**Siddhesh Singh**

**23070521150**

Practical 09 Part 02

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

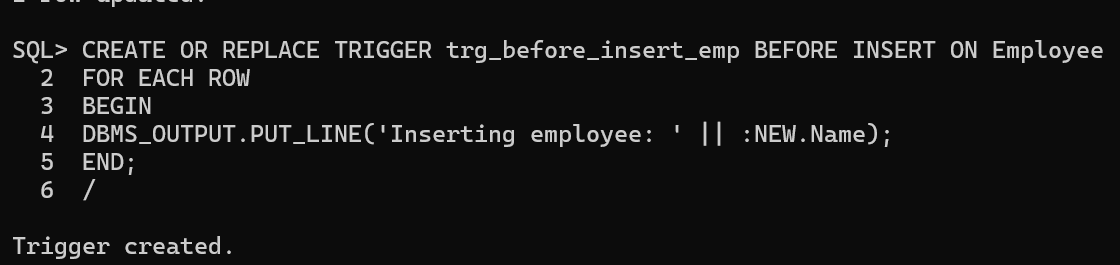
DBMS\_OUTPUT.PUT\_LINE('Inserting employee: ' || :NEW.Name);

END;

/

**Test:**

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



**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

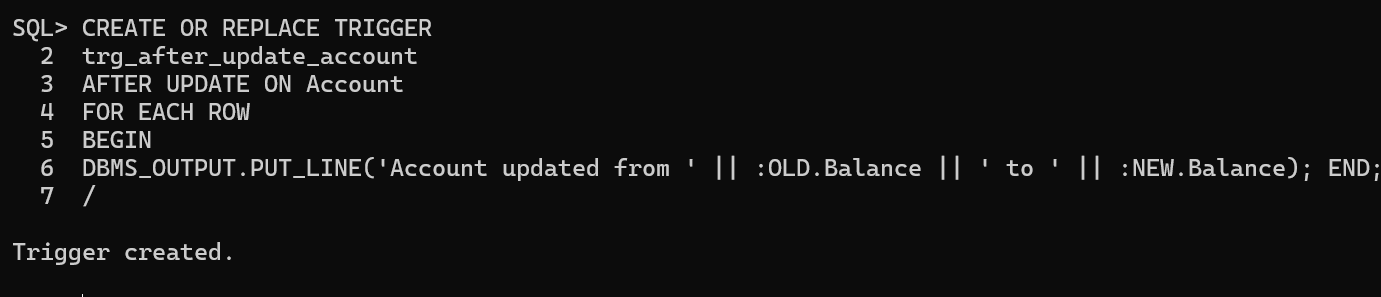
BEGIN

DBMS\_OUTPUT.PUT\_LINE('Account updated from ' || :OLD.Balance || ' to ' || :NEW.Balance); END;

/

**Test:**

UPDATE Account SET Balance = 3000 WHERE AccountID = 1001;



**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

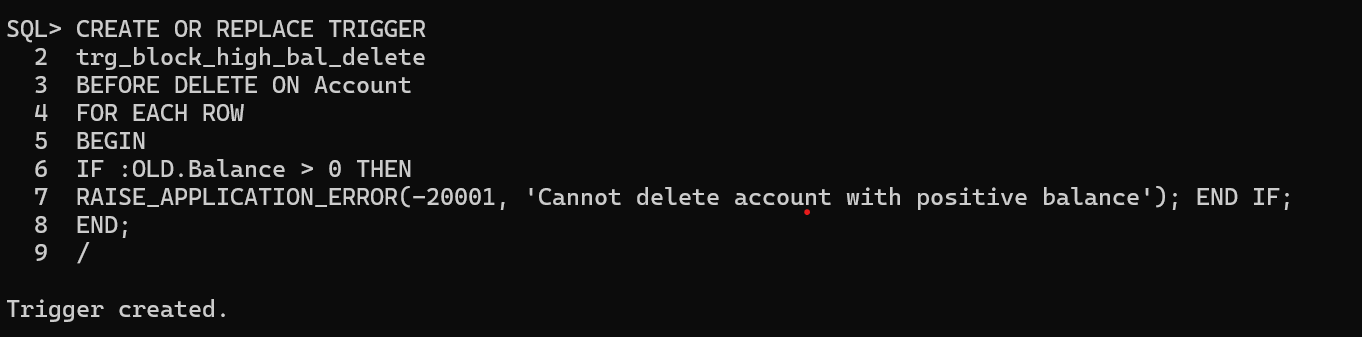
RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete account with positive balance'); END IF;

END;

/

**Test:**

DELETE FROM Account WHERE AccountID = 1001;



**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

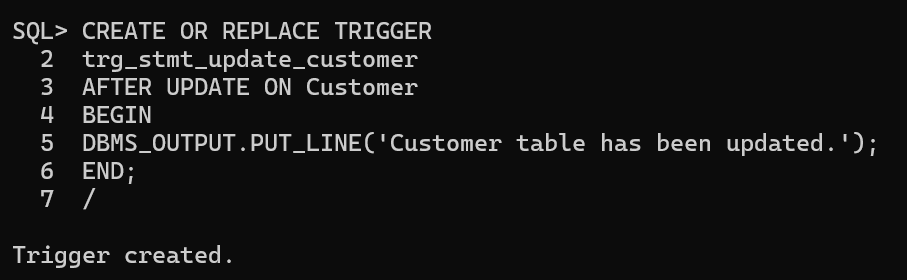
DBMS\_OUTPUT.PUT\_LINE('Customer table has been updated.');

END;

/

**Test:**

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



**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

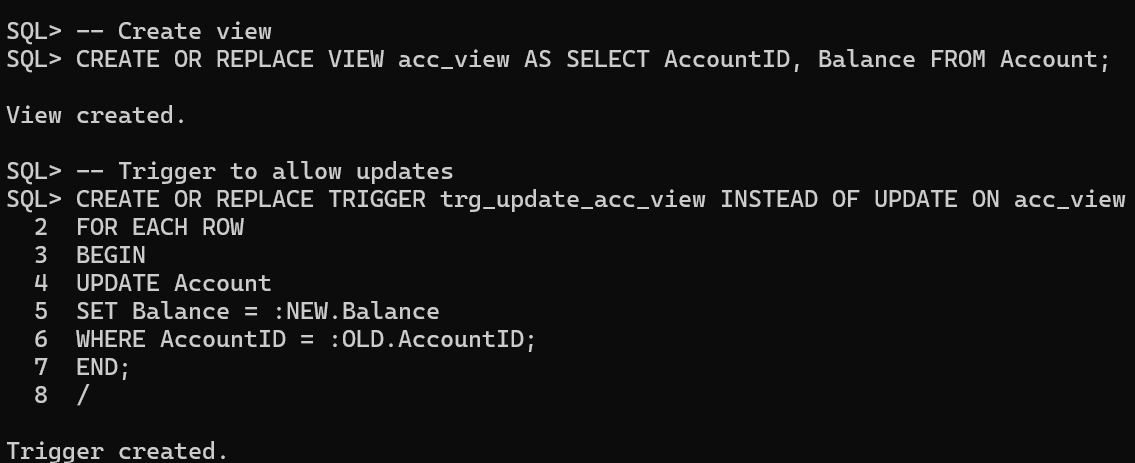
WHERE AccountID = :OLD.AccountID;

END;

/

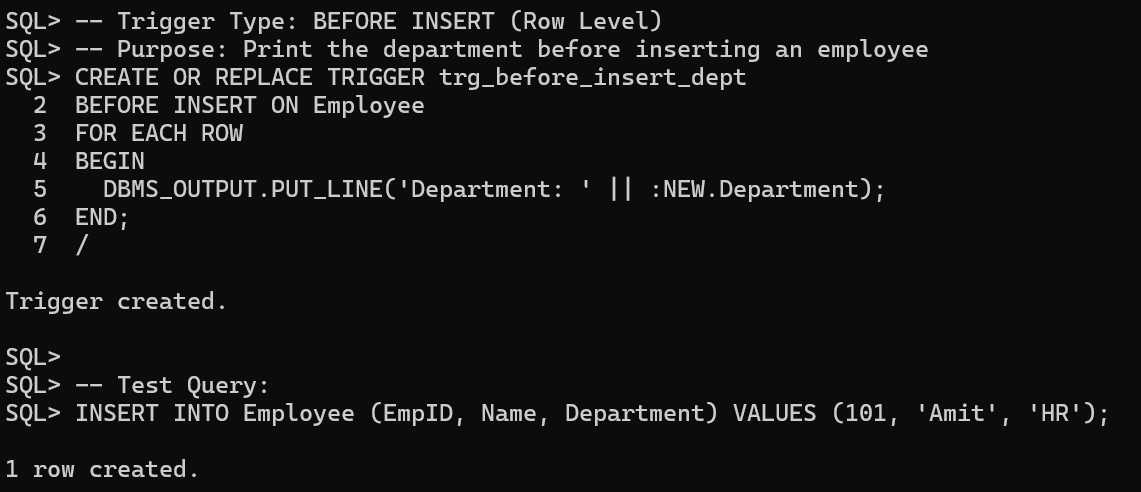
**Test:**

UPDATE acc\_view SET Balance = 6000 WHERE AccountID = 1001;



**Lab Tasks (SQL\*Plus Based)**

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



Type: BEFORE INSERT (Row Level)

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

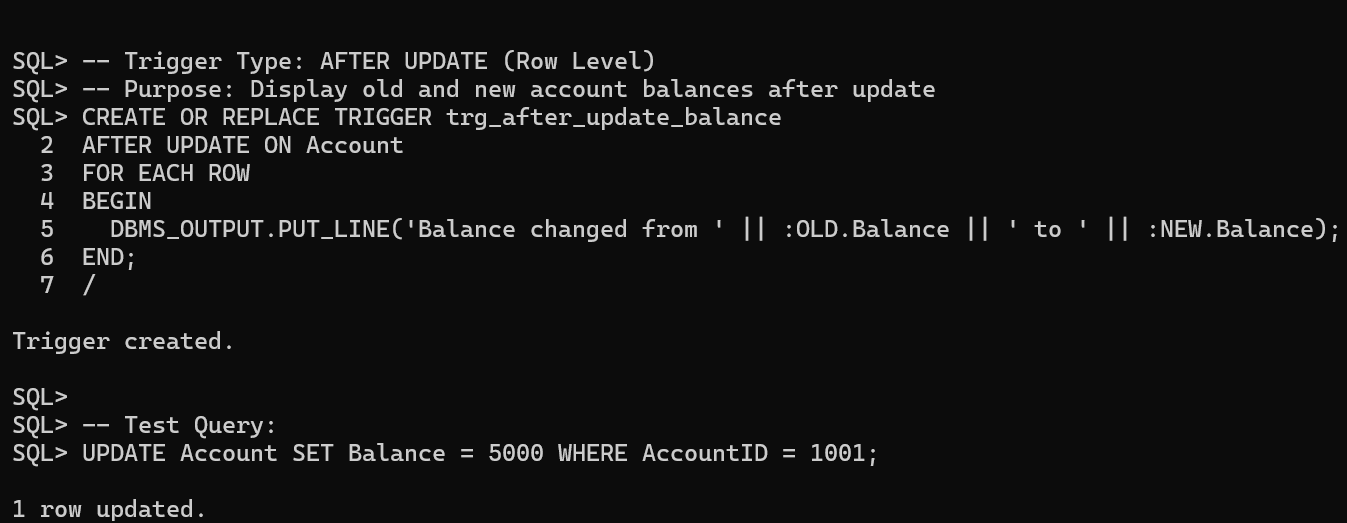
-- Test Query: INSERT INTO Employee VALUES (101, 'Amit', 'HR');

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;

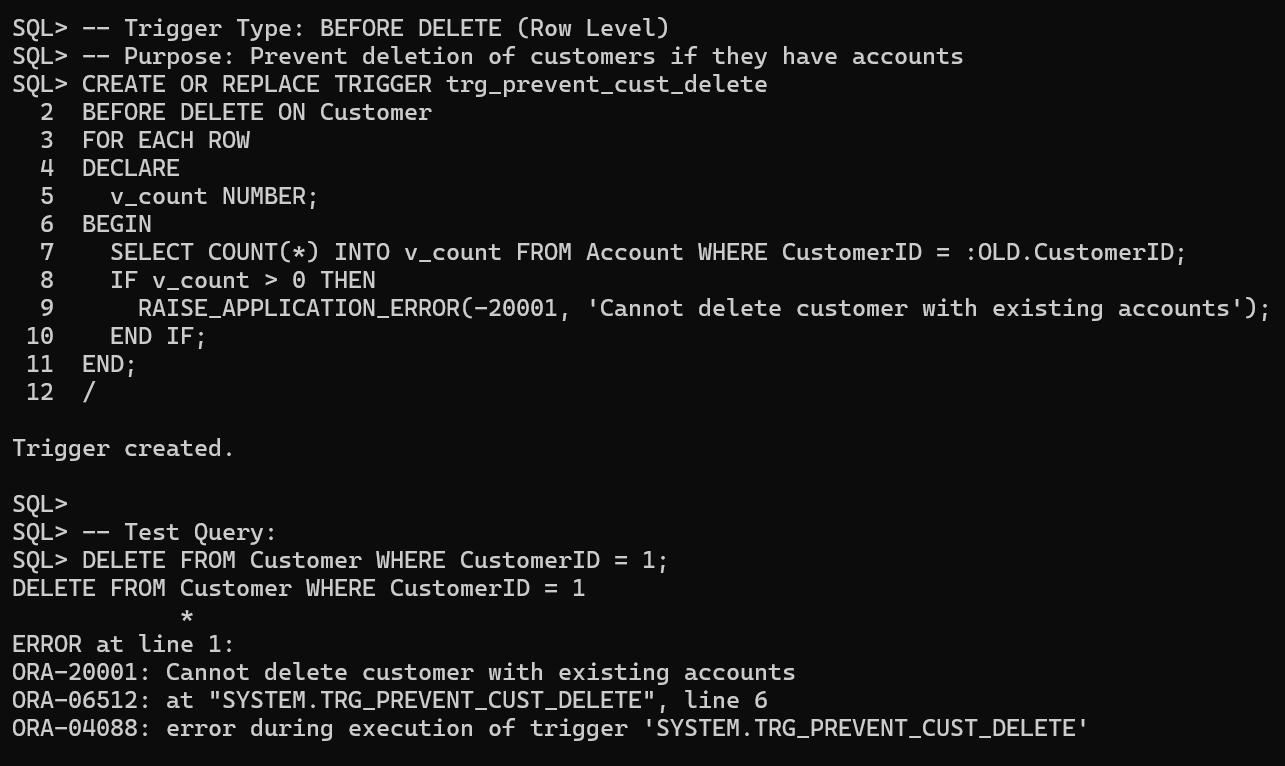


3. **Write** a trigger to **prevent deletion** of customers if they have accounts.

Type: BEFORE DELETE (Row Level)

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

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

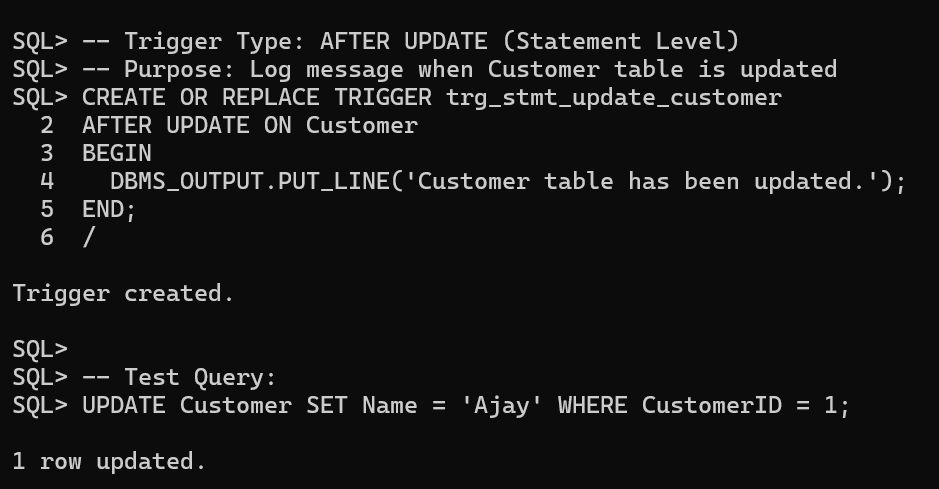


4. **Create** a **statement-level trigger** to log updates on the Customer table.

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;

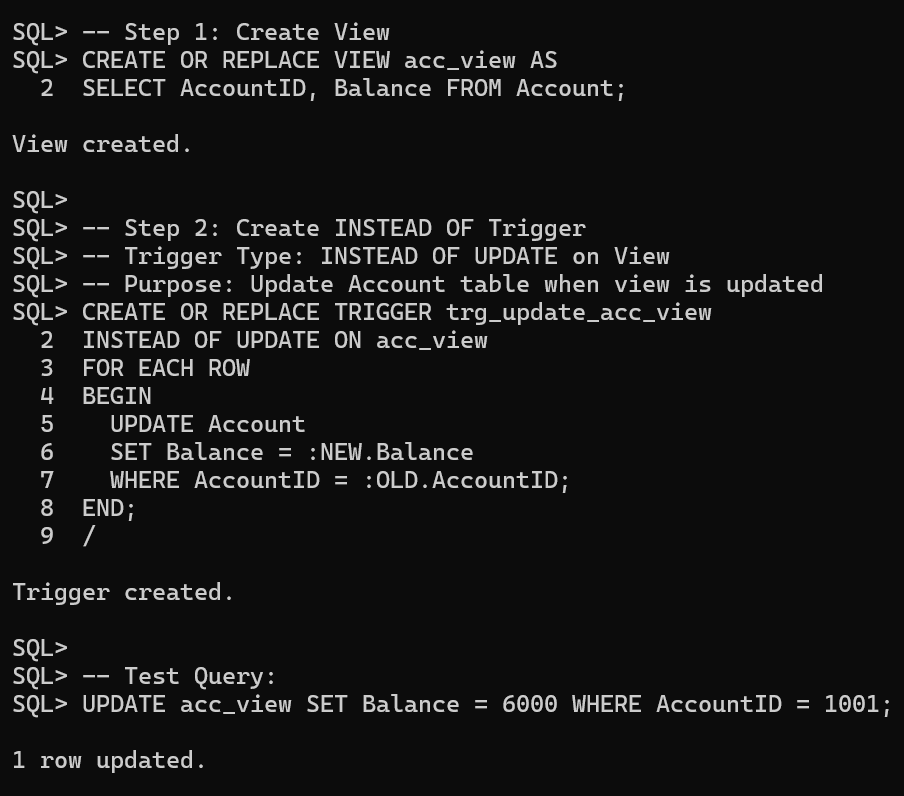


5. **Write** an **INSTEAD OF trigger** on a view for updating balances. \

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;



**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**.