## Exercise 1: Control Structures Solutions

```
-- Scenario 1
DECLARE
   CURSOR customer_cursor IS
      SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate
      FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID;
   v_age NUMBER;
BEGIN
   FOR cust_rec IN customer_cursor LOOP
      v_age := FLOOR(MONTHS_BETWEEN(SYSDATE, cust_rec.DOB) / 12);
      IF v_age > 60 THEN
         UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID = cust_rec.LoanID;
      END IF;
   END LOOP;
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
/

-- Scenario 2
DECLARE
   CURSOR customer_cursor IS SELECT CustomerID, Balance FROM Customers;
BEGIN
   FOR cust_rec IN customer_cursor LOOP
      IF cust_rec.Balance > 10000 THEN
         UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = cust_rec.CustomerID;
      END IF;
   END LOOP;
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
/

-- Scenario 3
DECLARE
   CURSOR loan_cursor IS
      SELECT l.LoanID, c.CustomerID, c.Name, l.EndDate
      FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID
      WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;
BEGIN
   FOR loan_rec IN loan_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Reminder: Customer ' || loan_rec.Name ||
                           ' has loan due on ' || TO_CHAR(loan_rec.EndDate, 'DD-MON-YYYY'));
   END LOOP;
EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM); END;
```

## Exercise 2: Error Handling Solutions

**-- Scenario 1**

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(
   p_from_account IN NUMBER, p_to_account IN NUMBER, p_amount IN NUMBER) AS
   v_from_balance NUMBER; v_to_account_exists NUMBER;
BEGIN
    SELECT Balance INTO v_from_balance FROM Accounts WHERE AccountID = p_from_account FOR UPDATE;
     IF v_from_balance < p_amount THEN RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds');
END IF;
   SELECT COUNT(*) INTO v_to_account_exists FROM Accounts WHERE AccountID = p_to_account;
     IF v_to_account_exists = 0 THEN RAISE_APPLICATION_ERROR(-20002, 'Account not found');
END IF;
   UPDATE Accounts SET Balance = Balance - p_amount WHERE AccountID = p_from_account;
   UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_account;
       INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, p_from_account, SYSDATE, p_amount, 'Transfer');
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

**-- Scenario 2**

```
CREATE OR REPLACE PROCEDURE UpdateSalary(p_employee_id IN NUMBER, p_percentage IN NUMBER) AS
   v_employee_exists NUMBER;
BEGIN
       SELECT COUNT(*) INTO v_employee_exists FROM Employees WHERE EmployeeID = p_employee_id;
     IF v_employee_exists = 0 THEN RAISE_APPLICATION_ERROR(-20003, 'Employee not found');
END IF;
      UPDATE Employees SET Salary = Salary * (1 + p_percentage/100) WHERE EmployeeID = p_employee_id;
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

**-- Scenario 3**

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(
     p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE, p_balance IN NUMBER DEFAULT 0) AS
   v_customer_exists NUMBER;
```

```
BEGIN
        SELECT COUNT(*) INTO v_customer_exists FROM Customers WHERE CustomerID = p_customer_id;
   IF v_customer_exists > 0 THEN RAISE_APPLICATION_ERROR(-20004, 'Customer exists'); END IF;
   INSERT INTO Customers VALUES (p_customer_id, p_name, p_dob, p_balance, SYSDATE);
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

## Exercise 3: Stored Procedures Solutions

**-- Scenario 1**

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
   CURSOR savings_accounts IS SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings' FOR UPDATE;
   v_interest_rate NUMBER := 1;
BEGIN
   FOR acc_rec IN savings_accounts LOOP
         UPDATE Accounts SET Balance = Balance * (1 + v_interest_rate/100) WHERE AccountID = acc_rec.AccountID;
      INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, acc_rec.AccountID, SYSDATE,
                        acc_rec.Balance * v_interest_rate/100, 'Interest');
   END LOOP;
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

**-- Scenario 2**

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p_department IN VARCHAR2, p_bonus_percentage IN NUMBER) AS
BEGIN
     UPDATE Employees SET Salary = Salary * (1 + p_bonus_percentage/100) WHERE Department = p_department;
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

**-- Scenario 3**

```
CREATE OR REPLACE PROCEDURE TransferFunds(
   p_from_account IN NUMBER, p_to_account IN NUMBER, p_amount IN NUMBER) AS
   v_from_balance NUMBER;
BEGIN
```

```
    SELECT Balance INTO v_from_balance FROM Accounts WHERE AccountID = p_from_account FOR
UPDATE;
     IF v_from_balance < p_amount THEN RAISE_APPLICATION_ERROR(-20005, 'Insufficient funds');
END IF;
   UPDATE Accounts SET Balance = Balance - p_amount WHERE AccountID = p_from_account;
   UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_account;
       INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, p_from_account, SYSDATE,
p_amount, 'Withdrawal');
        INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, p_to_account, SYSDATE,
p_amount, 'Deposit');
   COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

## Exercise 4: Functions Solutions

**-- Scenario 1**

```
CREATE OR REPLACE FUNCTION CalculateAge(p_dob IN DATE) RETURN NUMBER IS
BEGIN RETURN FLOOR(MONTHS_BETWEEN(SYSDATE, p_dob) / 12); EXCEPTION WHEN
OTHERS THEN RETURN NULL; END;
```

**-- Scenario 2**

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
      p_loan_amount IN NUMBER, p_interest_rate IN NUMBER, p_years IN NUMBER) RETURN
NUMBER IS
   v_monthly_rate NUMBER := p_interest_rate / 12 / 100;
   v_num_payments NUMBER := p_years * 12;
BEGIN
          RETURN ROUND(p_loan_amount * v_monthly_rate * POWER(1 + v_monthly_rate,
v_num_payments) /
           (POWER(1 + v_monthly_rate, v_num_payments) - 1), 2);
EXCEPTION WHEN OTHERS THEN RETURN NULL; END;
```

**-- Scenario 3**

```
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;
```

EXCEPTION WHEN NO_DATA_FOUND THEN RETURN FALSE; WHEN OTHERS THEN RETURN FALSE; END;

## Exercise 5: Triggers Solutions

**-- Scenario 1**

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers FOR EACH ROW
BEGIN :NEW.LastModified := SYSDATE; END;
```

**-- Scenario 2**

```
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions FOR EACH ROW
BEGIN
   INSERT INTO AuditLog VALUES (AuditLogSeq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID,
                        SYSDATE, :NEW.TransactionType, :NEW.Amount, USER);
END;
```

**-- Scenario 3**

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions FOR EACH ROW
DECLARE v_balance NUMBER;
BEGIN
   IF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN
      RAISE_APPLICATION_ERROR(-20006, 'Deposit must be positive');
   END IF;
   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(-20007, 'Insufficient balance');
      END IF;
   END IF;
END;
```

## Exercise 6: Cursors Solutions

```
-- Scenario 1
CREATE OR REPLACE PROCEDURE GenerateMonthlyStatements AS
   CURSOR customer_cursor IS
      SELECT DISTINCT c.CustomerID, c.Name FROM Customers c
      JOIN Accounts a ON c.CustomerID = a.CustomerID
      JOIN Transactions t ON a.AccountID = t.AccountID
      WHERE EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)
      AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);
   CURSOR transaction_cursor(p_customer_id NUMBER) IS
      SELECT t.TransactionID, t.TransactionDate, t.Amount, t.TransactionType, a.AccountID
      FROM Transactions t JOIN Accounts a ON t.AccountID = a.AccountID
      WHERE a.CustomerID = p_customer_id
      AND EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)
      AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE)
      ORDER BY t.TransactionDate;
BEGIN
   FOR cust_rec IN customer_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Statement for ' || cust_rec.Name);
      FOR trans_rec IN transaction_cursor(cust_rec.CustomerID) LOOP
         DBMS_OUTPUT.PUT_LINE('Account: ' || trans_rec.AccountID || ' | Date: ' ||
                              TO_CHAR(trans_rec.TransactionDate, 'DD-MON-YYYY') ||
                              ' | Type: ' || trans_rec.TransactionType ||
                              ' | Amount: ' || trans_rec.Amount);
      END LOOP;
   END LOOP;
EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM); END;


-- Scenario 2
CREATE OR REPLACE PROCEDURE ApplyAnnualFee AS
   CURSOR account_cursor IS
        SELECT AccountID, Balance FROM Accounts WHERE AccountType IN ('Checking', 'Savings')
FOR UPDATE;
   v_annual_fee NUMBER := 25;
BEGIN
   FOR acc_rec IN account_cursor LOOP
      IF acc_rec.Balance >= v_annual_fee THEN
                 UPDATE Accounts SET Balance = Balance - v_annual_fee WHERE AccountID =
acc_rec.AccountID;
               INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, acc_rec.AccountID,
SYSDATE, v_annual_fee, 'Fee');
```

```
    END IF;
  END LOOP;
  COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
```

**-- Scenario 3**

```
CREATE OR REPLACE PROCEDURE UpdateLoanInterestRates AS
  CURSOR loan_cursor IS SELECT LoanID, InterestRate, LoanAmount FROM Loans FOR UPDATE;
  v_new_rate NUMBER;
BEGIN
  FOR loan_rec IN loan_cursor LOOP
    IF loan_rec.LoanAmount > 100000 THEN v_new_rate := 3.5;
    ELSIF loan_rec.LoanAmount > 50000 THEN v_new_rate := 4.0;
    ELSE v_new_rate := 5.0; END IF;
    UPDATE Loans SET InterestRate = v_new_rate WHERE LoanID = loan_rec.LoanID;
  END LOOP;
  COMMIT;
EXCEPTION WHEN OTHERS THEN ROLLBACK; END;
/
```

## Exercise 7: Packages Solutions

**-- Scenario 1**

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
   PROCEDURE AddCustomer(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE,
p_balance IN NUMBER DEFAULT 0);
   PROCEDURE UpdateCustomerDetails(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob
IN DATE);
  FUNCTION GetCustomerBalance(p_customer_id IN NUMBER) RETURN NUMBER;
END;

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
   PROCEDURE AddCustomer(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE,
p_balance IN NUMBER DEFAULT 0) AS
    v_customer_exists NUMBER;
  BEGIN
        SELECT COUNT(*) INTO v_customer_exists FROM Customers WHERE CustomerID =
p_customer_id;
     IF v_customer_exists > 0 THEN RAISE_APPLICATION_ERROR(-20008, 'Customer exists'); END
IF;
    INSERT INTO Customers VALUES (p_customer_id, p_name, p_dob, p_balance, SYSDATE);
```

```
    COMMIT;
  EXCEPTION WHEN OTHERS THEN ROLLBACK; RAISE; END;

   PROCEDURE UpdateCustomerDetails(p_customer_id IN NUMBER, p_name IN VARCHAR2, p_dob IN DATE) AS
    v_customer_exists NUMBER;
  BEGIN
        SELECT COUNT(*) INTO v_customer_exists FROM Customers WHERE CustomerID = p_customer_id;
      IF v_customer_exists = 0 THEN RAISE_APPLICATION_ERROR(-20009, 'Customer not found'); END IF;
      UPDATE Customers SET Name = p_name, DOB = p_dob, LastModified = SYSDATE WHERE CustomerID = p_customer_id;
    COMMIT;
  EXCEPTION WHEN OTHERS THEN ROLLBACK; RAISE; END;

  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; WHEN OTHERS THEN RETURN NULL; END;
END;
```

**-- Scenario 2**

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
   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;
/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
   PROCEDURE HireEmployee(p_employee_id IN NUMBER, p_name IN VARCHAR2, p_position IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2) AS
    v_employee_exists NUMBER;
  BEGIN
        SELECT COUNT(*) INTO v_employee_exists FROM Employees WHERE EmployeeID = p_employee_id;
```

```
        IF v_employee_exists > 0 THEN RAISE_APPLICATION_ERROR(-20010, 'Employee exists'); END IF;
        INSERT INTO Employees VALUES (p_employee_id, p_name, p_position, p_salary, p_department, SYSDATE);
       COMMIT;
   EXCEPTION WHEN OTHERS THEN ROLLBACK; RAISE; END;

       PROCEDURE UpdateEmployeeDetails(p_employee_id IN NUMBER, p_name IN VARCHAR2, p_position IN VARCHAR2, p_salary IN NUMBER, p_department IN VARCHAR2) AS
       v_employee_exists NUMBER;
   BEGIN
            SELECT COUNT(*) INTO v_employee_exists FROM Employees WHERE EmployeeID = p_employee_id;
        IF v_employee_exists = 0 THEN RAISE_APPLICATION_ERROR(-20011, 'Employee not found'); END IF;
        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 ROLLBACK; RAISE; END;

   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; WHEN OTHERS THEN RETURN NULL; END;
END;
/

-- Scenario 3
CREATE OR REPLACE PACKAGE AccountOperations AS
         PROCEDURE OpenAccount(p_account_id IN NUMBER, p_customer_id IN NUMBER, p_account_type IN VARCHAR2, p_initial_balance IN NUMBER DEFAULT 0);
   PROCEDURE CloseAccount(p_account_id IN NUMBER);
   FUNCTION GetTotalBalance(p_customer_id IN NUMBER) RETURN NUMBER;
END;
/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS
         PROCEDURE OpenAccount(p_account_id IN NUMBER, p_customer_id IN NUMBER, p_account_type IN VARCHAR2, p_initial_balance IN NUMBER DEFAULT 0) AS
      v_account_exists NUMBER; v_customer_exists NUMBER;
   BEGIN
```

```
    SELECT COUNT(*) INTO v_account_exists FROM Accounts WHERE AccountID = p_account_id;
    IF v_account_exists > 0 THEN RAISE_APPLICATION_ERROR(-20012, 'Account exists'); END IF;
        SELECT COUNT(*) INTO v_customer_exists FROM Customers WHERE CustomerID = p_customer_id;
      IF v_customer_exists = 0 THEN RAISE_APPLICATION_ERROR(-20013, 'Customer not found'); END IF;
    INSERT INTO Accounts VALUES (p_account_id, p_customer_id, p_account_type, p_initial_balance, SYSDATE);
    IF p_initial_balance > 0 THEN
        INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, p_account_id, SYSDATE, p_initial_balance, 'Deposit');
    END IF;
    COMMIT;
  EXCEPTION WHEN OTHERS THEN ROLLBACK; RAISE; END;

  PROCEDURE CloseAccount(p_account_id IN NUMBER) AS
    v_account_balance NUMBER; v_account_exists NUMBER;
  BEGIN
    SELECT COUNT(*) INTO v_account_exists FROM Accounts WHERE AccountID = p_account_id;
    IF v_account_exists = 0 THEN RAISE_APPLICATION_ERROR(-20014, 'Account not found'); END IF;
    SELECT Balance INTO v_account_balance FROM Accounts WHERE AccountID = p_account_id;
    IF v_account_balance > 0 THEN
      DECLARE
        v_primary_account NUMBER; v_customer_id NUMBER;
      BEGIN
          SELECT CustomerID INTO v_customer_id FROM Accounts WHERE AccountID = p_account_id;
        SELECT MIN(AccountID) INTO v_primary_account FROM Accounts WHERE CustomerID = v_customer_id AND AccountID != p_account_id;
        IF v_primary_account IS NOT NULL THEN
            UPDATE Accounts SET Balance = Balance + v_account_balance WHERE AccountID = v_primary_account;
          INSERT INTO Transactions VALUES (TransactionSeq.NEXTVAL, v_primary_account, SYSDATE, v_account_balance, 'Transfer');
        ELSE RAISE_APPLICATION_ERROR(-20015, 'Withdraw balance first'); END IF;
      END;
    END IF;
    DELETE FROM Accounts WHERE AccountID = p_account_id;
    COMMIT;
  EXCEPTION WHEN OTHERS THEN ROLLBACK; RAISE; END;

  FUNCTION GetTotalBalance(p_customer_id IN NUMBER) RETURN NUMBER IS
    v_total_balance NUMBER := 0;
```

```
   BEGIN
        SELECT NVL(SUM(Balance), 0) INTO v_total_balance FROM Accounts WHERE CustomerID =
p_customer_id;
      RETURN v_total_balance;
   EXCEPTION WHEN OTHERS THEN RETURN 0; END;
END;
/
```