PL/SQL

EXERCISE 4

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 4: Functions

Scenario 1: Calculate the age of customers for eligibility checks.

Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

```
CREATE OR REPLACE FUNCTION CalculateAge (
    p_dob IN DATE
) RETURN NUMBER AS
    v_age NUMBER;
BEGIN
```

```
    v_age := EXTRACT(YEAR FROM SYSDATE) -
EXTRACT(YEAR FROM p_dob);


    RETURN v_age;
END;
/
```

Scenario 2: The bank needs to compute the monthly installment for a loan.

Question: Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (
    p_loan_amount IN NUMBER,
    p_interest_rate IN NUMBER,
    p_duration_years IN NUMBER
) RETURN NUMBER AS
    v_monthly_installment NUMBER;
    v_rate_per_month NUMBER;
    v_months NUMBER;
BEGIN
    v_rate_per_month := p_interest_rate / 12 / 100;
    v_months := p_duration_years * 12;


    v_monthly_installment := (p_loan_amount * v_rate_per_month) /
```

```
                    (1 - POWER(1 + v_rate_per_month, -v_months));

    RETURN v_monthly_installment;
END;
/
```

Scenario 3: Check if a customer has sufficient balance before making a transaction.

Question: Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

```
CREATE OR REPLACE FUNCTION HasSufficientBalance (
    p_account_id IN Accounts.AccountID%TYPE,
    p_amount IN NUMBER
) RETURN BOOLEAN AS
    v_balance Accounts.Balance%TYPE;
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;
/
```