**WEEK 2 -PL/SQL PROGRAMMING**

**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.

SOLUTION:

**Banking System – Stored Procedure for Monthly Interest Application**

**Objective**

Create a PL/SQL Stored Procedure to apply a 1% monthly interest rate to all savings accounts in the Accounts table.

**STEPS**:

1)Created the tables for customers, accounts, transactions, loans, Employees .

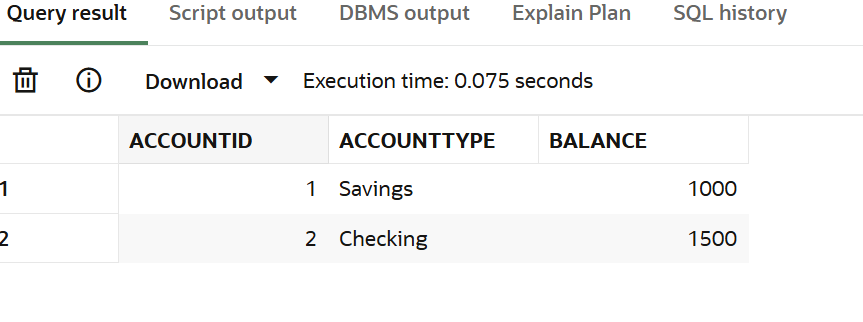
2)Inserted Sample Data

Inserted records into each table customers, accounts, transactions, loans, Employees .

3)Verified Data Insertion-Queried the Accounts table to ensure the data was present.

4)I got the result

SELECT AccountID, AccountType, Balance FROM Accounts;



5)**Created the Stored Procedure**

Created a ProcessMonthlyInterest procedure to update savings account balances:

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS**

**BEGIN**

**UPDATE Accounts**

**SET Balance = Balance \* 1.01,**

**LastModified = SYSDATE**

**WHERE AccountType = 'Savings';**

**END ProcessMonthlyInterest;**

**/**

**6)**Executed the Procedure

Run the procedure to apply 1% interes

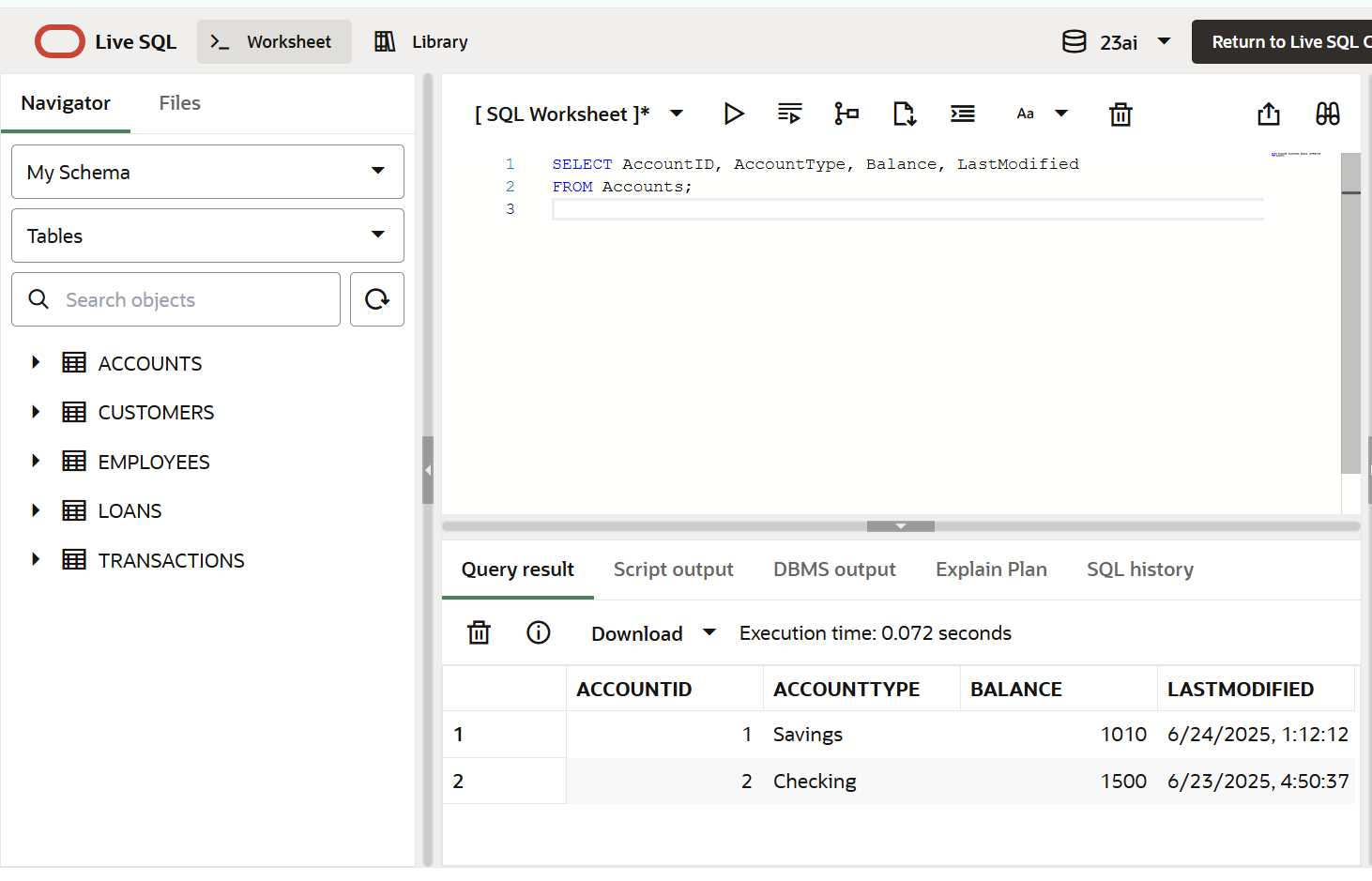
**EXEC ProcessMonthlyInterest;**

7)Checked Updated Data

Queried the accounts after execution

**SELECT AccountID, AccountType, Balance FROM Accounts;**

**OUTPUT:**

****

**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

**SOLUTION:**

**STEPS:**

**Step 1:**

* Based on the original schema provided (Employees, Customers, Accounts, etc.), there was no column or table to store employee performance information.  
  Yet, the business requirement states.
* “The bank wants to implement a bonus scheme for employees based on their performance.”
* To meet this requirement, we needed a way to record performance data — such as ratings or review scores — so that bonuses could be applied selectively to top performers.

**Step 2**:We introduced a new table called PerformanceRatings:

**CREATE TABLE PerformanceRatings (**

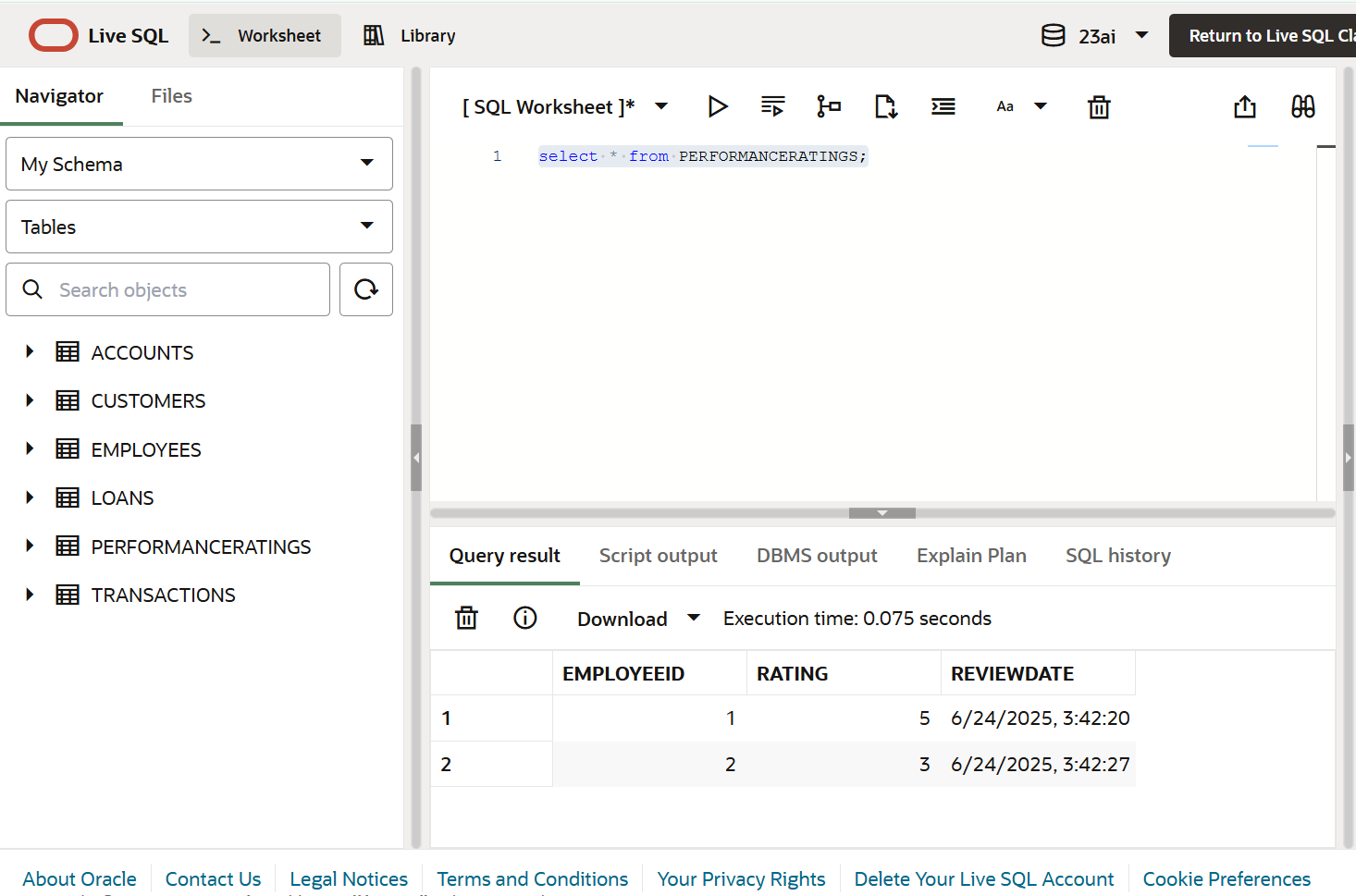
**EmployeeID NUMBER PRIMARY KEY,**

**Rating NUMBER,**

**ReviewDate DATE,**

**FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)**

**);**



**Step 3:**

* The existing Employees table contains only basic details like Salary and Department.
* Without PerformanceRatings, we cannot identify which employees deserve a raise.

By creating this table:

* We can store performance ratings separately and review them periodically.
* Our stored procedure can target specific employees with high ratings (e.g. 4 or 5 out of 5) for bonuses.
* This matches real-world HR processes where bonuses are given based on measured performance.

**Step 4:**Inserting a new values into Performance Ratings table

I inserted a new rows because we need some example data to verify that the table and logic work as intended.

INSERT INTO PerformanceRatings (EmployeeID, Rating, ReviewDate)

VALUES (1, 5, SYSDATE); -- Employee 1 is top performer

INSERT INTO PerformanceRatings (EmployeeID, Rating, ReviewDate)

VALUES (2, 3, SYSDATE); -- Employee 2 is average performer

**Step 5**:Create the Stored Procedure for the logic

**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(**

**p\_department IN VARCHAR2,**

**p\_bonus\_percentage IN NUMBER**

**) IS**

**BEGIN**

**UPDATE Employees e**

**SET e.Salary = e.Salary + (e.Salary \* p\_bonus\_percentage / 100)**

**WHERE e.Department = p\_department**

**AND e.EmployeeID IN (**

**SELECT pr.EmployeeID**

**FROM PerformanceRatings pr**

**WHERE pr.Rating >= 4**

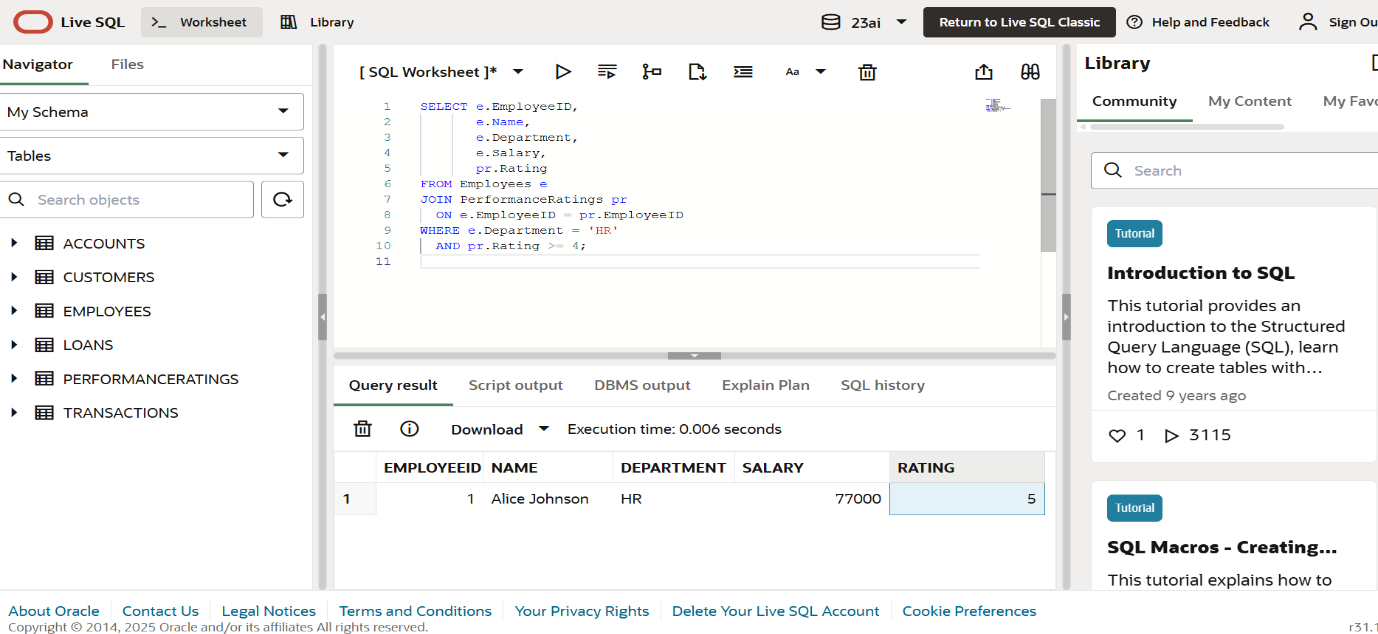
**);**

**COMMIT;**

**END;**

**/**

**Step 6:**Before adding the bonus



**Step 7**: Test the Procedure

EXEC UpdateEmployeeBonus('HR', 10);

After adding bonus of employee

**Step 8**: OUTPUT

SELECT e.EmployeeID,

       e.Name,

       e.Department,

       e.Salary,

       pr.Rating

FROM Employees e

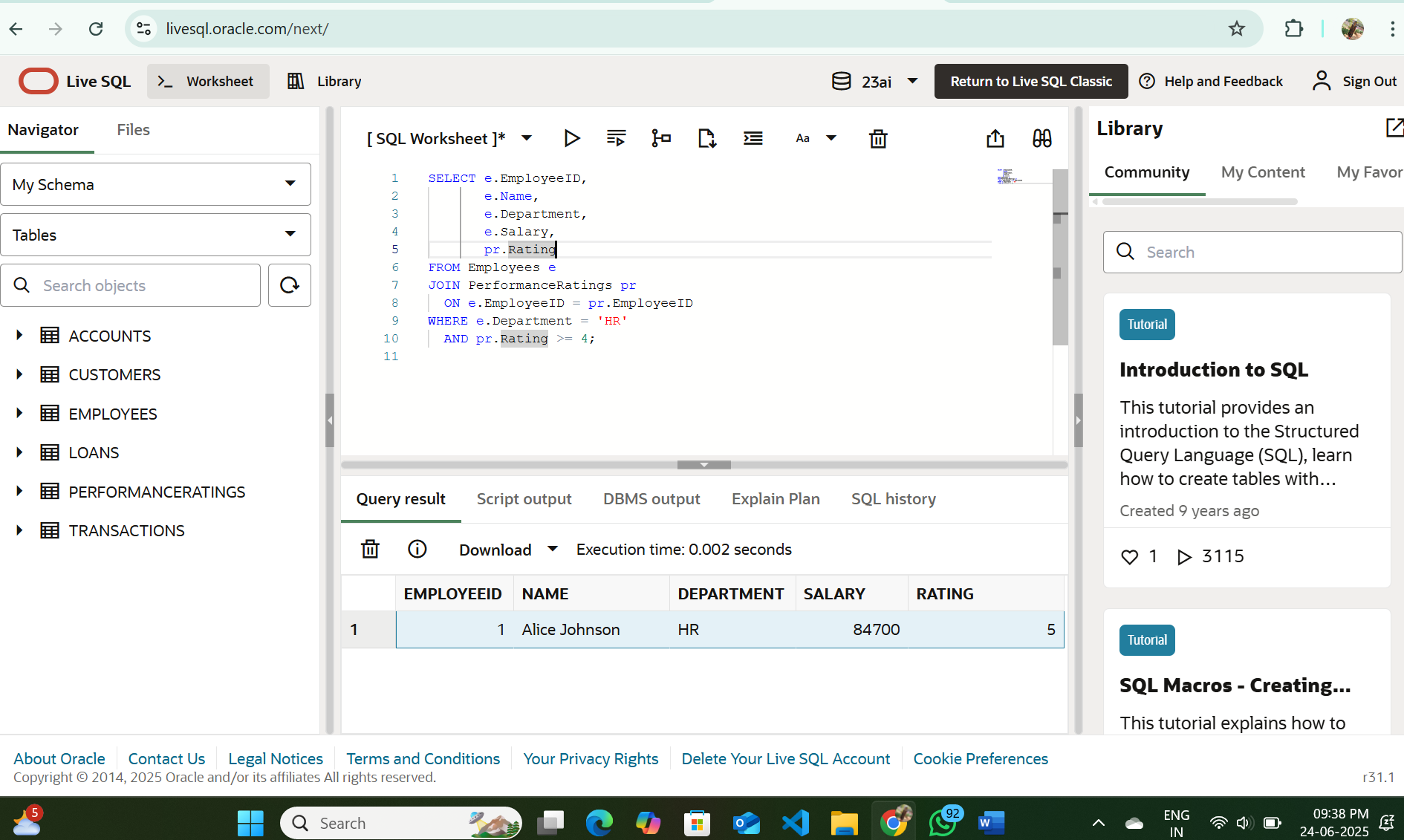
JOIN PerformanceRatings pr

  ON e.EmployeeID = pr.EmployeeID

WHERE e.Department = 'HR'

  AND pr.Rating >= 4;

**OUTPUT:**



This bonus was added to the salary of the alice employee .

**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.**

SOLUTION:

**Step 1:** Write the Procedure

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance >= p\_amount THEN

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account.');

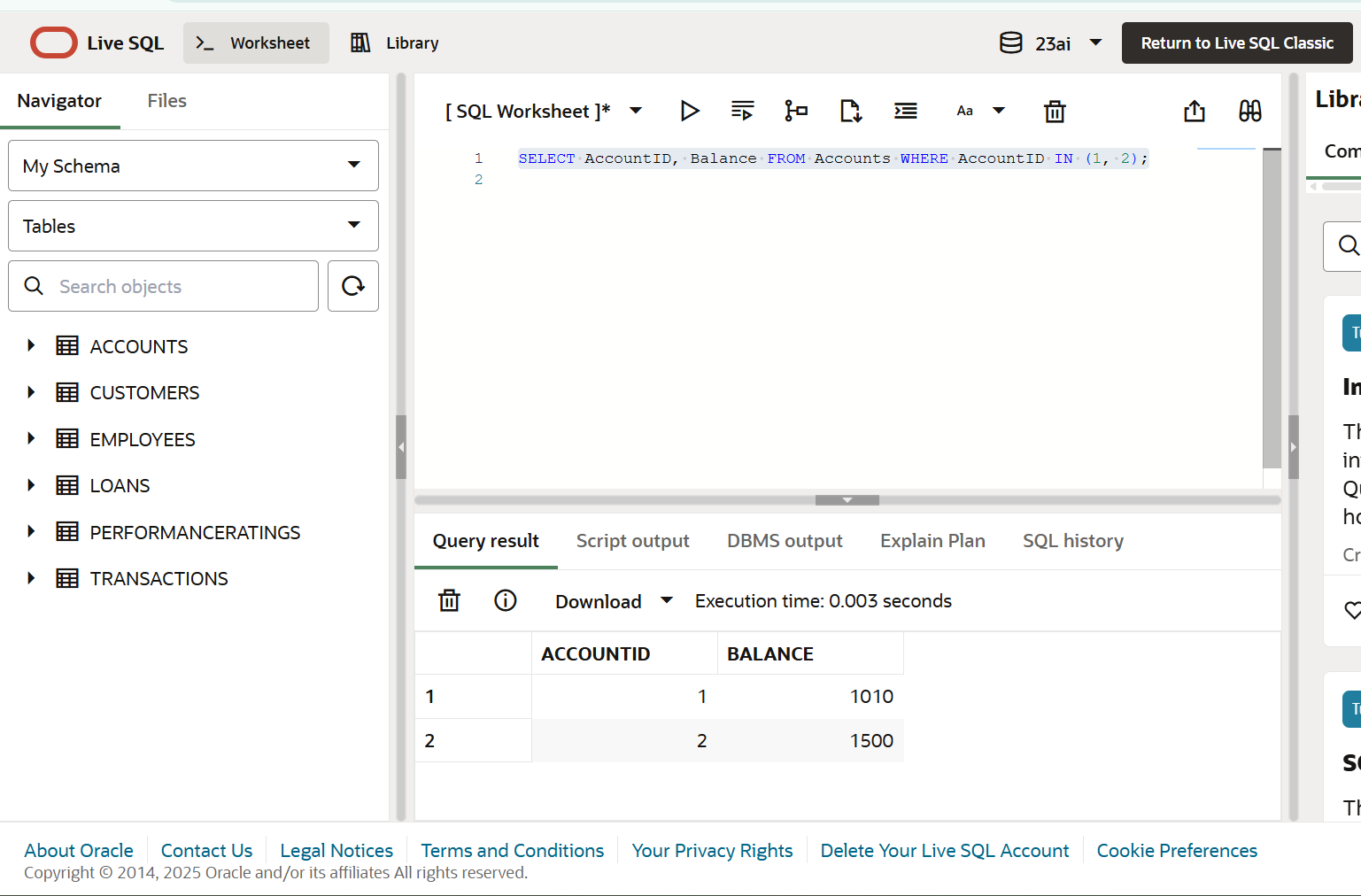
END IF;

END;

/

**Step 2: Check account balances before the transfer**

**SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1, 2);**



**Step 3:** Test the transfer

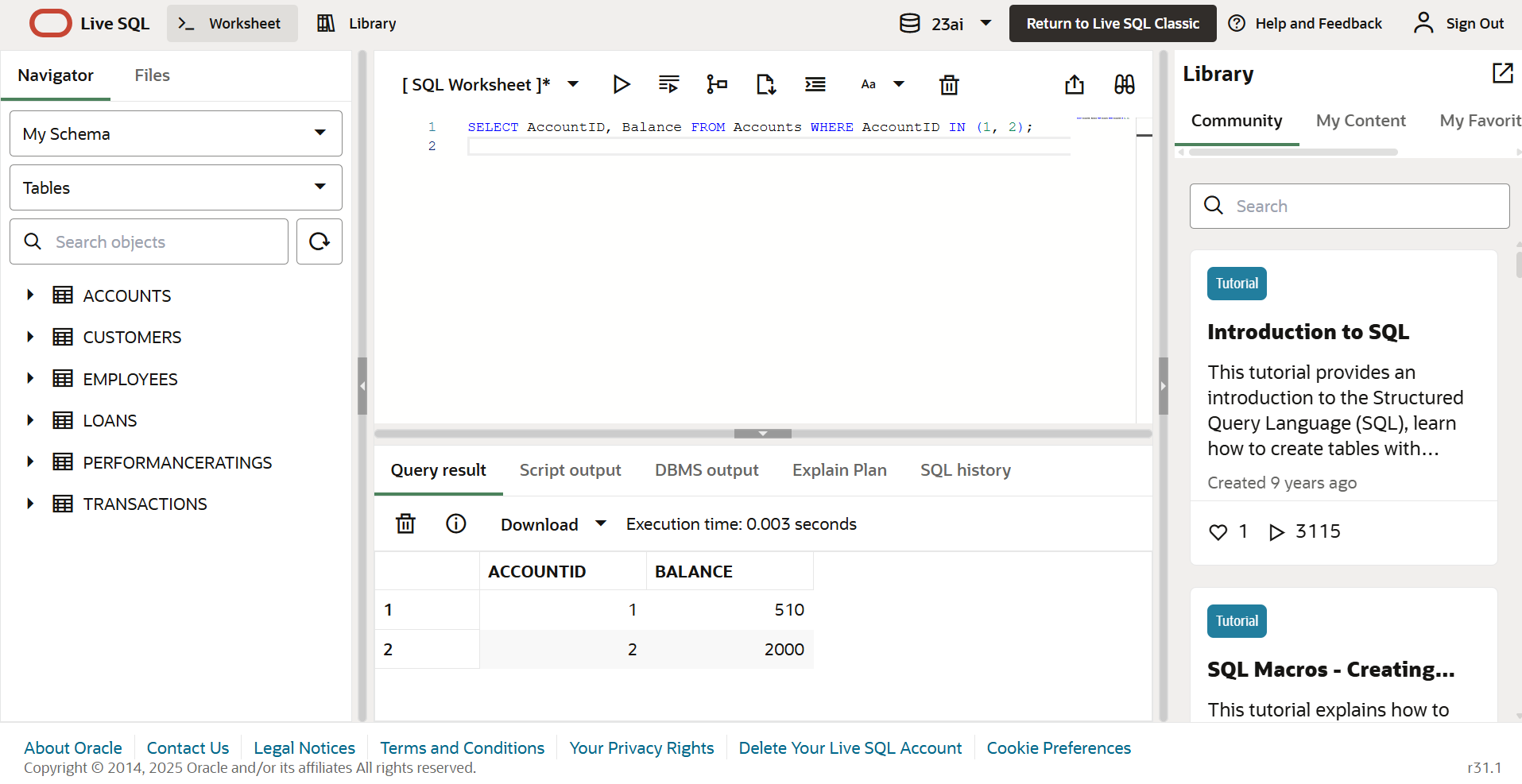
SET SERVEROUTPUT ON;

EXEC TransferFunds(1, 2, 500);

**Step 4:** Check account balances after the transfer

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1, 2);

**OUTPUT:**



The amount was added from account 1 (500) to account 2 (500).