Practical 09 Part 02

Shivansh Iohani
23070521141

Aim: Write and execute triggers using PL/SQL.

1. BEFORE INSERT Trigger (Row Level)

Goal: Print a message before inserting into Employee.

```
CREATE OR REPLACE TRIGGER trg_before_insert_emp
BEFORE INSERT ON Employee
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Inserting employee: ' ||
 :NEW.Name);
END;
/
```

Test:

```
INSERT INTO Employee VALUES (201, 'Ravi', 'Clerk');
```

```
SQL> CREATE OR REPLACE TRIGGER trg_before_insert_emp BEFORE INSERT ON Employee
  2  FOR EACH ROW
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE('Inserting employee: ' || :NEW.Name);
  5  END;
  6  /

Trigger created.
```

2. AFTER UPDATE Trigger (Row Level)

Goal: Show old and new balances after update.

```
CREATE OR REPLACE TRIGGER
trg_after_update_account
```

```
AFTER UPDATE ON Account
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Account updated from ' ||
:OLD.Balance || ' to ' || :NEW.Balance); END;
/
```

Test:

```
UPDATE Account SET Balance = 3000 WHERE
AccountID = 1001;
```

```
SQL> CREATE OR REPLACE TRIGGER
  2  trg_after_update_account
  3  AFTER UPDATE ON Account
  4  FOR EACH ROW
  5  BEGIN
  6  DBMS_OUTPUT.PUT_LINE('Account updated from ' || :OLD.Balance || ' to ' || :NEW.Balance); END;
  7  /

Trigger created.
```

3. BEFORE DELETE Trigger (Row Level)

Goal: Prevent deleting accounts with balance > 0.

```
CREATE OR REPLACE TRIGGER
trg_block_high_bal_delete
BEFORE DELETE ON Account
FOR EACH ROW
BEGIN
    IF :OLD.Balance > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot
delete account with positive balance'); END IF;
END;
/
```

Test:

```
DELETE FROM Account WHERE AccountID = 1001;
```

```
SQL> CREATE OR REPLACE TRIGGER
  2  trg_block_high_bal_delete
  3  BEFORE DELETE ON Account
  4  FOR EACH ROW
  5  BEGIN
  6  IF :OLD.Balance > 0 THEN
  7  RAISE_APPLICATION_ERROR(-20001, 'Cannot delete account with positive balance'); END IF;
  8  END;
  9  /

Trigger created.
```

4. AFTER UPDATE (Statement-Level Trigger)

Goal: Notify that an update has occurred.

```
CREATE OR REPLACE TRIGGER
trg_stmt_update_customer
AFTER UPDATE ON Customer
BEGIN
    DBMS_OUTPUT.PUT_LINE('Customer table has been
updated.');
END;
/
```

Test:

```
UPDATE Customer SET Name = 'Ajay' WHERE
CustomerID = 1;
```

```
SQL> CREATE OR REPLACE TRIGGER
  2  trg_stmt_update_customer
  3  AFTER UPDATE ON Customer
  4  BEGIN
  5  DBMS_OUTPUT.PUT_LINE('Customer table has been updated.');
  6  END;
  7  /

Trigger created.
```

5. INSTEAD OF Trigger on View

```
-- Create view
CREATE OR REPLACE VIEW acc_view AS
SELECT AccountID, Balance FROM Account;

-- Trigger to allow updates
CREATE OR REPLACE TRIGGER trg_update_acc_view
INSTEAD OF UPDATE ON acc_view
FOR EACH ROW
BEGIN
    UPDATE Account
    SET Balance = :NEW.Balance
    WHERE AccountID = :OLD.AccountID;
END;
/
```

Test:

```
UPDATE acc_view SET Balance = 6000 WHERE
AccountID = 1001;
```

```
SQL> -- Create view
SQL> CREATE OR REPLACE VIEW acc_view AS SELECT AccountID, Balance FROM Account;

View created.

SQL> -- Trigger to allow updates
SQL> CREATE OR REPLACE TRIGGER trg_update_acc_view INSTEAD OF UPDATE ON acc_view
  2  FOR EACH ROW
  3  BEGIN
  4  UPDATE Account
  5  SET Balance = :NEW.Balance
  6  WHERE AccountID = :OLD.AccountID;
  7  END;
  8  /

Trigger created.
```

Lab Tasks (SQL*Plus Based)

1. **Create** a BEFORE INSERT trigger to print employee department.

```
SQL> -- Trigger Type: BEFORE INSERT (Row Level)
SQL> -- Purpose: Print the department before inserting an employee
SQL> CREATE OR REPLACE TRIGGER trg_before_insert_dept
  2  BEFORE INSERT ON Employee
  3  FOR EACH ROW
  4  BEGIN
  5    DBMS_OUTPUT.PUT_LINE('Department: ' || :NEW.Department);
  6  END;
  7  /

Trigger created.

SQL>
SQL> -- Test Query:
SQL> INSERT INTO Employee (EmpID, Name, Department) VALUES (101, 'Amit', 'HR');

1 row created.
```

Type: BEFORE INSERT (Row Level)

-- Purpose: Print the department of the employee before inserting

-- Test Query: INSERT INTO Employee VALUES (101, 'Amit', 'HR');

2. **Create** an AFTER UPDATE trigger to display old and new account balances.

Type: AFTER UPDATE (Row Level)

-- Purpose: Display old and new balance after account update

-- Test Query: UPDATE Account SET Balance = 5000 WHERE AccountID = 1001;

```
SQL> -- Trigger Type: AFTER UPDATE (Row Level)
SQL> -- Purpose: Display old and new account balances after update
SQL> CREATE OR REPLACE TRIGGER trg_after_update_balance
  2  AFTER UPDATE ON Account
  3  FOR EACH ROW
  4  BEGIN
  5    DBMS_OUTPUT.PUT_LINE('Balance changed from ' || :OLD.Balance || ' to ' || :NEW.Balance);
  6  END;
  7  /

Trigger created.

SQL>
SQL> -- Test Query:
SQL> UPDATE Account SET Balance = 5000 WHERE AccountID = 1001;

1 row updated.
```

3. **Write** a trigger to **prevent deletion** of customers if they have accounts.

Type: BEFORE DELETE (Row Level)

-- Purpose: Prevent deletion of customers who still have accounts

-- Test Query: DELETE FROM Customer WHERE CustomerID = 1;

```
SQL> -- Trigger Type: BEFORE DELETE (Row Level)
SQL> -- Purpose: Prevent deletion of customers if they have accounts
SQL> CREATE OR REPLACE TRIGGER trg_prevent_cust_delete
  2  BEFORE DELETE ON Customer
  3  FOR EACH ROW
  4  DECLARE
  5    v_count NUMBER;
  6  BEGIN
  7    SELECT COUNT(*) INTO v_count FROM Account WHERE CustomerID = :OLD.CustomerID;
  8    IF v_count > 0 THEN
  9      RAISE_APPLICATION_ERROR(-20001, 'Cannot delete customer with existing accounts');
 10    END IF;
 11  END;
 12  /

Trigger created.

SQL>
SQL> -- Test Query:
SQL> DELETE FROM Customer WHERE CustomerID = 1;
DELETE FROM Customer WHERE CustomerID = 1
            *
ERROR at line 1:
ORA-20001: Cannot delete customer with existing accounts
ORA-06512: at "SYSTEM.TRG_PREVENT_CUST_DELETE", line 6
ORA-04088: error during execution of trigger 'SYSTEM.TRG_PREVENT_CUST_DELETE'
```

4. **Create** a **statement-level trigger** to log updates on the Customer table.

Type: AFTER UPDATE (Statement-Level)

-- Purpose: Log a message when the Customer table is updated

-- Test Query: UPDATE Customer SET Name = 'Ajay' WHERE CustomerID = 2;

```
SQL> -- Trigger Type: AFTER UPDATE (Statement Level)
SQL> -- Purpose: Log message when Customer table is updated
SQL> CREATE OR REPLACE TRIGGER trg_stmt_update_customer
  2  AFTER UPDATE ON Customer
  3  BEGIN
  4    DBMS_OUTPUT.PUT_LINE('Customer table has been updated.');
  5  END;
  6  /

Trigger created.

SQL>
SQL> -- Test Query:
SQL> UPDATE Customer SET Name = 'Ajay' WHERE CustomerID = 1;

1 row updated.
```

5. **Write** an **INSTEAD OF trigger** on a view for updating balances. \

Type: INSTEAD OF UPDATE (on View)

-- Purpose: Allow update on view that reflects Account table balances

-- Test Query: UPDATE acc_view SET Balance = 6000 WHERE AccountID = 1001;

```
SQL> -- Step 1: Create View
SQL> CREATE OR REPLACE VIEW acc_view AS
  2  SELECT AccountID, Balance FROM Account;

View created.

SQL>
SQL> -- Step 2: Create INSTEAD OF Trigger
SQL> -- Trigger Type: INSTEAD OF UPDATE on View
SQL> -- Purpose: Update Account table when view is updated
SQL> CREATE OR REPLACE TRIGGER trg_update_acc_view
  2  INSTEAD OF UPDATE ON acc_view
  3  FOR EACH ROW
  4  BEGIN
  5    UPDATE Account
  6    SET Balance = :NEW.Balance
  7    WHERE AccountID = :OLD.AccountID;
  8  END;
  9  /

Trigger created.

SQL>
SQL> -- Test Query:
SQL> UPDATE acc_view SET Balance = 6000 WHERE AccountID = 1001;

1 row updated.
```

Submit the following in PDF

- Submit the .sql script with all trigger definitions.
- Provide screen output showing successful trigger execution.
- Comment on each trigger with its **type**, **purpose**, and **test query**.