BEGIN
PRINT 'A new user has logged in.';
END;
```

Trigger in SQL

Viewing Triggers Using SQL Query

If we are working with many tables across multiple **databases**, we can use a simple query to list all available triggers in our SQL Server **instance**. This is helpful for **tracking** and **managing triggers**, especially when dealing with tables that have similar names across **databases**.

Syntax

```
SELECT name, is_instead_of_trigger
FROM sys.triggers
WHERE type = 'TR';
```

Key Terms

name: The name of the trigger.

is_instead_of_trigger: Whether the trigger is an **INSTEAD OF** trigger.

type = 'TR': This filters the results to show only triggers.

Trigger in SQL

BEFORE and AFTER Trigger

BEFORE triggers run the **trigger action before** the triggering statement is run. **AFTER triggers** run the trigger action after the triggering statement is run.

Example: Using BEFORE Trigger for Calculations

Given **Student Report Database**, in which student marks **assessment** is recorded. In such a schema, create a trigger so that the **total** and **percentage** of specified marks are automatically inserted whenever a record is inserted. Here, a **trigger will invoke** before the record is inserted so **BEFORE Tag** can be used.

Query

mysql>>desc Student;

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

Trigger in SQL

BEFORE and AFTER Trigger

BEFORE triggers run the **trigger action before** the triggering statement is run. **AFTER triggers** run the trigger action after the triggering statement is run.

Example: Using BEFORE Trigger for Calculations

Given **Student Report Database**, in which student marks **assessment** is recorded. In such a schema, create a trigger so that the **total** and **percentage** of specified marks are automatically inserted whenever a record is inserted. Here, a **trigger will invoke** before the record is inserted so **BEFORE Tag** can be used.

Query

mysql>>desc Student;

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

Trigger in SQL

SQL Trigger to the problem statement.

Stud_marks

Above **SQL statement** will create a trigger in the **student database** in which whenever subjects marks are entered, before inserting this data into the **database**, the trigger will compute those **two values** and insert them with the entered values. In this way, triggers can be **created** and **executed** in the **databases**.

Output

```
+------------+--------+-----------+--------------+---------------------+----
| Trigger    | Event  | Table     | Timing       | Created             | Stat
+------------+--------+-----------+--------------+---------------------+----
| stud_marks | BEFORE | Student   | BEFORE INSERT | 2023-05-02 00:00:00 | ACT
+------------+--------+-----------+--------------+---------------------+----
```

Trigger in SQL

Advantage of Triggers

Data Integrity: Triggers help enforce consistency and business rules, ensuring that data follows the correct format.

Automation: Triggers eliminate the need for manual intervention by automatically performing tasks such as updating, inserting, or deleting records when certain conditions are met.

Audit Trail: Triggers can track changes in a database, providing an audit trail of **INSERT**, **UPDATE**, and **DELETE** operations.

Performance: By automating repetitive tasks, triggers improve **SQL query performance** and reduce manual workload.

References

1. https://www.tutorialspoint.com/plsql/plsql_triggers.htm
2. https://www.geeksforgeeks.org/sql-trigger-student-database/
3. https://www.javatpoint.com/trigger-in-sql