PL/SQL

EXERCISE 3

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 3: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

Question: Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

    v_account_id Accounts.AccountID%TYPE;

    v_balance Accounts.Balance%TYPE;

BEGIN
```

```
   FOR account_record IN (SELECT AccountID, Balance FROM
Accounts WHERE AccountType = 'Savings') LOOP
      v_account_id := account_record.AccountID;
      v_balance := account_record.Balance;

      -- Calculate and update interest
      UPDATE Accounts
      SET Balance = v_balance + (v_balance * 0.01)
      WHERE AccountID = v_account_id;
   END LOOP;

   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Monthly interest processed for all
savings accounts.');
END;
/
```

Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

Question: Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
   p_department IN Employees.Department%TYPE,
   p_bonus_percentage IN NUMBER
) AS
```

```
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * p_bonus_percentage / 100)
    WHERE Department = p_department;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Employee bonuses updated for department: ' || p_department);
END;
/
```

Scenario 3: Customers should be able to transfer funds between their accounts.

Question: Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

```
CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account_id IN Accounts.AccountID%TYPE,
    p_to_account_id IN Accounts.AccountID%TYPE,
    p_amount IN NUMBER
) AS
    v_from_balance Accounts.Balance%TYPE;
BEGIN
    -- Check balance of the source account
    SELECT Balance INTO v_from_balance FROM Accounts WHERE AccountID = p_from_account_id FOR UPDATE;
    IF v_from_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in the source account.');
    END IF;

    -- Perform the transfer
    UPDATE Accounts SET Balance = Balance - p_amount WHERE AccountID = p_from_account_id;
    UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to_account_id;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Funds transferred successfully.');
END;
/
```