Exercise 1

-- Scenario 1: Apply a 1% discount to loan interest rates for customers above 60 years old.

DECLARE

v\_dob DATE;

v\_age NUMBER;

v\_customer\_id NUMBER;

v\_loan\_id NUMBER;

v\_interest\_rate NUMBER;

BEGIN

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

v\_dob := cust.DOB;

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

IF v\_age > 60 THEN

FOR loan IN (SELECT LoanID, InterestRate FROM Loans WHERE CustomerID = cust.CustomerID) LOOP

v\_interest\_rate := loan.InterestRate - 1;

IF v\_interest\_rate < 0 THEN

v\_interest\_rate := 0;

END IF;

UPDATE Loans

SET InterestRate = v\_interest\_rate

WHERE LoanID = loan.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Updated loan ' || loan.LoanID || ' for customer ' || cust.CustomerID || '. New interest rate: ' || v\_interest\_rate || '%');

END LOOP;

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 2: Set IsVIP to TRUE for customers with a balance over $10,000.

DECLARE

v\_is\_vip BOOLEAN;

BEGIN

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

v\_is\_vip := cust.Balance > 10000;

IF v\_is\_vip THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || cust.CustomerID || ' promoted to VIP status.');

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 3: Print reminder messages for loans due in the next 30 days.

DECLARE

v\_customer\_name VARCHAR2(100);

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

v\_customer\_name := loan.Name;

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

', your loan (ID: ' || loan.LoanID ||

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

'. Please arrange for repayment.');

END LOOP;

END;

/

Exercise 2

-- Scenario 1

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) AS

v\_from\_balance NUMBER;

v\_insufficient\_funds EXCEPTION;

v\_invalid\_account EXCEPTION;

BEGIN

-- Check if accounts exist

IF NOT EXISTS (SELECT 1 FROM Accounts WHERE AccountID = p\_from\_account) OR

NOT EXISTS (SELECT 1 FROM Accounts WHERE AccountID = p\_to\_account) THEN

RAISE v\_invalid\_account;

END IF;

-- Check for sufficient funds

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_from\_balance < p\_amount THEN

RAISE v\_insufficient\_funds;

END IF;

-- Perform transfer

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;

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

EXCEPTION

WHEN v\_insufficient\_funds THEN

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

ROLLBACK;

WHEN v\_invalid\_account THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Invalid account');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END SafeTransferFunds;

/

-- Scenario 2

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id NUMBER,

p\_percentage NUMBER

) AS

v\_current\_salary NUMBER;

v\_employee\_not\_found EXCEPTION;

BEGIN

-- Check if employee exists

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

-- Update salary

UPDATE Employees

SET Salary = Salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE v\_employee\_not\_found;

WHEN v\_employee\_not\_found THEN

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

-- Log error to a table (you would need to create this table)

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES ('Employee not found: ' || p\_employee\_id, SYSDATE);

COMMIT;

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateSalary;

/

-- Scenario 3

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) AS

v\_customer\_exists EXCEPTION;

BEGIN

-- Check if customer already exists

IF EXISTS (SELECT 1 FROM Customers WHERE CustomerID = p\_customer\_id) THEN

RAISE v\_customer\_exists;

END IF;

-- Insert new customer

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully');

EXCEPTION

WHEN v\_customer\_exists THEN

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

-- Log error to a table (you would need to create this table)

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

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

COMMIT;

WHEN OTHERS THEN

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

ROLLBACK;

END AddNewCustomer;

/

Exercise 3

-- Scenario 1

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

CURSOR c\_savings\_accounts IS

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings';

v\_interest NUMBER;

v\_new\_balance NUMBER;

BEGIN

FOR acc IN c\_savings\_accounts LOOP

-- Calculate interest (1% of current balance)

v\_interest := acc.Balance \* 0.01;

-- Calculate new balance

v\_new\_balance := acc.Balance + v\_interest;

-- Update account balance

UPDATE Accounts

SET Balance = v\_new\_balance,

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Updated Account ' || acc.AccountID ||

': Old Balance = ' || acc.Balance ||

', New Balance = ' || v\_new\_balance);

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processing completed.');

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END ProcessMonthlyInterest;

/

-- Scenario 2

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department VARCHAR2,

p\_bonus\_percentage NUMBER

) AS

v\_count NUMBER := 0;

BEGIN

-- Update salaries for employees in the given department

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage / 100)

WHERE Department = p\_department;

v\_count := SQL%ROWCOUNT;

IF v\_count > 0 THEN

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || v\_count ||

' employees in ' || p\_department ||

' department.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('No employees found in ' || p\_department ||

' department.');

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateEmployeeBonus;

/

-- Scenario 3

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) AS

v\_from\_balance NUMBER;

v\_insufficient\_funds EXCEPTION;

v\_same\_account EXCEPTION;

v\_invalid\_account EXCEPTION;

BEGIN

-- Check if accounts are different

IF p\_from\_account = p\_to\_account THEN

RAISE v\_same\_account;

END IF;

-- Check if accounts exist

IF NOT EXISTS (SELECT 1 FROM Accounts WHERE AccountID = p\_from\_account) OR

NOT EXISTS (SELECT 1 FROM Accounts WHERE AccountID = p\_to\_account) THEN

RAISE v\_invalid\_account;

END IF;

-- Check for sufficient funds

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_from\_balance < p\_amount THEN

RAISE v\_insufficient\_funds;

END IF;

-- Perform transfer

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;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_amount ||

' from Account ' || p\_from\_account ||

' to Account ' || p\_to\_account ||

' completed successfully.');

EXCEPTION

WHEN v\_same\_account THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Cannot transfer to the same account');

WHEN v\_invalid\_account THEN

DBMS\_OUTPUT.PUT\_LINE('Error: One or both accounts are invalid');

WHEN v\_insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the source account');

WHEN OTHERS THEN

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

ROLLBACK;

END TransferFunds;

/

Exercise 4

-- Scenario 1

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAge;

/

-- Scenario 2

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

)

RETURN NUMBER

IS

v\_monthly\_rate NUMBER;

v\_num\_payments NUMBER;

v\_installment NUMBER;

BEGIN

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

v\_num\_payments := p\_loan\_duration\_years \* 12;

v\_installment := p\_loan\_amount \*

(v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_num\_payments)) /

(POWER(1 + v\_monthly\_rate, v\_num\_payments) - 1);

RETURN ROUND(v\_installment, 2);

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateMonthlyInstallment;

/

-- Scenario 3

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

)

RETURN BOOLEAN

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN FALSE;

WHEN OTHERS THEN

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

RETURN FALSE;

END HasSufficientBalance;

/

Exercise 5

-- Scenario 1

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

-- Scenario 2

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

CREATE SEQUENCE AuditLog\_Seq;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (LogID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate)

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

EXCEPTION

WHEN OTHERS THEN

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

END;

/

-- Scenario 3

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Check if amount is positive

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Transaction amount must be positive');

END IF;

-- For withdrawals, check if there's sufficient balance

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF v\_balance < :NEW.Amount THEN

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

END IF;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Account not found');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Error checking transaction rules: ' || SQLERRM);

END;

/

Exercise 6

-- Scenario 1

DECLARE

CURSOR c\_customer\_transactions IS

SELECT c.CustomerID, c.Name, a.AccountID, a.AccountType,

t.TransactionID, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

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

ORDER BY c.CustomerID, a.AccountID, t.TransactionDate;

v\_current\_customer NUMBER := -1;

v\_current\_account NUMBER := -1;

v\_total\_deposits NUMBER := 0;

v\_total\_withdrawals NUMBER := 0;

BEGIN

FOR trans IN c\_customer\_transactions LOOP

-- Start new customer statement

IF trans.CustomerID != v\_current\_customer THEN

IF v\_current\_customer != -1 THEN

DBMS\_OUTPUT.PUT\_LINE('End of statement for customer ' || v\_current\_customer);

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

END IF;

DBMS\_OUTPUT.PUT\_LINE('Monthly Statement for ' || trans.Name || ' (CustomerID: ' || trans.CustomerID || ')');

v\_current\_customer := trans.CustomerID;

v\_current\_account := -1;

END IF;

-- Start new account section

IF trans.AccountID != v\_current\_account THEN

IF v\_current\_account != -1 THEN

DBMS\_OUTPUT.PUT\_LINE(' Total Deposits: ' || v\_total\_deposits);

DBMS\_OUTPUT.PUT\_LINE(' Total Withdrawals: ' || v\_total\_withdrawals);

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

END IF;

DBMS\_OUTPUT.PUT\_LINE('Account: ' || trans.AccountID || ' (' || trans.AccountType || ')');

v\_current\_account := trans.AccountID;

v\_total\_deposits := 0;

v\_total\_withdrawals := 0;

END IF;

-- Print transaction details

DBMS\_OUTPUT.PUT\_LINE(' ' || trans.TransactionDate || ' - ' ||

trans.TransactionType || ': ' || trans.Amount);

-- Update totals

IF trans.TransactionType = 'Deposit' THEN

v\_total\_deposits := v\_total\_deposits + trans.Amount;

ELSIF trans.TransactionType = 'Withdrawal' THEN

v\_total\_withdrawals := v\_total\_withdrawals + trans.Amount;

END IF;

END LOOP;

-- Print final totals for last account and customer

IF v\_current\_account != -1 THEN

DBMS\_OUTPUT.PUT\_LINE(' Total Deposits: ' || v\_total\_deposits);

DBMS\_OUTPUT.PUT\_LINE(' Total Withdrawals: ' || v\_total\_withdrawals);

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

END IF;

IF v\_current\_customer != -1 THEN

DBMS\_OUTPUT.PUT\_LINE('End of statement for customer ' || v\_current\_customer);

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

END IF;

END;

/

-- Scenario 2

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance, AccountType

FROM Accounts

FOR UPDATE OF Balance;

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

v\_updated\_count NUMBER := 0;

BEGIN

FOR acc IN c\_accounts LOOP

IF acc.Balance >= v\_annual\_fee THEN

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE CURRENT OF c\_accounts;

v\_updated\_count := v\_updated\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to Account ' || acc.AccountID ||

' (' || acc.AccountType || '). New balance: ' ||

(acc.Balance - v\_annual\_fee));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ' || acc.AccountID ||

' (' || acc.AccountType || ') to apply annual fee.');

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Annual fee applied to ' || v\_updated\_count || ' accounts.');

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

-- Scenario 3

DECLARE

CURSOR c\_loans IS

SELECT LoanID, LoanAmount, InterestRate

FROM Loans

FOR UPDATE OF InterestRate;

v\_new\_rate NUMBER;

BEGIN

FOR loan IN c\_loans LOOP

-- Example new policy:

-- Loans under 10000: increase by 0.5%

-- Loans 10000-50000: no change

-- Loans over 50000: decrease by 0.5%

IF loan.LoanAmount < 10000 THEN

v\_new\_rate := loan.InterestRate + 0.5;

ELSIF loan.LoanAmount > 50000 THEN

v\_new\_rate := loan.InterestRate - 0.5;

ELSE

v\_new\_rate := loan.InterestRate;

END IF;

-- Ensure interest rate doesn't go below 1%

v\_new\_rate := GREATEST(v\_new\_rate, 1);

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE CURRENT OF c\_loans;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ' || loan.LoanID ||

': Old rate = ' || loan.InterestRate ||

'%, New rate = ' || v\_new\_rate || '%');

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Loan interest rates updated successfully.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error updating loan interest rates: ' || SQLERRM);

ROLLBACK;

END;

/

Exercise 7

-- Scenario 1: CustomerManagement Package

CREATE OR REPLACE PACKAGE CustomerManagement AS

-- Add a new customer

PROCEDURE AddCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_initial\_balance IN NUMBER

);

-- Update customer details

PROCEDURE UpdateCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE

);

-- Get customer balance

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_initial\_balance IN NUMBER

) IS

BEGIN

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

END AddCustomer;

PROCEDURE UpdateCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE

) IS

BEGIN

UPDATE Customers

SET Name = p\_name,

DOB = p\_dob,

LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer updated successfully');

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateCustomer;

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

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

-- Scenario 2: EmployeeManagement Package

-- Package Specification

CREATE OR REPLACE PACKAGE EmployeeManagement AS

-- Hire a new employee

PROCEDURE HireEmployee(

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

);

-- Update employee details

PROCEDURE UpdateEmployee(

p\_employee\_id IN NUMBER,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

);

-- Calculate annual salary

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(

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);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Employee hired successfully');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID already exists');

WHEN OTHERS THEN

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

ROLLBACK;

END HireEmployee;

PROCEDURE UpdateEmployee(

p\_employee\_id IN NUMBER,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Position = p\_position,

Salary = p\_salary,

Department = p\_department

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Employee updated successfully');

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateEmployee;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER IS

v\_monthly\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_monthly\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_monthly\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

-- Scenario 3: AccountOperations Package

CREATE OR REPLACE PACKAGE AccountOperations AS

-- Open a new account

PROCEDURE OpenAccount(

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2,

p\_initial\_balance IN NUMBER

);

-- Close an account

PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

-- Get total balance of a customer

FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2,

p\_initial\_balance IN NUMBER

) IS

BEGIN

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Account opened successfully');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID already exists');

WHEN OTHERS THEN

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

ROLLBACK;

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF v\_balance > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account has non-zero balance');

ELSE

DELETE FROM Accounts WHERE AccountID = p\_account\_id;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Account closed successfully');

END IF;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

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

DBMS\_OUTPUT.PUT\_LINE('Error: Customer not found or has no accounts');

RETURN 0;

WHEN OTHERS THEN

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

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/