SQL Triggers:

A trigger is a special stored procedure in a database that automatically executes when specific events (like INSERT, UPDATE, or DELETE) occur on a table. Triggers help automate tasks, maintain data consistency, and record database activities. Each trigger is tied to a particular table and runs without manual execution.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

FOR EACH ROW

BEGIN

END;

In the above syntax:

trigger_name: The name of the trigger to be created

BEFORE | AFTER: Specifies whether the trigger is fired before or after the triggering event (INSERT, UPDATE, DELETE).

{INSERT | UPDATE | DELETE}: Specifies the operation that will activate the trigger.

table_name: The name of the table the trigger is associated with.

FOR EACH ROW: Indicates that the trigger is row-level, meaning it executes once for each affected row.

trigger_body: The SQL statements to be executed when the trigger is fired.

Example: Automatically Track when a new workers table record is updated

Create a table / Use an existing table:

CREATE TABLE Workers (id INT PRIMARY KEY, name VARCHAR(50), email VARCHAR(100), updated_at TIMESTAMP);

Create Trigger:

This trigger automatically updates the updated_at field whenever the user record is modified.

CREATE TRIGGER update_timestamp

BEFORE UPDATE ON workers

FOR EACH ROW

BEGIN

  SET NEW.updated_at = CURRENT_TIMESTAMP;

END;

Inserting data into the table:

INSERT INTO Workers (id, name, email) VALUES (1, 'Amit', 'amit@example.com');

Display the table values:

Select * from Workers;

Updating a record in the table:

UPDATE Workers SET email = 'amit_new@example.com' WHERE id = 1; **and display the contents of a table**

Select * from Workers;

Example: Use the CUSTOMERS table we had created and used in the previously –

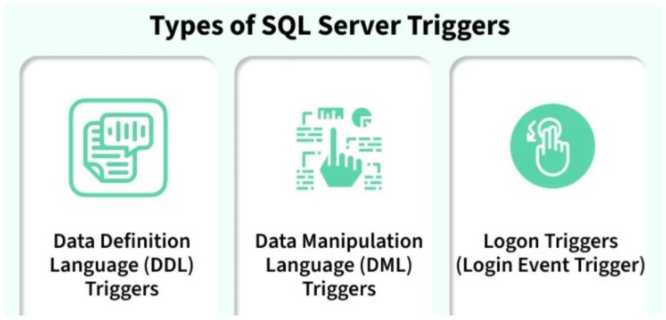
create a row-level trigger for displaying the salary difference between the old values and new values –

Display the table customers: `Select * from customers;`

Now, 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 will display the salary difference between the old values and new values –

```
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;
/
```

Output: Trigger created.


Types of SQL Server Triggers
Data Definition Language (DDL) Triggers
Data Manipulation Language (DML) Triggers
Logon Triggers (Login Event Trigger)

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.

Example: Prevent Table Deletions

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

DML Triggers

DML triggers fire when we manipulate data with commands like INSERT, UPDATE, or DELETE. These triggers are perfect for scenarios where we need to validate data before it is inserted, log changes to a table, or cascade updates across related tables.

Example: Prevent Unauthorized Updates Let’s say you want to prevent users from updating the data in a sensitive students table. We can set up a trigger to handle that:

```
CREATE TRIGGER prevent_update
ON students
FOR UPDATE, INSERT, DELETE
AS
BEGIN
    RAISERROR ('You can not insert, update and delete rows in this table.', 16, 1);
END;
```

```
Output: Message: you cannot insert, update and delete rows in this table.
```

Logon Triggers

Logon triggers fire in response to user logon events. They are used to monitor login activity, restrict access, or limit active sessions for a login. Messages and errors from these triggers appear in the SQL Server error log.

Example: Track User Logins

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

```
Output: Message: A new user has logged in.
```

Try Automatically Updating Related Tables:

Example: DML Triggers on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table –

```
INSERT INTO CUSTOMERS values 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 -

```
Output: Old salary:
        New salary: 7500
        Salary difference:
```

As there is 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;
```

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 –

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

Example: Imagine we have a database for students, where the student_gradestable holds individual subject grades. If the grade of a student is updated, we may also need to update the total_scorestable.

```
CREATE TRIGGER update_student_score
AFTER UPDATE ON student_grades
FOR EACH ROW
BEGIN
  UPDATE total_scores
  SET score = score + :new.grade
  WHERE student_id = :new.student_id;
END;
```

This above triggers ensures that every time a student's grade is updated, the total score in the total_scores table is automatically recalculated.

Data Validation (Before Insert Trigger Example)

Triggers can be used to validate data before it is inserted into a table, ensuring that the data follows specific business rules. For instance, we may want to ensure that the grades being inserted are within a valid range (say 0 to 100).

```
CREATE TRIGGER validate_grade
BEFORE INSERT ON student_grades
FOR EACH ROW
BEGIN
  IF :new.grade < 0 OR :new.grade > 100 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Invalid grade value.');
  END IF;
END;
```

The trigger checks if the inserted grade is valid. If not, it throws an error and prevents the insertion.

****