Oracle PL/SQL – Triggers

Introduction to Triggers:

- A trigger is a PL/SQL named block structure which is invoked/ executed automatically in response to a specific event.
- Triggers are 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).
  - System events such as startup, shutdown, and error messages.
  - User events such as logon and logoff.

- Triggers could be defined on the table, view, schema, or database with which the event is associated.
- The program structure of trigger is similar to that of procedures. The main differences between triggers and procedures are as follows;

  - Procedures can be invoked explicitly by a user or an application whereas a trigger is implicitly fired by Oracle when the triggering event occurs.
  - Arguments can be passed to procedures but triggers will not accept arguments.
  - Triggers must be stored as stand-alone objects in database and cannot be local to a block or package whereas procedures can be local to a block or package.

Importance of Triggers :

Triggers help Oracle in providing a customized database management system. Triggers can be used to

- Automatically generate derived column values.
- Prevent invalid transactions.
- Enforce complex business rules.
- Provide transparent event logging.
- Provide auditing.
- Gather statistics on table access.
- Modify table data when DML statements are issued against views.
- Publishes information about database events, user events, and SQL statements to subscribing applications.

Trigger Restrictions:

- A trigger may not issue a TCL statement.
- Any function or procedure called by a trigger cannot issue a TCL statement.
- It is not permitted to declare a LONG or LONG RAW variables in the body of a trigger.

Trigger Components:

A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction
- A trigger action

a) Triggering event or statement:

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire.

A triggering event can be one or more of the following:

- An INSERT, UPDATE, or DELETE statement on a specific table (or view, in some cases)
- A CREATE, ALTER, or DROP statement on any schema object
- A database startup or instance shutdown
- A specific error message or any error message
- A user logon or logoff

b) Trigger restriction:

- A trigger restriction specifies a Boolean expression that must be true for the trigger to fire.
- The trigger action is not run if the trigger restriction evaluates to false or unknown.

c) Trigger action:

A trigger action is the procedure that contains the SQL statements and code to be run when the following events occur:

- A triggering statement or event is issued.
- The trigger restriction evaluates to true.

A trigger action can:

- Contain SQL, PL/SQL or Java statements
- Define PL/SQL language constructs such as variables, constants, cursors, exceptions.
- Define Java language constructs
- Call stored procedures

Types of Triggers:

There are three main kinds of triggers:

- DML triggers
- System triggers
- Instead-of triggers

DML triggers:

- A DML trigger is fired by a DML statement, and the type of statement determines the type of DML trigger.
- DML triggers can be defined for INSERT, UPDATE, or DELETE operations.
- They can be triggered before or after the DML operation.
- Triggers can either act on all or some rows.
- The triggers can be further classified as below:
  - Row level or statement level
  - Before or after triggers

The below table explains the different factors of triggers types.

Category	Values	Description
DML Operation	INSERT,DELETE or UPDATE	Defines which kind of DML statement causes the trigger to fire.
Timing	Before or After	Defines whether the trigger fires **before** or **after** the statement is executed. Before and after triggers can be triggered only on tables and not on views.
Level	Row or Statement	If the trigger is a **row-level** trigger, it fires once for each row affected by the triggering statement. If it is **statement level** trigger, it triggers only once regardless of the number of rows affected by the triggering event.

The different combination of all these factors determine the trigger type. Some examples are

- Before INSERT statement level,
- After UPDATE row level,
- Before DELETE row level,etc

System triggers:

A system trigger is fired for the following events:

- System events
  - Database startup and shutdown
  - Server error message events
- User events
  - User logon and logoff
  - DDL statements (CREATE,ALTER and DROP)

Instead-of triggers:

- Instead-of triggers can be defined on views only.
- It provides a transparent way of modifying non editable views that cannot be modified directly through INSERT / UPDATE / DELETE statements.
- They are called Instead Of because Oracle database fires the trigger instead of running the statement which fired the trigger.

- The check option for views is not enforced when inserts or updates to the views are done using INSTEAD OF triggers. The INSTEAD OF Triggers body must enforce the check.
- These triggers must be row level.

Creating Triggers:

The general syntax for creating a trigger is

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;
```

Where

- **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 would 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 would 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, like INSERT, UPDATE, and DELETE.
- **[FOR EACH ROW]** :- This specifies a row level trigger, i.e., the trigger would 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.

Example: 1

In the below example, trig_display_salary_change is the trigger which will be triggered whenever there is an UPDATE to the salary column of employee table. Before updating each row, the difference between the to be updated and old salary value is calculated and printed.

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER trig_display_salary_change
BEFORE UPDATE OF salary
ON employee
FOR EACH ROW
WHEN (NEW.emp_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

SQL> update employee set salary =17000 where emp_id = 1200;
Old salary: 16000
New salary: 17000
Salary difference: 1000

1 row updated
```

Example 2 : Statement Trigger

trig_secure_emp is the trigger created to prevent from inserting records to Employee table during non business hours. If we try to insert a record during a weekend within the given time frame then it will raise an error.

```
CREATE or REPLACE TRIGGER trig_secure_emp
BEFORE INSERT ON employee
BEGIN
IF (TO_CHAR(sysdate,'DY') IN ('SAT','SUN')) OR
   (TO_CHAR(sysdate,'HH24:MI') NOT BETWEEN '08:00' and '18:00') THEN
raise_application_error(-20500,'Records can be inserted TO Employee TABLE only during business hours');
END IF;
END;
/
```

Example 3 : AFTER Trigger

trig_update_emp_history trigger will insert one record to emp_history table immediately after inserting a record to employee table with the same values.

```
CREATE OR REPLACE TRIGGER trig_update_emp_history
AFTER INSERT ON employee
FOR EACH ROW
BEGIN
IF(:NEW.emp_id > 0) THEN
INSERT INTO emp_history(emp_id,emp_name,designation) VALUES
                    (:NEW.emp_id,:NEW.emp_name,:NEW.designation);
END IF;
END;
/
```

Dropping and Disabling Triggers:

The syntax for permanently deleting existing trigger from database is as follows:

DROP TRIGGER <triggername>;

Unlike procedures and packages, however, a trigger can be disabled without dropping it. When a trigger is disabled, it still exists in the data dictionary but is never fired.

To disable a trigger, use the ALTER TRIGGER statement:

ALTER TRIGGER <triggername> {DISABLE | ENABLE};

where trigger name is the name of the trigger. All triggers are enabled by default when they are created. ALTER TRIGGER can disable and then reenable any trigger.