**Exercise 1: Control Structures**

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

**PL/SQL Query:**

BEGIN

FOR rec IN (SELECT \* FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID) LOOP

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

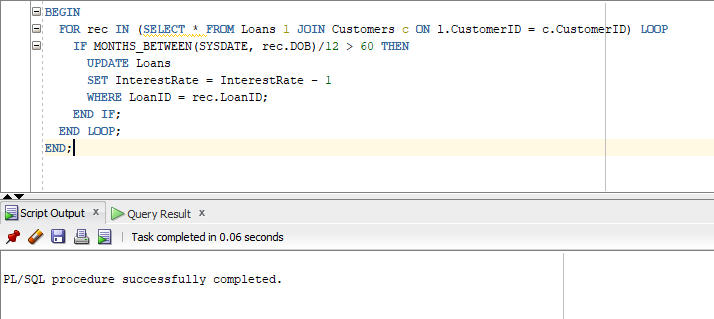
UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

END;

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

**PL/SQL Query:**

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

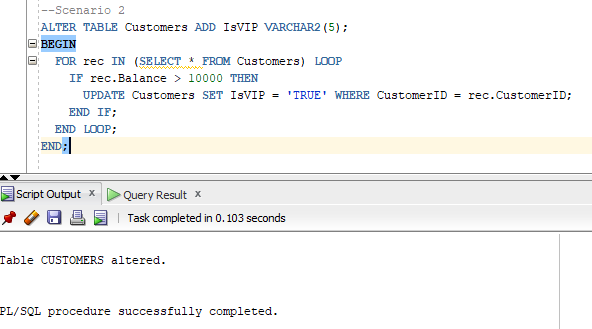
FOR rec IN (SELECT \* FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

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

**PL/SQL Query:**

BEGIN

FOR rec IN (

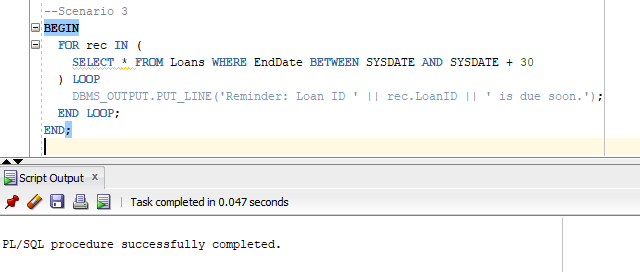
SELECT \* FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID || ' is due soon.');

END LOOP;

END;



**Exercise 2: Error Handling**

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

**PL/SQL Query:**

CREATE PROCEDURE SafeTransferFunds(

p\_from NUMBER,

p\_to NUMBER,

p\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

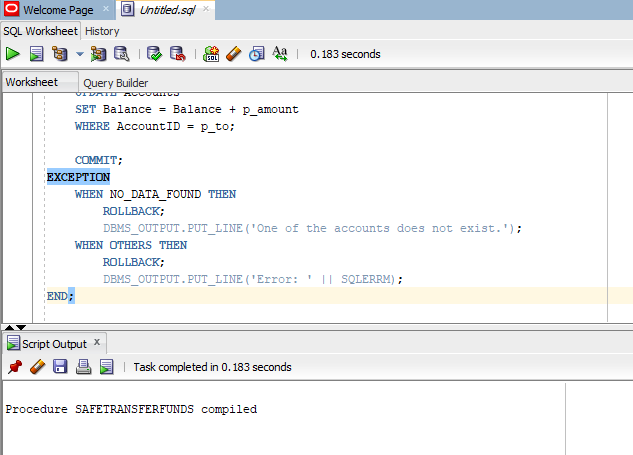
DBMS\_OUTPUT.PUT\_LINE('One of the accounts does not exist.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;



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

**PL/SQL Query:**

CREATE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_percent NUMBER) IS

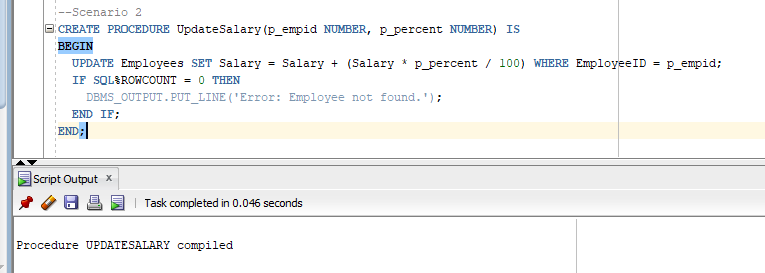
BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100) WHERE EmployeeID = p\_empid;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found.');

END IF;

END;

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

**PL/SQL Query:**

CREATE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with this ID already exists.');

A screenshot of a computer

AI-generated content may be incorrect.END;

**Exercise 3: Stored Procedures**

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

**PL/SQL Query:**

CREATE PROCEDURE ProcessMonthlyInterest AS

BEGIN

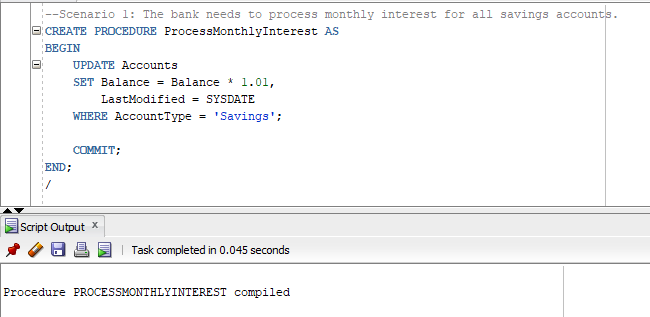
UPDATE Accounts

SET Balance = Balance \* 1.01,

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

END;

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

**PL/SQL Query:**

CREATE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

**A screenshot of a computer

AI-generated content may be incorrect.**END;

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

**PL/SQL Query:**

CREATE SEQUENCE TRANSACTIONS\_SEQ START WITH 1001 INCREMENT BY 1;

CREATE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

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

VALUES (TRANSACTIONS\_SEQ.NEXTVAL, p\_from\_account, SYSDATE, p\_amount, 'Transfer Out');

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

VALUES (TRANSACTIONS\_SEQ.NEXTVAL, p\_to\_account, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

END;

/

A screenshot of a computer screen

AI-generated content may be incorrect.

**Exercise 4: Functions**

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

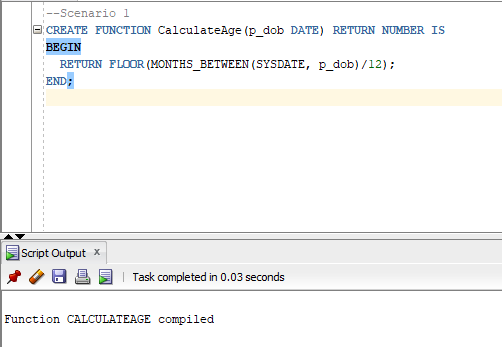
**PL/SQL Query:**

CREATE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);

END;



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

**PL/SQL Query:**

CREATE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER,

p\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

r NUMBER := p\_rate / (12 \* 100);

n NUMBER := p\_years \* 12;

BEGIN

RETURN (p\_amount \* r \* POWER(1 + r, n)) / (POWER(1 + r, n) - 1);

END;

**A screenshot of a computer program

AI-generated content may be incorrect.**

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

**PL/SQL Query:**

CREATE FUNCTION HasSufficientBalance(p\_accountid NUMBER, p\_amount NUMBER) RETURN BOOLEAN IS

bal NUMBER;

BEGIN

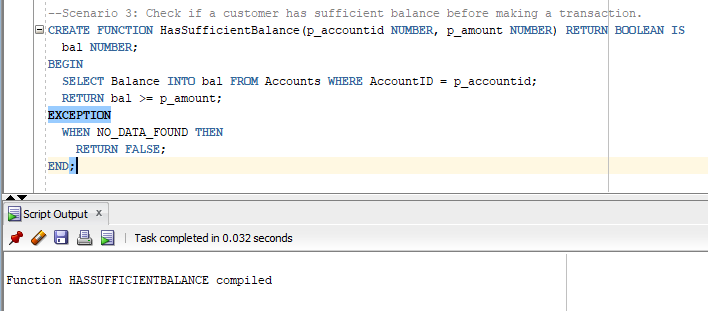
SELECT Balance INTO bal FROM Accounts WHERE AccountID = p\_accountid;

RETURN bal >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

**Exercise 5: Triggers**

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

**PL/SQL Query:**

CREATE TRIGGER UpdateCustomerLastModified

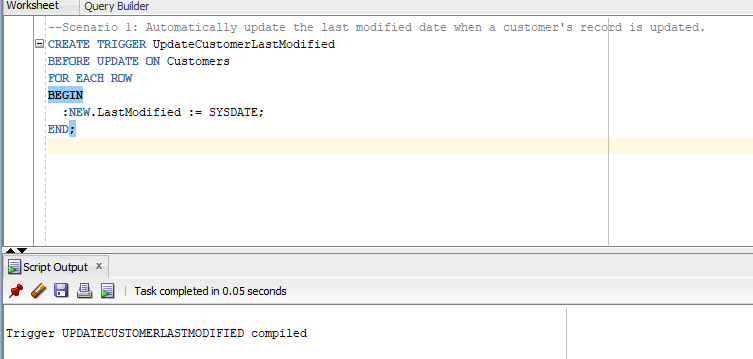
BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

****

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

**PL/SQL Query:**

CREATE SEQUENCE AuditLog\_SEQ

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Message VARCHAR2(100)

);

CREATE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

LogID,

TransactionID,

LogDate,

Message

) VALUES (

AuditLog\_SEQ.NEXTVAL,

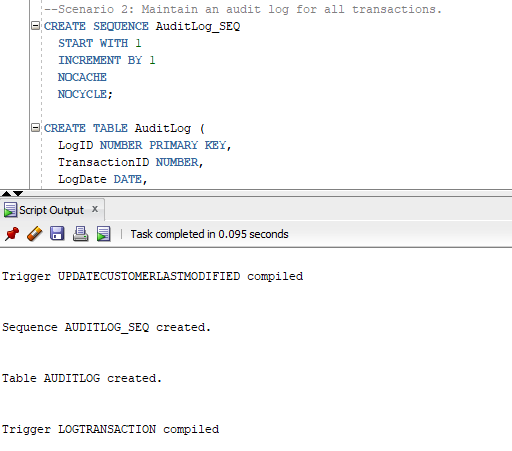
:NEW.TransactionID,

SYSDATE,

'Transaction successfully recorded in audit log.'

);

END;



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

**PL/SQL Query:**

CREATE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > bal THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance');

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit must be positive');

END IF;

END;

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 6: Cursors**

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

**PL/SQL Query:**

DECLARE

CURSOR cur IS

SELECT c.Name, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

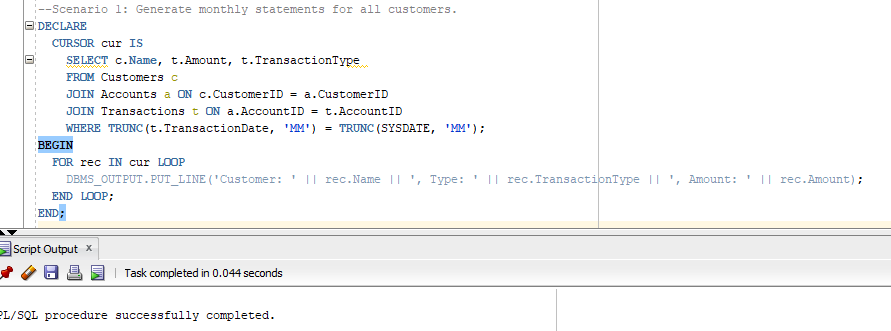
BEGIN

FOR rec IN cur LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ', Type: ' || rec.TransactionType || ', Amount: ' || rec.Amount);

END LOOP;

END;



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

**PL/SQL Query:**

DECLARE

CURSOR cur IS SELECT AccountID FROM Accounts;

BEGIN

FOR rec IN cur LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = rec.AccountID;

END LOOP;

END;

**A screenshot of a computer

AI-generated content may be incorrect.**

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

**PL/SQL Query:**

DECLARE

CURSOR cur IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR rec IN cur LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate + 0.5

WHERE LoanID = rec.LoanID;

END LOOP;

END;

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 7: Packages**

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

**PL/SQL Query:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_balance NUMBER);

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

-- Create the package body

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

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

END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_balance NUMBER) IS

BEGIN

UPDATE Customers

SET Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_id;

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

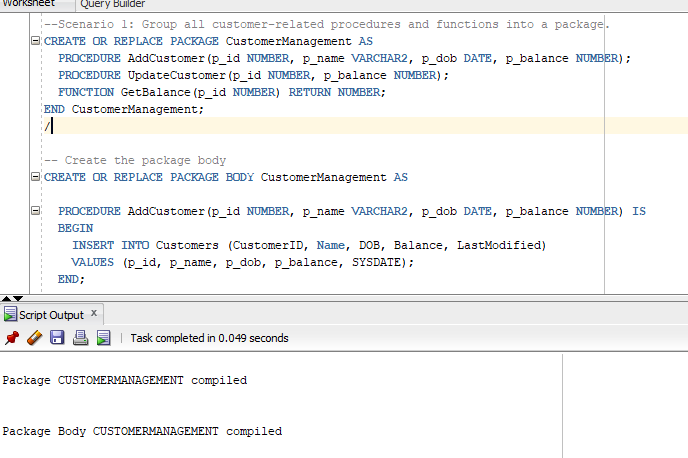
SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_id;

RETURN v\_balance;

END;

****END CustomerManagement;  
/

**Scenario 2: Group all customer-related procedures and functions into a package.**

**PL/SQL Query:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(

p\_EmpID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

);

PROCEDURE UpdateEmployee(

p\_EmpID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

);

FUNCTION GetAnnualSalary(

p\_EmpID IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(

p\_EmpID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

) IS

BEGIN

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

VALUES (p\_EmpID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

END HireEmployee;

PROCEDURE UpdateEmployee(

p\_EmpID 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\_EmpID;

END UpdateEmployee;

FUNCTION GetAnnualSalary(

p\_EmpID IN NUMBER

) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary

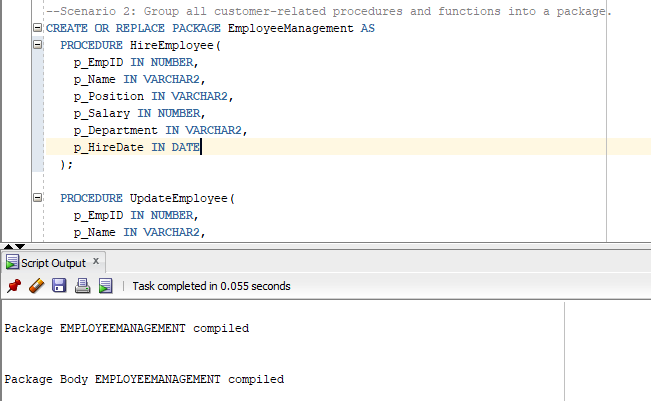
FROM Employees

WHERE EmployeeID = p\_EmpID;

RETURN v\_Salary \* 12;

END GetAnnualSalary;

END EmployeeManagement;

/

**Scenario 3: Group all customer-related procedures and functions into a package.**

**PL/SQL Query:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_InitialBalance IN NUMBER

);

PROCEDURE CloseAccount(

p\_AccountID IN NUMBER

);

FUNCTION GetTotalBalance(

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_InitialBalance IN NUMBER

) IS

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_InitialBalance, SYSDATE);

END OpenAccount;

PROCEDURE CloseAccount(

p\_AccountID IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

END CloseAccount;

FUNCTION GetTotalBalance(

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_TotalBalance;

END GetTotalBalance;

END AccountOperations;

/

