**Week 2**

**PL/SQL Programming**

**Exercise 1: Control Structures :-**

For executing the scenarios given we first need to insert sample data   
code : -

CREATE TABLE customers (

  customer\_id NUMBER PRIMARY KEY,

  name VARCHAR2(100),

  age NUMBER,

  balance NUMBER,

  isVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE loans (

  loan\_id NUMBER PRIMARY KEY,

  customer\_id NUMBER,

  interest\_rate NUMBER,

  due\_date DATE,

  FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers VALUES (1, 'Alice', 65, 12000, 'FALSE');

INSERT INTO customers VALUES (2, 'Bob', 45, 8000, 'FALSE');

INSERT INTO customers VALUES (3, 'Charlie', 70, 15000, 'FALSE');

INSERT INTO loans VALUES (101, 1, 7.5, SYSDATE + 15);

INSERT INTO loans VALUES (102, 2, 8.0, SYSDATE + 40);

INSERT INTO loans VALUES (103, 3, 6.5, SYSDATE + 10);

COMMIT;

---------------------------------------------------------------------------------------------------------

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Solution for Scenario 1:**

BEGIN  
  FOR rec IN (  
    SELECT l.loan\_id  
    FROM loans l  
    JOIN customers c ON l.customer\_id = c.customer\_id  
    WHERE c.age > 60  
  ) LOOP  
    UPDATE loans  
    SET interest\_rate = interest\_rate - 1  
    WHERE loan\_id = rec.loan\_id;  
  END LOOP;  
  
  COMMIT;  
END;

**Output :-**

PL/SQL procedure successfully completed.  
  
Elapsed: 00:00:00.106

**---------------------------------------------------------------------------------------------------------------------------------------**

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Solution for Scenario 2:**

BEGIN  
  FOR rec IN (  
    SELECT customer\_id FROM customers WHERE balance > 10000  
  ) LOOP  
    UPDATE customers  
    SET IsVIP = 'TRUE'  
    WHERE customer\_id = rec.customer\_id;  
  END LOOP;  
  
  COMMIT;  
END;

**Output :-**

PL/SQL procedure successfully completed.  
  
Elapsed: 00:00:00.012

**---------------------------------------------------------------------------------------------------------------------------------------------------**

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Solution for Scenario 3:**

BEGIN  
  FOR rec IN (  
    SELECT c.name, l.due\_date  
    FROM customers c  
    JOIN loans l ON c.customer\_id = l.customer\_id  
    WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30  
  ) LOOP  
    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for ' || rec.name ||  
                         ' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));  
  END LOOP;  
END;

**Output :-**

Reminder: Loan for Alice is due on 13-JUL-2025  
Reminder: Loan for Charlie is due on 08-JUL-2025  
  
  
PL/SQL procedure successfully completed.  
  
Elapsed: 00:00:00.018

**Exercise 3: Stored Procedures :-**

For executing the scenarios given we first need to insert sample data   
code : -

- Drop if exists

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

-- Create accounts table

CREATE TABLE accounts (

  account\_id NUMBER PRIMARY KEY,

  account\_type VARCHAR2(20),

  balance NUMBER

);

-- Insert sample data

INSERT INTO accounts VALUES (1001, 'SAVINGS', 15000);

INSERT INTO accounts VALUES (1002, 'CURRENT', 5000);

INSERT INTO accounts VALUES (1003, 'SAVINGS', 25000);

INSERT INTO accounts VALUES (1004, 'SAVINGS', 8000);

INSERT INTO accounts VALUES (1005, 'CURRENT', 10000);

COMMIT;

-- Drop if exists

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

-- Create employees table

CREATE TABLE employees (

  employee\_id NUMBER PRIMARY KEY,

  name VARCHAR2(100),

  department\_id NUMBER,

  salary NUMBER

);

-- Insert Indian employee names

INSERT INTO employees VALUES (1, 'Rajesh Kumar', 101, 50000);

INSERT INTO employees VALUES (2, 'Anita Sharma', 101, 52000);

INSERT INTO employees VALUES (3, 'Vikram Patel', 102, 48000);

INSERT INTO employees VALUES (4, 'Priya Verma', 102, 47000);

INSERT INTO employees VALUES (5, 'Amitabh Joshi', 103, 60000);

COMMIT;

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Solution for Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
  UPDATE accounts  
  SET balance = balance + (balance \* 0.01)  
  WHERE account\_type = 'SAVINGS';  
  
  COMMIT;  
END;

**Output :-**

Procedure PROCESSMONTHLYINTEREST compiled  
  
Elapsed: 00:00:00.016

------------------------------------------------------------------------------

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Solution for Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  
  dept\_id IN NUMBER,  
  bonus\_percent IN NUMBER  
) IS  
BEGIN  
  UPDATE employees  
  SET salary = salary + (salary \* bonus\_percent / 100)  
  WHERE department\_id = dept\_id;  
  
  COMMIT;  
END;

**Output :-**

Procedure UPDATEEMPLOYEEBONUS compiled  
  
Elapsed: 00:00:00.017

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Solution for Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (  
  from\_account\_id IN NUMBER,  
  to\_account\_id IN NUMBER,  
  amount IN NUMBER  
) IS  
  from\_balance NUMBER;  
BEGIN  
  -- Check balance of source account  
  SELECT balance INTO from\_balance  
  FROM accounts  
  WHERE account\_id = from\_account\_id;  
  
  IF from\_balance < amount THEN  
    RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account');  
  END IF;  
  
  -- Debit source account  
  UPDATE accounts  
  SET balance = balance - amount  
  WHERE account\_id = from\_account\_id;  
  
  -- Credit destination account  
  UPDATE accounts  
  SET balance = balance + amount  
  WHERE account\_id = to\_account\_id;  
  
  COMMIT;  
END;

**Output :-**

Procedure TRANSFERFUNDS compiled  
  
Elapsed: 00:00:00.017