Ex. No.: 13 **WORKING WITH TRIGGER** ARUL JOTHI P

230701034

Initial:

```
CREATE TABLE orders (     order_id
NUMBER PRIMARY KEY,     item_id NUMBER,
quantity NUMBER,     order_date DATE,
running_total
NUMBER,     user_id NUMBER,
   FOREIGN KEY (item_id) REFERENCES items(item_id)
);

INSERT INTO orders (order_id, item_id, quantity, order_date, running_total, user_id)
VALUES (1, 1, 20, SYSDATE, 20, 101);
INSERT INTO orders (order_id, item_id, quantity, order_date, running_total, user_id)
VALUES (2, 2, 30, SYSDATE, 50, 102);

CREATE TABLE items (     item_id
NUMBER PRIMARY KEY,     item_name VARCHAR2(50),
stock_level
NUMBER,     pending_orders NUMBER
DEFAULT 0
);

INSERT INTO items (item_id, item_name, stock_level, pending_orders)
VALUES (1, 'Item A', 100, 0);
INSERT INTO items (item_id, item_name, stock_level, pending_orders)
VALUES (2, 'Item B', 50, 0);
INSERT INTO items (item_id, item_name, stock_level, pending_orders) VALUES (3,
'Item C', 150, 0);
```

```
CREATE TABLE audit_log (     log_id
NUMBER PRIMARY KEY,
table_name VARCHAR2(50),
operation VARCHAR2(10),
   change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
user_id NUMBER,     details VARCHAR2(200)
);

CREATE SEQUENCE audit_log_seq
START WITH 1
INCREMENT BY 1;
```

1. 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 items
FOR EACH ROW DECLARE
child_count NUMBER;
BEGIN
        SELECT COUNT(*) INTO child_count FROM orders
        WHERE item_id = :OLD.item_id;

        IF child_count > 0 THEN
                RAISE_APPLICATION_ERROR(-20001, 'Cannot delete item; dependent
                orders exist.');
        END IF;
END; /
```

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_for_duplicates
BEFORE INSERT OR UPDATE ON orders
```

```
FOR EACH ROW DECLARE
duplicate_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO duplicate_count FROM orders
    WHERE item_id = :NEW.item_id AND order_id != :NEW.order_id;

    IF duplicate_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Duplicate item entry found in
        orders.');
    END IF;
END; /
```

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 restrict_insertion
BEFORE INSERT ON orders
FOR EACH ROW DECLARE
total_quantity NUMBER;
BEGIN
    SELECT SUM(quantity) INTO total_quantity FROM orders;
    IF (total_quantity + :NEW.quantity) > 500 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Cannot insert order; total
        quantity exceeds threshold.');
    END IF;
END; /
```

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 OR REPLACE TRIGGER log_changes
AFTER UPDATE ON orders
FOR EACH ROW
BEGIN
    INSERT INTO audit_log (log_id, table_name, operation, user_id, details) VALUES
```

```
(audit_log_seq.NEXTVAL, 'orders', 'UPDATE', :NEW.user_id, 'Order ' ||
:NEW.order_id || ' changed from ' || :OLD.quantity || ' to ' || :NEW.quantity ); END;
/
```

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 OR REPLACE TRIGGER log_user_activity
AFTER INSERT OR DELETE OR UPDATE ON orders
FOR EACH ROW
BEGIN
    INSERT INTO audit_log (log_id, table_name, operation, user_id, details) VALUES
    (audit_log_seq.NEXTVAL, 'orders',
        CASE
            WHEN INSERTING THEN 'INSERT'
            WHEN UPDATING THEN 'UPDATE'
            WHEN DELETING THEN 'DELETE'
        END,
    NVL(:NEW.user_id, :OLD.user_id), 'User action recorded on order ' ||
    NVL(:NEW.order_id, :OLD.order_id));
END; /
```

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 orders SET running_total = (SELECT SUM(quantity) FROM orders)
    WHERE order_id = :NEW.order_id;
END; /
```

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_item_availability
BEFORE INSERT ON orders
```

```
        FOR EACH ROW DECLARE
available_stock NUMBER;
        BEGIN
                SELECT stock_level - pending_orders INTO available_stock FROM items
                WHERE item_id = :NEW.item_id;

                IF :NEW.quantity > available_stock THEN
                        RAISE_APPLICATION_ERROR(-20004, 'Insufficient stock available for the
                        order.');
                END IF;

                UPDATE items SET pending_orders = pending_orders + :NEW.quantity
                WHERE item_id = :NEW.item_id;
        END; /
```

Result:

The given programs are performed successfully.