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_parent_delete
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.parent_primary_key;
  IF v_child_count > 0 THEN
      RAISE_APPLICATION_ERROR (-20001, 'Cannot delete parent
row : child records exist!');
   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 some_unique_column = :new.some_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
AFTER INSERT ON your_table
DECLARE
    v_total NUMBER;
    v_threshold NUMBER := 100000;
BEGIN
    SELECT SUM (some_column)
    INTO v_total
    FROM your_table;

    IF v_total > v_threshold THEN
        RAISE_APPLICATION_ERROR (-20003, 'Insertion failed: Total
exceeds 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 column_audit_log (
    log_id NUMBER GENERATED AS IDENTITY,
    table_audited VARCHAR2(30),
    column_audited VARCHAR2(30),
    row_pk VARCHAR2(100),
    old_value VARCHAR2(1000),
    new_value VARCHAR2(1000),
    changed_by VARCHAR2(30),
    change_date TIMESTAMP
);
CREATE OR REPLACE TRIGGERE 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.primark_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)
    VALUES ('your_table', 'job_id', : old.primary_key, :old.job_id,
: new.job_id, USER, SYSTIMESTAMP);
    END IF;
END;
/
```

200

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 AS IDENTITY,
    table_audited VARCHAR2(30),
    dml_action VARCHAR2(10),
    row_pk VARCHAR2(100),
    action_by VARCHAR2(30),
    action_date TIMESTAMP
);
CREATE OR REPLACE TRIGGER log_user_activity
AFTER INSERT OR UPDATE OR DELETE ON your_table
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_pk, action_by, action_date)
        VALUES ('your_table', v_action, : new. primary_key, USER,
SYSTIMESTAMP);
```

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
  ELSE
    UPDATE products
    SET stock_level = stock_level = :new.quantity
    WHERE product_id = :new.product_id;
  END IF;
END;
```