**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 TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

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

VALUES (1, 'Ankit Sharma', TO\_DATE('1990-05-20', 'YYYY-MM-DD'), 5000, SYSDATE);

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

VALUES (2, 'Meera Nair', TO\_DATE('1985-12-10', 'YYYY-MM-DD'), 10000, SYSDATE);

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

VALUES (3, 'Ravi Verma', TO\_DATE('1978-09-15', 'YYYY-MM-DD'), 15000, SYSDATE);

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

VALUES (4, 'Neha Patil', TO\_DATE('1995-03-25', 'YYYY-MM-DD'), 8000, SYSDATE);

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

VALUES (5, 'Suresh Iyer', TO\_DATE('1982-07-30', 'YYYY-MM-DD'), 12000, SYSDATE);

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

VALUES (6, 'Pooja Reddy', TO\_DATE('2000-11-10', 'YYYY-MM-DD'), 6000, SYSDATE);

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

VALUES (7, 'Anil Kumar', TO\_DATE('1965-04-05', 'YYYY-MM-DD'), 25000, SYSDATE);

COMMIT;

COMMIT;

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE)

RETURN NUMBER

IS

v\_Age NUMBER;

BEGIN

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/

SELECT CustomerID, CalculateAge(DOB) AS Age

FROM Customers;

**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 TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

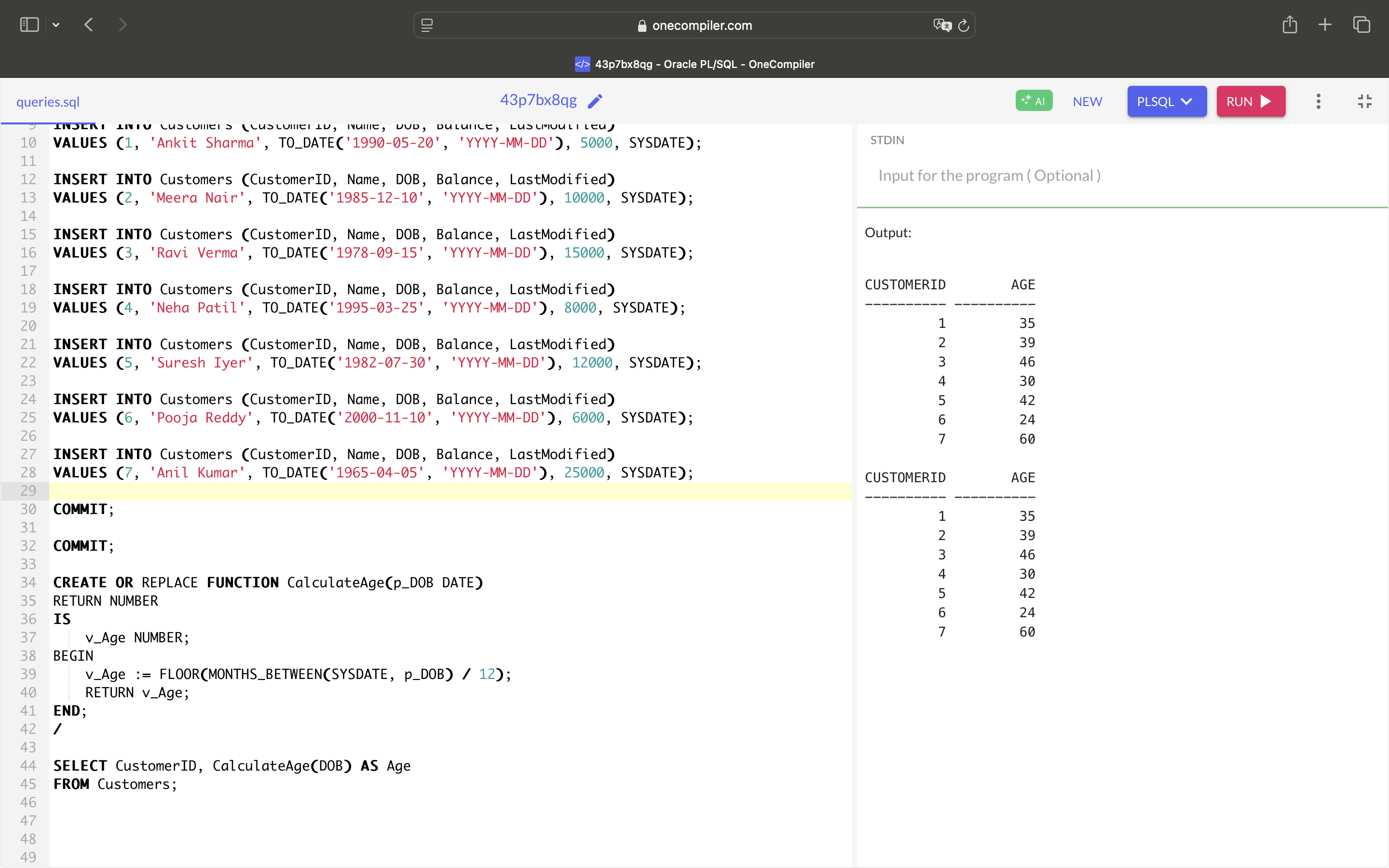
StartDate DATE,

EndDate DATE

);

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

VALUES (1, 1, 500000, 8, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

VALUES (2, 2, 300000, 7.5, SYSDATE, ADD\_MONTHS(SYSDATE, 48));

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

VALUES (3, 3, 750000, 9, SYSDATE, ADD\_MONTHS(SYSDATE, 72));

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

VALUES (4, 4, 250000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

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

VALUES (5, 5, 1000000, 8.5, SYSDATE, ADD\_MONTHS(SYSDATE, 120));

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

VALUES (6, 6, 400000, 7.2, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

VALUES (7, 7, 300000, 9.5, SYSDATE, ADD\_MONTHS(SYSDATE, 48));

COMMIT;

COMMIT;

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount NUMBER,

p\_InterestRate NUMBER,

p\_LoanDurationYears NUMBER

)

RETURN NUMBER

IS

v\_MonthlyRate NUMBER;

v\_NumPayments NUMBER;

v\_Installment NUMBER;

BEGIN

v\_MonthlyRate := p\_InterestRate / 1200;

v\_NumPayments := p\_LoanDurationYears \* 12;

v\_Installment := (p\_LoanAmount \* v\_MonthlyRate) /

(1 - POWER(1 + v\_MonthlyRate, -v\_NumPayments));

RETURN v\_Installment;

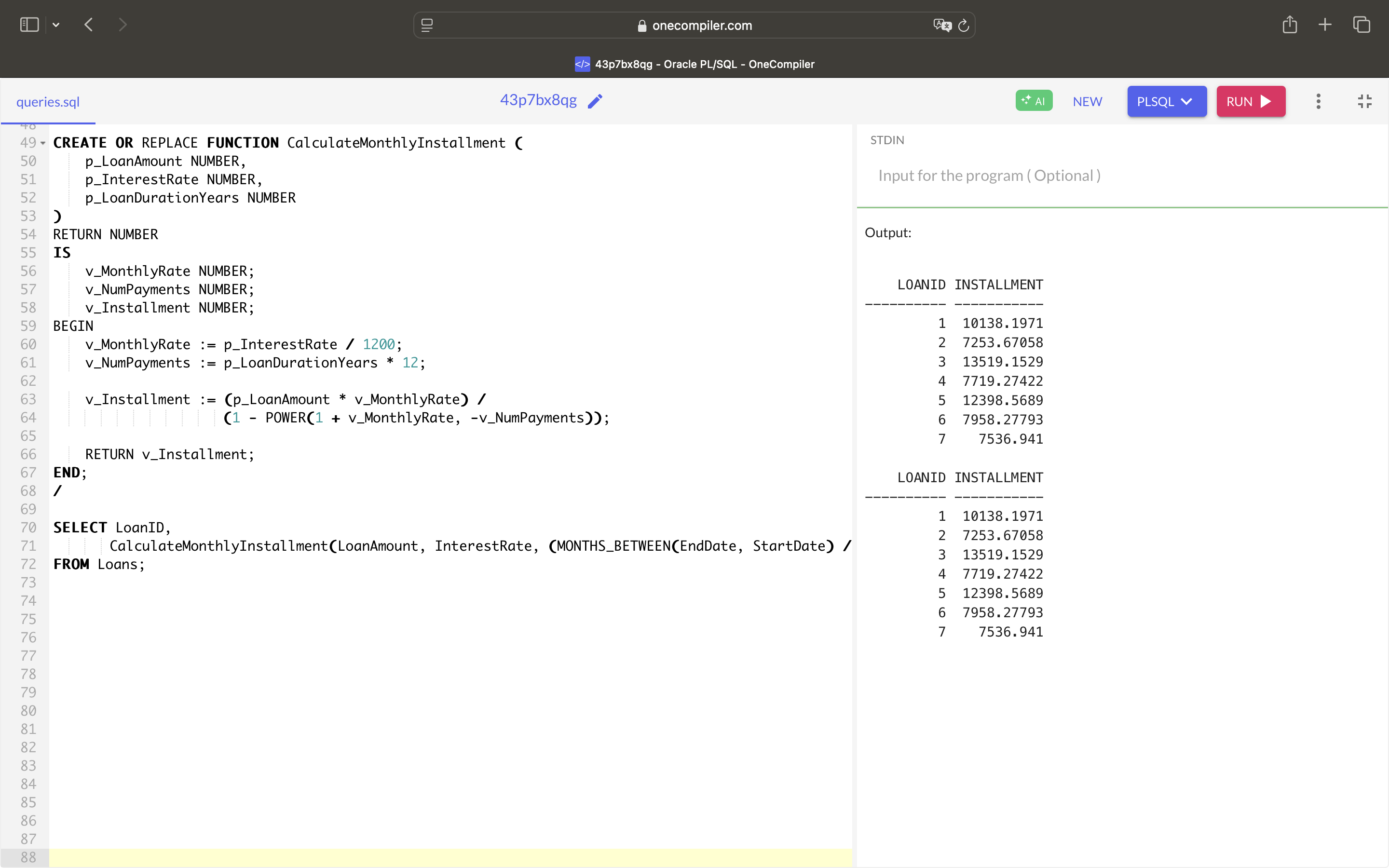
END;

/

SELECT LoanID,

CalculateMonthlyInstallment(LoanAmount, InterestRate, (MONTHS\_BETWEEN(EndDate, StartDate) / 12)) AS Installment

FROM Loans;



**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 TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

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

VALUES (101, 1, 'Savings', 10000, SYSDATE);

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

VALUES (102, 1, 'Checking', 5000, SYSDATE);

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

VALUES (103, 2, 'Savings', 15000, SYSDATE);

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

VALUES (104, 3, 'Savings', 20000, SYSDATE);

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

VALUES (105, 4, 'Checking', 3000, SYSDATE);

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

VALUES (106, 5, 'Savings', 1500, SYSDATE);

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

VALUES (107, 6, 'Checking', 9000, SYSDATE);

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

VALUES (108, 7, 'Savings', 50000, SYSDATE);

COMMIT;

COMMIT;

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID NUMBER,

p\_Amount NUMBER

)

RETURN BOOLEAN

IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

IF v\_Balance >= p\_Amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

SET SERVEROUTPUT ON;

DECLARE

v\_Result BOOLEAN;

BEGIN

FOR rec IN (SELECT AccountID FROM Accounts) LOOP

v\_Result := HasSufficientBalance(rec.AccountID, 7000);

IF v\_Result THEN

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || rec.AccountID || ' : Sufficient Balance');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || rec.AccountID || ' : Insufficient Balance');

END IF;

END LOOP;

END;

/

