Exercise 18

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.

PL/SQL Code:

```
CREATE TABLE departments (
department_id NUMBER PRIMARY KEY,
department_name VARCHAR2(50)
);
 CREATE TABLE employees (
employee_id NUMBER PRIMARY KEY,
first_name VARCHAR2(50),
last_name VARCHAR2(50),
department_id NUMBER,
    CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES departments (department_id)
);
```

```
CREATE OR REPLACE TRIGGER prevent_parent_deletion
BEFORE DELETE ON departments
FOR EACH ROW
DECLARE
v_count NUMBER;
BEGIN
    -- Check if there are any associated child records
    SELECT COUNT(*) INTO v_count FROM employees WHERE department_id = :OLD.department_id;

    -- If child records exist, raise an error
    IF v_count> 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot delete department with associated employees.');
    END IF;
END;
/
INSERT INTO departments (department_id, department_name) VALUES (1, 'Sales');
INSERT INTO employees (employee_id, first_name, last_name, department_id) VALUES (1, 'John', 'Doe', 1);

DELETE FROM departments WHERE department_id = 1; --This will raise an error
```

Output:

```
ORA-20001: Cannot delete department with associated employees.
ORA-06512: at "WKSP_CHUTTI.PREVENT_PARENT_DELETION", line 9
ORA-04088: error during execution of trigger 'WKSP_CHUTTI.PREVENT_PARENT_DELETION'
```

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.

PL/SQL Code:

```
CREATE TABLE products (product_id NUMBER PRIMARY KEY,product_name VARCHAR2(50));

CREATE OR REPLACE TRIGGER prevent_duplicates BEFORE INSERT ON products FOR EACH ROW
DECLARE
v_count NUMBER;
BEGIN
    -- Check if the new product_name already exists
    SELECT COUNT(*) INTO v_count FROM products WHERE product_name = :NEW.product_name;
        -- If duplicate value found, raise an error
    IF v_count> 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Product name already exists.');
    END IF;
END;
/
INSERT INTO products (product_id, product_name) VALUES (1, 'Widget');
INSERT INTO products (product_id, product_name) VALUES (2, 'Widget'); -- This will raise an error
```

Output:

```
Error at line 1/69: ORA-00933: SQL command not properly ended


1. INSERT INTO products (product_id, product_name) VALUES (2, 'Widget'); -- This
will raise an error
```

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.

PL/SQL Code:

```
CREATE TABLE orders (order_id NUMBER PRIMARY KEY,customer_id NUMBER,order_amount NUMBER);
CREATE OR REPLACE TRIGGER check_order_amount
BEFORE INSERT ON orders
FOR EACH ROW
DECLARE
total_amount NUMBER;
max_threshold NUMBER := 10000; -- Change this to your desired threshold
BEGIN
    -- Calculate the current total order amount for the customer
    SELECT NVL(SUM(order_amount), 0) INTO total_amount
    FROM orders
    WHERE customer_id= :NEW.customer_id;

    -- Check if inserting the new row will exceed the threshold
    IF total_amount+ :NEW.order_amount>max_threshold THEN
        RAISE_APPLICATION_ERROR(-20001, 'Total order amount exceeds the threshold.');
    END IF;
END;
/
```

```
-- Inserting rows that don't exceed the threshold
INSERT INTO orders (order_id, customer_id, order_amount) VALUES (1, 101, 5000);
INSERT INTO orders (order_id, customer_id, order_amount) VALUES (2, 101, 3000);
INSERT INTO orders (order_id, customer_id, order_amount) VALUES (3, 102, 8000);
-- Attempting to insert a row that would exceed the threshold
-- This should raise an error and prevent the insertion
BEGIN
    INSERT INTO orders (order_id, customer_id, order_amount) VALUES (4, 102, 5000);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

Output:

Results | Explain | Describe | Saved SQL | History

```
Error: ORA-20001: Total order amount exceeds the threshold.
ORA-06512: at "WKSP_CHUTTI.CHECK_ORDER_AMOUNT", line 12
ORA-04088: error during execution of trigger 'WKSP_CHUTTI.CHECK_ORDER_AMOUNT'
```

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.

PL/SQL Code:

```
-- Create the salary_audit table to store changes
CREATE TABLE salary_audit (audit_id NUMBER PRIMARY KEY,employee_id NUMBER,old_salary NUMBER,new_salary NUMBER,change_date TIMESTAMP);
-- Create a sequence for generating unique audit IDs
CREATE SEQUENCE seq_salary_audit START WITH 1 INCREMENT BY 1;

-- Create a sequence for generating unique audit IDs
CREATE SEQUENCE seq_salary_audit START WITH 1 INCREMENT BY 1;
```

```
-- Create a trigger to capture changes in salary
CREATE OR REPLACE TRIGGER salary_change_auditAFTER UPDATE ON employees FOR EACH ROW
WHEN (NEW.salary<>OLD.salary) -- Only capture changes in the salary column
DECLARE
v_audit_id NUMBER;
BEGIN
    -- Generate a unique audit ID
    SELECT seq_salary_audit.NEXTVAL INTO v_audit_id FROM DUAL;

    -- Insert the change details into the audit table
    INSERT INTO salary_audit (audit_id, employee_id, old_salary, new_salary, change_date)
    VALUES (v_audit_id, :OLD.employee_id, :OLD.salary, :NEW.salary, SYSTIMESTAMP);
END;
/
```

```
-- Inserting a sample employee record
INSERT INTO employees (employee_id, employee_name, salary)
VALUES (1, 'John Doe', 50000);

-- Updating the salary of the employee
UPDATE employees SET salary = 60000 WHERE employee_id = 1;

SELECT * FROM salary_audit;
```

Output:

Results | Explain | Describe | Saved SQL | History

AUDIT_ID	EMPLOYEE_ID	OLD_SALARY	NEW_SALARY	CHANGE_DATE
1	1	50000	60000	20-MAY-24 12.58.34.534932 PM

1 rows returned in 0.02 seconds Download

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.

PL/SQL Code:

```
CREATE TABLE Employee (emp_id NUMBER PRIMARY KEY,emp_name VARCHAR2(100),emp_salary NUMBER);
-- Create Audit_Log table
CREATE TABLE Audit_Log (log_id NUMBER PRIMARY KEY,table_name VARCHAR2(100),activity_type VARCHAR2(20),
activity_date TIMESTAMP,user_id VARCHAR2(50));
CREATE SEQUENCE Audit_Seq START WITH 1 INCREMENT BY 1;
CREATE OR REPLACE TRIGGER Employee_Audit_Trigger AFTER INSERT OR UPDATE OR DELETE ON Employee
FOR EACH ROW
DECLARE
v_activity_type VARCHAR2(20);
BEGIN
    IF INSERTING THEN
v_activity_type := 'INSERT';
    ELSIF UPDATING THEN
v_activity_type := 'UPDATE';
    ELSIF DELETING THEN
v_activity_type := 'DELETE';
    END IF;
    INSERT INTO Audit_Log (log_id, table_name, activity_type, activity_date, user_id)
    VALUES (Audit_Seq.NEXTVAL, 'Employee', v_activity_type, SYSTIMESTAMP, USER);
END;
/
```

```
-- Insert a new employee
INSERT INTO Employee (emp_id, emp_name, emp_salary)VALUES (1, 'John Doe', 50000);
-- Update an employee's salary
UPDATE Employee SET emp_salary = 55000 WHERE emp_id = 1;
-- Delete an employee
DELETE FROM Employee WHERE emp_id = 1;
SELECT * FROM Audit_Log;
```

Output:

```
Table created.
Table created.
Sequence created.
Trigger created.
1 row(s) inserted.
1 row(s) updated.
1 row(s) deleted.
Result Set 1
LOG_ID  TABLE_NAME      ACTIVITY_TYPE   ACTIVITY_DATE   USER_ID
1       Employee        INSERT  18-AUG-23 12.40.22.286572 PM    APEX_PUBLIC_USER
2       Employee        UPDATE  18-AUG-23 12.40.22.297518 PM    APEX_PUBLIC_USER
3       Employee        DELETE  18-AUG-23 12.40.22.301028 PM    APEX_PUBLIC_USER
```

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.

PL/SQL Code:

```
CREATE TABLE Sales (sale_id NUMBER PRIMARY KEY,sale_date DATE,amount NUMBER,running_total NUMBER);
CREATE OR REPLACE TRIGGER Update_Running_Total
BEFORE INSERT ON Sales
FOR EACH ROW
BEGIN
IF :NEW.running_total IS NULL THEN
        SELECT NVL(MAX(running_total), 0) + :NEW.amount
INTO :NEW.running_total
        FROM Sales;
    ELSE
:NEW.running_total := :NEW.running_total + :NEW.amount;
    END IF;
END;
/
INSERT INTO Sales (sale_id, sale_date, amount) VALUES (1, TO_DATE('2023-08-01', 'YYYY-MM-DD'), 100);
INSERT INTO Sales (sale_id, sale_date, amount) VALUES (2, TO_DATE('2023-08-02', 'YYYY-MM-DD'), 200);
INSERT INTO Sales (sale_id, sale_date, amount) VALUES (3, TO_DATE('2023-08-03', 'YYYY-MM-DD'), 150);
SELECT * FROM Sales;
```

Output:

Results | Explain | Describe | Saved SQL | History

SALE_ID	SALE_DATE	AMOUNT	RUNNING_TOTAL
1	08/01/2023	100	100
3	08/03/2023	150	450
2	08/02/2023	200	300

3 rows returned in 0.01 seconds Download

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.

PL/SQL Code:

```
-- Create Products table
CREATE TABLE Products (product_id NUMBER PRIMARY KEY,product_name VARCHAR2(100),stock_quantity NUMBER);

-- Create Orders table
CREATE TABLE Orders (order_id NUMBER PRIMARY KEY,product_id NUMBER,order_quantity NUMBER);
```

```
CREATE OR REPLACE TRIGGER Validate_Order_Availability
BEFORE INSERT ON Orders
FOR EACH ROW
DECLARE
v_current_stock NUMBER;
v_pending_orders NUMBER;
BEGIN
    -- Get current stock for the product
    SELECT stock_quantity INTO v_current_stock
    FROM Products
    WHERE product_id= :NEW.product_id;

    -- Get total quantity of pending orders for the product
    SELECT NVL(SUM(order_quantity), 0) INTO v_pending_orders
    FROM Orders
    WHERE product_id= :NEW.product_id;
```

```
    -- Calculate total available quantity (stock - pending orders)
    IF v_current_stock - v_pending_orders - :NEW.order_quantity< 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient stock for the order');
    END IF;
END;
/
-- Insert sample data into Products table
INSERT INTO Products (product_id, product_name, stock_quantity)
VALUES (1, 'Product A', 100);

-- Attempt to place an order with insufficient stock
INSERT INTO Orders (order_id, product_id, order_quantity)
VALUES (1, 1, 150);
-- This should fail due to insufficient stock
-- Place an order within available stock
INSERT INTO Orders (order_id, product_id, order_quantity)
VALUES (2, 1, 50);
-- This should succeed
-- Query the Orders table to see the placed orders
SELECT * FROM Orders;
```

Output:

```
ORA-20001: Insufficient stock for the order
ORA-06512: at "WKSP_CHUTTI.VALIDATE_ORDER_AVAILABILITY", line 17
ORA-04088: error during execution of trigger
'WKSP_CHUTTI.VALIDATE_ORDER_AVAILABILITY'


1. INSERT INTO Orders (order_id, product_id, order_quantity)
2. VALUES (1, 1, 150);
```