**PL/SQL**

**Exercise 1: Control Structures**

**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.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**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.

A screenshot of a computer

AI-generated content may be incorrect.

**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.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3: Stored Procedures**

**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.

A screenshot of a computer

AI-generated content may be incorrect.

**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.

A screenshot of a computer

AI-generated content may be incorrect.

**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

A screenshot of a computer

AI-generated content may be incorrect.

CODE:

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

select \* from employees;

BEGIN

  FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP

    IF (MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12) > 60 THEN

      UPDATE Loans

      SET InterestRate = InterestRate - 1

      WHERE CustomerID = cust.CustomerID;

    END IF;

  END LOOP;

END;

-- Add a customer aged 70

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Senior Customer', TO\_DATE('1954-01-01', 'YYYY-MM-DD'), 800, SYSDATE);

-- Add a loan for that customer

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 3, 7000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

SELECT \* FROM Loans;

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

  FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

    IF cust.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = cust.CustomerID;

    END IF;

  END LOOP;

END;

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Wealthy Customer', TO\_DATE('1970-01-01', 'YYYY-MM-DD'), 20000, SYSDATE);

BEGIN

  FOR loan IN (

    SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan.LoanID ||

                         ' for customer ' || loan.Name ||

                         ' is due on ' || TO\_CHAR(loan.EndDate, 'YYYY-MM-DD'));

  END LOOP;

END;

-- Add a loan that ends in 10 days

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (3, 2, 3000, 4.5, SYSDATE, SYSDATE + 10);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR acc IN (

    SELECT AccountID, Balance FROM Accounts

    WHERE AccountType = 'Savings'

  ) LOOP

    UPDATE Accounts

    SET Balance = acc.Balance + (acc.Balance \* 0.01)

    WHERE AccountID = acc.AccountID;

  END LOOP;

END;

BEGIN

  ProcessMonthlyInterest;

END;

SELECT AccountID, AccountType, Balance FROM Accounts;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  deptName IN VARCHAR2,

  bonusPercent IN NUMBER

) IS

BEGIN

  UPDATE Employees

  SET Salary = Salary + (Salary \* bonusPercent / 100)

  WHERE Department = deptName;

END;

BEGIN

  UpdateEmployeeBonus('IT', 10);

END;

SELECT Name, Department, Salary FROM Employees;

CREATE OR REPLACE PROCEDURE TransferFunds (

  fromAccountID IN NUMBER,

  toAccountID IN NUMBER,

  amount IN NUMBER

) IS

  fromBalance NUMBER;

BEGIN

  -- Get the balance of the source account

  SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID = fromAccountID;

  -- Check if there is enough money

  IF fromBalance >= amount THEN

    -- Subtract from source

    UPDATE Accounts

    SET Balance = Balance - amount

    WHERE AccountID = fromAccountID;

    -- Add to destination

    UPDATE Accounts

    SET Balance = Balance + amount

    WHERE AccountID = toAccountID;

    DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

  ELSE

    DBMS\_OUTPUT.PUT\_LINE('Insufficient funds.');

  END IF;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    DBMS\_OUTPUT.PUT\_LINE('One of the accounts does not exist.');

END;

BEGIN

  TransferFunds(1, 2, 200);  -- Transfer $200 from Account 1 to Account 2

END;

SELECT AccountID, Balance FROM Accounts;