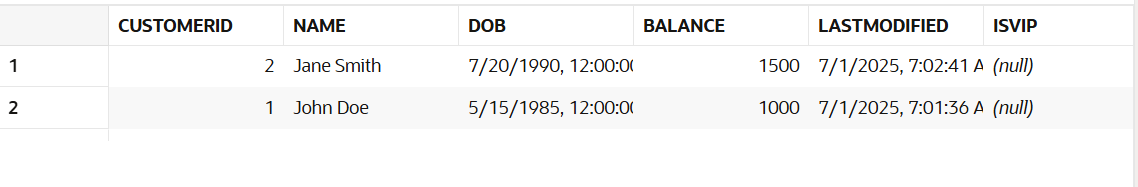
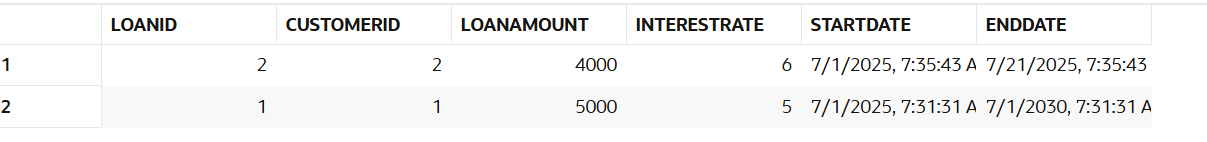
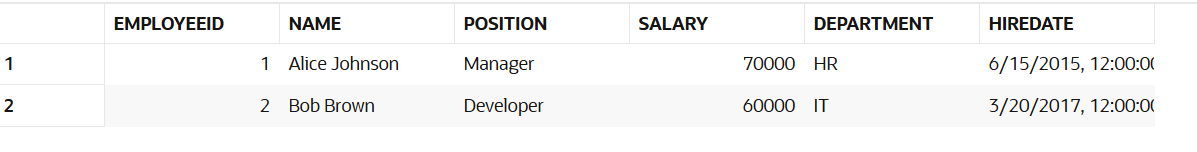
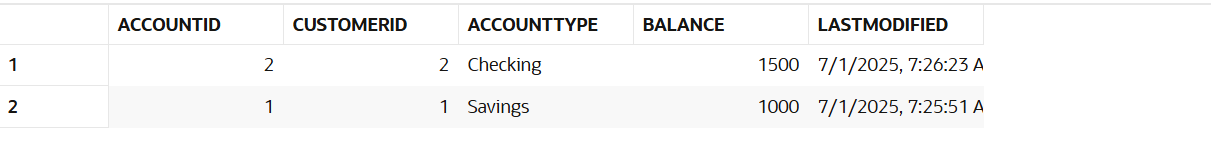
TABLES:

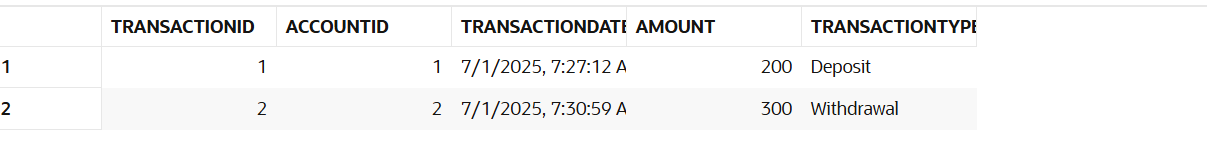
Given create and insert queries.











**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates

BEGIN

FOR cust\_rec IN (

SELECT CustomerID, DOB

FROM Customers

)

LOOP

DECLARE

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

END IF;

END;

END LOOP;

COMMIT;

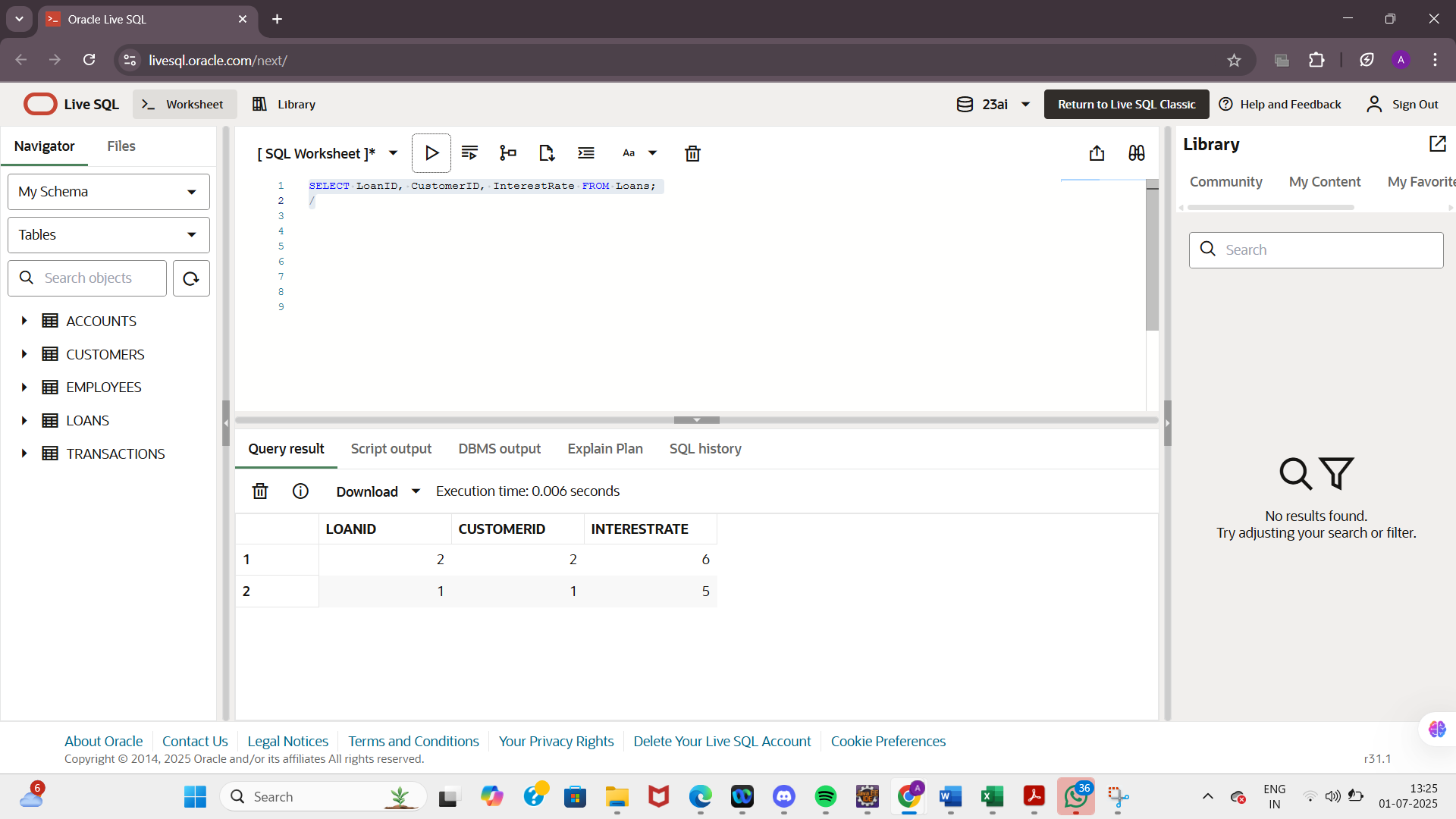
END;

/

OUTPUT:

SELECT LoanID, CustomerID, InterestRate FROM Loans;

/



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

BEGIN

FOR cust IN (

SELECT CustomerID, Balance

FROM Customers

)

LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

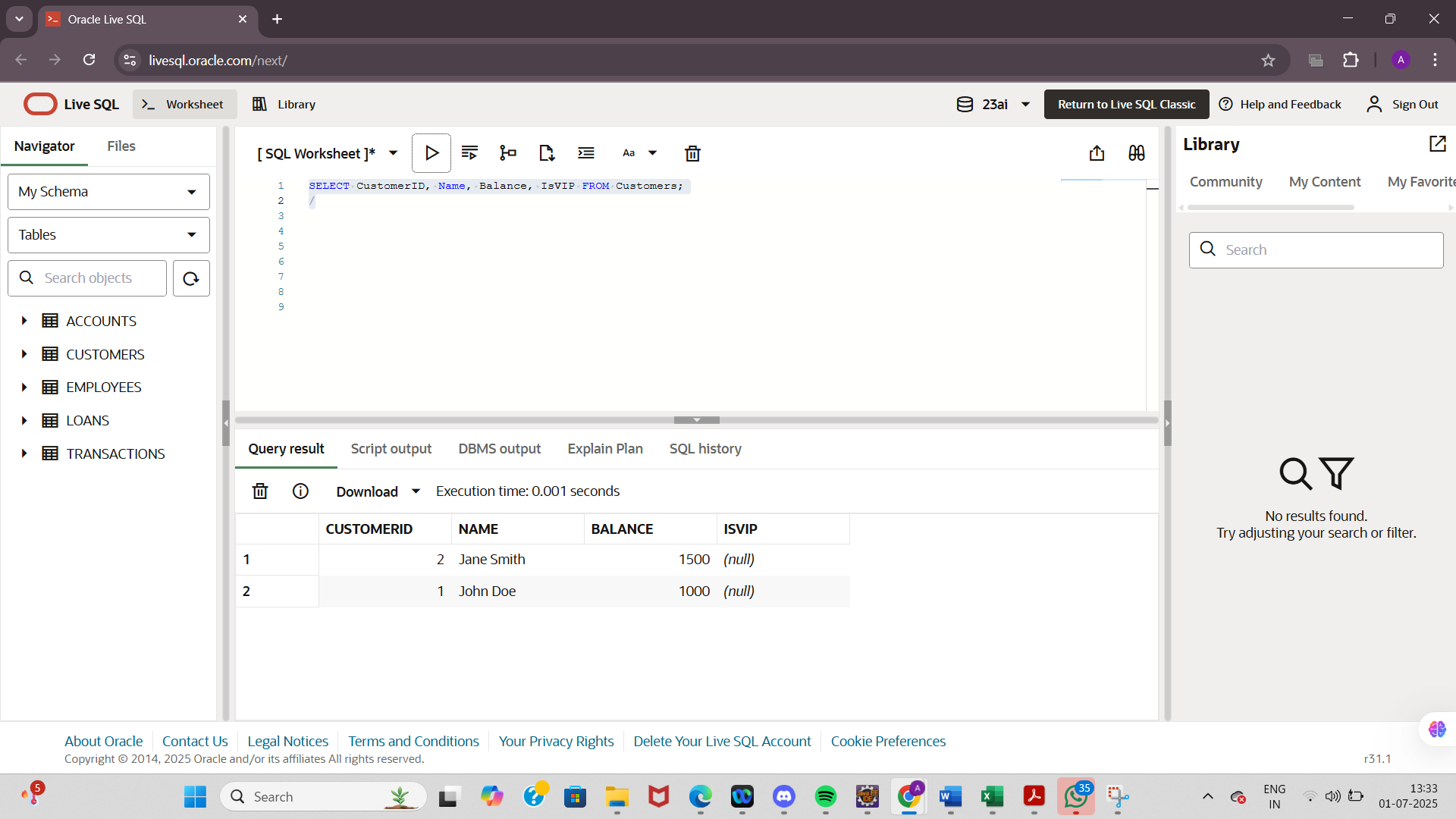
END;

/

OUTPUT:

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

/



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer

BEGIN

FOR loan\_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 ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

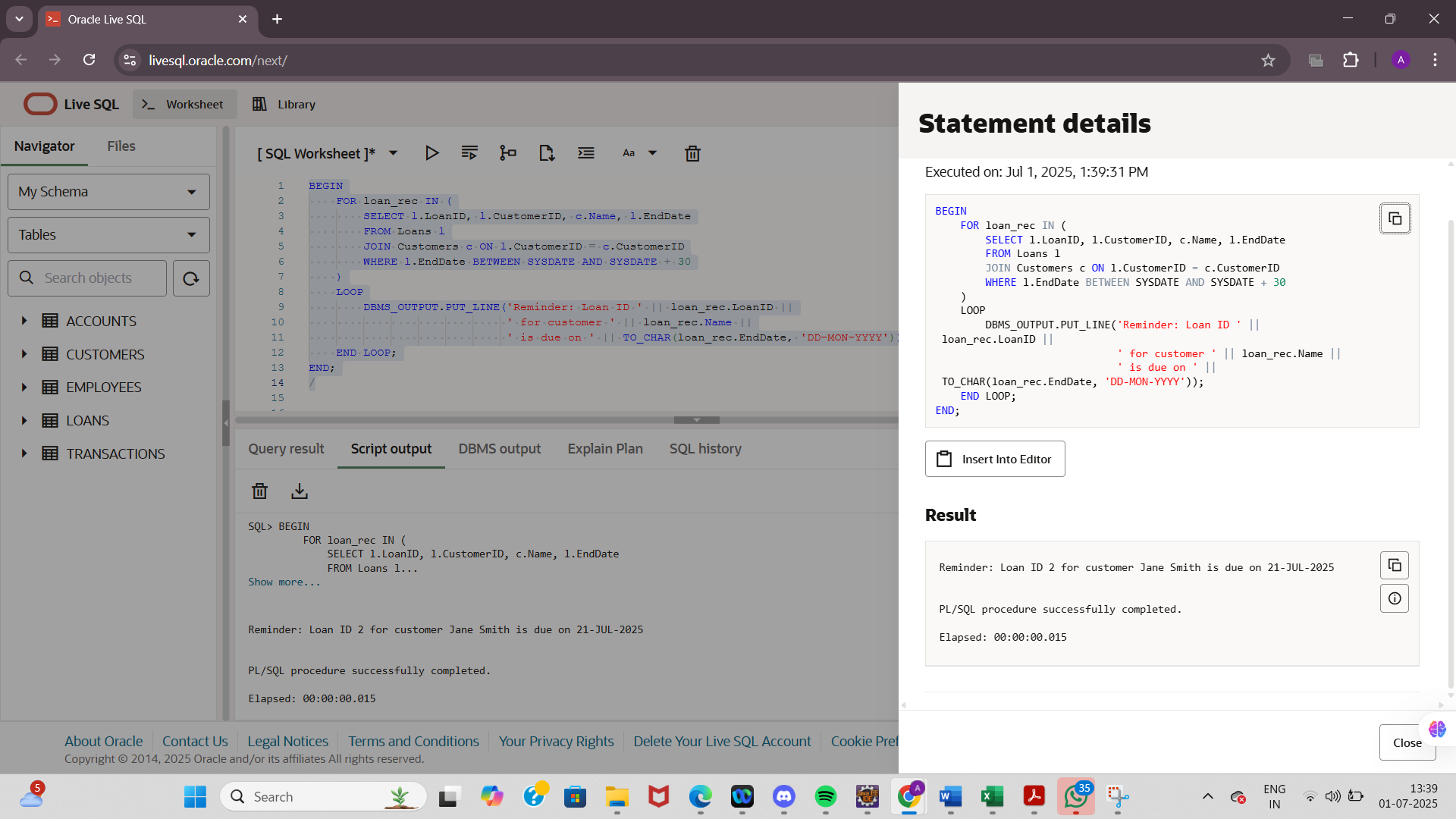
' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

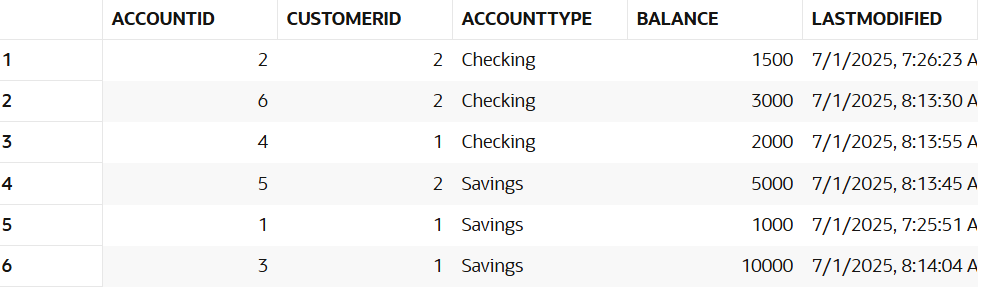
/

OUTPUT:



**Exercise 3: Stored Procedures**

**TABLES:**

****

****

**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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

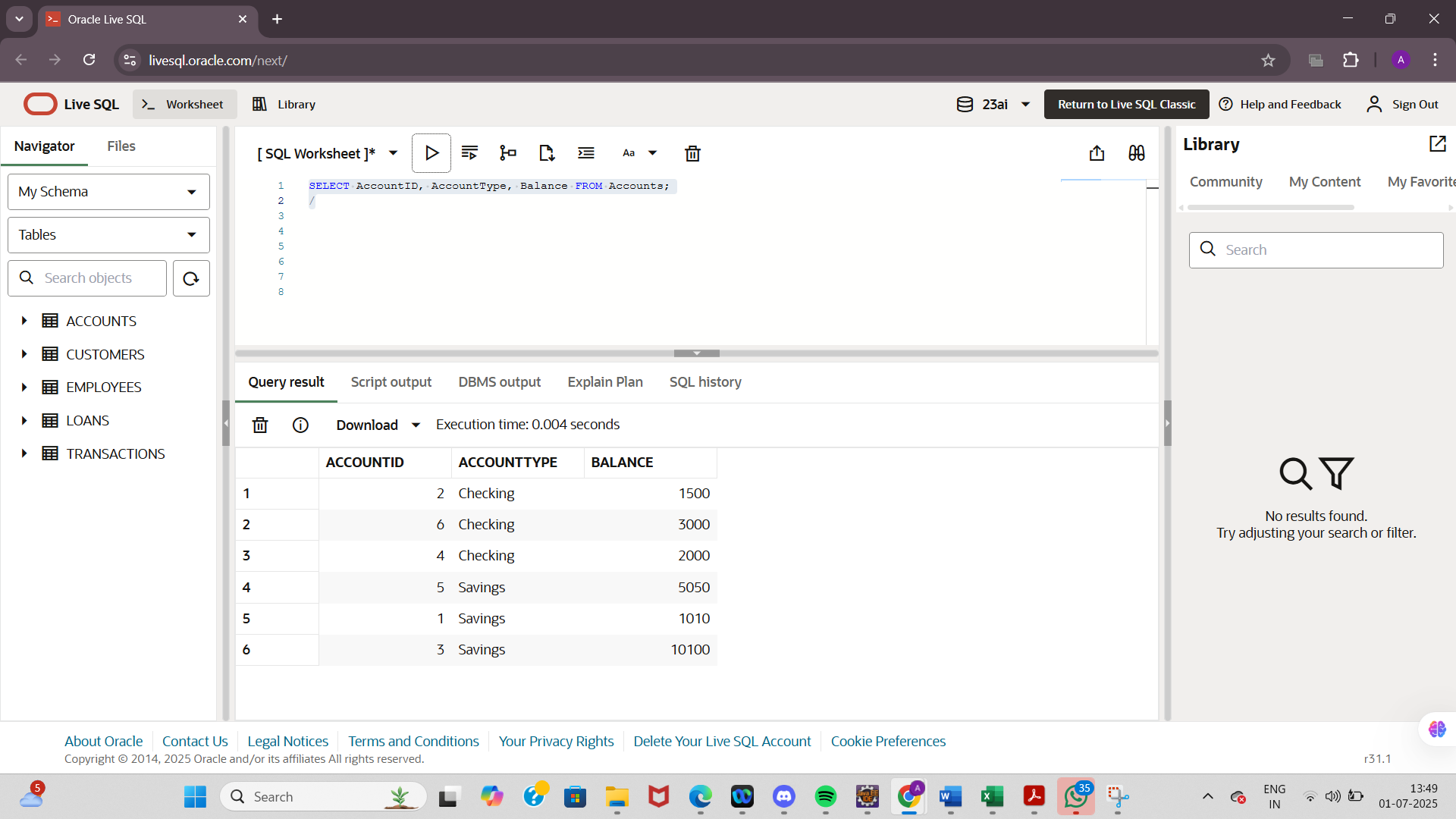
END;

/

OUTPUT:

SELECT AccountID, AccountType, Balance FROM Accounts;

/



**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE Department = p\_department;

COMMIT;

END;

/

BEGIN

UpdateEmployeeBonus('Finance', 10);

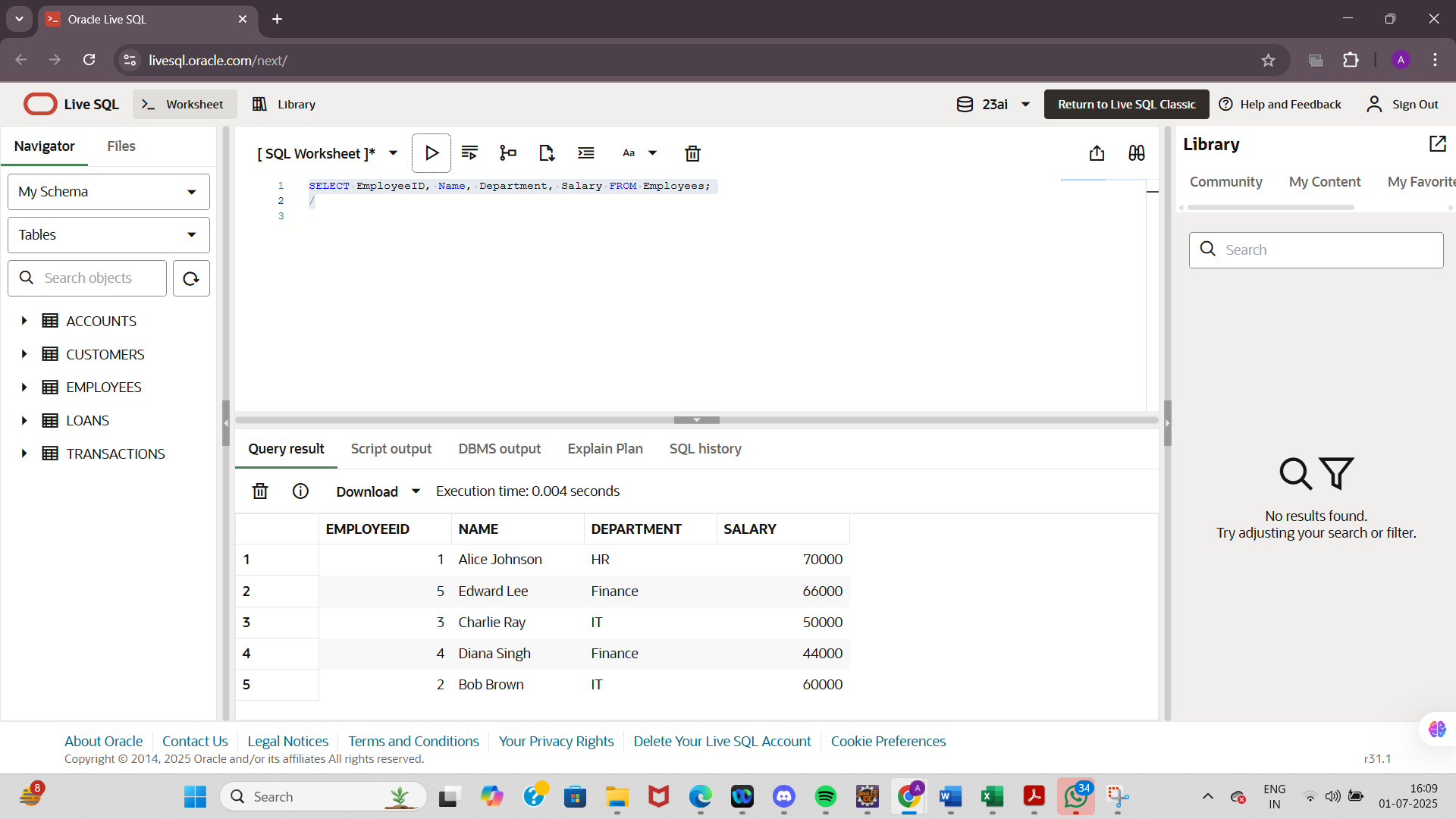
END;

/

OUTPUT:

SELECT EmployeeID, Name, Department, Salary FROM Employees;

/



**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.

DECLARE

v\_from\_account\_id NUMBER := 4; -- source account (Checking for Customer 1)

v\_to\_account\_id NUMBER := 3; -- destination account (Savings for Customer 1)

v\_amount NUMBER := 500;

v\_from\_balance NUMBER;

BEGIN

-- Get current balance of source account

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = v\_from\_account\_id;

IF v\_from\_balance >= v\_amount THEN

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - v\_amount

WHERE AccountID = v\_from\_account\_id;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + v\_amount

WHERE AccountID = v\_to\_account\_id;

COMMIT;

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

ELSE

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

END IF;

END;

/

OUTPUT:

SELECT AccountID, AccountType, Balance FROM Accounts

WHERE AccountID IN (3, 4);

/

