Topic - 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 −

1.Generating some derived column values automatically

2.Enforcing referential integrity

3.Event logging and storing information on table access

4. Auditing

5.Synchronous replication of tables

6.Imposing security authorizations

7.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;

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 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.

Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters −

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 −

```
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.

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;

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

Sanjeev Kumar Sinha 9931917742 MCA Course, Department of Statistics, P. U.