**Exercise 1: Control Structures Scenario**

**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.

DECLARE

    CURSOR c\_customers IS

        SELECT customer\_id, age, loan\_interest\_rate

        FROM customers

        WHERE age > 60;

   v\_customer\_id customers.customer\_id%TYPE;

    v\_age customers.age%TYPE;

   v\_loan\_interest\_rate customers.loan\_interest\_rate%TYPE;

BEGIN

    OPEN c\_customers;

    LOOP

        FETCH c\_customers INTO v\_customer\_id, v\_age, v\_loan\_interest\_rate;

        EXIT WHEN c\_customers%NOTFOUND;

        UPDATE customers

        SET loan\_interest\_rate = v\_loan\_interest\_rate - 1

        WHERE customer\_id = v\_customer\_id;

    END LOOP;

    CLOSE c\_customers;

    COMMIT;

END;

/

**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.

DECLARE

    CURSOR c\_customers IS

        SELECT customer\_id, balance

        FROM customers

        WHERE balance > 10000;

   v\_customer\_id customers.customer\_id%TYPE;

    v\_balance customers.balance%TYPE;

BEGIN

    OPEN c\_customers;

    LOOP

        FETCH c\_customers INTO v\_customer\_id, v\_balance;

        EXIT WHEN c\_customers%NOTFOUND;

        UPDATE customers

        SET IsVIP = TRUE

        WHERE customer\_id = v\_customer\_id;

    END LOOP;

    CLOSE c\_customers;

    COMMIT;

END;

/

**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.

DECLARE

    CURSOR c\_loans IS

        SELECT customer\_id, loan\_id, due\_date

        FROM loans

        WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30;

   v\_customer\_id loans.customer\_id%TYPE;

    v\_loan\_id loans.loan\_id%TYPE;

    v\_due\_date loans.due\_date%TYPE;

BEGIN

    OPEN c\_loans;

    LOOP

        FETCH c\_loans INTO v\_customer\_id, v\_loan\_id, v\_due\_date;

        EXIT WHEN c\_loans%NOTFOUND;

        -- Replace this with actual code to send a reminder

       DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || v\_loan\_id || ' for customer ' || v\_customer\_id || ' is due on ' || TO\_CHAR(v\_due\_date, 'YYYY-MM-DD'));

    END LOOP;

    CLOSE c\_loans;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

   p\_from\_account\_id IN NUMBER,

   p\_to\_account\_id IN NUMBER,

    p\_amount IN NUMBER

)

IS

   v\_from\_balance NUMBER;

    v\_to\_balance NUMBER;

   ex\_insufficient\_funds EXCEPTION;

   ex\_account\_not\_found EXCEPTION;

BEGIN

    -- Get the balance of the from account

    SELECT balance INTO v\_from\_balance

    FROM Accounts

    WHERE AccountID = p\_from\_account\_id

    FOR UPDATE;

    -- Check if the from account has sufficient funds

    IF v\_from\_balance < p\_amount THEN

        RAISE ex\_insufficient\_funds;

    END IF;

    -- Get the balance of the to account

    SELECT balance INTO v\_to\_balance

    FROM Accounts

    WHERE AccountID = p\_to\_account\_id

    FOR UPDATE;

    -- Deduct the amount from the from account

    UPDATE Accounts

    SET Balance = Balance - p\_amount,

       LastModified = SYSDATE

    WHERE AccountID = p\_from\_account\_id;

    -- Add the amount to the to account

    UPDATE Accounts

    SET Balance = Balance + p\_amount,

       LastModified = SYSDATE

    WHERE AccountID = p\_to\_account\_id;

    -- Commit the transaction

    COMMIT;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RAISE ex\_account\_not\_found;

    WHEN ex\_insufficient\_funds THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

    WHEN ex\_account\_not\_found THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('Error: One of the accounts not found.');

    WHEN OTHERS THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END SafeTransferFunds;

/

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

CREATE OR REPLACE PROCEDURE UpdateSalary (

   p\_employee\_id IN NUMBER,

    p\_percentage IN NUMBER

)

IS

   ex\_employee\_not\_found EXCEPTION;

BEGIN

    -- Update the employee's salary

    UPDATE Employees

    SET Salary = Salary + (Salary \* p\_percentage / 100),

       LastModified = SYSDATE

    WHERE EmployeeID = p\_employee\_id;

    IF SQL%ROWCOUNT = 0 THEN

        RAISE ex\_employee\_not\_found;

    END IF;

    -- Commit the transaction

    COMMIT;

EXCEPTION

    WHEN ex\_employee\_not\_found THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' not found.');

    WHEN OTHERS THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END UpdateSalary;

/

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

   p\_customer\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_dob IN DATE,

    p\_balance IN NUMBER

)

IS

   ex\_duplicate\_customer EXCEPTION;

    PRAGMA EXCEPTION\_INIT(ex\_duplicate\_customer, -1); -- Duplicate value exception

BEGIN

    -- Insert the new customer

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

    VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

    -- Commit the transaction

    COMMIT;

EXCEPTION

    WHEN ex\_duplicate\_customer THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_customer\_id || ' already exists.');

    WHEN OTHERS THEN

       ROLLBACK;

       DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END AddNewCustomer;

/

**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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

    -- Update the balance for all savings accounts by applying a 1% interest

    UPDATE Accounts

    SET Balance = Balance \* 1.01,

        LastModified = SYSDATE

    WHERE AccountType = 'Savings';

    -- Commit the transaction

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END ProcessMonthlyInterest;

/

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_percentage IN NUMBER

)

IS

BEGIN

    -- Update the salary for all employees in the given department by adding the bonus percentage

    UPDATE Employees

    SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100),

        LastModified = SYSDATE

    WHERE Department = p\_department;

    -- Commit the transaction

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Bonus updated for all employees in department ' || p\_department);

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END UpdateEmployeeBonus;

/

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_from\_account\_id IN NUMBER,

    p\_to\_account\_id IN NUMBER,

    p\_amount IN NUMBER

)

IS

    v\_from\_balance NUMBER;

    ex\_insufficient\_funds EXCEPTION;

BEGIN

    -- Get the balance of the from account

    SELECT Balance INTO v\_from\_balance

    FROM Accounts

    WHERE AccountID = p\_from\_account\_id

    FOR UPDATE;

    -- Check if the from account has sufficient funds

    IF v\_from\_balance < p\_amount THEN

        RAISE ex\_insufficient\_funds;

    END IF;

    -- Deduct the amount from the from account

    UPDATE Accounts

    SET Balance = Balance - p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_from\_account\_id;

    -- Add the amount to the to account

    UPDATE Accounts

    SET Balance = Balance + p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_to\_account\_id;

    -- Commit the transaction

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Funds transferred from account ' || p\_from\_account\_id || ' to account ' || p\_to\_account\_id);

EXCEPTION

    WHEN ex\_insufficient\_funds THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: One of the accounts not found.');

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

CREATE OR REPLACE FUNCTION CalculateAge (

    p\_dob IN DATE

) RETURN NUMBER

IS

    v\_age NUMBER;

BEGIN

    -- Calculate age

    v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

    RETURN v\_age;

END CalculateAge;

/

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

    p\_loan\_amount IN NUMBER,

    p\_interest\_rate IN NUMBER,

    p\_loan\_duration\_years IN NUMBER

) RETURN NUMBER

IS

    v\_monthly\_rate NUMBER;

    v\_total\_months NUMBER;

    v\_monthly\_installment NUMBER;

BEGIN

    -- Calculate monthly interest rate

    v\_monthly\_rate := p\_interest\_rate / 12 / 100;

    -- Calculate total number of months

    v\_total\_months := p\_loan\_duration\_years \* 12;

    -- Calculate monthly installment using the formula for an annuity

    v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_rate / (1 - POWER(1 + v\_monthly\_rate, -v\_total\_months));

    RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

    p\_account\_id IN NUMBER,

    p\_amount IN NUMBER

) RETURN BOOLEAN

IS

    v\_balance NUMBER;

BEGIN

    -- Get the balance of the account

    SELECT Balance INTO v\_balance

    FROM Accounts

    WHERE AccountID = p\_account\_id;

    -- Check if the balance is sufficient

    IF v\_balance >= p\_amount THEN

        RETURN TRUE;

    ELSE

        RETURN FALSE;

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario 1:**Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

   :NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

CREATE TABLE AuditLog (

    AuditID NUMBER PRIMARY KEY,

   TransactionID NUMBER,

    AccountID NUMBER,

   TransactionDate DATE,

    Amount NUMBER,

   TransactionType VARCHAR2(10),

    LogDate DATE

);

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

    INSERT INTO AuditLog (

        AuditID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate

    ) VALUES (

       AuditLog\_seq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType, SYSDATE

    );

END LogTransaction;

/

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

DECLARE

    CURSOR cur\_transactions IS

        SELECT t.CustomerID, t.TransactionDate, t.Amount, t.TransactionType

        FROM Transactions t

        JOIN Accounts a ON t.AccountID = a.AccountID

        WHERE EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

        AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

   v\_customer\_id Customers.CustomerID%TYPE;

   v\_transaction\_date Transactions.TransactionDate%TYPE;

    v\_amount Transactions.Amount%TYPE;

   v\_transaction\_type Transactions.TransactionType%TYPE;

BEGIN

    OPEN cur\_transactions;

    LOOP

        FETCH cur\_transactions INTO v\_customer\_id, v\_transaction\_date, v\_amount, v\_transaction\_type;

        EXIT WHEN cur\_transactions%NOTFOUND;

       DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);

       DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_transaction\_date);

       DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_amount);

       DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_transaction\_type);

       DBMS\_OUTPUT.PUT\_LINE('------------------------------');

    END LOOP;

    CLOSE cur\_transactions;

END;

/

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

DECLARE

    CURSOR cur\_accounts IS

        SELECT AccountID, Balance

        FROM Accounts;

    v\_account\_id Accounts.AccountID%TYPE;

    v\_balance Accounts.Balance%TYPE;

    v\_annual\_fee CONSTANT NUMBER := 50; -- Annual maintenance fee

BEGIN

    OPEN cur\_accounts;

    LOOP

        FETCH cur\_accounts INTO v\_account\_id, v\_balance;

        EXIT WHEN cur\_accounts%NOTFOUND;

        -- Deduct annual fee

        UPDATE Accounts

        SET Balance = Balance - v\_annual\_fee,

           LastModified = SYSDATE

        WHERE AccountID = v\_account\_id;

       DBMS\_OUTPUT.PUT\_LINE('Annual fee applied to Account ID: ' || v\_account\_id);

    END LOOP;

    CLOSE cur\_accounts;

    -- Commit the transaction

    COMMIT;

END;

/

**Scenario 3:** Update the interest rate for all loans based on a new policy.

* + **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

DECLARE

    CURSOR cur\_accounts IS

        SELECT AccountID, Balance

        FROM Accounts;

    v\_account\_id Accounts.AccountID%TYPE;

    v\_balance Accounts.Balance%TYPE;

    v\_annual\_fee CONSTANT NUMBER := 50; -- Annual maintenance fee

BEGIN

    OPEN cur\_accounts;

    LOOP

        FETCH cur\_accounts INTO v\_account\_id, v\_balance;

        EXIT WHEN cur\_accounts%NOTFOUND;

        -- Deduct annual fee

        UPDATE Accounts

        SET Balance = Balance - v\_annual\_fee,

           LastModified = SYSDATE

        WHERE AccountID = v\_account\_id;

       DBMS\_OUTPUT.PUT\_LINE('Annual fee applied to Account ID: ' || v\_account\_id);

    END LOOP;

    CLOSE cur\_accounts;

    -- Commit the transaction

    COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

CREATE OR REPLACE PACKAGE CustomerManagement AS

    PROCEDURE AddNewCustomer(

       p\_customer\_id IN NUMBER,

        p\_name IN VARCHAR2,

        p\_dob IN DATE,

       p\_balance IN NUMBER

    );

    PROCEDURE UpdateCustomerDetails(

       p\_customer\_id IN NUMBER,

        p\_name IN VARCHAR2,

        p\_dob IN DATE,

       p\_balance IN NUMBER

    );

    FUNCTION GetCustomerBalance(

       p\_customer\_id IN NUMBER

    ) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

    PROCEDURE AddNewCustomer(

       p\_customer\_id IN NUMBER,

        p\_name IN VARCHAR2,

        p\_dob IN DATE,

       p\_balance IN NUMBER

    ) IS

    BEGIN

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

        VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

    END AddNewCustomer;

    PROCEDURE UpdateCustomerDetails(

       p\_customer\_id IN NUMBER,

        p\_name IN VARCHAR2,

        p\_dob IN DATE,

       p\_balance IN NUMBER

    ) IS

    BEGIN

        UPDATE Customers

        SET Name = p\_name,

            DOB = p\_dob,

           Balance = p\_balance,

           LastModified = SYSDATE

        WHERE CustomerID = p\_customer\_id;

    END UpdateCustomerDetails;

    FUNCTION GetCustomerBalance(

       p\_customer\_id IN NUMBER

    ) RETURN NUMBER IS

       v\_balance NUMBER;

    BEGIN

        SELECT Balance INTO v\_balance

        FROM Customers

        WHERE CustomerID = p\_customer\_id;

        RETURN v\_balance;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

           RETURN NULL;

    END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Package Specification (EmployeeManagement.pks)**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

    PROCEDURE HireNewEmployee(

       p\_employee\_id IN NUMBER,

        p\_name IN VARCHAR2,

       p\_position IN VARCHAR2,

        p\_salary IN NUMBER,

       p\_department IN VARCHAR2

    );

    PROCEDURE UpdateEmployeeDetails(

       p\_employee\_id IN NUMBER,

        p\_name IN VARCHAR2,

       p\_position IN VARCHAR2,

        p\_salary IN NUMBER,

       p\_department IN VARCHAR2

    );

    FUNCTION CalculateAnnualSalary(

       p\_employee\_id IN NUMBER

    ) RETURN NUMBER;

END EmployeeManagement;

/

**Package Body (EmployeeManagement.pkb)**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

    PROCEDURE HireNewEmployee(

       p\_employee\_id IN NUMBER,

        p\_name IN VARCHAR2,

       p\_position IN VARCHAR2,

        p\_salary IN NUMBER,

       p\_department IN VARCHAR2

    ) IS

    BEGIN

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

        VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, SYSDATE);

    END HireNewEmployee;

    PROCEDURE UpdateEmployeeDetails(

       p\_employee\_id IN NUMBER,

        p\_name IN VARCHAR2,

       p\_position IN VARCHAR2,

        p\_salary IN NUMBER,

       p\_department IN VARCHAR2

    ) IS

    BEGIN

        UPDATE Employees

        SET Name = p\_name,

           Position = p\_position,

           Salary = p\_salary,

           Department = p\_department

        WHERE EmployeeID = p\_employee\_id;

    END UpdateEmployeeDetails;

    FUNCTION CalculateAnnualSalary(

       p\_employee\_id IN NUMBER

    ) RETURN NUMBER IS

        v\_salary NUMBER;

    BEGIN

        SELECT Salary INTO v\_salary

        FROM Employees

        WHERE EmployeeID = p\_employee\_id;

        RETURN v\_salary \* 12;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

           RETURN NULL;

    END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Package Specification (AccountOperations.pks)**

CREATE OR REPLACE PACKAGE AccountOperations AS

    PROCEDURE OpenNewAccount(

        p\_account\_id IN NUMBER,

        p\_customer\_id IN NUMBER,

        p\_account\_type IN VARCHAR2,

        p\_balance IN NUMBER

    );

    PROCEDURE CloseAccount(

        p\_account\_id IN NUMBER

    );

    FUNCTION GetTotalBalance(

        p\_customer\_id IN NUMBER

    ) RETURN NUMBER;

END AccountOperations;

/

**Package Body (AccountOperations.pkb)**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

    PROCEDURE OpenNewAccount(

        p\_account\_id IN NUMBER,

        p\_customer\_id IN NUMBER,

        p\_account\_type IN VARCHAR2,

        p\_balance IN NUMBER

    ) IS

    BEGIN

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

        VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

    END OpenNewAccount;

    PROCEDURE CloseAccount(

        p\_account\_id IN NUMBER

    ) IS

    BEGIN

        DELETE FROM Accounts

        WHERE AccountID = p\_account\_id;

    END CloseAccount;

    FUNCTION GetTotalBalance(

        p\_customer\_id IN NUMBER

    ) RETURN NUMBER IS

        v\_total\_balance NUMBER;

    BEGIN

        SELECT SUM(Balance) INTO v\_total\_balance

        FROM Accounts

        WHERE CustomerID = p\_customer\_id;

        RETURN v\_total\_balance;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RETURN 0;

    END GetTotalBalance;

END AccountOperations;

/

**Schema to be Created**

*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*

*);*

**Example Scripts for Sample Data Insertion**

*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);*

-- Create Customers table

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Transactions table

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Create Loans table

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Employees table

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

-- Insert sample data into Customers

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 sample data into Accounts

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);