**Cognizant Digital Nurture 4.0-Week(2)**

**Name: Gandu Lasya Sri**

**Email: gandulasyasri@gmail.com**

**Superset ID:6428164**

**Mandatory Hands-On Exercises**

**PL/SQL Programming**

**Exercise 1: Control Structures**

In this task, I worked on 3 different scenarios using PL/SQL control structures. I used Oracle Live SQL to write and run the queries. I first created the required tables, inserted the data, and then wrote the logic using loops and if conditions. Below is everything I did step-by-step.

**Step 1: Creating the Tables**

I created two tables:

* Customers table stores customer details like ID, name, date of birth, balance, and a flag to mark them as VIP.
* Loans table keeps track of loans taken by customers along with interest rate and end date.

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

After running this, Oracle Live SQL confirmed both tables were created successfully.

**Step 2: Inserting Sample Data**

I added 6 customers and 6 loans. Each loan is linked to a customer using CustomerID. I only gave 4 values while inserting into the Customers table because IsVIP will take its default value ('FALSE').

-- Customers

INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (1, 'John Doe', TO\_DATE('1950-05-01', 'YYYY-MM-DD'), 12000);

INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (2, 'Jane Smith', TO\_DATE('1985-06-15', 'YYYY-MM-DD'), 9000);

INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (3, 'Alice Green', TO\_DATE('1948-03-22', 'YYYY-MM-DD'), 11000);

INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (4, 'Bob White', TO\_DATE('2000-08-10', 'YYYY-MM-DD'), 5000);

INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (5, 'Carol King', TO\_DATE('1960-02-28', 'YYYY-MM-DD'), 13000);

INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (6, 'David Lee', TO\_DATE('1972-12-11', 'YYYY-MM-DD'), 3000);

-- Loans

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000, 7, SYSDATE, SYSDATE + 20);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (2, 2, 4000, 8, SYSDATE, SYSDATE + 40);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (3, 3, 6000, 6.5, SYSDATE, SYSDATE + 25);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (4, 4, 3000, 7.5, SYSDATE, SYSDATE + 15);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (5, 5, 5500, 8.2, SYSDATE, SYSDATE + 10);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (6, 6, 3500, 7.8, SYSDATE, SYSDATE + 45);

After running all the insert statements together, it showed 6 rows inserted in both tables.

**Scenario 1: Interest Rate Discount for Senior Citizens**

This code checks each customer's age. If they are older than 60, it reduces their loan interest rate by 1%. It also prints a confirmation message using DBMS\_OUTPUT.PUT\_LINE.

BEGIN

FOR rec IN (

SELECT c.CustomerID, c.Name, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN

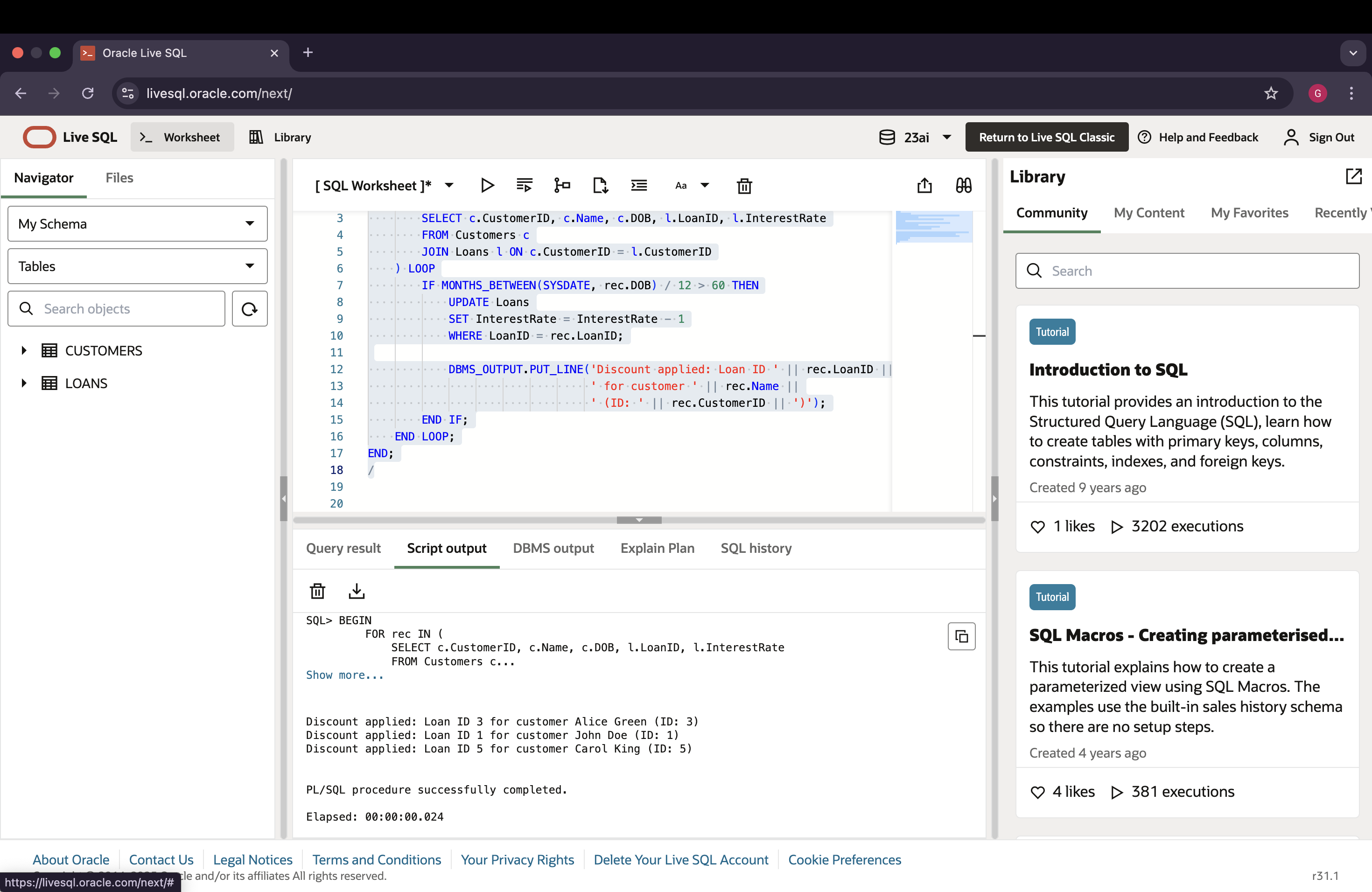
UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied: Loan ID ' || rec.LoanID || ' for customer ' || rec.Name || ' (ID: ' || rec.CustomerID || ')');

END IF;

END LOOP;

END;

**Output:**

**Scenario 2: Promote VIPs Based on Balance**

In this one, I checked each customer’s balance. If it was more than 10,000, I updated their IsVIP to 'TRUE'. Again, I printed a confirmation message.

BEGIN

FOR rec IN (SELECT CustomerID, Name, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

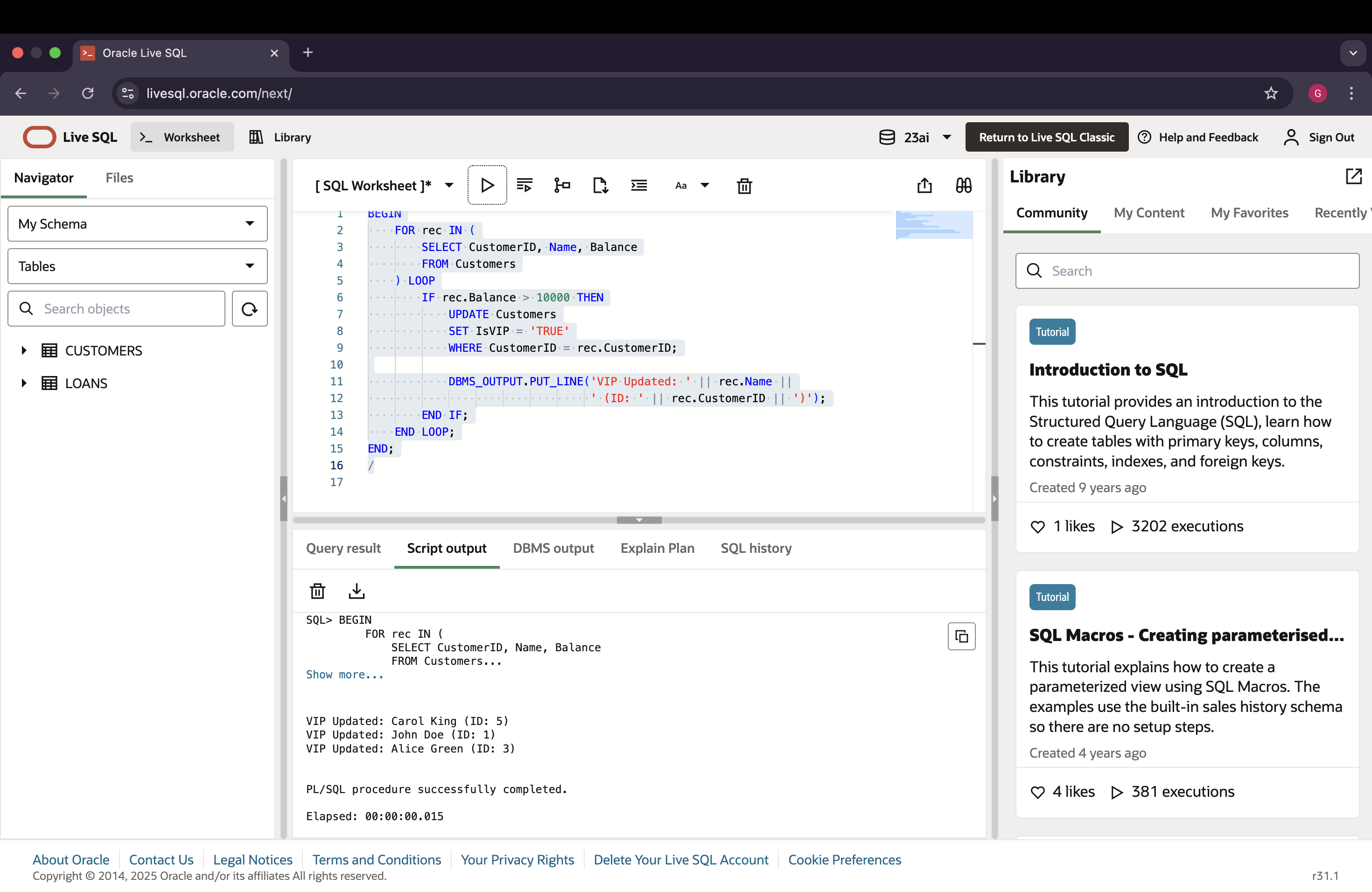
UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('VIP Updated: ' || rec.Name || ' (ID: ' || rec.CustomerID || ')');

END IF;

END LOOP;

END;

**Output:**

**Scenario 3: Reminders for Upcoming Loan Due Dates**

Here, I checked loans that are ending in the next 30 days. For each one, I printed a reminder message with the due date.

BEGIN

FOR rec IN (

SELECT l.LoanID, l.CustomerID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

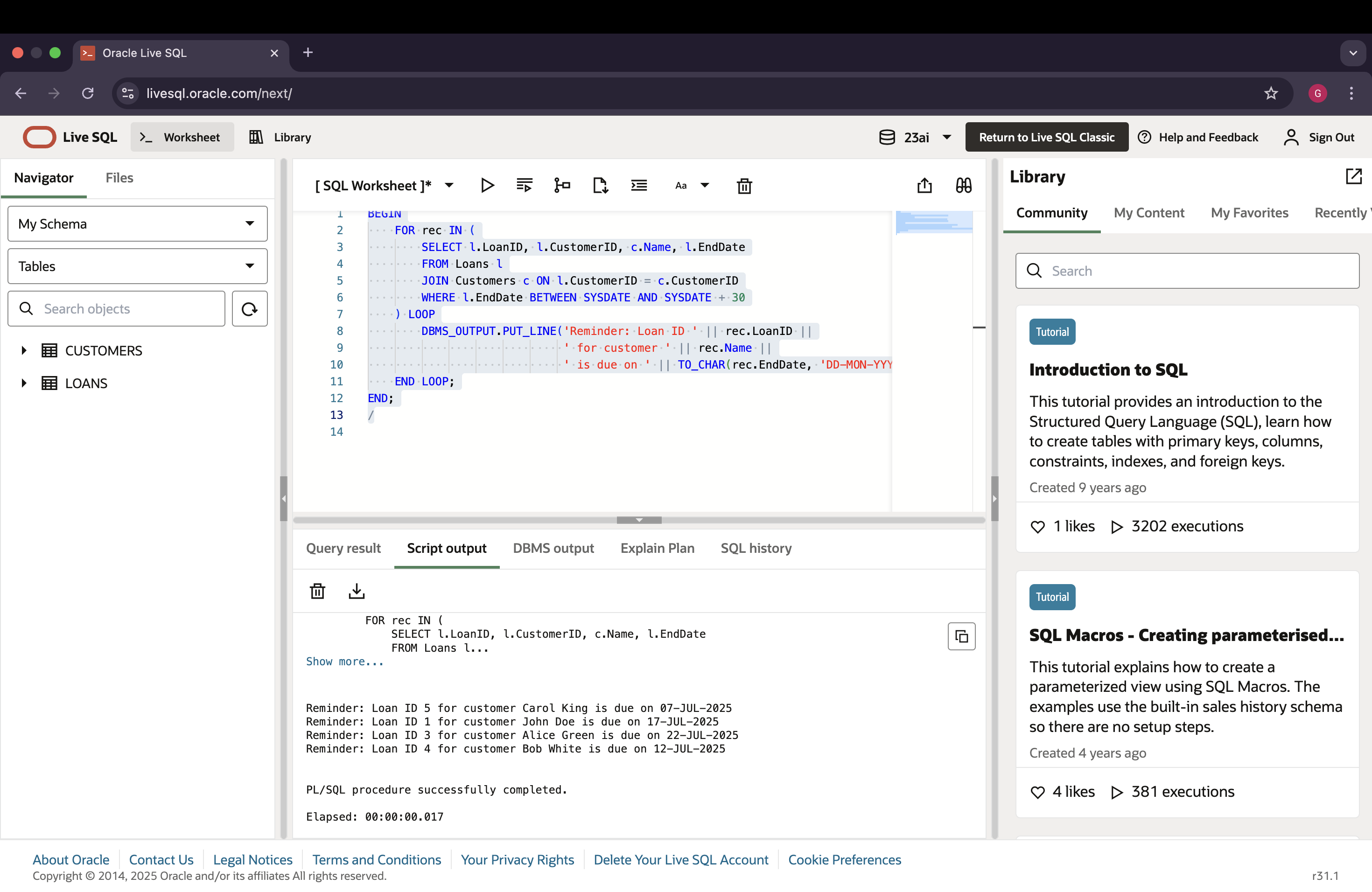
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for customer ' || rec.Name ||

' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**Output:**

Exercise 3: Stored Procedures

**Exercise 3: Stored Procedures – Step-by-Step Execution in Oracle Live SQL (Explained as a Student)**

In this part of the hands-on, I worked on creating and executing stored procedures for 3 real-world banking scenarios. I used Oracle Live SQL, and here is how I did everything from scratch.

**Scenario 1: Apply Monthly Interest to Savings Accounts**

**Step 1: Create the Accounts Table**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

**Step 2: Insert 6 Rows into Accounts**

INSERT INTO Accounts VALUES (1, 1, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 3000, SYSDATE);

INSERT INTO Accounts VALUES (3, 3, 'Savings', 4000, SYSDATE);

INSERT INTO Accounts VALUES (4, 4, 'Checking', 6000, SYSDATE);

INSERT INTO Accounts VALUES (5, 5, 'Savings', 8000, SYSDATE);

INSERT INTO Accounts VALUES (6, 6, 'Savings', 2000, SYSDATE);

**Step 3: Create the Procedure**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (

SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added for Account ID: ' || rec.AccountID);

END LOOP;

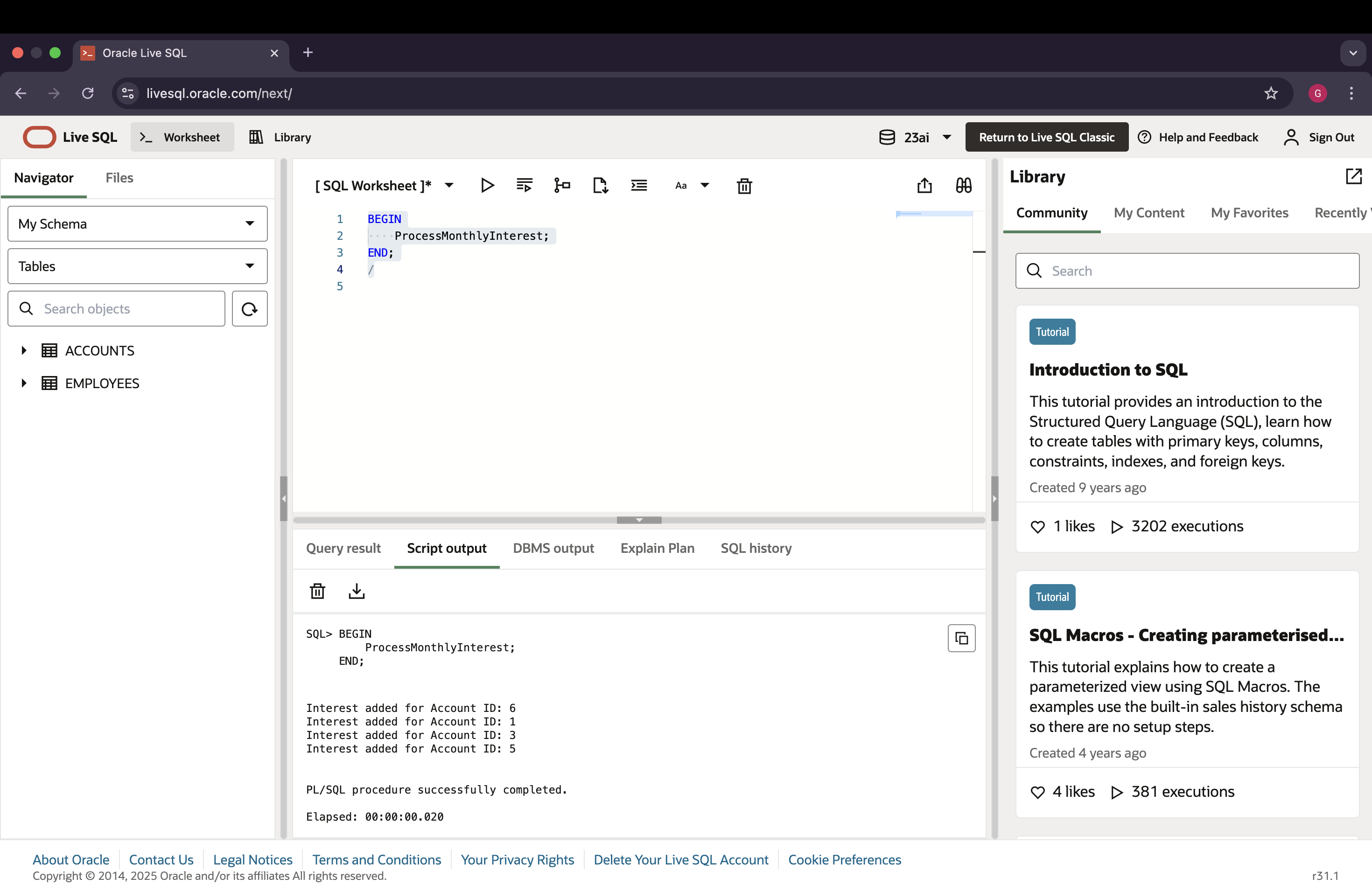
END;

**Step 4: Execute the Procedure**

BEGIN

ProcessMonthlyInterest;

END;

Output:

**Scenario 2: Apply Bonus to Employees in a Department**

**Step 1: Create the Employees Table**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

**Step 2: Insert Rows into Employees**

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (3, 'Cathy Ray', 'Tester', 50000, 'QA', TO\_DATE('2019-01-10', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (4, 'David Smith', 'Analyst', 65000, 'Finance', TO\_DATE('2016-11-12', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (5, 'Eva Green', 'HR Executive', 48000, 'HR', TO\_DATE('2018-05-05', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (6, 'Frank Roy', 'Developer', 62000, 'IT', TO\_DATE('2020-08-18', 'YYYY-MM-DD'));

**Step 3: Create the Procedure**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

deptName IN VARCHAR2,

bonusPercent IN NUMBER

) IS

BEGIN

FOR rec IN (

SELECT EmployeeID FROM Employees WHERE Department = deptName

) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE EmployeeID = rec.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || rec.EmployeeID);

END LOOP;

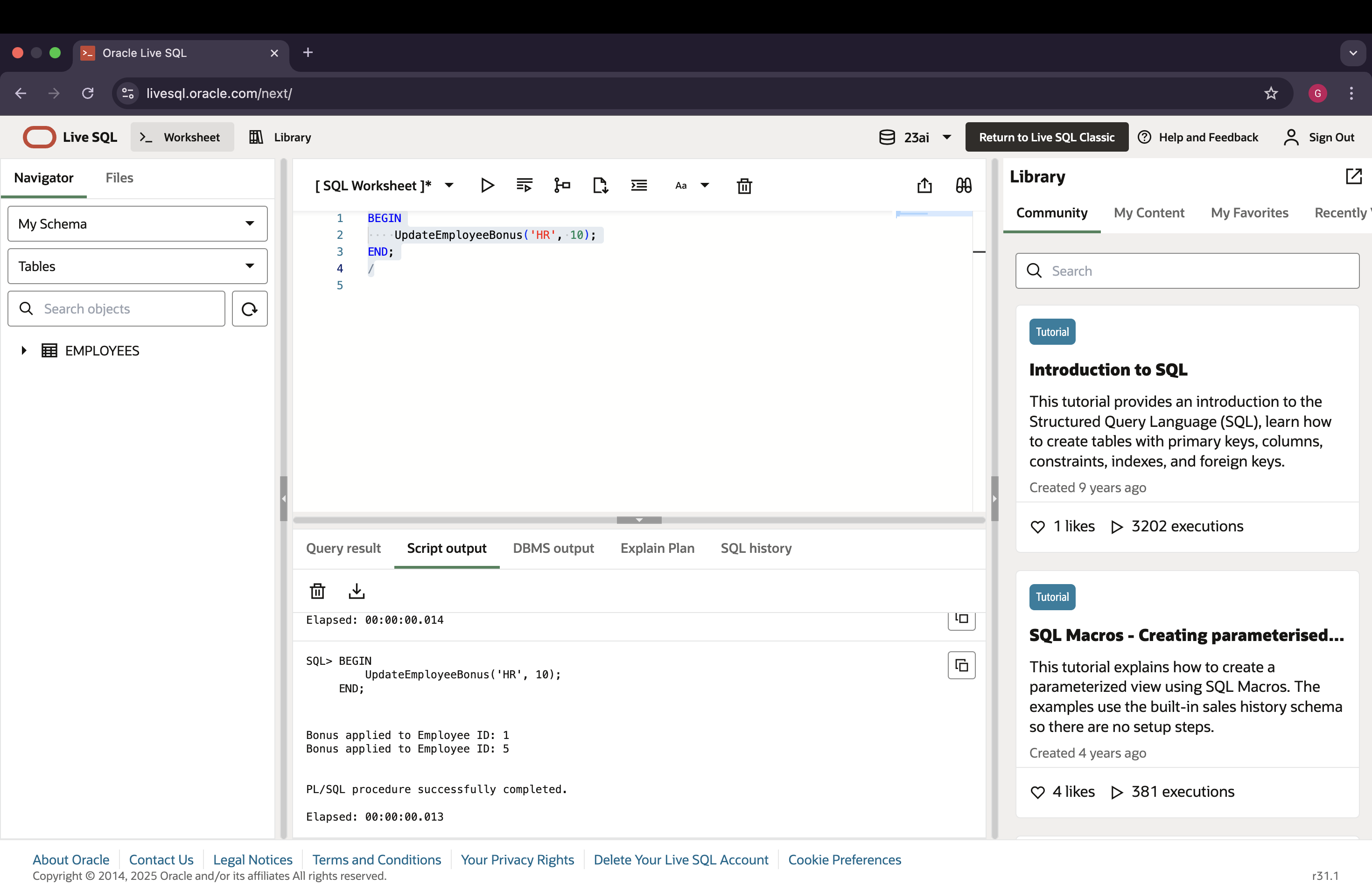
END;

**Step 4: Run the Procedure for Department "HR"**

BEGIN

UpdateEmployeeBonus('HR', 10);

END;

Output:

**Scenario 3: Transfer Funds Between Accounts**

**Step 1: Create the TransferFunds Procedure**

CREATE OR REPLACE PROCEDURE TransferFunds(

fromAcc IN NUMBER,

toAcc IN NUMBER,

amt IN NUMBER

) IS

fromBal NUMBER;

BEGIN

SELECT Balance INTO fromBal FROM Accounts WHERE AccountID = fromAcc;

IF fromBal >= amt THEN

UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = fromAcc;

UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = toAcc;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amt || ' from Account ' || fromAcc || ' to Account ' || toAcc);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ' || fromAcc);

END IF;

END;

**Step 2: Run the Procedure (Transfer 1000 from Acc 1 to Acc 2)**

BEGIN

TransferFunds(1, 2, 1000);

END;

Output: