**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.

**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.

**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.

**Scenario 1: CalculateAge — Compute Customer Age**

CODE:-

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB IN DATE)

RETURN NUMBER

IS

  v\_Age NUMBER;

BEGIN

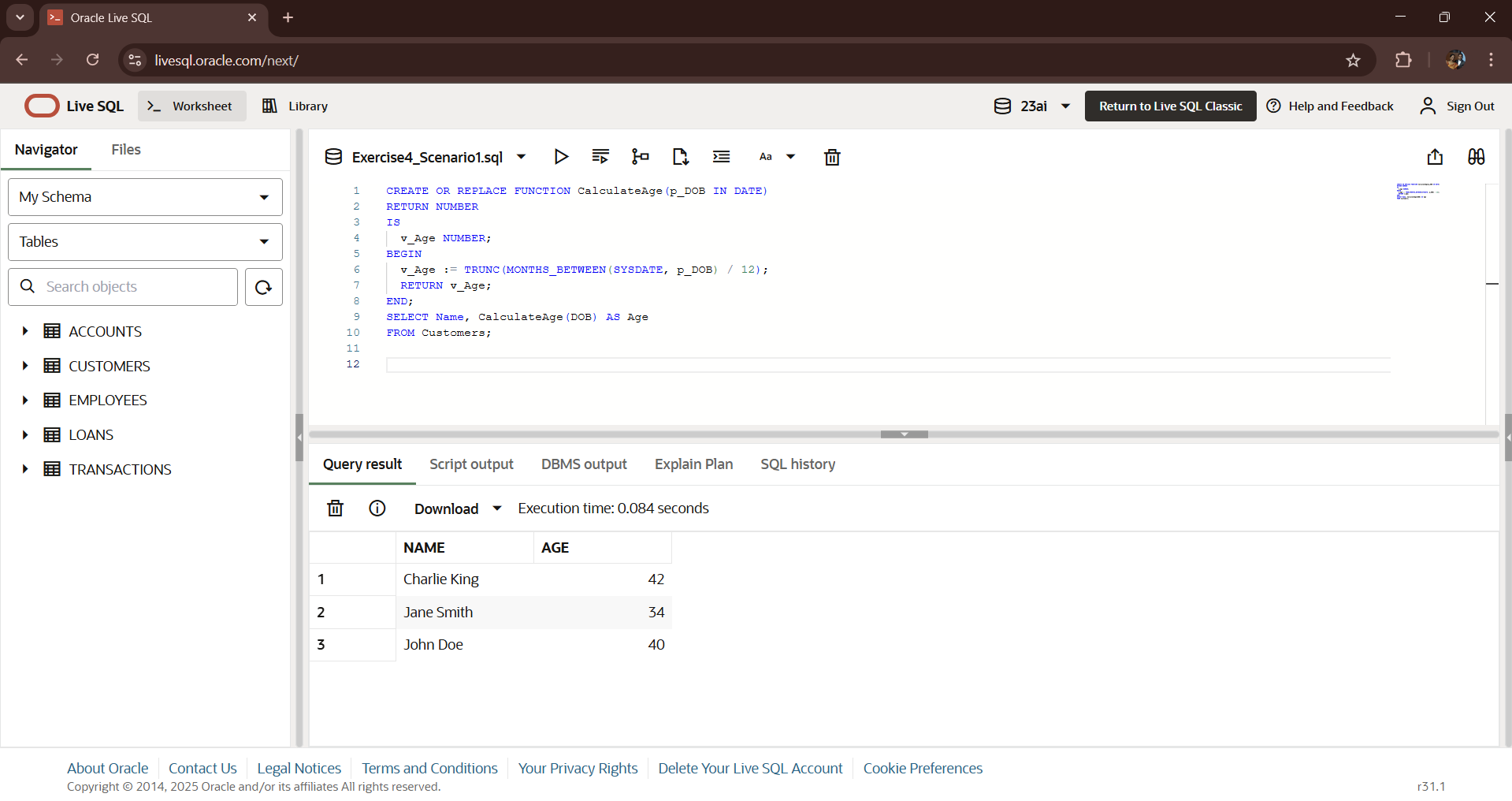
  v\_Age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

  RETURN v\_Age;

END;

SELECT Name, CalculateAge(DOB) AS Age

FROM Customers;

**OUTPUT:-**

**Scenario 2: CalculateMonthlyInstallment — Compute EMI**

**Formula used:**

****

Where:

* **P** = Loan amount
* **R** = Monthly interest rate (annual rate / 12 / 100)
* **N** = Total months (loan duration in years × 12)

**CODE:-**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

  p\_LoanAmount IN NUMBER,

  p\_AnnualRate IN NUMBER,

  p\_Years IN NUMBER

)

RETURN NUMBER

IS

  v\_MonthlyRate NUMBER;

  v\_Months NUMBER;

  v\_EMI NUMBER;

BEGIN

  v\_MonthlyRate := p\_AnnualRate / 12 / 100;

  v\_Months := p\_Years \* 12;

  v\_EMI := (p\_LoanAmount \* v\_MonthlyRate \* POWER(1 + v\_MonthlyRate, v\_Months)) /

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

  RETURN ROUND(v\_EMI, 2);

EXCEPTION

  WHEN ZERO\_DIVIDE THEN

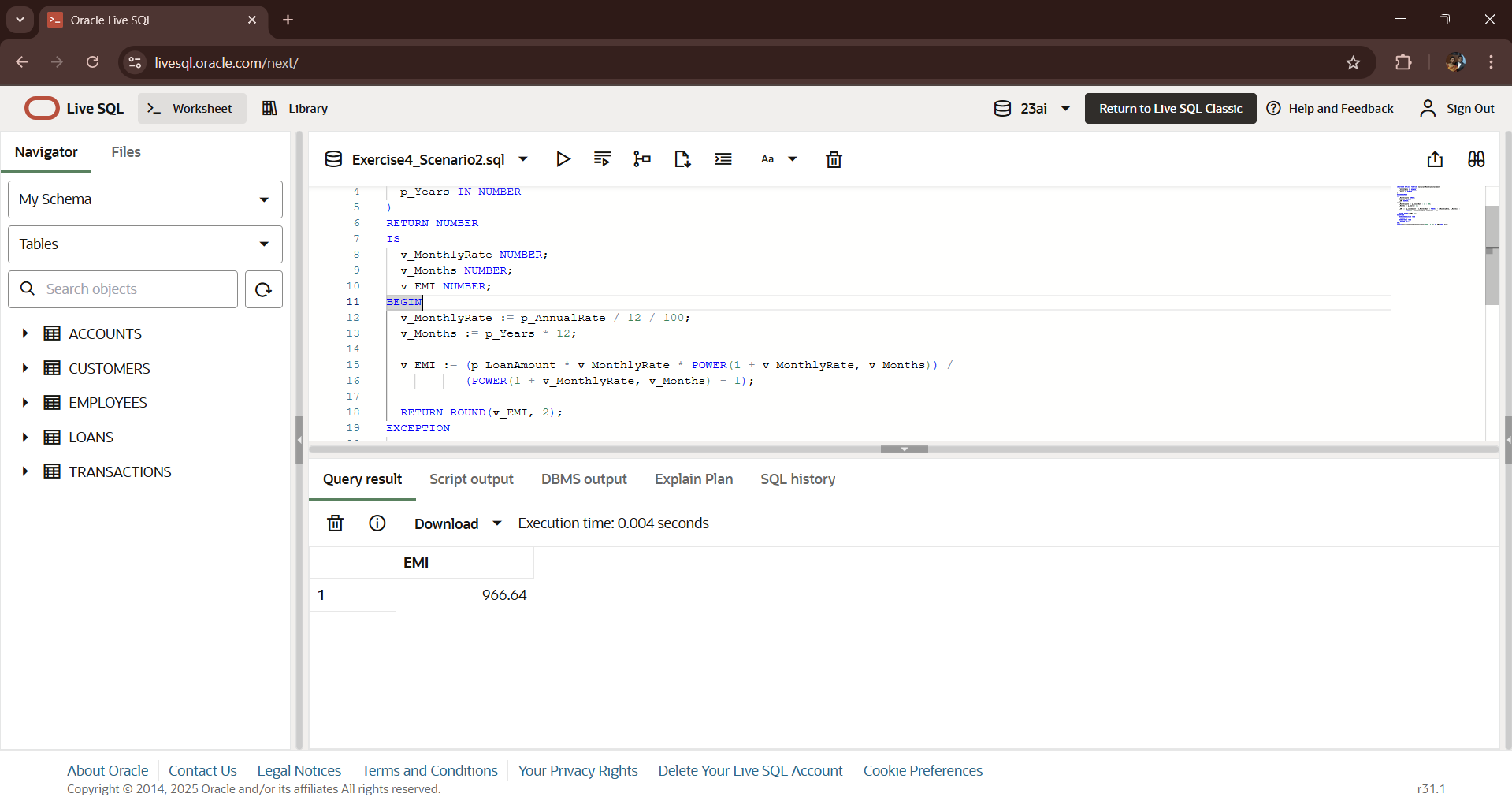
    RETURN 0;

  WHEN OTHERS THEN

    RETURN NULL;

END;

SELECT CalculateMonthlyInstallment(50000, 6, 5) AS EMI FROM dual;

**OUTPUT:-**

**Scenario 3: HasSufficientBalance — Validate Account Balance**

**CODE:-**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

  p\_AccountID IN NUMBER,

  p\_Amount IN NUMBER

)

RETURN BOOLEAN

IS

  v\_Balance NUMBER;

BEGIN

  SELECT Balance INTO v\_Balance

  FROM Accounts

  WHERE AccountID = p\_AccountID;

  RETURN v\_Balance >= p\_Amount;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    RETURN FALSE;

  WHEN OTHERS THEN

    RETURN FALSE;

END;

DECLARE

  result BOOLEAN;

BEGIN

  result := HasSufficientBalance(1, 1000);

  IF result THEN

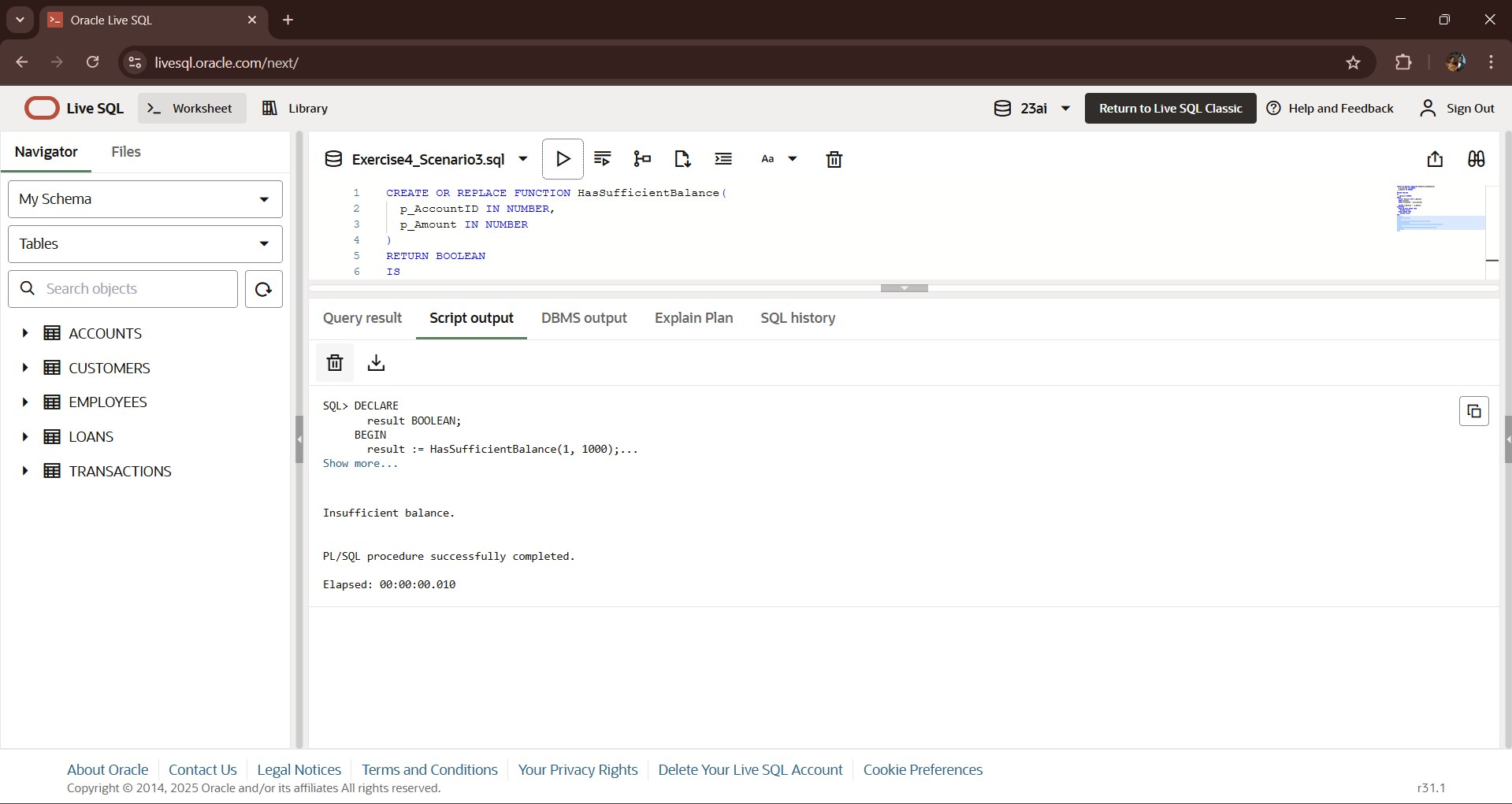
    DBMS\_OUTPUT.PUT\_LINE('Sufficient balance available.');

  ELSE

    DBMS\_OUTPUT.PUT\_LINE('Insufficient balance.');

  END IF;

END;

**OUTPUT:-**