**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE

CURSOR customers\_cursor IS

SELECT CustomerID, DOB, LoanID, InterestRate

FROM Customers

INNER JOIN Loans ON Customers.CustomerID = Loans.CustomerID;

customer\_rec customers\_cursor%ROWTYPE;

BEGIN

OPEN customers\_cursor;

LOOP

FETCH customers\_cursor INTO customer\_rec;

EXIT WHEN customers\_cursor%NOTFOUND;

IF TRUNC(MONTHS\_BETWEEN(SYSDATE, customer\_rec.DOB) / 12) > 60 THEN

UPDATE Loans

SET InterestRate = customer\_rec.InterestRate \* 0.99

WHERE LoanID = customer\_rec.LoanID;

END IF;

END LOOP;

CLOSE customers\_cursor;

END;

**Scenario 2:**

DECLARE

CURSOR customers\_cursor IS

SELECT CustomerID, Balance

FROM Customers;

customer\_rec customers\_cursor%ROWTYPE;

BEGIN

OPEN customers\_cursor;

LOOP

FETCH customers\_cursor INTO customer\_rec;

EXIT WHEN customers\_cursor%NOTFOUND;

IF customer\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

CLOSE customers\_cursor;

END;

**Scenario 3:**

DECLARE

CURSOR loans\_cursor IS

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

loan\_rec loans\_cursor%ROWTYPE;

BEGIN

OPEN loans\_cursor;

LOOP

FETCH loans\_cursor INTO loan\_rec;

EXIT WHEN loans\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan\_rec.LoanID || ' for customer ' || loan\_rec.CustomerID || ' is due on ' || loan\_rec.EndDate);

END LOOP;

CLOSE loans\_cursor;

END;

**Exercise 2: Error Handling**

**Scenario 1:** CREATE OR REPLACE PROCEDURE SafeTransferFunds(

from\_account\_id NUMBER,

to\_account\_id NUMBER,

amount NUMBER

) AS

BEGIN

IF amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Invalid transfer amount');

END IF;

IF from\_account\_id = to\_account\_id THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Cannot transfer to the same account');

END IF;

DECLARE

from\_balance NUMBER;

to\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account\_id;

IF from\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient funds');

END IF;

SELECT Balance INTO to\_balance

FROM Accounts

WHERE AccountID = to\_account\_id;

UPDATE Accounts

SET Balance = from\_balance - amount

WHERE AccountID = from\_account\_id;

UPDATE Accounts

SET Balance = to\_balance + amount

WHERE AccountID = to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

employee\_id NUMBER,

percentage NUMBER

) AS

BEGIN

IF percentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Invalid percentage');

END IF;

DECLARE

current\_salary NUMBER;

BEGIN

SELECT Salary INTO current\_salary

FROM Employees

WHERE EmployeeID = employee\_id;

IF current\_salary IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Employee not found');

END IF;

UPDATE Employees

SET Salary = current\_salary \* (1 + percentage / 100)

WHERE EmployeeID = employee\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

customer\_id NUMBER,

name VARCHAR2,

dob DATE,

balance NUMBER

) AS

BEGIN

IF customer\_id IS NULL OR name IS NULL OR dob IS NULL OR balance IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20006, 'Invalid customer data');

END IF;

DECLARE

existing\_customer NUMBER;

BEGIN

SELECT CustomerID INTO existing\_customer

FROM Customers

WHERE CustomerID = customer\_id;

IF existing\_customer IS NOT NULL THEN

RAISE\_APPLICATION\_ERROR(-20007, 'Customer already exists');

END IF;

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

VALUES (customer\_id, name, dob, balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

department VARCHAR2,

bonus\_percentage NUMBER

) AS

BEGIN

IF bonus\_percentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20008, 'Invalid bonus percentage');

END IF;

UPDATE Employees

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

WHERE Department = department;

COMMIT;

END;

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_account\_id NUMBER,

to\_account\_id NUMBER,

amount NUMBER

) AS

from\_balance NUMBER;

to\_balance NUMBER;

BEGIN

IF amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20009, 'Invalid transfer amount');

END IF;

IF from\_account\_id = to\_account\_id THEN

RAISE\_APPLICATION\_ERROR(-20010, 'Cannot transfer to the same account');

END IF;

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account\_id;

IF from\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Insufficient funds');

END IF;

SELECT Balance INTO to\_balance

FROM Accounts

WHERE AccountID = to\_account\_id;

UPDATE Accounts

SET Balance = from\_balance - amount

WHERE AccountID = from\_account\_id;

UPDATE Accounts

SET Balance = to\_balance + amount

WHERE AccountID = to\_account\_id;

COMMIT;

END;

**Exercise 4: Functions**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE) RETURN NUMBER AS

age NUMBER;

BEGIN

age := TRUNC(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

RETURN age;

END;

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loan\_amount NUMBER,

interest\_rate NUMBER,

loan\_duration NUMBER

) RETURN NUMBER AS

monthly\_installment NUMBER;

BEGIN

monthly\_installment := loan\_amount \* (interest\_rate / 1200) \* POWER(1 + interest\_rate / 1200, loan\_duration \* 12) / (POWER(1 + interest\_rate / 1200, loan\_duration \* 12) - 1);

RETURN monthly\_installment;

END;

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

account\_id NUMBER,

amount NUMBER

) RETURN BOOLEAN AS

balance NUMBER;

BEGIN

SELECT Balance INTO balance

FROM Accounts

WHERE AccountID = account\_id;

IF balance >= amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END;

**Exercise 5: Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSTIMESTAMP;

END;

**Scenario 2:**

CREATE TABLE AuditLog (

AuditLogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE,

FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)

);

**Scenario 3:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

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

END;

**Exercise 6: 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 TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

statement\_rec GenerateMonthlyStatements%ROWTYPE;

BEGIN

OPEN GenerateMonthlyStatements;

LOOP

FETCH GenerateMonthlyStatements INTO statement\_rec;

EXIT WHEN GenerateMonthlyStatements%NOTFOUND;

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

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

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

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

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

END LOOP;

CLOSE GenerateMonthlyStatements;

END;

**Scenario 2:**

DECLARE

annual\_fee NUMBER := 50; -- assume annual fee is 50

CURSOR ApplyAnnualFee IS

SELECT AccountID, Balance

FROM Accounts;

account\_rec ApplyAnnualFee%ROWTYPE;

BEGIN

OPEN ApplyAnnualFee;

LOOP

FETCH ApplyAnnualFee INTO account\_rec;

EXIT WHEN ApplyAnnualFee%NOTFOUND;

UPDATE Accounts

SET Balance = account\_rec.Balance - annual\_fee

WHERE AccountID = account\_rec.AccountID;

END LOOP;

CLOSE ApplyAnnualFee;

END;

**Scenario 3:**

DECLARE

new\_interest\_rate NUMBER := 0.05; -- assume new interest rate is 5%

CURSOR UpdateLoanInterestRates IS

SELECT LoanID, InterestRate

FROM Loans;

loan\_rec UpdateLoanInterestRates%ROWTYPE;

BEGIN

OPEN UpdateLoanInterestRates;

LOOP

FETCH UpdateLoanInterestRates INTO loan\_rec;

EXIT WHEN UpdateLoanInterestRates%NOTFOUND;

UPDATE Loans

SET InterestRate = new\_interest\_rate

WHERE LoanID = loan\_rec.LoanID;

END LOOP;

CLOSE UpdateLoanInterestRates;

END;

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

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) IS

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, 0, SYSTIMESTAMP);

END AddNewCustomer;

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

BEGIN

UPDATE Customers

SET Name = p\_Name, DOB = p\_DOB, LastModified = SYSTIMESTAMP

WHERE CustomerID = p\_CustomerID;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_Balance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

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, p\_HireDate DATE);

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 AS

PROCEDURE HireNewEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE) IS

BEGIN

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

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

END HireNewEmployee;

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:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2);

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) IS

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, 0, SYSTIMESTAMP);

END OpenNewAccount;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

UPDATE Accounts

SET LastModified = SYSTIMESTAMP

WHERE AccountID = p\_AccountID;

END CloseAccount;

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_Balance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

END GetTotalBalance;

END AccountOperations;