**Exercise 1: Control Structures**

**Scenario 1: Discount for customers above 60 years old**

DECLARE

CURSOR customer\_cur IS

SELECT CustomerID, DOB, Balance

FROM Customers;

v\_customer customer\_cur%ROWTYPE;

v\_age NUMBER;

BEGIN

FOR v\_customer IN customer\_cur LOOP

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

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = v\_customer.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 2: Promote customers 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;

COMMIT;

END;

**Scenario 3: Reminders for due loans**

DECLARE

CURSOR loan\_cur IS

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR v\_loan IN loan\_cur LOOP

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

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1: SafeTransferFunds stored procedure**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN Accounts.AccountID%TYPE,

p\_to\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

BEGIN

-- Check balance

DECLARE

v\_balance Accounts.Balance%TYPE;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

END;

-- Perform transfer

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

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

**Scenario 2: UpdateSalary stored procedure**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_percentage IN NUMBER

) IS

e\_employee\_not\_found EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_employee\_not\_found, -01403); -- NO\_DATA\_FOUND

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%NOTFOUND THEN

RAISE e\_employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN e\_employee\_not\_found THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

**Scenario 3: AddNewCustomer stored procedure**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

) IS

e\_duplicate\_customer EXCEPTION;

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

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest stored procedure**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

**Scenario 2: UpdateEmployeeBonus stored procedure**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN Employees.Department%TYPE,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END;

**Scenario 3: TransferFunds stored procedure**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN Accounts.AccountID%TYPE,

p\_to\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) IS

BEGIN

DECLARE

v\_balance Accounts.Balance%TYPE;

BEGIN

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

IF v\_balance < p\_amount THEN

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

END IF;

END;

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 OTHERS THEN

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

ROLLBACK;

END;

**Exercise 4: Functions**

**Scenario 1: CalculateAge function**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END;

**Scenario 2: CalculateMonthlyInstallment function**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_loan\_duration IN NUMBER

) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_installment := p\_loan\_amount \* (p\_interest\_rate / 1200) /

(1 - POWER(1 + (p\_interest\_rate / 1200), -p\_loan\_duration \* 12));

RETURN v\_monthly\_installment;

END;

**Scenario 3: HasSufficientBalance function**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) RETURN BOOLEAN IS

v\_balance Accounts.Balance%TYPE;

BEGIN

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

RETURN v\_balance >= p\_amount;

END;

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified trigger**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Scenario 2: LogTransaction trigger**

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 trigger**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

DECLARE

v\_balance Accounts.Balance%TYPE;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF v\_balance < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal amount exceeds account balance.');

END IF;

END;

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 using explicit cursor**

DECLARE

CURSOR transaction\_cur IS

SELECT CustomerID, TransactionDate, Amount, TransactionType

FROM Transactions

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

v\_transaction transaction\_cur%ROWTYPE;

BEGIN

OPEN transaction\_cur;

LOOP

FETCH transaction\_cur INTO v\_transaction;

EXIT WHEN transaction\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || v\_transaction.CustomerID || ': ' || v\_transaction.TransactionType || ' of ' || v\_transaction.Amount || ' on ' || v\_transaction.TransactionDate);

END LOOP;

CLOSE transaction\_cur;

END;

**Scenario 2: ApplyAnnualFee using explicit cursor**

DECLARE

CURSOR account\_cur IS

SELECT AccountID, Balance

FROM Accounts;

v\_account account\_cur%ROWTYPE;

BEGIN

OPEN account\_cur;

LOOP

FETCH account\_cur INTO v\_account;

EXIT WHEN account\_cur%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - 100 -- Annual fee

WHERE AccountID = v\_account.AccountID;

END LOOP;

CLOSE account\_cur;

COMMIT;

END;

**Scenario 3: UpdateLoanInterestRates using explicit cursor**

DECLARE

CURSOR loan\_cur IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan loan\_cur%ROWTYPE;

BEGIN

OPEN loan\_cur;

LOOP

FETCH loan\_cur INTO v\_loan;

EXIT WHEN loan\_cur%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_loan.InterestRate + 1 -- New policy

WHERE LoanID = v\_loan.LoanID;

END LOOP;

CLOSE loan\_cur;

COMMIT;

END;

**Exercise 7: Packages**

**Scenario 1: CustomerManagement package**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE);

FUNCTION GetCustomerBalance(p\_customer\_id IN Customers.CustomerID%TYPE) RETURN NUMBER;

END CustomerManagement;

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE) IS

BEGIN

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

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

COMMIT;

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE) IS

BEGIN

UPDATE Customers

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

WHERE CustomerID = p\_customer\_id;

COMMIT;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id IN Customers.CustomerID%TYPE) 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 AS

PROCEDURE HireEmployee(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE, p\_hiredate IN Employees.HireDate%TYPE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE, p\_hiredate IN Employees.HireDate%TYPE);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN Employees.EmployeeID%TYPE) RETURN NUMBER;

END EmployeeManagement;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE, p\_hiredate IN Employees.HireDate%TYPE) IS

BEGIN

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

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hiredate);

COMMIT;

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE, p\_hiredate IN Employees.HireDate%TYPE) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary, Department = p\_department, HireDate = p\_hiredate

WHERE EmployeeID = p\_employee\_id;

COMMIT;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN Employees.EmployeeID%TYPE) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary;

END CalculateAnnualSalary;

END EmployeeManagement;

**Scenario 3: AccountOperations package**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN Accounts.AccountID%TYPE, p\_customer\_id IN Accounts.CustomerID%TYPE, p\_account\_type IN Accounts.AccountType%TYPE, p\_balance IN Accounts.Balance%TYPE);

PROCEDURE CloseAccount(p\_account\_id IN Accounts.AccountID%TYPE);

FUNCTION GetTotalBalance(p\_customer\_id IN Accounts.CustomerID%TYPE) RETURN NUMBER;

END AccountOperations;

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN Accounts.AccountID%TYPE, p\_customer\_id IN Accounts.CustomerID%TYPE, p\_account\_type IN Accounts.AccountType%TYPE, p\_balance IN Accounts.Balance%TYPE) IS

BEGIN

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

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

COMMIT;

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN Accounts.AccountID%TYPE) IS

BEGIN

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

COMMIT;

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id IN Accounts.CustomerID%TYPE) 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;