**Exercise 1: Control Structures**

**Scenario 1: Discount for Senior Customers**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers WHERE TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

END LOOP;

END;

**Scenario 2: Promote to VIP Status**

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

**Scenario 3: Loan Due Reminders**

BEGIN

FOR rec IN (SELECT CustomerID, LoanID, EndDate FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for Customer ' || rec.CustomerID || ' is due within 30 days.');

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1: SafeTransferFunds**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_SourceAccountID NUMBER, p\_TargetAccountID NUMBER, p\_Amount NUMBER) IS

insufficient\_funds EXCEPTION;

current\_balance NUMBER;

BEGIN

SELECT Balance INTO current\_balance FROM Accounts WHERE AccountID = p\_SourceAccountID;

IF current\_balance < p\_Amount THEN

RAISE insufficient\_funds;

ELSE

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_TargetAccountID;

END IF;

EXCEPTION

WHEN insufficient\_funds THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END SafeTransferFunds;

**Scenario 2: UpdateSalary**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_EmployeeID NUMBER, p\_Percentage NUMBER) IS

employee\_not\_found EXCEPTION;

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM Employees WHERE EmployeeID = p\_EmployeeID;

IF v\_count = 0 THEN

RAISE employee\_not\_found;

ELSE

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

END IF;

EXCEPTION

WHEN employee\_not\_found THEN

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

WHEN OTHERS THEN

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

END UpdateSalary;

**Scenario 3: AddNewCustomer**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

customer\_exists EXCEPTION;

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM Customers WHERE CustomerID = p\_CustomerID;

IF v\_count > 0 THEN

RAISE customer\_exists;

ELSE

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

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

END IF;

EXCEPTION

WHEN customer\_exists THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountID = rec.AccountID;

END LOOP;

END ProcessMonthlyInterest;

**Scenario 2: UpdateEmployeeBonus**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_Department VARCHAR2, p\_BonusPercentage NUMBER) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

END UpdateEmployeeBonus;

**Scenario 3: TransferFunds**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_SourceAccountID NUMBER, p\_TargetAccountID NUMBER, p\_Amount NUMBER) IS

insufficient\_balance EXCEPTION;

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_SourceAccountID;

IF v\_balance < p\_Amount THEN

RAISE insufficient\_balance;

ELSE

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_TargetAccountID;

END IF;

EXCEPTION

WHEN insufficient\_balance THEN

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

WHEN OTHERS THEN

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

END TransferFunds;

**Exercise 4: Functions**

**Scenario 1: CalculateAge**

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;

END CalculateAge;

**Scenario 2: CalculateMonthlyInstallment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_LoanAmount NUMBER, p\_InterestRate NUMBER, p\_DurationYears NUMBER) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_total\_months NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

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

v\_total\_months := p\_DurationYears \* 12;

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

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

**Scenario 3: HasSufficientBalance**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_AccountID NUMBER, p\_Amount NUMBER) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_AccountID;

RETURN v\_balance >= p\_Amount;

END HasSufficientBalance;

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Scenario 2: LogTransaction**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);

END;

**Scenario 3: CheckTransactionRules**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

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(-20001, 'Insufficient balance for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END;

**Exercise 6: Cursors**

**Scenario 1: GenerateMonthlyStatements**

DECLARE

CURSOR c\_statements IS

SELECT CustomerID, AccountID, TransactionDate, Amount, TransactionType

FROM Transactions

WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

BEGIN

FOR rec IN c\_statements LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ' Account ' || rec.AccountID ||

' Date: ' || rec.TransactionDate || ' Amount: ' || rec.Amount ||

' Type: ' || rec.TransactionType);

END LOOP;

END;

**Scenario 2: ApplyAnnualFee**

DECLARE

CURSOR c\_apply\_fee IS

SELECT AccountID, Balance FROM Accounts;

v\_fee NUMBER := 50;

BEGIN

FOR rec IN c\_apply\_fee LOOP

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = rec.AccountID;

END LOOP;

END;

**Scenario 3: UpdateLoanInterestRates**

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_new\_rate NUMBER := 7;

BEGIN

FOR rec IN c\_loans LOOP

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = rec.LoanID;

END LOOP;

END;

**Exercise 7: Packages**

**Scenario 1: CustomerManagement Package**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomerDetails(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

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\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name, DOB = p\_DOB, 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;

END GetCustomerBalance;

END CustomerManagement;

**Scenario 2: EmployeeManagement Package**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2);

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(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 HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2) IS

BEGIN

UPDATE Employees

SET Name = p\_Name, Position = p\_Position, Salary = p\_Salary, Department = p\_Department

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;

END CalculateAnnualSalary;

END EmployeeManagement;

**Scenario 3: AccountOperations Package**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(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 IS

PROCEDURE OpenAccount(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 OpenAccount;

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\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance FROM Accounts WHERE CustomerID = p\_CustomerID;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;