Ex.No.: 13 WORKING WITH TRIGGER
Date :

Program 1
Trigger to Enforce Referential Integrity by Preventing Deletion of a Parent Record if Child Records Exist
```sql
CREATE OR REPLACE TRIGGER prevent_parent_delete
BEFORE DELETE ON parent_table
FOR EACH ROW
DECLARE
   child_count NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO child_count
   FROM child_table
   WHERE parent_id = :OLD.parent_id;

   IF child_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Cannot delete parent record as child records
exist.');
   END IF;
END prevent_parent_delete;
/
```

Program 2
Trigger to Check for Duplicate Values in a Specific Column and Raise an Exception if Found
```sql
CREATE OR REPLACE TRIGGER check_duplicate_column
BEFORE INSERT OR UPDATE ON table_name
FOR EACH ROW
DECLARE
   duplicate_count NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO duplicate_count
   FROM table_name
   WHERE column_name = :NEW.column_name
     AND ROWID != :NEW.ROWID;
   IF duplicate_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Duplicate value found in column_name.');
   END IF;
END check_duplicate_column;
/
```

Program 3
Trigger to Restrict Insertion of New Rows if the Total of a Column’s Values Exceeds a Certain Threshold
```sql
CREATE OR REPLACE TRIGGER restrict_insert_on_threshold
BEFORE INSERT ON table_name
FOR EACH ROW
DECLARE
   column_total NUMBER;
   threshold NUMBER := 100000; -- Set the threshold limit here
BEGIN
   SELECT SUM(column_name)
   INTO column_total
   FROM table_name;

   IF (column_total + :NEW.column_name) > threshold THEN
      RAISE_APPLICATION_ERROR(-20003, 'Cannot insert as the column total exceeds the threshold.');
   END IF;
END restrict_insert_on_threshold;
/
```

Program 4
Trigger to Capture Changes Made to Specific Columns and Log Them in an Audit Table
```sql
CREATE OR REPLACE TRIGGER capture_changes
AFTER UPDATE ON target_table
FOR EACH ROW
BEGIN
   IF :OLD.column1 != :NEW.column1 OR :OLD.column2 != :NEW.column2 THEN
      INSERT INTO audit_table (record_id, old_value1, new_value1, old_value2, new_value2, change_date)
```

~~~~
        VALUES (:OLD.id, :OLD.column1, :NEW.column1, :OLD.column2, :NEW.column2,
SYSDATE);
    END IF;
END capture_changes;
/
```

---

### Program 5
**Trigger to Record User Activity (Inserts, Updates, Deletes) in an Audit Log**
```sql
CREATE OR REPLACE TRIGGER record_user_activity
AFTER INSERT OR UPDATE OR DELETE ON target_table
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO audit_log (activity_type, record_id, activity_date, user_id)
        VALUES ('INSERT', :NEW.id, SYSDATE, USER);
    ELSIF UPDATING THEN
        INSERT INTO audit_log (activity_type, record_id, activity_date, user_id)
        VALUES ('UPDATE', :OLD.id, SYSDATE, USER);
    ELSIF DELETING THEN
        INSERT INTO audit_log (activity_type, record_id, activity_date, user_id)
        VALUES ('DELETE', :OLD.id, SYSDATE, USER);
    END IF;
END record_user_activity;
/
```

---

### Program 7
**Trigger to Automatically Calculate and Update a Running Total Column Whenever New Rows
Are Inserted**
```sql
CREATE OR REPLACE TRIGGER update_running_total
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
    UPDATE table_name
    SET running_total_column = NVL(running_total_column, 0) + :NEW.value_column
    WHERE id = :NEW.id;
END update_running_total;
~~~~

/

Program 8
Trigger to Validate Availability of Items Before Allowing an Order to Be Placed (Considering Stock Levels and Pending Orders)
```sql
CREATE OR REPLACE TRIGGER validate_stock_availability
BEFORE INSERT ON orders
FOR EACH ROW
DECLARE
   available_stock NUMBER;
   pending_orders NUMBER;
BEGIN
   -- Check current stock
   SELECT stock_quantity INTO available_stock FROM inventory WHERE item_id =
:NEW.item_id;

   -- Calculate total pending orders for this item
   SELECT SUM(quantity) INTO pending_orders FROM orders WHERE item_id =
:NEW.item_id AND status = 'PENDING';

   IF (available_stock - pending_orders - :NEW.quantity) < 0 THEN
      RAISE_APPLICATION_ERROR(-20004, 'Not enough stock available to fulfill this order.');
   END IF;
END validate_stock_availability;
/
```