**Exercise 1: Control Structures**

**Scenario 1: Discount on Loan Interest Rates for Customers Above 60 Years Old**

DECLARE

CURSOR c\_customers IS

SELECT CustomerID FROM Customers WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) > 60;

BEGIN

FOR r\_customer IN c\_customers LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = r\_customer.CustomerID;

END LOOP;

COMMIT;

END;

/

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

DECLARE

CURSOR c\_customers IS

SELECT CustomerID FROM Customers WHERE Balance > 10000;

BEGIN

FOR r\_customer IN c\_customers LOOP

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = r\_customer.CustomerID;

END LOOP;

COMMIT;

END;

/

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

DECLARE

CURSOR c\_loans IS

SELECT l.LoanID, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR r\_loan IN c\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || r\_loan.LoanID || ' for customer ' || r\_loan.Name || ' is due within the next 30 days.');

END LOOP;

END;

/

### Exercise 2: Error Handling

#### Scenario 1: SafeTransferFunds Stored Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

BEGIN

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;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

#### Scenario 2: UpdateSalary Stored Procedure

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END;

/

#### Scenario 3: AddNewCustomer Stored Procedure

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

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 ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

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

END;

/

### Exercise 3: Stored Procedures

#### Scenario 1: ProcessMonthlyInterest Stored Procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

/

#### Scenario 2: UpdateEmployeeBonus Stored Procedure

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;

/

#### Scenario 3: TransferFunds Stored Procedure

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

BEGIN

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance >= p\_amount THEN

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;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in account ' || p\_from\_account);

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

/

### Exercise 4: Functions

#### Scenario 1: CalculateAge Function

CREATE OR REPLACE FUNCTION CalculateAge(

p\_dob IN DATE

) RETURN NUMBER IS

BEGIN

RETURN FLOOR((SYSDATE - p\_dob) / 365);

END;

/

#### Scenario 2: CalculateMonthlyInstallment Function

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_duration\_years IN NUMBER

) RETURN NUMBER IS

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_months NUMBER;

BEGIN

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

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

RETURN p\_loan\_amount \* v\_monthly\_interest\_rate / (1 - POWER(1 + v\_monthly\_interest\_rate, -v\_number\_of\_months));

END;

/

#### Scenario 3: HasSufficientBalance Function

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;

/

### 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

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 PL/SQL Block

DECLARE

CURSOR c\_transactions IS

SELECT CustomerID, TransactionDate, Amount, TransactionType

FROM Transactions

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

BEGIN

FOR r\_transaction IN c\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || r\_transaction.CustomerID ||

', Date: ' || r\_transaction.TransactionDate ||

', Amount: ' || r\_transaction.Amount ||

', Type: ' || r\_transaction.TransactionType);

END LOOP;

END;

/

#### Scenario 2: ApplyAnnualFee PL/SQL Block

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

BEGIN

FOR r\_account IN c\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 50

WHERE AccountID = r\_account.AccountID;

END LOOP;

COMMIT;

END;

/

#### Scenario 3: UpdateLoanInterestRates PL/SQL Block

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

BEGIN

FOR r\_loan IN c\_loans LOOP

UPDATE Loans

SET InterestRate = r\_loan.InterestRate + 0.5

WHERE LoanID = r\_loan.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 UpdateCustomerDetails(

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

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

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

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END AddCustomer;

PROCEDURE UpdateCustomerDetails(

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;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END UpdateCustomerDetails;

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

RETURN NULL;

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,

p\_hire\_date IN DATE

);

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,

p\_hire\_date IN DATE

) IS

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

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;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

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);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END OpenAccount;

PROCEDURE CloseAccount(

p\_account\_id IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

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

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/