

EXERCISE-17TRIGGERDEFINITION

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 trig - prevent-parent-delete
BEFORE DELETE ON department
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM employee
    WHERE dept_id = :OLD.dept_id;
    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Can't delete
        parent record Employee table. ');
    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 trg_check_email
BEFORE INSERT OR UPDATE ON students
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM students
    WHERE email = :NEW.email;
    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Duplicate
        email detected, each email must be unique);
    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 trig_total_salary
BEFORE INSERT ON employee
FOR EACH ROW
DECLARE
    v_total NUMBER;
    v_threshold CONSTANT NUMBER := 100000;
BEGIN
    SELECT NVL (SUM(salary), 0) INTO v_total FROM
    employee;
    IF (v_total + :NEW salary) > v_threshold THEN
        RAISE_APPLICATION_ERROR (-20003, 'Total salary
        exceeds allowed threshold. ');
    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 employee_audit (  
    emp_id NUMBER,  
    old_salary NUMBER,  
    new_salary NUMBER,  
    change_date DATE,  
    changed_by VARCHAR2 (50),  
);  
  
CREATE OR REPLACE TRIGGER trig_audit_salary  
AFTER UPDATE OF salary ON employee  
FOR EACH ROW  
BEGIN  
    VALUES (: OLD.emp_id; OLD salary; NEW  
salary, SYSDATE, USER);  
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.

```
CREATE TABLE activity_log (  
    table_name VARCHAR2(50),  
    operation_type VARCHAR2(20),  
    user_name VARCHAR2(50),  
    activity_date DATE );
```

```
CREATE OR REPLACE TRIGGER trg-user-act  
AFTER INSERT OR UPDATE OR DELETE ON employee  
BEGIN
```


```
    VALVER ( EMPLOYEE , ORA - SYSEVENT, USER, SYSDATE )  
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 TABLE sales(  
    sale_id NUMBER,  
    amount NUMBER,  
    running_total NUMBER  
);  
  
CREATE OR REPLACE TRIGGER trig_update  
AFTER INSERT ON sales  
FOR EACH ROW  
DECLARE  
    v_total NUMBER;  
BEGIN  
    SELECT NVL(SUM(amount), 0) INTO v_total  
    FROM sales;  
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 TABLE OR REPLACE trig-check,act  
BEFORE INSERT ON ORDER  
FOR EACH ROW  
DECLARE  
    v-stock NUMBER;  
BEGIN  
    SELECT quantity INTO v-stock FROM  
inventory WHERE item-id = :NEW.item-id;  
    IF v-stock < :NEW.order-quantity  
THEN  
    END IF;  
END;
```



<u>Evaluation Procedure</u>	<u>Marks awarded</u>
<u>Practice Evaluation (5)</u>	5
<u>Viva(5)</u>	5
<u>Total (10)</u>	10
<u>Faculty Signature</u>	