

## **PL/SQL programming**

### **Mandatory Exercises:**

#### **Exercise 1: Control Structures**

##### **1. IF-ELSE Statement**

###### **Code:**

```
DECLARE

    num NUMBER := 8;

BEGIN

    IF MOD(num, 2) = 0 THEN

        DBMS_OUTPUT.PUT_LINE('Even Number');

    ELSE

        DBMS_OUTPUT.PUT_LINE('Odd Number');

    END IF;

END;

/
```

###### **Output:**

```
Even Number

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.005
```

##### **2. CASE Statement**

###### **Code:**

```
DECLARE

    marks NUMBER := 85;

    grade CHAR(1);

BEGIN
```

CASE

WHEN marks >= 90 THEN grade := 'A';

WHEN marks >= 80 THEN grade := 'B';

WHEN marks >= 70 THEN grade := 'C';

WHEN marks >= 60 THEN grade := 'D';

ELSE grade := 'F';

END CASE;

DBMS\_OUTPUT.PUT\_LINE('Grade: ' || grade);

END;

/

```
Grade: B
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.007
```

### 3. Simple LOOP

DECLARE

i NUMBER := 1;

BEGIN

LOOP

EXIT WHEN i > 5;

DBMS\_OUTPUT.PUT\_LINE('Value: ' || i);

i := i + 1;

END LOOP;

END;

/

**Output:**

```
Value: 1  
Value: 2  
Value: 3  
Value: 4  
Value: 5
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.006
```

#### 4. WHILE LOOP

##### Code:

```
DECLARE
```

```
    num NUMBER := 5;
```

```
    result NUMBER := 1;
```

```
BEGIN
```

```
    WHILE num > 0 LOOP
```

```
        result := result * num;
```

```
        num := num - 1;
```

```
    END LOOP;
```

```
    DBMS_OUTPUT.PUT_LINE('Factorial: ' || result);
```

```
END;
```

```
/
```

##### Output:

```
Factorial: 120
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.006
```

## 5. FOR LOOP

### Code:

```
DECLARE  
    i NUMBER;  
BEGIN  
    FOR i IN 1..10 LOOP  
        DBMS_OUTPUT.PUT_LINE('3 x ' || i || ' = ' || (3 * i));  
    END LOOP;  
END;  
/
```

### Output:

```
3 x 1 = 3  
3 x 2 = 6  
3 x 3 = 9  
3 x 4 = 12  
3 x 5 = 15  
3 x 6 = 18  
3 x 7 = 21  
3 x 8 = 24  
3 x 9 = 27  
3 x 10 = 30
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.006
```

## Exercise 3: Stored Procedures

### 1.Procedure with IN and OUT Parameters

#### Code:

```
-- Create procedure  
CREATE OR REPLACE PROCEDURE get_factorial (  
    num IN NUMBER,
```

```
fact OUT NUMBER
)
IS
    i NUMBER := 1;
BEGIN
    fact := 1;
    WHILE i <= num LOOP
        fact := fact * i;
        i := i + 1;
    END LOOP;
END;
/
-- Call the procedure
DECLARE
    result NUMBER;
BEGIN
    get_factorial(5, result);
    DBMS_OUTPUT.PUT_LINE('Factorial is: ' || result);
END;
/
```

**Output:**

```
Factorial is: 120
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.005
```

## 2. Stored Procedure with IN Parameter

```
-- Create procedure  
  
CREATE OR REPLACE PROCEDURE print_square (  
    num IN NUMBER  
)  
IS  
    result NUMBER;  
  
BEGIN  
    result := num * num;  
    DBMS_OUTPUT.PUT_LINE('Square of ' || num || ' is ' || result);  
END;  
/
```

```
Procedure PRINT_SQUARE compiled  
Elapsed: 00:00:00.013
```

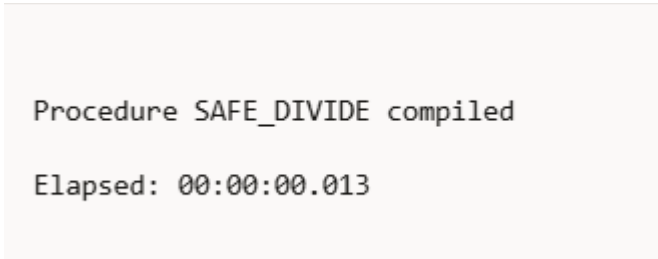
```
-- Call the procedure  
  
BEGIN  
    print_square(6);  
END;  
/  
  
Square of 6 is 36  
  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.004
```

### 3. Procedure with Exception Handling

-- Create procedure

```
CREATE OR REPLACE PROCEDURE safe_divide (  
    a IN NUMBER,  
    b IN NUMBER  
)  
IS  
    result NUMBER;  
BEGIN  
    result := a / b;  
    DBMS_OUTPUT.PUT_LINE('Result: ' || result);  
EXCEPTION  
    WHEN ZERO_DIVIDE THEN  
        DBMS_OUTPUT.PUT_LINE('Error: Cannot divide by zero.');
```

END;  
/



```
Procedure SAFE_DIVIDE compiled  
Elapsed: 00:00:00.013
```

-- Call the procedure

```
BEGIN  
    safe_divide(10, 0);  
END;  
/  

```

```
Error: Cannot divide by zero.
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.006
```

## **Additional Exercises:**

### **Schema**

#### **Step 1: Tables Creation**

```
BEGIN
EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';
EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';
EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';
EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';
EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
-- Create tables
CREATE TABLE Customers (
  CustomerID NUMBER PRIMARY KEY,
  Name VARCHAR2(100),
  DOB DATE,
  Balance NUMBER,
  LastModified DATE
);
CREATE TABLE Accounts (
  AccountID NUMBER PRIMARY KEY,
  CustomerID NUMBER,
  AccountType VARCHAR2(20),
  Balance NUMBER,
  LastModified DATE,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Transactions (
  TransactionID NUMBER PRIMARY KEY,
```



```
AccountID NUMBER,  
TransactionDate DATE,  
Amount NUMBER,  
TransactionType VARCHAR2(10),  
FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)  
);
```

```
CREATE TABLE Loans (  
    LoanID NUMBER PRIMARY KEY,  
    CustomerID NUMBER,  
    LoanAmount NUMBER,  
    InterestRate NUMBER,  
    StartDate DATE,  
    EndDate DATE,  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

```
CREATE TABLE Employees (  
    EmployeeID NUMBER PRIMARY KEY,  
    Name VARCHAR2(100),  
    Position VARCHAR2(50),  
    Salary NUMBER,  
    Department VARCHAR2(50),  
    HireDate DATE  
);
```

Step 2: Insert Sample Data

-- Insert sample data

```
INSERT INTO Customers VALUES (1, 'John Doe', TO_DATE('1960-05-15', 'YYYY-MM-DD'),  
12000, SYSDATE);
```

```
INSERT INTO Customers VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'),  
8000, SYSDATE);
```

```
INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);
```

```
INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);
```

```
INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');
```

```
INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');
```

```
INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));
```

```
INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR',  
TO_DATE('2015-06-15', 'YYYY-MM-DD'));  
INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT',  
TO_DATE('2017-03-20', 'YYYY-MM-DD'));
```

## Exercise 1: Control Structures

### Scenario 1: Apply Discount to Senior Citizens' Loan

#### Code:

```
BEGIN  
FOR loan_rec IN (  
  SELECT  
    l.LoanID AS loan_id,  
    l.InterestRate,  
    c.CustomerID AS cust_id,  
    c.DOB  
  FROM Loans l  
  JOIN Customers c ON l.CustomerID = c.CustomerID  
)  
LOOP  
  IF MONTHS_BETWEEN(SYSDATE, loan_rec.DOB)/12 > 60 THEN  
    UPDATE Loans  
    SET InterestRate = InterestRate - 1  
    WHERE LoanID = loan_rec.loan_id;  
  
    DBMS_OUTPUT.PUT_LINE('Loan ID ' || loan_rec.loan_id ||  
      ' updated for Customer ID ' || loan_rec.cust_id);  
  END IF;  
END LOOP;  
END;  
/
```

#### Output:

```
Loan ID 1 updated for Customer ID 1  
  
PL/SQL procedure successfully completed.  
  
Elapsed: 00:00:00.009
```

## Scenario 2 Promote VIP Customers

### Code:

```
BEGIN

FOR cust IN (SELECT * FROM Customers) LOOP

    IF cust.Balance > 10000 THEN

        DBMS_OUTPUT.PUT_LINE('Customer ' || cust.Name || ' promoted to VIP.');
```

### Output:

```
Customer John Doe promoted to VIP.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

## Scenario 3: Reminders for Upcoming Loan Dues

### Code:

```
BEGIN

FOR rec IN (

    SELECT * FROM Loans WHERE EndDate <= SYSDATE + 30

) LOOP

    DBMS_OUTPUT.PUT_LINE('Reminder: Loan for Customer ID ' || rec.CustomerID ||

        ' is due on ' || TO_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/
```

**Output:**

```
Reminder: Loan for Customer ID 2 is due on 08-JUL-2025
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.005
```

**Exercise 2: Error Handling****Scenario 1: Safe Transfer Funds with Output****Code:**

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds (  
    fromAcc NUMBER,  
    toAcc NUMBER,  
    amount NUMBER  
) IS  
    v_balance NUMBER;  
BEGIN  
    SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = fromAcc;  
    IF v_balance < amount THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds');  
    END IF;  
    UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;  
    UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAcc;  
    COMMIT;  
    DBMS_OUTPUT.PUT_LINE('Transfer successful.');
```

```
EXCEPTION  
    WHEN OTHERS THEN
```

```
ROLLBACK;

DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);

END;

/
```

**Output:**

```
Procedure SAFETRANSFERFUNDS compiled
```

```
Elapsed: 00:00:00.023
```

**Code:**

```
BEGIN

  SafeTransferFunds(1, 2, 100); -- Example: Transfer 100 from AccountID 1 to AccountID
  2

END;

/
```

**Output:**

```
Transfer successful.
```

```
PL/SQL procedure successfully completed.
```

**Scenario 2: Update Salary with Output**

**Code:**

```
CREATE OR REPLACE PROCEDURE UpdateSalary (

  emplId NUMBER,

  percent IN NUMBER

) IS

BEGIN

  UPDATE Employees SET Salary = Salary + (Salary * percent / 100) WHERE EmployeeID
  = emplId;

  IF SQL%ROWCOUNT = 0 THEN

    DBMS_OUTPUT.PUT_LINE('Error: Employee ID not found.');
```

```
ELSE
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Salary updated successfully.');
```

END IF;

END;

/

**Output:**

```
Procedure UPDATESALARY compiled
```

```
Elapsed: 00:00:00.018
```

**Code:**

```
BEGIN
    UpdateSalary(1, 10); -- Updates salary of employee with ID 1 by 10%
END;
```

/

**Output:**

```
Salary updated successfully.
```

```
PL/SQL procedure successfully completed.
```

	EMPLOYEEID	NAME	SALARY
1	1	Alice Johnson	77000

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

**Code:**

```
CREATE OR REPLACE PROCEDURE AddNewCustomer (
    p_id NUMBER,
    p_name VARCHAR2,
    p_dob DATE,
```

```

    p_balance NUMBER
) IS
BEGIN
    INSERT INTO Customers VALUES (p_id, p_name, p_dob, p_balance, SYSDATE);
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Customer added successfully.');
```

EXCEPTION

```

    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Customer ID already exists.');
```

END;

/

#### Output:

```

Procedure ADDNEWCUSTOMER compiled
Elapsed: 00:00:00.016
```

### Exercise 3: Stored Procedures

#### Scenario 1: Process Monthly Interest

##### Code:


```


CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
    UPDATE Accounts
    SET Balance = Balance + (Balance * 0.01)
    WHERE AccountType = 'Savings';
END;

/

-- Test
EXEC ProcessMonthlyInterest;
```

SELECT \* FROM Accounts;





Download

▼

Execution time: 0.006 seconds

	ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	1	1	Savings	909	6/28/2025, 7:14:41
2	2	2	Checking	1600	6/28/2025, 7:14:41

Scenario 2: Update Employee Bonus

Code:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(  
    dept IN VARCHAR2,  
    bonus_percent IN NUMBER  
) IS  
BEGIN  
    UPDATE Employees  
    SET Salary = Salary + (Salary * bonus_percent / 100)  
    WHERE Department = dept;  
END;  
  
/  
  
-- Test  
  
EXEC UpdateEmployeeBonus('IT', 10);  
  
SELECT * FROM Employees;
```

Output:

	EMPLOYEEID	NAME	POSITION	SALARY	DEPARTMENT
1	1	Alice Johnson	Manager	77000	HR
2	2	Bob Brown	Developer	66000	IT

Scenario 3: Transfer Funds

Code:



```
CREATE OR REPLACE PROCEDURE TransferFunds(
    from_acc IN NUMBER,
    to_acc IN NUMBER,
    amount IN NUMBER
) IS
    insufficient_balance EXCEPTION;
BEGIN
    DECLARE
        from_balance NUMBER;
    BEGIN
        SELECT Balance INTO from_balance FROM Accounts WHERE AccountID =
from_acc FOR UPDATE;

        IF from_balance < amount THEN
            RAISE insufficient_balance;
        END IF;

        UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from_acc;
        UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to_acc;

        INSERT INTO Transactions VALUES (
            Transactions_seq.NEXTVAL, from_acc, SYSDATE, amount, 'Transfer'
        );
        INSERT INTO Transactions VALUES (
            Transactions_seq.NEXTVAL, to_acc, SYSDATE, amount, 'Transfer'
        );
    END;
EXCEPTION
```

```

    WHEN insufficient_balance THEN

        DBMS_OUTPUT.PUT_LINE('Insufficient funds.');
```

END;

/

```

-- Test

EXEC TransferFunds(1, 2, 200);

SELECT * FROM Accounts;

SELECT * FROM Transactions;
```

**Output:**

	ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	1	1	Savings	909	6/28/2025, 7:14:41
2	2	2	Checking	1600	6/28/2025, 7:14:41

## Exercise 4: Functions

### Scenario 1: Calculate Age

**Code:**

```

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE) RETURN NUMBER IS

BEGIN

    RETURN FLOOR(MONTHS_BETWEEN(SYSDATE, dob)/12);

END;
```

/

```

-- Test

SELECT Name, CalculateAge(DOB) AS Age FROM Customers;
```

**Output:**

	NAME	AGE
1	John Doe	65
2	Jane Smith	34

## Scenario 2: Calculate Monthly Installment

### Code:

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(  
    loan_amount IN NUMBER,  
    annual_rate IN NUMBER,  
    years IN NUMBER  
) RETURN NUMBER IS  
    monthly_rate NUMBER;  
    n NUMBER;  
  
BEGIN  
    monthly_rate := annual_rate / (12 * 100);  
    n := years * 12;  
    RETURN loan_amount * monthly_rate / (1 - POWER(1 + monthly_rate, -n));  
  
END;  
  
/  
  
-- Test  
  
SELECT CalculateMonthlyInstallment(5000, 5, 5) AS MonthlyInstallment FROM DUAL;
```

### Output:

Function CALCULATEMONTHLYINSTALLMENT compiled

Elapsed: 00:00:00.003

	MONTHLYINSTALLMENT
1	4.35616822005467

### Scenario 3: Has Sufficient Balance

#### Code:

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(  
    acc_id IN NUMBER,  
    amt IN NUMBER  
) RETURN BOOLEAN IS  
    bal NUMBER;  
BEGIN  
    SELECT Balance INTO bal FROM Accounts WHERE AccountID = acc_id;  
    RETURN bal >= amt;  
END;  
  
/  
-- Test  
  
DECLARE  
    result BOOLEAN;  
BEGIN  
    result := HasSufficientBalance(1, 500);  
    IF result THEN  
        DBMS_OUTPUT.PUT_LINE('Sufficient');  
    ELSE  
        DBMS_OUTPUT.PUT_LINE('Insufficient');  
    END IF;  
END;  
  
/
```

#### Output:

```
Function HASSUFFICIENTBALANCE compiled
```

```
Elapsed: 00:00:00.013
```

Sufficient

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.006

## Exercise 5: Triggers

### Scenario 1: Update Last Modified

#### Code:

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
    :NEW.LastModified := SYSDATE;
END;
/
-- Test
UPDATE Customers SET Name = 'John D.' WHERE CustomerID = 1;
SELECT * FROM Customers;
```

#### Output:

Trigger UPDATECUSTOMERLASTMODIFIED compiled

Elapsed: 00:00:00.014

	CUSTOMERID	NAME	DOB	BALANCE	LASTMODIFIED
1	1	John D.	5/15/1960, 12:00:00	12000	6/28/2025, 8:40:10
2	2	Jane Smith	7/20/1990, 12:00:00	8000	6/28/2025, 7:14:41

### Scenario 2: Audit Log

#### Code:

```
CREATE TABLE AuditLog (
```

```

LogID NUMBER GENERATED ALWAYS AS IDENTITY,
AccountID NUMBER,
TransactionDate DATE,
Amount NUMBER,
TransactionType VARCHAR2(10)
);

CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
    INSERT INTO AuditLog(AccountID, TransactionDate, Amount, TransactionType)
    VALUES (:NEW.AccountID, :NEW.TransactionDate, :NEW.Amount,
:NEW.TransactionType);
END;
/

-- Test

INSERT INTO Transactions VALUES (3, 1, SYSDATE, 100, 'Deposit');

SELECT * FROM AuditLog;

```

### Output:

Trigger LOGTRANSACTION compiled

Elapsed: 00:00:00.021

	LOGID	ACCOUNTID	TRANSACTIONDATE	AMOUNT	TRANSACTIONTYPE
1	1	1	6/28/2025, 8:41:10	100	Deposit

### Scenario 3: Transaction Rules

#### Code:

```

CREATE OR REPLACE TRIGGER CheckTransactionRules

```

```

BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
    bal NUMBER;
BEGIN
    IF :NEW.TransactionType = 'Withdrawal' THEN
        SELECT Balance INTO bal FROM Accounts WHERE AccountID = :NEW.AccountID;
        IF :NEW.Amount > bal 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;
/

-- Test (this will raise error if rules break)
-- INSERT INTO Transactions VALUES (4, 1, SYSDATE, 50000, 'Withdrawal');

```

### Output:

```

Trigger CHECKTRANSACTIONRULES compiled
Elapsed: 00:00:00.016

```

## Exercise 6: Cursors

### Scenario 1: Generate Monthly Statements

#### Code:

```

DECLARE

```

```

CURSOR cur IS

    SELECT AccountID, TransactionDate, Amount, TransactionType
    FROM Transactions

    WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

BEGIN

    FOR rec IN cur LOOP

        DBMS_OUTPUT.PUT_LINE('Account: ' || rec.AccountID ||

            ', Date: ' || rec.TransactionDate ||

            ', Amount: ' || rec.Amount ||

            ', Type: ' || rec.TransactionType);

    END LOOP;

END;

/

```

### Output:

Trigger CHECKTRANSACTIONRULES compiled

Elapsed: 00:00:00.023

```

Account: 1, Date: 28-JUN-25, Amount: 200, Type: Deposit
Account: 2, Date: 28-JUN-25, Amount: 300, Type: Withdrawal
Account: 1, Date: 28-JUN-25, Amount: 100, Type: Deposit

```

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013

### Scenario 2: Apply Annual Fee

#### Code:

```

DECLARE

    CURSOR c IS SELECT AccountID, Balance FROM Accounts FOR UPDATE;

    fee NUMBER := 100;

```



```

BEGIN
    FOR r IN c LOOP
        UPDATE Accounts SET Balance = Balance - fee WHERE AccountID = r.AccountID;
    END LOOP;
END;
/
-- Test
SELECT * FROM Accounts;

```

### Output:

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.009

	ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	1	1	Savings	809	6/28/2025, 7:14:41
2	2	2	Checking	1500	6/28/2025, 7:14:41

### Scenario 3: Update Loan Interest

#### Code:

```

DECLARE
    CURSOR c IS SELECT LoanID, InterestRate FROM Loans FOR UPDATE;
BEGIN
    FOR r IN c LOOP
        UPDATE Loans
            SET InterestRate = r.InterestRate + 0.5
            WHERE LoanID = r.LoanID;
    END LOOP;
END;
/
-- Test

```

```
SELECT * FROM Loans;
```

### Output:

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.013
```

---

	LOANID	CUSTOMERID	LOANAMOUNT	INTERESTRATE	STARTDATE	ENDDATE
1	1	1	5000	3.5	6/28/2025, 7:14:41	6/28/2025, 7:14:41
2	2	2	10000	6.5	6/28/2025, 7:35:42	6/28/2025, 7:35:42

## Exercise 7: Packages

### Scenario 1: CustomerManagement

#### Code:

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
```

```
    PROCEDURE AddCustomer(
```

```
        id NUMBER,
```

```
        name VARCHAR2,
```

```
        dob DATE,
```

```
        balance NUMBER
```

```
    );
```

```
    PROCEDURE UpdateCustomer(
```

```
        id NUMBER,
```

```
        name VARCHAR2
```

```
    );
```

```
    FUNCTION GetBalance(
```

```
        id NUMBER
```

```
    ) RETURN NUMBER;
```

```
END CustomerManagement;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
```

```
    PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance  
NUMBER) IS
```

```
    BEGIN
```

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

```
        VALUES(id, name, dob, balance, SYSDATE);
```

```
    END;
```

```
    PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2) IS
```

```
    BEGIN
```

```
        UPDATE Customers
```

```
        SET Name = name,
```

```
            LastModified = SYSDATE
```

```
        WHERE CustomerID = id;
```

```
    END;
```

```
    FUNCTION GetBalance(id NUMBER) RETURN NUMBER IS
```

```
        bal NUMBER;
```

```
    BEGIN
```

```
        SELECT Balance INTO bal FROM Customers WHERE CustomerID = id;
```

```
        RETURN bal;
```

```
    END;
```

```
END CustomerManagement;
```

```
/
```

```
SET SERVEROUTPUT ON;
```

```
BEGIN
```

```
-- Add a new customer

CustomerManagement.AddCustomer(3, 'Sam Wilson', TO_DATE('1992-12-12', 'YYYY-MM-DD'), 1200);


-- Update customer name

CustomerManagement.UpdateCustomer(3, 'Samuel Wilson');


-- Fetch and display balance

DECLARE

    bal NUMBER;

BEGIN

    bal := CustomerManagement.GetBalance(3);

    DBMS_OUTPUT.PUT_LINE('Balance for Customer 3: ' || bal);

END;

END;

/
```

### Output:

```
Package CUSTOMERMANAGEMENT compiled
```

```
Elapsed: 00:00:00.011
```

```
Package Body CUSTOMERMANAGEMENT compiled
```

```
Elapsed: 00:00:00.017
```

```
Balance for Customer 3: 1200
```

```
PL/SQL procedure successfully completed.
```

## Scenario 2: EmployeeManagement

### Code:

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
```

```
    PROCEDURE HireEmployee(
```

```
        id NUMBER,
```

```
        name VARCHAR2,
```

```
        pos VARCHAR2,
```

```
        sal NUMBER,
```

```
        dept VARCHAR2,
```

```
        hiredate DATE
```

```
    );
```

```
    PROCEDURE UpdateEmployee(
```

```
        id NUMBER,
```

```
        name VARCHAR2
```

```
    );
```

```
    FUNCTION GetAnnualSalary(
```

```
        id NUMBER
```

```
    ) RETURN NUMBER;
```

```
END EmployeeManagement;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
```

```
    PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal  
NUMBER, dept VARCHAR2, hiredate DATE) IS
```

```
    BEGIN
```

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

```

VALUES(id, name, pos, sal, dept, hiredate);

END;

PROCEDURE UpdateEmployee(id NUMBER, name VARCHAR2) IS
BEGIN
    UPDATE Employees
    SET Name = name
    WHERE EmployeeID = id;
END;

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER IS
    sal NUMBER;
BEGIN
    SELECT Salary INTO sal FROM Employees WHERE EmployeeID = id;
    RETURN sal * 12;
END;

END EmployeeManagement;

/

SET SERVEROUTPUT ON;

BEGIN
    -- Step 1: Add a new employee
    EmployeeManagement.HireEmployee(
        3,
        'Charlie Stark',
        'Analyst',
        50000,
        'Finance',

```

```

        TO_DATE('2020-01-10', 'YYYY-MM-DD')
    );

-- Step 2: Update employee name
EmployeeManagement.UpdateEmployee(3, 'Charles Stark');

-- Step 3: Fetch and print annual salary
DECLARE
    annual_salary NUMBER;
BEGIN
    annual_salary := EmployeeManagement.GetAnnualSalary(3);
    DBMS_OUTPUT.PUT_LINE('Annual Salary for Employee 3: ' || annual_salary);
END;
END;
/

```

### Output:

```
Package EMPLOYEEMANAGEMENT compiled
```

```
Elapsed: 00:00:00.010
```

```
Package Body EMPLOYEEMANAGEMENT compiled
```

```
Elapsed: 00:00:00.014
```

```
Annual Salary for Employee 3: 600000
```

```
PL/SQL procedure successfully completed.
```

```
Elapsed: 00:00:00.009
```

### Scenario 3: AccountOperations

#### Code:

```
CREATE OR REPLACE PACKAGE AccountOperations AS
```

```
    PROCEDURE OpenAccount(
```

```
        accid NUMBER,
```

```
        cid NUMBER,
```

```
        type VARCHAR2,
```

```
        balance NUMBER
```

```
    );
```

```
    PROCEDURE CloseAccount(
```

```
        accid NUMBER
```

```
    );
```

```
    FUNCTION GetTotalBalance(
```

```
        cid NUMBER
```

```
    ) RETURN NUMBER;
```

```
END AccountOperations;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
```

```
    PROCEDURE OpenAccount(accid NUMBER, cid NUMBER, type VARCHAR2, balance  
NUMBER) IS
```

```
    BEGIN
```

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

```
        VALUES(accid, cid, type, balance, SYSDATE);
```

```
    END;
```

```
    PROCEDURE CloseAccount(accid NUMBER) IS
```

```
    BEGIN
```

```
        DELETE FROM Accounts WHERE AccountID = accid;
```

```
    END;
```



```

FUNCTION GetTotalBalance(cid NUMBER) RETURN NUMBER IS
    total NUMBER;
BEGIN
    SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = cid;
    RETURN total;
END;
END AccountOperations;
/

SET SERVEROUTPUT ON;

BEGIN
    -- Open a new account
    AccountOperations.OpenAccount(3, 1, 'Savings', 2000);

    -- Get and display total balance
    DECLARE
        total_bal NUMBER;
    BEGIN
        total_bal := AccountOperations.GetTotalBalance(1);
        DBMS_OUTPUT.PUT_LINE('Total Balance for Customer 1: ' || total_bal);
    END;
END;
/

```

**Output:**

Package ACCOUNTOPERATIONS compiled

Elapsed: 00:00:00.013

Package Body ACCOUNTOPERATIONS compiled

Elapsed: 00:00:00.013

Total Balance for Customer 1: 2809

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014