

## Assignment No A8

**Title:** Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table.

**Objective:** Understand the concept of Triggers and use of triggers on database table.

**Theory:**

**Definition :**

A trigger is defined for a specific table and one or more events. In most database management systems you can only define one trigger per table.

**Syntax :**

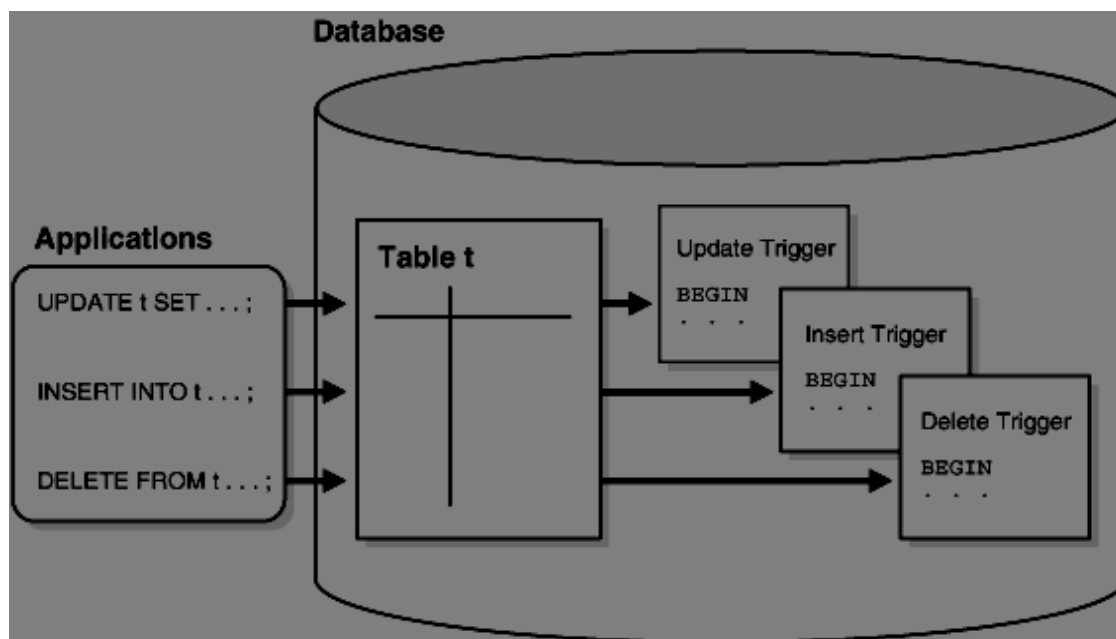
-- SQL Server Syntax --

Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name ON {  
table | view }  
[ WITH <dml_trigger_option> [ ,...n ] ]  
{ FOR | AFTER | INSTEAD OF }  
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] } [  
WITH APPEND ]  
[ NOT FOR REPLICATION ]  
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

```
<dml_trigger_option> ::= [  
    ENCRYPTION ]  
[ EXECUTE AS Clause ]
```

```
<method_specifier> ::=  
    assembly_name.class_name.method_name
```



### Explanation :

### What is a Database Trigger?

A database trigger is special stored procedure that is run when specific actions occur within a database. Most triggers are defined to run when changes are made to a table's data. Triggers can be defined to run instead of or after DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.

Triggers help the database designer ensure certain actions, such as maintaining an audit file, are completed regardless of which program or user makes changes to the data.

The programs are called triggers since an event, such as adding a record to a table, fires their execution.

Triggers and their implementations are specific to database vendors. In this article we'll focus on Microsoft SQL server; however, the concepts are the same or similar in Oracle and MySQL.

Note: All the examples for this lesson are based on Microsoft SQL Server Management Studio and the AdventureWorks2012 database. You can get started using these free tools using my Guide Getting Started Using SQL Server.

### Events

The triggers can occur AFTER or INSTEAD OF a DML action. Triggers are associated with the database DML actions INSERT, UPDATE, and DELETE. Triggers are defined to run when these actions are executed on a specific table.

### Triggering Event or Statement

A triggering event or statement is the SQL statement that causes a trigger to be fired. A triggering event can be an INSERT, UPDATE, or DELETE statement on a table.

For example, in Figure 15 - 3, the triggering statement is

... UPDATE OF parts\_on\_hand ON inventory ...

which means that when the PARTS\_ON\_HAND column of a row in the INVENTORY table is updated, fire the trigger. Note that when the triggering event is an UPDATE statement, you can include a column list to identify which columns must be updated to fire the trigger. Because INSERT and DELETE statements affect entire rows of information, a column list cannot be specified for these options.

A triggering event can specify multiple DML statements, as in

... INSERT OR UPDATE OR DELETE OF inventory ...

which means that when an INSERT, UPDATE, or DELETE statement is issued against the INVENTORY table, fire the trigger. When multiple types of DML statements can fire a trigger, conditional predicates can be used to detect the type of triggering statement. Therefore, a single trigger can be created that executes different code based on the type of statement that fired the trigger.

### **Trigger Restriction**

A trigger restriction specifies a Boolean (logical) expression that must be TRUE for the trigger to fire. The trigger action is not executed if the trigger restriction evaluates to FALSE or UNKNOWN.

A trigger restriction is an option available for triggers that are fired for each row. Its function is to control the execution of a trigger conditionally. You specify a trigger restriction using a WHEN clause. For example, the REORDER trigger in Figure 15 - 3 has a trigger restriction. The trigger is fired by an UPDATE statement affecting the PARTS\_ON\_HAND column of the INVENTORY table, but the trigger action only fires if the following expression is TRUE:

```
new.parts_on_hand < new.reorder_point
```

### **Trigger Action**

A trigger action is the procedure (PL/SQL block) that contains the SQL statements and PL/SQL code to be executed when a triggering statement is issued and the trigger restriction evaluates to TRUE.

Similar to stored procedures, a trigger action can contain SQL and PL/SQL statements, define PL/SQL language constructs (variables, constants, cursors, exceptions, and so on), and call stored procedures. Additionally, for row trigger, the statements in a trigger action have access to column values (new and old) of the current row being processed by the trigger. Two correlation names provide access to the old and new values for each column.

### **Types of Triggers**

When you define a trigger, you can specify the number of times the trigger action is to be executed: once for every row affected by the triggering statement (such as might be fired by an UPDATE statement that updates many rows), or once for the triggering statement, no matter how many rows it affects.

**Row Triggers** A row trigger is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not executed at all.

Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected. For example, Figure 15 - 3 illustrates a row trigger that uses the values of each row affected by the triggering statement.

**Statement Triggers** A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects (even if no rows are affected). For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once, regardless of how many rows are deleted from the table.

Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected. For example, if a trigger makes a complex security check on the current time or user, or if a trigger generates a single audit record based on the type of triggering statement, a statement trigger is used.

### **BEFORE vs. AFTER Triggers**

When defining a trigger, you can specify the trigger timing. That is, you can specify whether the trigger action is to be executed before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

**BEFORE Triggers** BEFORE triggers execute the trigger action before the triggering statement. This type of trigger is commonly used in the following situations:

- BEFORE triggers are used when the trigger action should determine whether the triggering statement should be allowed to complete. By using a BEFORE trigger for this purpose, you can eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action.
- BEFORE triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

**AFTER Triggers** AFTER triggers execute the trigger action after the triggering statement is executed. AFTER triggers are used in the following situations:

- AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.
- If a BEFORE trigger is already present, an AFTER trigger can perform different actions on the same triggering statement.

### Combinations

Using the options listed in the previous two sections, you can create four types of triggers:

- BEFORE statement trigger Before executing the triggering statement, the trigger action is executed.
- BEFORE row trigger Before modifying each row affected by the triggering statement and before checking appropriate integrity constraints, the trigger action is executed provided that the trigger restriction was not violated.
- AFTER statement trigger After executing the triggering statement and applying any deferred integrity constraints, the trigger action is executed.
- AFTER row trigger After modifying each row affected by the triggering statement and possibly applying appropriate integrity constraints, the trigger action is executed for the current row provided the trigger restriction was not violated. Unlike BEFORE row triggers, AFTER row triggers lock rows.

You can have multiple triggers of the same type for the same statement for any given table. For example you may have two BEFORE STATEMENT triggers for UPDATE statements on the EMP table. Multiple triggers of the same type permit modular installation of applications that have triggers on the same tables. Also, Oracle snapshot logs use AFTER ROW triggers, so you can design your own AFTER ROW trigger in addition to the Oracle-defined AFTER ROW trigger.

You can create as many triggers of the preceding different types as you need for each type of DML statement (INSERT, UPDATE, or DELETE). For example, suppose you have a table, SAL, and you want to know when the table is being accessed and the types of queries being issued. A global session variable, STAT.ROWCNT, is initialized to zero by a BEFORE statement trigger, then it is increased each time the row trigger is executed, and finally the statistical information is saved in the table STAT\_TAB by the AFTER statement trigger.

### Example - ROW LEVEL AFTER UPDATE TRIGGER

```
CREATE OR REPLACE TRIGGER "TRIGGER_ROW_LVL_AFTER_UPDATE"
AFTER UPDATE OF BOOK_STATUS_ID ON LIBRARY
FOR EACH ROW BEGIN
INSERT INTO LIBRARY_AUDIT_ROW_LVL(BOOK_ID, BOOK_STATUS_ID,
UPDATE_TS) VALUES (:old.BOOK_ID, :old.BOOK_STATUS_ID,
CURRENT_TIMESTAMP);
END;
```

**Example - ROW LEVEL BEFORE UPDATE TRIGGER**

```
CREATE OR REPLACE TRIGGER "TRIGGER_ROW_LVL_BEFORE_UPDATE"  
BEFORE UPDATE OF BOOK_STATUS_ID ON LIBRARY  
FOR EACH ROW BEGIN  
    INSERT INTO LIBRARY_AUDIT_ROW_LVL(BOOK_ID, BOOK_STATUS_ID,  
    UPDATE_TS) VALUES (:old.BOOK_ID, :old.BOOK_STATUS_ID,  
    CURRENT_TIMESTAMP);  
END;
```

**Example - STATEMENT LEVEL AFTER DELETE TRIGGER**

```
CREATE OR REPLACE TRIGGER TRIGGER_STMT_LVL_AFTER_DELETE  
AFTER DELETE ON LIBRARY  
BEGIN  
    INSERT INTO LIBRARY_AUDIT_STMT_LVL (STMT_TYPE, UPDATE_TS) VALUES  
('AFTER DELETE', CURRENT_TIMESTAMP);  
END;
```

**Example - STATEMENT LEVEL BEFORE DELETE TRIGGER**

```
CREATE OR REPLACE TRIGGER TRIGGER_STMT_LVL_BEFORE_DELETE  
BEFORE DELETE ON LIBRARY  
BEGIN  
    INSERT INTO LIBRARY_AUDIT_STMT_LVL (STMT_TYPE, UPDATE_TS) VALUES  
('BEFORE DELETE', CURRENT_TIMESTAMP);  
END;
```

**Conclusion:** Here we understood what are triggers and events, types of triggers how triggers are used on database, and come to know that how use-full triggers are to manage the database.