--Creating a sample schema to implement the exercises

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 (3, 'Michael Johnson', TO\_DATE('1978-11-23', 'YYYY-MM-DD'), 2500, SYSDATE);

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

VALUES (5, 'Emily Davis', TO\_DATE('1982-03-17', 'YYYY-MM-DD'), 3200, SYSDATE);

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

VALUES (3, 3, 'Savings', 2500, SYSDATE);

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

VALUES (5, 5, 'Checking', 3200, SYSDATE);

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

VALUES (3, 3, SYSDATE, 500, 'Deposit');

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

VALUES (5, 5, SYSDATE, 600, 'Withdrawal');

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

VALUES (2, 3, 7000, 4.5, SYSDATE, ADD\_MONTHS(SYSDATE, 48));

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

VALUES (3, 5, 10000, 3.8, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

VALUES (3, 'David Miller', 'Analyst', 65000, 'Finance', TO\_DATE('2016-08-25', 'YYYY-MM-DD'));

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

VALUES (5, 'Sophia Wilson', 'Designer', 55000, 'Marketing', TO\_DATE('2018-11-12', 'YYYY-MM-DD'));

1. **Control Structures**

**Scenario 1**

DECLARE

CURSOR cur\_customers IS

SELECT CustomerID, DOB FROM Customers;

v\_customer\_age NUMBER;

v\_current\_loan Loans%ROWTYPE;

BEGIN

FOR customer\_rec IN cur\_customers LOOP

-- Calculate age

v\_customer\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, customer\_rec.DOB) / 12);

-- Check if age is above 60

IF v\_customer\_age > 60 THEN

-- Fetch current loan for the customer

SELECT \* INTO v\_current\_loan

FROM Loans

WHERE CustomerID = customer\_rec.CustomerID;

-- Apply 1% discount to interest rate

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = v\_current\_loan.LoanID;

END IF;

END LOOP;

END;

/

**OUTPUT:**

Update operation is done.

**Scenario 2**

--altering table to set default value of IsVIP to NO

ALTER TABLE Customers

ADD IsVIP VARCHAR2(3) DEFAULT 'NO';

DECLARE

CURSOR cur\_customers IS

SELECT CustomerID, Balance FROM Customers;

BEGIN

FOR customer\_rec IN cur\_customers LOOP

-- Check if balance is over 10000

IF customer\_rec.Balance > 10000 THEN

-- Set IsVIP to TRUE

UPDATE Customers

SET IsVIP = 'YES'

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

END;

/

**OUTPUT:**

Update operation is done.

**Scenario 3**

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

v\_customer\_name Customers.Name%TYPE;

BEGIN

FOR loan\_rec IN cur\_loans LOOP

-- Fetch customer name

SELECT Name INTO v\_customer\_name

FROM Customers

WHERE CustomerID = loan\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || v\_customer\_name ||

', your loan (Loan ID: ' || loan\_rec.LoanID ||

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

END LOOP;

END;

/

**OUTPUT:**

Reminder: Customer Michael Johnson, your loan (Loan ID: 2) is due on 2024-08-22.

1. **Error Handling**

--Creating an error log table to store error messages

CREATE TABLE ErrorLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

ErrorMessage VARCHAR2(4000),

ErrorDate DATE

);

Scenario 1

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) IS

v\_from\_balance Accounts.Balance%TYPE;

v\_to\_balance Accounts.Balance%TYPE;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (SQLERRM, SYSDATE);

RAISE;

END SafeTransferFunds;

/

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

) IS

v\_old\_salary Employees.Salary%TYPE;

BEGIN

SELECT Salary INTO v\_old\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES ('Employee ID ' || p\_employee\_id || ' does not exist.', SYSDATE);

ROLLBACK;

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (SQLERRM, SYSDATE);

RAISE;

END UpdateSalary;

/

**Scenario 3**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) IS

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES ('Customer ID ' || p\_customer\_id || ' already exists.', SYSDATE);

ROLLBACK;

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (SQLERRM, SYSDATE);

RAISE;

END AddNewCustomer;

/

--To check for any logged errors

SELECT \* FROM ErrorLog;

1. **Stored Procedures**

**Scenario 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

CURSOR cur\_savings\_accounts IS

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings';

v\_new\_balance Accounts.Balance%TYPE;

BEGIN

FOR account\_rec IN cur\_savings\_accounts LOOP

v\_new\_balance := account\_rec.Balance + (account\_rec.Balance \* 0.01);

UPDATE Accounts

SET Balance = v\_new\_balance

WHERE AccountID = account\_rec.AccountID;

END LOOP;

COMMIT;

END ProcessMonthlyInterest;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department VARCHAR2,

p\_bonus\_percentage NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END UpdateEmployeeBonus;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

**Scenario 3**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) IS

v\_from\_balance Accounts.Balance%TYPE;

v\_to\_balance Accounts.Balance%TYPE;

BEGIN

-- Check if the from\_account has sufficient funds

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (SQLERRM, SYSDATE);

RAISE;

END TransferFunds;

/

BEGIN

TransferFunds(3, 4, 100);

END;

/

1. **Functions:**

**Scenario 1**

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

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END CalculateAge;

/

--Usage of function

SELECT CalculateAge(TO\_DATE('1985-05-15', 'YYYY-MM-DD')) AS Age FROM DUAL;

**Scenario 2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_monthly\_installment NUMBER;

v\_total\_months NUMBER;

BEGIN

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

-- Total number of monthly installments

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

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

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

--Usage of function

SELECT CalculateMonthlyInstallment(10000, 5, 5) AS MonthlyInstallment FROM DUAL;

**Scenario 3**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance Accounts.Balance%TYPE;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END HasSufficientBalance;

/

--Usage of function

DECLARE

v\_has\_balance BOOLEAN;

BEGIN

v\_has\_balance := HasSufficientBalance(1, 500);

IF v\_has\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Sufficient balance.');

ELSE

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

END IF;

END;

/

1. **Triggers**

**Scenario 1**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

Scenario 2

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

AuditDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AuditID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, AuditDate)

VALUES (AuditLog\_seq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END LogTransaction;

/

**Scenario 3**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance Accounts.Balance%TYPE;

BEGIN

-- Fetch the current balance of the account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID

FOR UPDATE;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF v\_balance < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive.');

END IF;

END IF;

END CheckTransactionRules;

/

--Usage of the above created triggers and to invoke them

--Scenario 1

UPDATE Customers

SET Name = 'John Doe Updated'

WHERE CustomerID = 1;

SELECT CustomerID, Name, LastModified FROM Customers WHERE CustomerID = 1;

--Scenario 2

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

VALUES (3, 1, SYSDATE, 200, 'Deposit');

SELECT \* FROM AuditLog WHERE TransactionID = 3;

--Scenario 3

--Valid deposit

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

VALUES (4, 1, SYSDATE, 300, 'Deposit');

--Invalid withdrawal

BEGIN

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

VALUES (5, 1, SYSDATE, 5000, 'Withdrawal');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

--Invalid deposit (non-positive amount)

BEGIN

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

VALUES (6, 1, SYSDATE, -100, 'Deposit');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

1. **Cursors**

**Scenario 1**

DECLARE

CURSOR GenerateMonthlyStatements IS

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

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate >= TRUNC(SYSDATE, 'MM') AND t.TransactionDate < ADD\_MONTHS(TRUNC(SYSDATE, 'MM'), 1);

v\_customerID Customers.CustomerID%TYPE;

v\_customerName Customers.Name%TYPE;

v\_transactionDate Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transactionType Transactions.TransactionType%TYPE;

BEGIN

FOR rec IN GenerateMonthlyStatements LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || rec.Name);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || rec.TransactionDate);

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

DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || rec.TransactionType);

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

END LOOP;

END;

/

**Scenario 2**

DECLARE

CURSOR ApplyAnnualFee IS

SELECT AccountID, Balance

FROM Accounts;

v\_accountID Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_new\_balance Accounts.Balance%TYPE;

annual\_fee CONSTANT NUMBER := 50; -- Define the annual fee amount

BEGIN

FOR rec IN ApplyAnnualFee LOOP

v\_accountID := rec.AccountID;

v\_balance := rec.Balance;

v\_new\_balance := v\_balance - annual\_fee;

UPDATE Accounts

SET Balance = v\_new\_balance

WHERE AccountID = v\_accountID;

END LOOP;

COMMIT;

END;

/

**Scenario 3**

DECLARE

CURSOR UpdateLoanInterestRates IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loanID Loans.LoanID%TYPE;

v\_current\_rate Loans.InterestRate%TYPE;

v\_new\_rate Loans.InterestRate%TYPE;

BEGIN

FOR rec IN UpdateLoanInterestRates LOOP

v\_loanID := rec.LoanID;

v\_current\_rate := rec.InterestRate;

v\_new\_rate := v\_current\_rate + 1; -- Increase interest rate by 1%

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = v\_loanID;

END LOOP;

COMMIT;

END;

/

1. **Packages**

**Scenario 1**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_CustomerID NUMBER,

p\_Name VARCHAR2,

p\_DOB DATE,

p\_Balance NUMBER

);

PROCEDURE UpdateCustomerDetails(

p\_CustomerID NUMBER,

p\_Name VARCHAR2,

p\_Balance NUMBER

);

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_CustomerID NUMBER,

p\_Name VARCHAR2,

p\_DOB DATE,

p\_Balance NUMBER

) IS

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(

p\_CustomerID NUMBER,

p\_Name VARCHAR2,

p\_Balance NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_Name,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL; -- Return NULL if customer is not found

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_EmployeeID NUMBER,

p\_Name VARCHAR2,

p\_Position VARCHAR2,

p\_Salary NUMBER,

p\_Department VARCHAR2

);

PROCEDURE UpdateEmployeeDetails(

p\_EmployeeID NUMBER,

p\_Name VARCHAR2,

p\_Salary NUMBER,

p\_Position VARCHAR2

);

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_EmployeeID NUMBER,

p\_Name VARCHAR2,

p\_Position VARCHAR2,

p\_Salary NUMBER,

p\_Department VARCHAR2

) IS

BEGIN

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

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, SYSDATE);

END HireNewEmployee;

PROCEDURE UpdateEmployeeDetails(

p\_EmployeeID NUMBER,

p\_Name VARCHAR2,

p\_Salary NUMBER,

p\_Position VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Name = p\_Name,

Salary = p\_Salary,

Position = p\_Position

WHERE EmployeeID = p\_EmployeeID;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12; -- Assuming salary is monthly

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL; -- Return NULL if employee is not found

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(

p\_AccountID NUMBER,

p\_CustomerID NUMBER,

p\_AccountType VARCHAR2,

p\_Balance NUMBER

);

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(

p\_AccountID NUMBER,

p\_CustomerID NUMBER,

p\_AccountType VARCHAR2,

p\_Balance NUMBER

) IS

BEGIN

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

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

END OpenNewAccount;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

END CloseAccount;

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Return 0 if no accounts found

END GetTotalBalance;

END AccountOperations;

/

--Trying some work with the packages created

--adding a new customer

BEGIN

CustomerManagement.AddNewCustomer(3, 'Alice Cooper', TO\_DATE('1992-04-10', 'YYYY-MM-DD'), 2000);

END;

/

--updating customer details

BEGIN

CustomerManagement.UpdateCustomerDetails(3, 'Alice Cooper Updated', 2500);

END;

/

--getting customer balance

DECLARE

v\_balance NUMBER;

BEGIN

v\_balance := CustomerManagement.GetCustomerBalance(3);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_balance);

END;

/

--calculating annual salary

DECLARE

v\_annual\_salary NUMBER;

BEGIN

v\_annual\_salary := EmployeeManagement.CalculateAnnualSalary(3);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || v\_annual\_salary);

END;

/

--getting total balance of a customer

DECLARE

v\_total\_balance NUMBER;

BEGIN

v\_total\_balance := AccountOperations.GetTotalBalance(3);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || v\_total\_balance);

END;

/