**Exercise 1: Control Structures**

**Scenario 1: Apply Discount to Loan Interest Rates for Customers Above 60**

BEGIN

FOR loan\_rec IN (SELECT LoanID, InterestRate, CustomerID FROM Loans) LOOP

DECLARE

v\_age NUMBER;

BEGIN

SELECT ROUND(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) INTO v\_age

FROM Customers

WHERE CustomerID = loan\_rec.CustomerID;

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE LoanID = loan\_rec.LoanID;

END IF;

END;

END LOOP;

END;

**Scenario 2: Promote Customers to VIP Status Based on Balance**

BEGIN

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

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

END IF;

END LOOP;

END;

**Scenario 3: Send Reminders for Loans Due in the Next 30 Days**

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.Name || ', your loan is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1: Safe Transfer of Funds Between Accounts**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

BEGIN

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END SafeTransferFunds;

**Scenario 2: Update Salary with Exception Handling**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

e\_no\_data\_found EXCEPTION;

BEGIN

DECLARE

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_employee\_id FOR UPDATE;

UPDATE Employees SET Salary = Salary \* (1 + p\_percentage / 100) WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' does not exist.');

RAISE e\_no\_data\_found;

WHEN OTHERS THEN

ROLLBACK;

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

END;

END UpdateSalary;

**Scenario 3: Add New Customer with Exception Handling**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

e\_duplicate\_customer EXCEPTION;

BEGIN

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

RAISE e\_duplicate\_customer;

END;

END AddNewCustomer;

**Exercise 3: Stored Procedures**

**Scenario 1: Process Monthly Interest for Savings Accounts**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

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

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acct\_rec.AccountID;

END LOOP;

COMMIT;

END ProcessMonthlyInterest;

Scenario 2: Update Employee Bonus Based on Performance

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END UpdateEmployeeBonus;

**Scenario 3: Transfer Funds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

BEGIN

SafeTransferFunds(p\_from\_account\_id, p\_to\_account\_id, p\_amount);

END TransferFunds;

**Exercise 4: Functions**

**Scenario 1: Calculate Age of Customers**

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;

**Scenario 2: Calculate Monthly Installment for a Loan**

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\_months NUMBER;

v\_installment NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 1200;

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

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

RETURN v\_installment;

END CalculateMonthlyInstallment;

**Scenario 3: Check for Sufficient Balance**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN 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;

END HasSufficientBalance;

**Exercise 5: Triggers**

**Scenario 1: Update Last Modified Date**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

**Scenario 2: Maintain Audit Log for Transactions**

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 LogTransaction;

**Scenario 3: Enforce Business Rules on Transactions**

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 :NEW.Amount > v\_balance THEN

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

END IF;

ELSIF :NEW.Amount <= 0 THEN

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

END IF;

END CheckTransactionRules;

**Exercise 6: Cursors**

**Scenario 1: Generate Monthly Statements for All Customers**

BEGIN

FOR transaction\_rec IN (SELECT CustomerID, TransactionDate, Amount, TransactionType

FROM Transactions

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

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

DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(transaction\_rec.TransactionDate, 'DD-MON-YYYY'));

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

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

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

END

LOOP;

END;

**Scenario 2: Apply Annual Fee to All Accounts**

BEGIN

FOR acct\_rec IN (SELECT AccountID, Balance FROM Accounts) LOOP

UPDATE Accounts

SET Balance = Balance - 50 -- Assuming a flat $50 fee

WHERE AccountID = acct\_rec.AccountID;

END LOOP;

COMMIT;

END;

**Scenario 3: Update Loan Interest Rates Based on New Policy**

BEGIN

FOR loan\_rec IN (SELECT LoanID, InterestRate FROM Loans) LOOP

UPDATE Loans

SET InterestRate = InterestRate \* 1.05 -- Assuming a 5% increase

WHERE LoanID = loan\_rec.LoanID;

END LOOP;

COMMIT;

END;

**Exercise 7: Packages**

**Scenario 1: CustomerManagement Package**

CREATE OR REPLACE PACKAGE CustomerManagement IS

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

PROCEDURE UpdateCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER);

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

END CustomerManagement;

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

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

BEGIN

AddNewCustomer(p\_customer\_id, p\_name, p\_dob, p\_balance);

END AddCustomer;

PROCEDURE UpdateCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

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;

END GetCustomerBalance;

END CustomerManagement;

**Scenario 2: EmployeeManagement Package**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2);

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

END EmployeeManagement;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

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);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id 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\_employee\_id;

END UpdateEmployeeDetails;

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

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

END CalculateAnnualSalary;

END EmployeeManagement;

**Scenario 3: AccountOperations Package**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER);

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

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

END AccountOperations;

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

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

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

END OpenAccount;

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

BEGIN

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

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;

END GetTotalBalance;

END AccountOperations;