****

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science

**CS311 Introduction to Database**

**Spring 2023**

**Lab # 11**

**Objective: Database Triggers**

.

Maira Usman

**Name of Student:**

21B-011-SE

**B**

**Roll No: Section:**

15-06-2023

**Date of Experiment:**

**Date Submitted:**

Marks Obtained/Remarks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# What is a trigger?

A

***Trigger*** is a PL/SQL block that executes implicitly whenever a particular event takes place. A trigger can be either a database trigger or an application trigger.

***Database triggers*** execute implicitly when an INSERT, UPDATE, or DELETE statement is issued against the associated table, no matter which user is connected or which application is used.

***Application triggers*** execute implicitly whenever a particular event occurs within an application. An example of an application that uses triggers extensively is one developed with Developer/2000 Form Builder.

**Note**: Database triggers can be defined only on tables, not on views. However, if a DML operation is issued against a view, triggers on the base table(s) of a view are fired.

## Guidelines for designing triggers

* Only use database triggers for centralized, global operations that should be fired for the triggering statement, regardless of which user or application issues the statement.
* Do not define triggers to implement integrity rules that can be done by using declarative constraints.
* The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications. Only use triggers when necessary, and beware of recursive and cascading effects.

## Database Trigger Types

The trigger type determines the number of times the trigger action is to be executed: once for every row affected by the triggering statement (such as a multiple row UPDATE), or once for the triggering statement no matter how many rows it affects.

### **Statement Trigger**

A statement trigger is fired once on behalf of the triggering event, even if no rows are affected at all. Statement triggers are useful if the trigger action does not depend on data of rows that are affected or data provided by the triggering event itself. For example, a trigger that performs a complex security check on the current user.

### **Row Trigger**

A Row trigger fires each time the table is affected by the triggering event. If the triggering event affects no row(s), a row trigger is not executed at all.

Row triggers are useful if the trigger action depends on data of rows that are affected or data provided by the triggering event itself.

## Creating Statement Triggers

**Syntax for creating Statement Triggers**

CREATE [OR REPLACE] TRIGGER *trigger\_name*

*Timing* *event1* [OR *event2* OR *event3*] ON *table\_name*

### Trigger Components

Before coding the trigger block, decide on the components of the trigger: -

**Trigger timing**: BEFORE or AFTER

**Triggering event**: INSERT or UPDATE or DELETE

**Table Name**: ON table

**‘Trigger Type**: Row or Statement

**Trigger body**: DECLARE

BEGIN

END;

**Trigger Timing**

Indicates the time when the trigger fires in relation to the triggering event: BEFORE or AFTER.

BEFORE Triggers

This type of trigger is frequently used in the following situations:

* When the trigger action should determine whether that triggering statement should be allowed to complete. This allows to eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the triggering action.
* To derive column values before completing a triggering INSERT or UPDATE statement.

AFTER Triggers

This type of trigger is frequently used in the following situations:

* When the triggering statement is to be completed before executing the triggering action.
* If a BEFORE trigger is already present, and an after trigger can perform different actions on the same triggering statement.

**Triggering Event**

The triggering event or statement can be an INSERT, UPDATE, or DELETE statement on a table.

* When the triggering event is an UPDATE, we can include a column list to identify which column(s) must be changed to fire the trigger. We cannot specify a column list for an INSERT or for a DELETE statement, as they always affect entire rows.
* The triggering event can contain multiple DML statements. In this way, we can differentiate what code to execute depending on the statement that caused the triggers to fire.

**Trigger Body**

The trigger action defines what needs to be done when the triggering event is issued. It can contain SQL and PL/SQL statements, define PL/SQL constructs such as variables, cursors, exceptions and so on.

Additionally row triggers have access to the old and new column values of the row being processed by the trigger, using correlation names.

The trigger body is defined with an anonymous PL/SQL block.

[DECLARE]

BEGIN

[EXCEPTION]

END;

**Before Statement Trigger:**

We can create a *BEFORE statement* trigger in order to prevent the triggering operation from succeeding if a certain condition is violated.

For example, create a trigger to restrict inserts into the EMP table to certain business hours on Monday through Friday. If a user attempted to insert a row into the EMP table on Saturday, for example, the user will see the message, the trigger will fail, and the triggering statement will be rolled back.

RAISE\_APPLICATION\_ERROR is a server-side built-in procedure that prints a message to the user and causes the PL/SQL block to fail. When a database trigger fails, the triggering statement is automatically rolled back by the Oracle Server.

Creating Before Insert Trigger:

CREATE OR REPLACE TRIGGER **secure\_emp**

BEFORE INSERT ON emp

BEGIN

IF (TO\_CHAR(sysdate, 'DY') IN ('SAT', 'SUN'))

OR (TO\_CHAR(sysdate, 'HH24') NOT BETWEEN '08' AND '18') THEN RAISE\_APPLICATION\_ERROR (-20000, 'You may only insert into EMP during normal hours.');

END IF;

END**;**

**Using Conditional Predicates**

We can combine several triggering events into one by taking advantage of the special conditional predicates INSERTING, UPDATING, and DELETING within the trigger body. For example, create one trigger to restrict all data manipulation events on the EMP table to certain business hours, Monday through Friday. Also use BEFORE statement triggers to initialize global variables or flags, and to validate complex business rules.

CREATE OR REPLACE TRIGGER secure\_emp

BEFORE INSERT OR UPDATE OR DELETE ON emp

BEGIN

IF (TO\_CHAR (sysdate, 'DY') IN ('SAT', 'SUN')) OR (TO\_CHAR(sysdate, 'HH24') NOT BETWEEN '08' AND '18')

THEN

IF DELETING THEN

RAISE\_APPLICATION\_ERROR (-20502, 'You may only delete from EMP during normal hours.');

ELSIF INSERTING THEN

RAISE\_APPLICATION\_ERROR (-20500, 'You may only insert into EMP during normal hours.');

ELSIF UPDATING('SAL') THEN

RAISE\_APPLICATION\_ERROR (-20503, 'You may only update SAL during normal hours.');

ELSE

RAISE\_APPLICATION\_ERROR (-20504, 'You may only update EMP during normal hours.');

END IF;

END IF;

END;

After Statement Trigger:

We can create an *AFTER Statement* trigger in order to audit the triggering operation or perform a calculation after an operation has completed.

Suppose we have a user defined audit table that lists users and counts their data manipulation operations. After any user has updated the SAL column in the EMP table, use the audit table to ensure that the number of salary changes does not exceed the maximum permitted for that user.

**User Audit Table**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **USER\_ NAME** | **TABLE\_NAME** | **COLUMN\_NAME** | **INS** | **UPD** | **DEL** | **MAX\_INS** | **MAX\_UPD** | **MAX\_DEL** |
| UIT2 | EMP |  | 1 | 1 | 1 | 5 | 5 | 5 |
| UIT2 | EMP | SAL |  | 1 |  |  | 5 |  |
| UIT2 | EMP |  | 0 | 0 | 0 | 5 | 0 | 0 |

**After Update Trigger:**

|  |
| --- |
| CREATE OR REPLACE TRIGGER check\_salary\_count  AFTER UPDATE OF sal ON emp  DECLARE  v\_salary\_changes NUMBER;  v\_max\_changes NUMBER;  BEGIN  SELECT upd, max\_upd  INTO v\_salary\_changes, v\_max\_changes  FROM audit\_table  WHERE user\_name = 'UIT2'  AND tablename = 'EMP'  AND column\_name = 'SAL';  IF v\_salary\_changes > v\_max\_changes THEN  RAISE\_APPLICATION\_ERROR (-20501, 'You may only make a maximum of '  || to\_char(v\_max\_changes) || ' changes to the sal column');  END IF;  END; |

**After Delete Trigger: Example**

Creating an Audit Table:

|  |
| --- |
| CREATE TABLE orders\_audit1\_delete  (order\_id number (5),  quantity number (4),  cost\_per\_item number (6,2),  total\_cost number (8,2),  username varchar (50),  datetrans date,  detail varchar (30)  ); |

Creating After Delete Trigger:

|  |
| --- |
| CREATE OR REPLACE TRIGGER orders\_after\_delete1  AFTER delete ON orders  FOR EACH ROW  DECLARE  v\_username varchar2(10);  details varchar(30);  BEGIN  details:='record has been deleted'|| TO\_CHAR(:old.order\_id);  -- Find username of person performing the INSERT into the table  SELECT user INTO v\_username FROM dual;  -- Insert record into audit table  INSERT INTO orders\_audit1\_delete (order\_id, quantity, cost\_per\_item, total\_cost, username, datetrans, detail )  VALUES ( :old.order\_id, :old.quantity, :old.cost\_per\_item, :old.total\_cost, v\_username, sysdate, details);  END; |

**Disable/Enable a database trigger**

ALTER TRIGGER *trigger\_name* *DISABLE* | *ENABLE*;

**Removing a Trigger**

DROP TIGGER *trigger\_name*;

**EXERCISE**

1. What is meant by triggering event? Give examples.

In the context of databases, a triggering event refers to an action or condition that causes a trigger to execute. A trigger is a database object that is associated with a table and is automatically executed in response to a specific event or action, such as an insert, update, or delete operation on the table.

Examples of triggering events include:

Insert Event: The trigger can be programmed to execute when a new row is inserted into a specific table. For example, a trigger may be triggered when a new employee record is inserted into the "Employees" table, and it can perform actions such as updating related tables or sending notifications.

Update Event: The trigger can be configured to execute when one or more columns in a table are updated. For instance, a trigger may be triggered when the "Status" column of a "Orders" table is updated, and it can perform actions like validating the updated data or logging the changes.

Delete Event: The trigger can be activated when a row is deleted from a table. For example, a trigger may be triggered when a customer record is deleted from the "Customers" table, and it can perform actions like cascading deletes or archiving the deleted data.

1. Give three examples of a situation when a BEFORE statement trigger is needed?

* To validate data before it is inserted into a table. For example, you could create a trigger that checks to make sure that the data being inserted into a table is valid. For example, you could check to make sure that the data is within a certain range, or that it matches a certain pattern.
* To update data before it is updated in a table. For example, you could create a trigger that updates a field in a table with the current date and time whenever the row is updated. This could be useful for tracking changes to data.
* To insert data into another table before it is inserted into the current table. For example, you could create a trigger that inserts data into a log table whenever a row is inserted into another table. This could be useful for auditing changes to data.

1. Give three examples of a situation when a AFTER statement trigger is needed?

* To send an email notification after a row is inserted into a table. For example, you could create a trigger that sends an email notification to the owner of a table whenever a new row is inserted. This could be useful for notifying users of new data.
* To update a field in a table after it is updated. For example, you could create a trigger that updates a field in a table with the current date and time whenever the row is updated. This could be useful for tracking changes to data.
* To insert data into another table after it is inserted into the current table. For example, you could create a trigger that inserts data into a log table whenever a row is inserted into another table. This could be useful for auditing changes to data.

1. Create an Insert Trigger on EMP\_PROJECT to stop assign more than two project to single employee



