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 prevent_del
BEFORE DELETE ON departments
FOR EACH ROW
DECLARE
   v_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_count FROM employees
   WHERE department_id = :OLD.department_id;
   IF v_count >0 THEN
     raise_application_error(-20001,'Cannot delete department with
                                  employees');
   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 no_dup_email
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
   v_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_count FROM employees
   WHERE email = :NEW.email;
   IF v_count > 0 THEN
      raise_application_error(-20002,'Duplicate email 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 salary_limit
BEFORE INSERT ON employees
DECLARE
   v_total NUMBER;
BEGIN
   SELECT SUM(salary) INTO v_total FROM employees;
   IF v_total + :NEW.salary >1000000 THEN
        raise_application_error(-20003,'Salary limit exceeded!');
   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 emp-audit (
    emp_id NUMBER,
    old_salary NUMBER, new_salary NUMBER,
    changed_on DATE
);

CREATE OR REPLACE TRIGGER audit_salary
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
   INSERT INTO emp_audit VALUES
   (:OLD.employee_id, :OLD.salary, :NEW.salary, SYSDATE);
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 alog (
    username VARCHAR2(30),
    action VARCHAR2(10),
    table_name VARCHAR2(30);
    action_date DATE
);
CREATE OR REPLACE TRIGGER rec_act
AFTER INSERT OR UPDATE OR DELETE ON employees
BEGIN
   INSERT INTO alog VALUES (USER, ORA_SYSEVENT,
                        'EMPLOYEES', 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,
    total_amount NUMBER
);

CREATE OR REPLACE TRIGGER calc_total
AFTER INSERT ON sales
FOR EACH ROW
BEGIN
    UPDATE sales SET total_amount = total_amount + :NEW.amount
    WHERE sale_id = :NEW.sale_id;
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 check_stock
BEFORE INSERT ON orders
FOR EACH ROW
DECLARE
    v_stock NUMBER;
BEGIN
    SELECT stock INTO v_stock FROM items
    WHERE item_id = :NEW.item_id;
    IF v_stock < :NEW.qty THEN
        raise_application_error(-20004, 'Not enough stock');
    END IF;
END;
/
```