PL/SQL

EXERCISE 7

TABLE CREATION AND DATA INSERTION:

```
-- Create the Customers table
CREATE TABLE Customers (
   CustomerID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   DOB DATE,
   Balance NUMBER,
   LastModified DATE
);

-- Create the Accounts table
CREATE TABLE Accounts (
   AccountID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   AccountType VARCHAR2(20),
   Balance NUMBER,
   LastModified DATE,
   FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID)
);
```

```
-- Create the Transactions table
CREATE TABLE Transactions (
    TransactionID NUMBER PRIMARY KEY,
    AccountID NUMBER,
    TransactionDate DATE,
    Amount NUMBER,
    TransactionType VARCHAR2(10),
    FOREIGN KEY (AccountID) REFERENCES
Accounts(AccountID)
);

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

-- Create the Employees table
```

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

-- Create the AuditLog table

```
CREATE TABLE AuditLog (
    LogID NUMBER PRIMARY KEY,
    TransactionID NUMBER,
    LogDate DATE,
    Message VARCHAR2(255),
    FOREIGN KEY (TransactionID) REFERENCES
Transactions(TransactionID)
);
```

-- Insert sample data into the Customers table

```
INSERT INTO Customers (CustomerID, Name, DOB, Balance,
LastModified)
VALUES (1, 'John Doe', TO_DATE('1985-05-15', 'YYYY-MM-
DD'), 1000, SYSDATE);
```

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

VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

-- Insert sample data into the Accounts table

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

VALUES (1, 1, 'Savings', 1000, SYSDATE);

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

VALUES (2, 2, 'Checking', 1500, SYSDATE);

-- Insert sample data into the Transactions table

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

-- Insert sample data into the Loans table

```
INSERT INTO Loans (LoanID, CustomerID, LoanAmount,
InterestRate, StartDate, EndDate)

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

-- Insert sample data into the Employees table

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

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR',
TO_DATE('2015-06-15', 'YYYY-MM-DD'));


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

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO_DATE('2017-
03-20', 'YYYY-MM-DD'));
```

Exercise 7: Packages

Scenario 1: Group all customer-related procedures and functions into a package.

Question: Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

```
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
   ) AS
   BEGIN
       INSERT INTO Customers (CustomerID, Name, DOB, Balance,
LastModified)
       VALUES (p_customer_id, p_name, p_dob, p_balance,
SYSDATE);

       COMMIT;
       DBMS_OUTPUT.PUT_LINE('New customer added
successfully.');
   EXCEPTION
       WHEN DUP_VAL_ON_INDEX THEN
           DBMS_OUTPUT.PUT_LINE('Error: Customer with the same
ID already exists.');
       WHEN OTHERS THEN
           DBMS_OUTPUT.PUT_LINE('Error adding new customer: ' ||
SQLERRM);
   END;

   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
   ) AS
   BEGIN
```

```
        UPDATE Customers
        SET Name = p_name, DOB = p_dob, Balance = p_balance,
LastModified = SYSDATE
        WHERE CustomerID = p_customer_id;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Customer details updated
successfully.');
    END;

    FUNCTION GetCustomerBalance (
        p_customer_id IN Customers.CustomerID%TYPE
    ) RETURN NUMBER AS
        v_balance Customers.Balance%TYPE;
    BEGIN
        SELECT Balance INTO v_balance FROM Customers WHERE
CustomerID = p_customer_id;

        RETURN v_balance;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN 0;
    END;
END CustomerManagement;
/
```

<u>Scenario 2:</u> <u>Create a package to manage employee data.</u>

Question: Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

```
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_hire_date 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
    );

    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_hire_date IN Employees.HireDate%TYPE
    ) AS
    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;
        DBMS_OUTPUT.PUT_LINE('New employee hired
successfully.');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
```

```
            DBMS_OUTPUT.PUT_LINE('Error: Employee with the same
ID already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error hiring new employee: ' ||
SQLERRM);
    END;

    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
    ) AS
    BEGIN
        UPDATE Employees
        SET Name = p_name, Position = p_position, Salary = p_salary,
Department = p_department
        WHERE EmployeeID = p_employee_id;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Employee details updated
successfully.');
    END;

    FUNCTION CalculateAnnualSalary (
        p_employee_id IN Employees.EmployeeID%TYPE
    ) RETURN NUMBER AS
        v_salary Employees.Salary%TYPE;
    BEGIN
        SELECT Salary INTO v_salary FROM Employees WHERE
EmployeeID = p_employee_id;

        RETURN v_salary * 12;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN 0;
    END;
END EmployeeManagement;
/
```

Scenario 3: Group all account-related operations into a package.

Question: Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

```
CREATE OR REPLACE PACKAGE AccountOperations AS
   PROCEDURE OpenNewAccount (
      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 GetTotalCustomerBalance (
      p_customer_id IN Accounts.CustomerID%TYPE
   ) RETURN NUMBER;
END AccountOperations;
/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS
   PROCEDURE OpenNewAccount (
        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
    ) AS
    BEGIN
        INSERT INTO Accounts (AccountID, CustomerID,
AccountType, Balance, LastModified)
        VALUES (p_account_id, p_customer_id, p_account_type,
p_balance, SYSDATE);

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('New account opened
successfully.');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Account with the same
ID already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error opening new account: ' ||
SQLERRM);
    END;

    PROCEDURE CloseAccount (
        p_account_id IN Accounts.AccountID%TYPE
    ) AS
```

```
  BEGIN
    DELETE FROM Accounts WHERE AccountID = p_account_id;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Account closed successfully.');
  END;

  FUNCTION GetTotalCustomerBalance (
    p_customer_id IN Accounts.CustomerID%TYPE
  ) RETURN NUMBER AS
    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 0;
  END;
END AccountOperations;
/
```