## EXERCISE-17

## TRIGGER

### DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,

- **Trigger statement**: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.

- **Trigger body or trigger action**: It is a PL/SQL block that is executed when the triggering statement is used.

- **Trigger restriction**: Restrictions on the trigger can be achieved

**The different uses of triggers are as follows,**

- *To generate data automatically*
- *To enforce complex integrity constraints*
- *To customize complex securing authorizations*
- *To maintain the replicate table*
- To audit data modifications

### TYPES OF TRIGGERS

The various types of triggers are as follows,

- **Before**: It fires the trigger before executing the trigger statement.

- **After**: It fires the trigger after executing the trigger statement
- .
- **For each row**: It specifies that the trigger fires once per row
- .
- **For each statement**: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

### VARIABLES USED IN TRIGGERS

- :new

- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

### SYNTAX

```
create or replace trigger triggername [before/after] {DML statements}
on [tablename] [for each row/statement]
begin
------------------------
```

Program 1

Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.

```
CREATE or replace trigger parent_child
Before delete on parent_table
For EACH ROW
DECLARE
    v_child_count NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_child_count
    FROM child_table
    WHERE parent_id = :old.primary_key;
    IF v_child_count > 0 THEN
    RAISE_APPLICATION_ERROR
  END IF;
END;
/
```

## Program 2

Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
CREATE OR REPLACE TRIGGER check_duplicate_value
Before INSERT OR UPDATE ON 'your_table'
For Each ROW
DECLARE
  v_count NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO v_count
   From your_table
   WHERE your_unique_column = :new.your_unique_column;
   IF v_count > 0 THEN
   RAISE_APPLICATION_ERROR
    END IF;
END;
```

## Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
CREATE OR REPLACE TRIGGER check_total_threshold
BEFORE INSERT ON your_table
DECLARE
  v_total NUMBER(10);
  v_threshold NUMBER := 1000; -- Set your threshold value here
BEGIN
  SELECT SUM(your_column)
  INTO v_total
  FROM your_table;

  IF v_total > v_threshold THEN
    RAISE_APPLICATION_ERROR(
  END IF;
END;
```

Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
CREATE OR REPLACE TRIGGER check_total_threshold
BEFORE INSERT ON your_table
DECLARE
  v_total NUMBER;
  v_threshold NUMBER := 100000; -- Your specific threshold here
SELECT SUM(some_column)
INTO v_total
FROM your_table;
IF v_total > v_threshold THEN
  RAISE_APPLICATION_ERROR
  END IF;
END;
```

## Program 4

Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
CREATE TABLE column_audit_log (
    log_id NUMBER GENERATED AS IDENTITY,
    table_audited VARCHAR2(30),
    column_audited VARCHAR2(30),
    row_pk VARCHAR2(100),
    old_value VARCHAR2(4000),
    new_value VARCHAR2(4000),
    changed_by VARCHAR2(30),
    change_date TIMESTAMP
);

CREATE OR REPLACE Trigger log_column_changes
After update on your_table
FOR EACH ROW
BEGIN
If :old.salary != :new.salary THEN
  INSERT into column_audit_log (table_audited, column_audited,
row_pk, old_value, new_value, changed_by, change_date
VALUES ('your_table', 'salary', :old.first_key : old.salary,
:new.salary, USER, systimestamp);
END IF;
If :old.job_id != :new.job_id then
  INSERT INTO column_audit_log (table_audited, column_audited,
row_pk, old_value, new_value, changed_by, change_date)
  VALUES ('your_table', 'job_id', :old.first_key : old.job_id,
  :new.job_id, USER, systimestamp);
END IF;

END;
```

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

```
CREATE TABLE activity_audit_log (
    log_id number generated always as identity,
    table_audited varchar2(30),
    dml_action varchar2(10),
    row_id varchar2(100),
    action_by varchar2(30),
    action_date timestamp
    );

CREATE or REPLACE Trigger log_user_activity
BEFORE INSERT or UPDATE or DELETE on table_name
For each row
Declare
  v_action varchar2(10);
  Begin
    If inserting then
      v_action := 'INSERT';

    INSERT INTO activity_audit_log (table_audited, dml_action, row_id,
action_by, action_date)
      Values ('Your table', v_action, :new.primary_key, user, systimestamp);
Else if updating then
  v_action := 'UPDATE',

    Insert into activity_audit_log (table_audited, dml_action, row_id,
action_by, action_date)
      values ('your table', v_action, :old.primary_key, user, systimestamp);

  END IF
END>
```

Program 7

Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.

```
CREATE or REPLACE Trigger update_running_total
AFTER INSERT ON orders
For Each row
BEGIN
    UPDATE sales_summary
    SET total_sales = total_sales + new.amount
    WHERE summary_id = 1;
END;
/
```

Program 8

Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.

```
CREATE OR REPLACE TRIGGER validate_stock_level
BEFORE INSERT ON order_items
FOR EACH ROW
DECLARE
  v_stock NUMBER;
BEGIN
    SELECT stock_level
    INTO v_stock
    FROM products
    WHERE product_id = :new.product_id;
IF v_stock < :new.quantity THEN
    RAISE_APPLICATION_ERROR(-20004, 'Cannot place order. Insufficient
    stock for product' || :new.product_id);
ELSE
    UPDATE products
    SET stock_level = stock_level - :new.quantity
    WHERE product_id = :new.product_id;
  END IF;
END;
/
```

| Evaluation Procedure | Marks awarded |
|---|---|
| PL/SQL Procedure(5) | |
| Program/Execution (5) | |
| Viva(5) | |
| Total (15) | |
| Faculty Signature | |