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 dept
for each row
Declare
    vcount NUMBER;
BEGIN
    Select Count(*) into v_count from emp
    where deptno = :OLD.deptno;

    IF v_count > 0 then
        RAISE_APPLICATION_ERROR (-20001, 'Record
    not exist')
    END IF;
END;
/
```

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.

```
Create or replace Trigger
Chck_duplicate_rollno
Before insert or update on Student
for each row
Declare
    v_count number;
Begin
    Select Count(*) into v_count from Student where
    roll_no = :New.roll_no;
    if v_count > 0 then
        Raise_Application_Error(-20002, 'Duplicate Roll
    Number not allowed');
    END IF;
END;
/
```

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.

```
Create table account (
        acc_no NUMBER PRIMARY KEY,
        Cust_name VARCHAR2(30),
        balance NUMBER
);

Create or replace Trigger
check_total_balance
 Before INSERT OR UPDATE ON account
for each row
Declare
        v_total NUMBER;
        v_threshold CONSTANT NUMBER := 100000;

Begin
        Select NVL(SUM(balance),0) INTO v_total from account;
        v_total := v_total + :NEW.balance;
        IF v_total > v_threshold THEN
                RAISE_APPLICATION_ERROR(-20003, 'Total balance
        exceeds allowed limit of 100000');
        END IF;
END;
```

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.

```
Create table employee(
    emp_id NUMBER PRIMARY KEY,
    emp_name VARCHAR2(30),
    Salary NUMBER);

Create TABLE audit_log(
    log_id NUMBER GENERATED ALWAYS AS IDENTITY
    PRIMARY KEY,
    emp_id NUMBER,
    old_Salary NUMBER,
    new_Salary NUMBER,
    changed_on DATE,
    changed_by VARCHAR2(30));

Create or replace trigger Salary_audit_trigger After
    update OF Salary on employee
    for each Row
    Begin
        INSERT INTO audit_log Values (:old.emp_id, :old.Salary,
:New.Salary, SYSDATE, USER);
    END;
/
```

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.

```
Create or replace trigger emp_audit_trigger After insert
or update or delete on employee
BEGIN
    INSERT INTO auditlog values ('EMPLOYEE', ORA_SYSEVENT,
USER, SYSDATE );
END;
/
Create or replace trigger deptaudit_trigger
After insert or update or delete on department
BEGIN
    INSERT INTO audit-log values ('DEPARTMENT',
ORA_SYSEVENT, USER, SYSDATE);
END;
/
```

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.

```
Create Table Sales (
        Sale_id NUMBER PRIMARY KEY,
        Sale_amount NUMBER,
        running_total NUMBER );

Create or replace trigger update_running_total Before
INSERT ON Sales
for each row
declare
    v_total NUMBER;
BEGIN
    select NVL (Sum (Sale_amount), 0) INTO v_total from
Sales;
    :NEW.running_total := v_total + :NEW.Sale_amount;

END;
/
```

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.

```
Create or replace trigger Validate_Stock_Before_order
Before INSERT ON orders for each row
declare
    v_available_qty NUMBER;
BEGIN
    Select stock_qty into v_available_qty from items
where item_id = :NEW.item_id;

    IF :NEW.order_qty > v_available_qty ~~from items~~ then
~~Begin~~ RAISE_APPLICATION_ERROR (-20005, 'Insufficient Stock!
Cannot Place order.');

    END IF;

UPDATE items SET Stock_qty = Stock_qty - :NEW.order_qty
where item_id = :New.item_id;

END;
/
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	5
Program/Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	11/11