- **Exercise 1: Control Structures**

  1. **Scenario 1: Apply Discount to Loan Interest Rates**

```
BEGIN
  FOR rec IN (SELECT LoanID, InterestRate FROM Loans WHERE CustomerID IN (SELECT
CustomerID FROM Customers WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM
DOB) > 60)) LOOP
    UPDATE Loans
    SET InterestRate = InterestRate - 1
    WHERE LoanID = rec.LoanID;
  END LOOP;
END;
/
```

  2. **Scenario 2: Set VIP Status Based on Balance**

```
BEGIN
  FOR rec IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP
    UPDATE Customers
    SET IsVIP = TRUE
    WHERE CustomerID = rec.CustomerID;
  END LOOP;
END;
/
```

  3. **Scenario 3: Send Reminders for Loans Due**

```
BEGIN
  FOR rec IN (SELECT LoanID, CustomerID FROM Loans WHERE EndDate <= SYSDATE + 30)
LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Customer ' || rec.CustomerID || ', your loan ' ||
rec.LoanID || ' is due within 30 days.');
  END LOOP;
END;
/
```

- **Exercise 2: Error Handling**

  1. **Scenario 1: SafeTransferFunds Procedure**

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(
    p_source_account_id IN NUMBER,
    p_dest_account_id IN NUMBER,
    p_amount IN NUMBER
) AS
BEGIN
  UPDATE Accounts
  SET Balance = Balance - p_amount
  WHERE AccountID = p_source_account_id;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Source account not found or insufficient funds.');
  END IF;

  UPDATE Accounts
  SET Balance = Balance + p_amount
  WHERE AccountID = p_dest_account_id;
  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Destination account not found.');
  END IF;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

2. **Scenario 2: UpdateSalary Procedure**

```
CREATE OR REPLACE PROCEDURE UpdateSalary(
    p_employee_id IN NUMBER,
    p_percentage IN NUMBER
) AS
BEGIN
  UPDATE Employees
  SET Salary = Salary * (1 + p_percentage / 100)
  WHERE EmployeeID = p_employee_id;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20003, 'Employee ID not found.');
  END IF;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

3. **Scenario 3: AddNewCustomer Procedure**

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(
    p_customer_id IN NUMBER,
    p_name IN VARCHAR2,
    p_dob IN DATE,
    p_balance IN NUMBER
) AS
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
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Customer with this ID already exists.');
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;/
```

- ## Exercise 3: Stored Procedures

1. **Scenario 1: ProcessMonthlyInterest Procedure**

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
BEGIN
  FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings')
LOOP
    UPDATE Accounts
    SET Balance = Balance * 1.01
    WHERE AccountID = rec.AccountID;
  END LOOP;

  COMMIT;
END;
/
```

2. **Scenario 2: UpdateEmployeeBonus Procedure**

```
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;
END;
/
```

3. **Scenario 3: TransferFunds Procedure**

```
CREATE OR REPLACE PROCEDURE TransferFunds(
    p_source_account_id IN NUMBER,
    p_dest_account_id IN NUMBER,
    p_amount IN NUMBER
) AS
BEGIN
  UPDATE Accounts
  SET Balance = Balance - p_amount
  WHERE AccountID = p_source_account_id;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Source account not found or insufficient funds.');
  END IF;

  UPDATE Accounts
  SET Balance = Balance + p_amount
  WHERE AccountID = p_dest_account_id;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Destination account not found.');
  END IF;

  COMMIT;
```

```
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

- **Exercise 4: Functions**
  1. **Scenario 1: CalculateAge Function**

```
CREATE OR REPLACE FUNCTION CalculateAge(p_dob DATE) RETURN NUMBER IS
  v_age NUMBER;
BEGIN
  SELECT TRUNC(MONTHS_BETWEEN(SYSDATE, p_dob) / 12) INTO v_age FROM
DUAL;
  RETURN v_age;
END;
/
```

  2. **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_rate NUMBER;
  v_monthly_installment NUMBER;
BEGIN
  v_monthly_rate := p_interest_rate / 12 / 100;
  v_monthly_installment := p_loan_amount * v_monthly_rate / (1 - POWER(1 +
v_monthly_rate, -p_duration_years * 12));
  RETURN v_monthly_installment;
END;
/
```

  3. **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;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END;
/
```

- **Exercise 5: Triggers**

  1. **Scenario 1: UpdateCustomerLastModified Trigger**

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

  2. **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;
/
```

  3. **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 :NEW.Amount > v_balance THEN
      RAISE_APPLICATION_ERROR(-20004, 'Insufficient balance.');
    END IF;
  ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN
    RAISE_APPLICATION_ERROR(-20005, 'Deposit amount must be positive.');
  END IF;
END;
/
```

- ## Exercise 6: Cursors

  1. **Scenario 1: GenerateMonthlyStatements Block**

```
DECLARE
  CURSOR cur_transactions IS
    SELECT AccountID, SUM(Amount) AS Total FROM Transactions
    WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST_DAY(SYSDATE)
    GROUP BY AccountID;
BEGIN
  FOR rec IN cur_transactions LOOP
    DBMS_OUTPUT.PUT_LINE('Account ' || rec.AccountID || ' has total transactions of ' ||
rec.Total || ' this month.');
  END LOOP;
END;
/
```

  2. **Scenario 2: ApplyAnnualFee Block**

```
DECLARE
  CURSOR cur_accounts IS
    SELECT AccountID, Balance FROM Accounts;
BEGIN
  FOR rec IN cur_accounts LOOP
    UPDATE Accounts
    SET Balance = Balance - 100 -- Annual fee amount
    WHERE AccountID = rec.AccountID;
  END LOOP;

  COMMIT;
END;
/
```

  3. **Scenario 3: UpdateLoanInterestRates Block**

```
DECLARE
  CURSOR cur_loans IS
    SELECT LoanID, InterestRate FROM Loans;
BEGIN
  FOR rec IN cur_loans LOOP
    UPDATE Loans
    SET InterestRate = rec.InterestRate + 0.5 -- Example new policy increment
    WHERE LoanID = rec.LoanID;
  END LOOP;

  COMMIT;
END;
/
```

- **Exercise 7: Packages**

1. **Scenario 1: CustomerManagement Package**

```
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);
  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 AS
  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;
  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;
    COMMIT;
  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;
/
```

2. **Scenario 2: EmployeeManagement Package**

```
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, p_hire_date IN DATE);
  PROCEDURE UpdateEmployee(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 AS
  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;
 END HireEmployee;

 PROCEDURE UpdateEmployee(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;
 END UpdateEmployee;

 FUNCTION CalculateAnnualSalary(p_employee_id IN NUMBER) 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;
/
```

3. **Scenario 3: AccountOperations Package**

```
CREATE OR REPLACE PACKAGE AccountOperations AS
 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 AS
 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;
 END OpenAccount;

 PROCEDURE CloseAccount(p_account_id IN NUMBER) IS
 BEGIN
  DELETE FROM Accounts WHERE AccountID = p_account_id;
  COMMIT;
 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;/
```