**PL/SQL: Hands on practices**

**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.

**PL/SQL CODE:**

SET SERVEROUTPUT ON;

BEGIN

    FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP

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

            UPDATE Loans

            SET InterestRate = InterestRate - (InterestRate \* 0.01)

            WHERE CustomerID = cust.CustomerID;

        END IF;

    END LOOP;

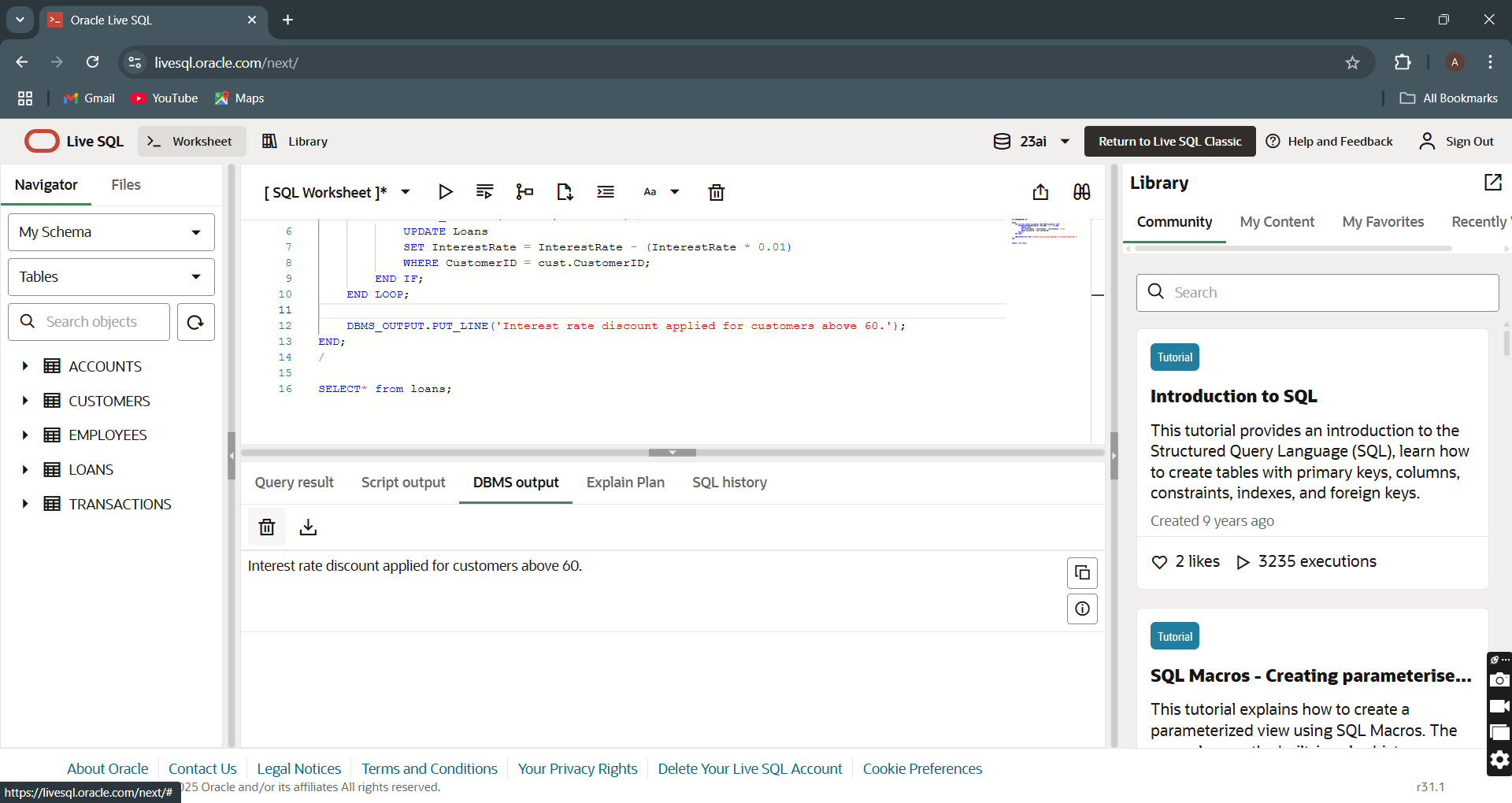
    DBMS\_OUTPUT.PUT\_LINE('Interest rate discount applied for customers above 60.');

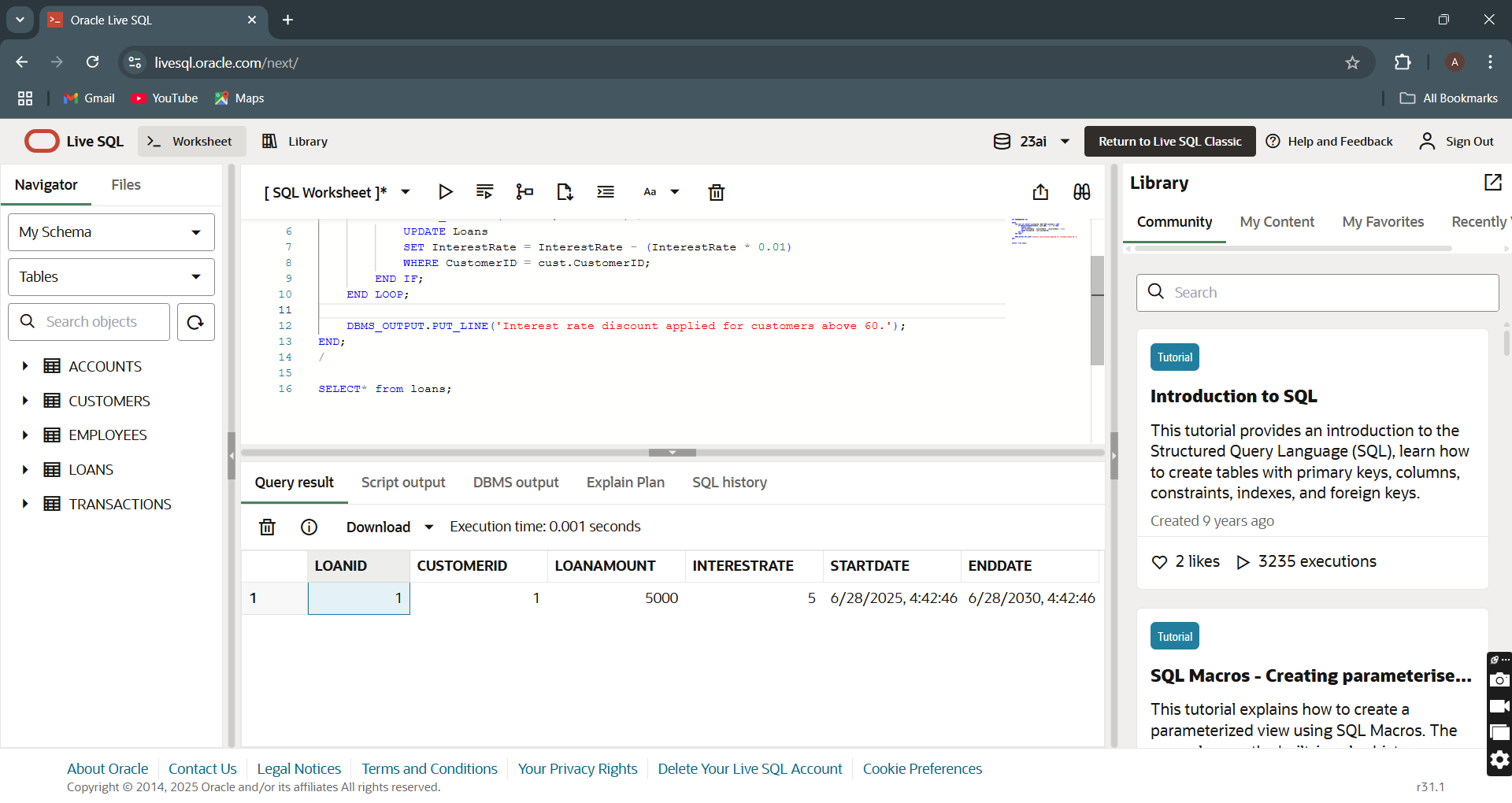
END;

/

SELECT\* from loans;

**OUTPUT:**

****



**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.

**PL/SQL CODE:**

SET SERVEROUTPUT ON;

BEGIN

    FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

        IF cust.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'TRUE'

            WHERE CustomerID = cust.CustomerID;

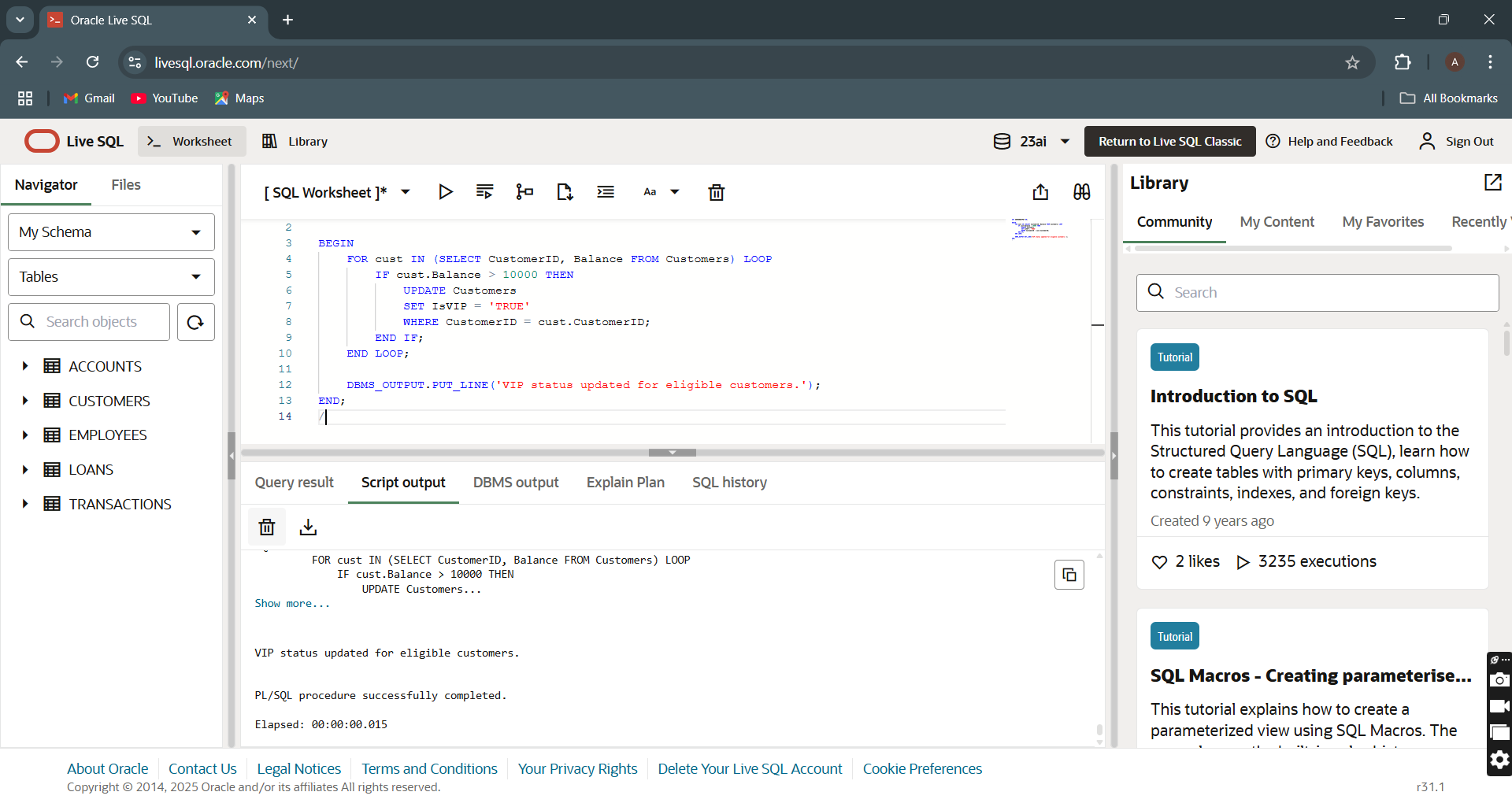
        END IF;

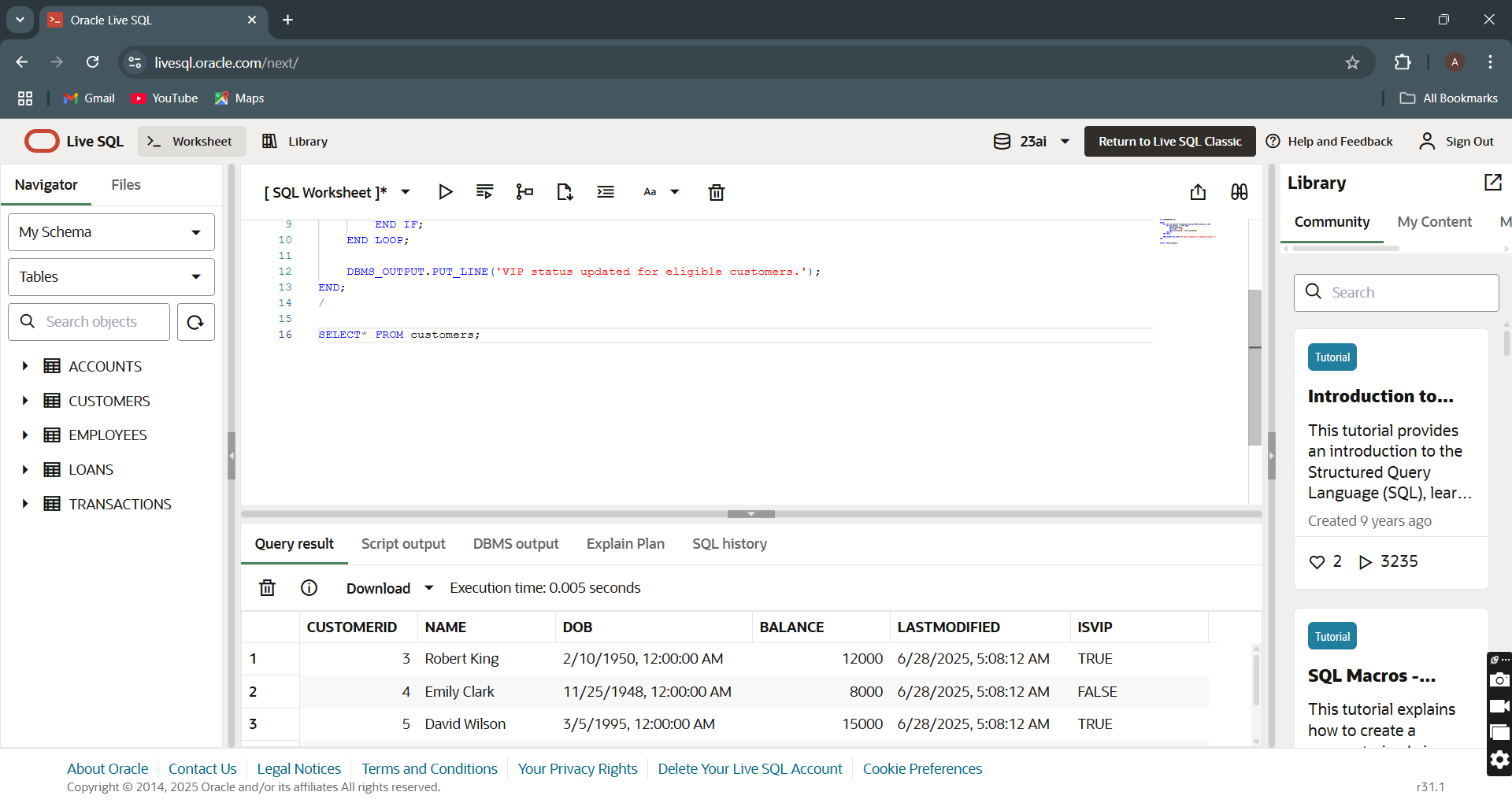
    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('VIP status updated for eligible customers.');

END;

/

**OUTPUT:**

****

**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.

**PL/SQL CODE:**

SET SERVEROUTPUT ON;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Loan Due Reminders (Next 30 Days):');

    FOR rec IN (

        SELECT L.LoanID, L.CustomerID, L.EndDate, C.Name

        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: LoanID ' || rec.LoanID || ' for Customer ' || rec.Name ||

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

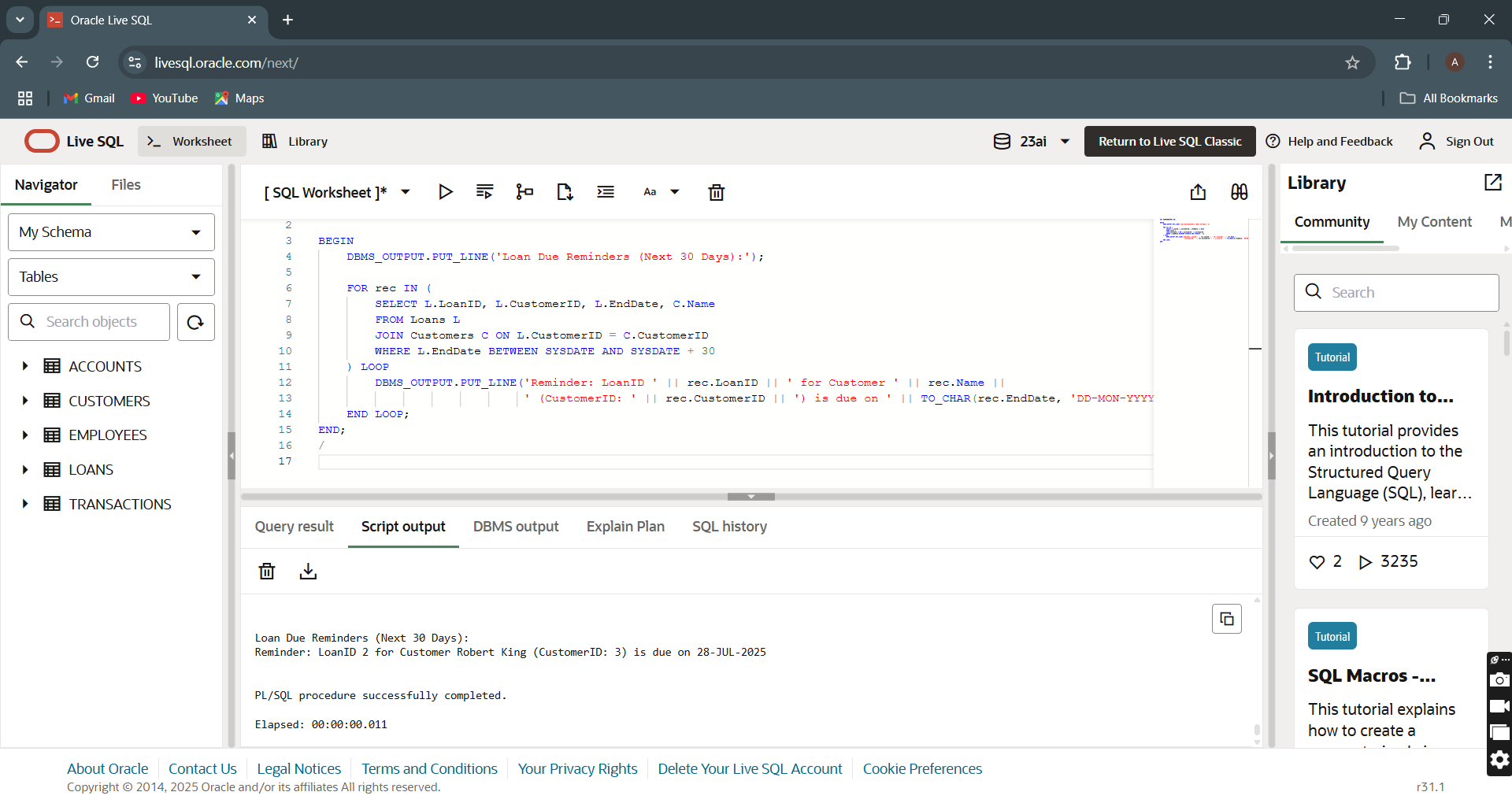
    END LOOP;

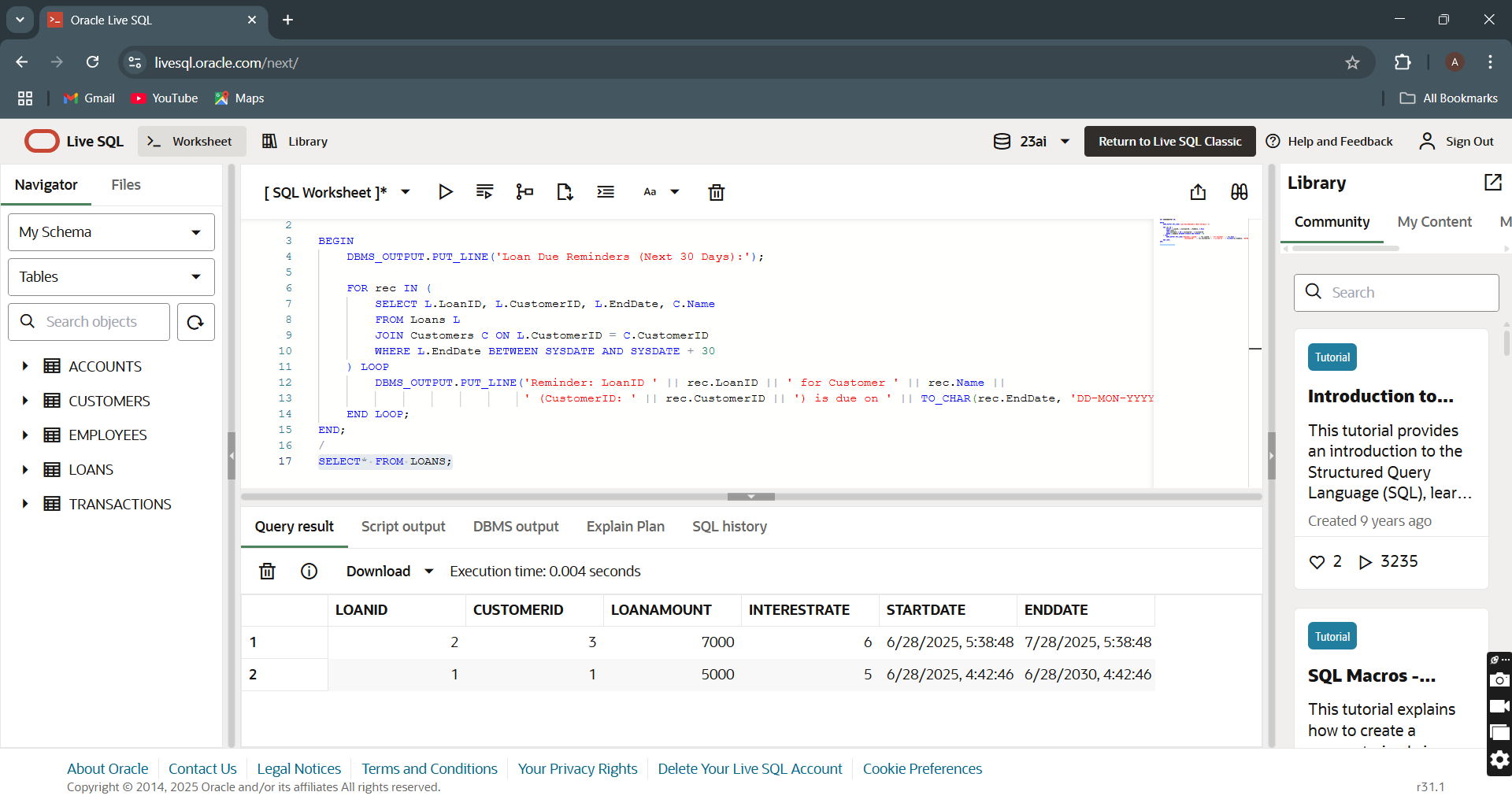
END;

/

SELECT\* FROM LOANS;

**OUTPUT:**

****

****

**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.

**PL/SQL CODE:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    -- Apply Interest

    UPDATE Accounts

    SET Balance = Balance + (Balance \* 0.01)

    WHERE AccountType = 'Savings';

    -- Print Header

    DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to savings accounts.');

    DBMS\_OUTPUT.PUT\_LINE('Affected Accounts:');

    -- Loop to print AccountID and Balance

    FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

        DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || rec.AccountID || ', Updated Balance: ' || rec.Balance);

    END LOOP;

END;

/

SET SERVEROUTPUT ON;

BEGIN

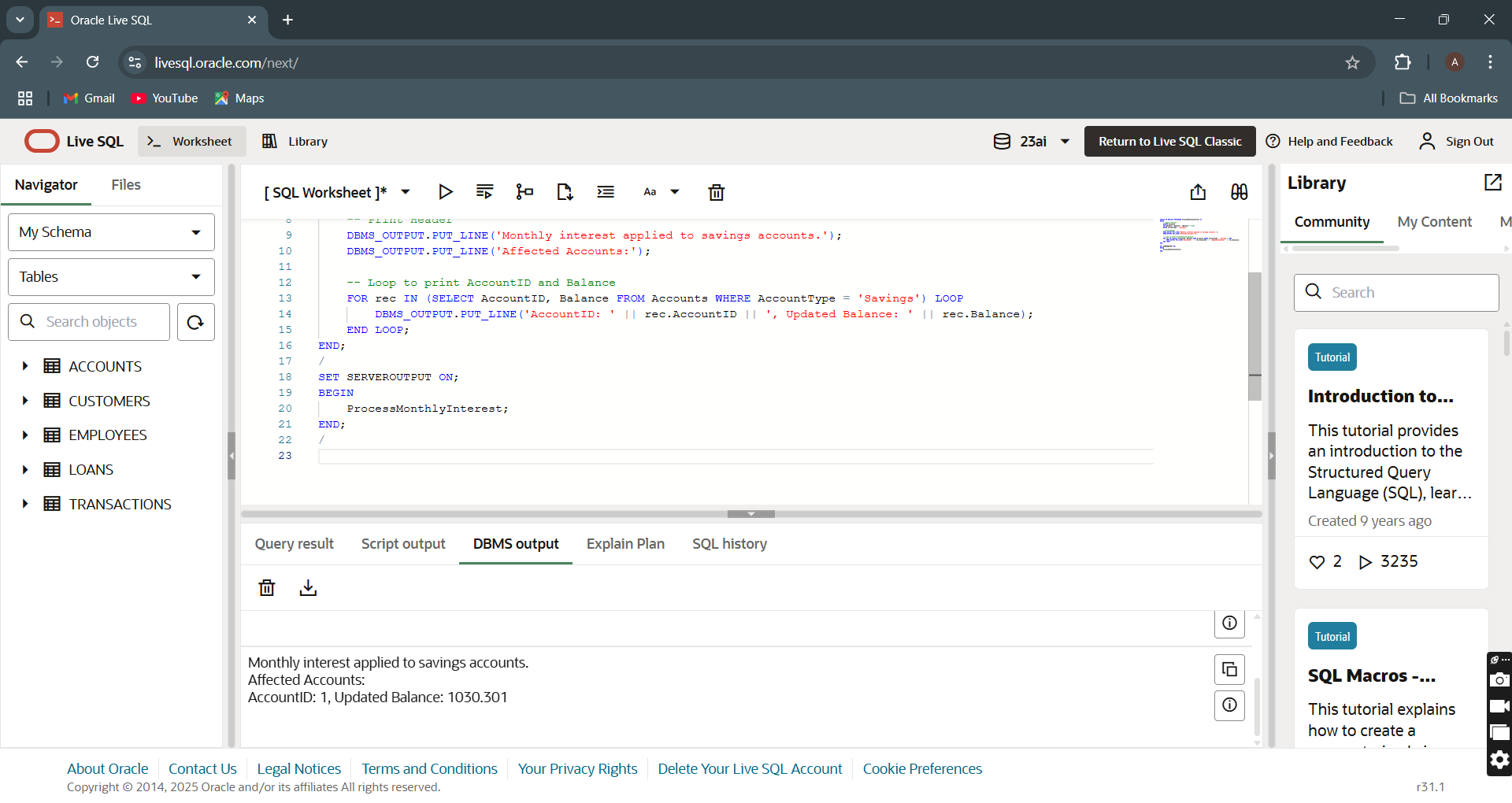
    ProcessMonthlyInterest;

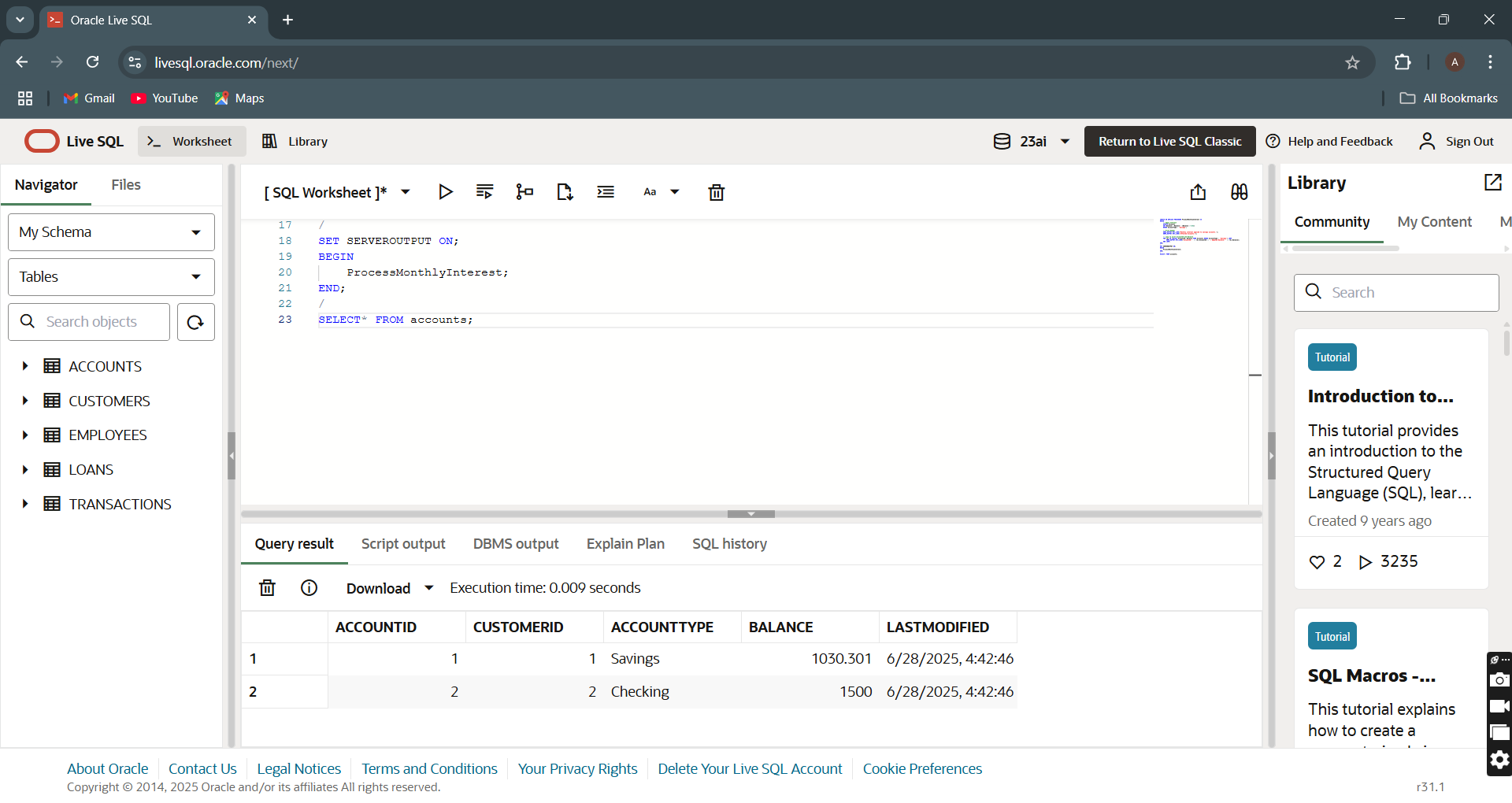
END;

/

SELECT\* 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.

**PL/SQL CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_Department IN VARCHAR2,

    p\_BonusPercent IN NUMBER

) IS

BEGIN

    UPDATE Employees

    SET Salary = Salary + (Salary \* (p\_BonusPercent / 100))

    WHERE Department = p\_Department;

    DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_BonusPercent || '% applied to Department: ' || p\_Department);

END;

/

SET SERVEROUTPUT ON;

BEGIN

    UpdateEmployeeBonus('HR', 10);

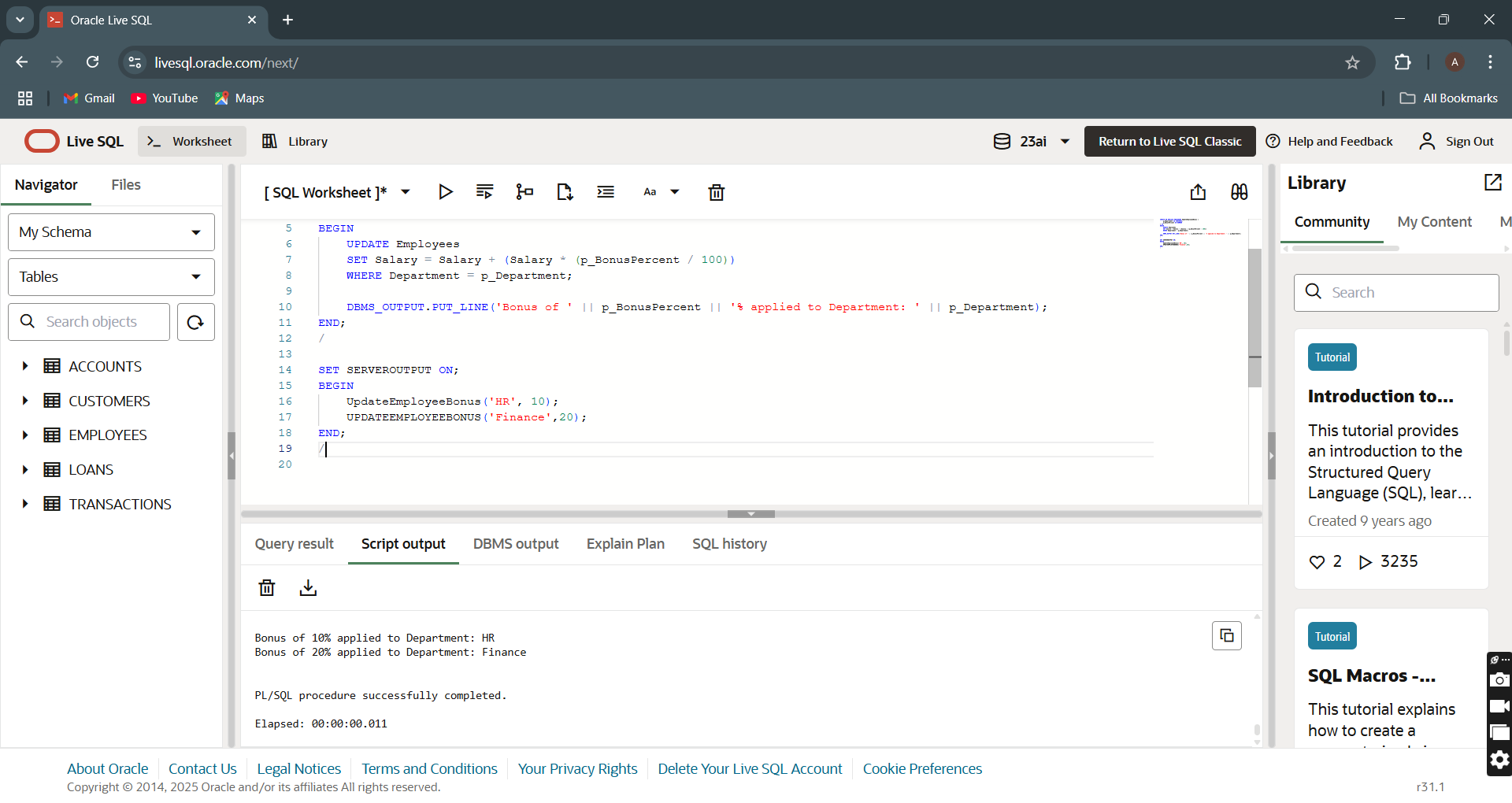
    UPDATEEMPLOYEEBONUS('Finance',20);

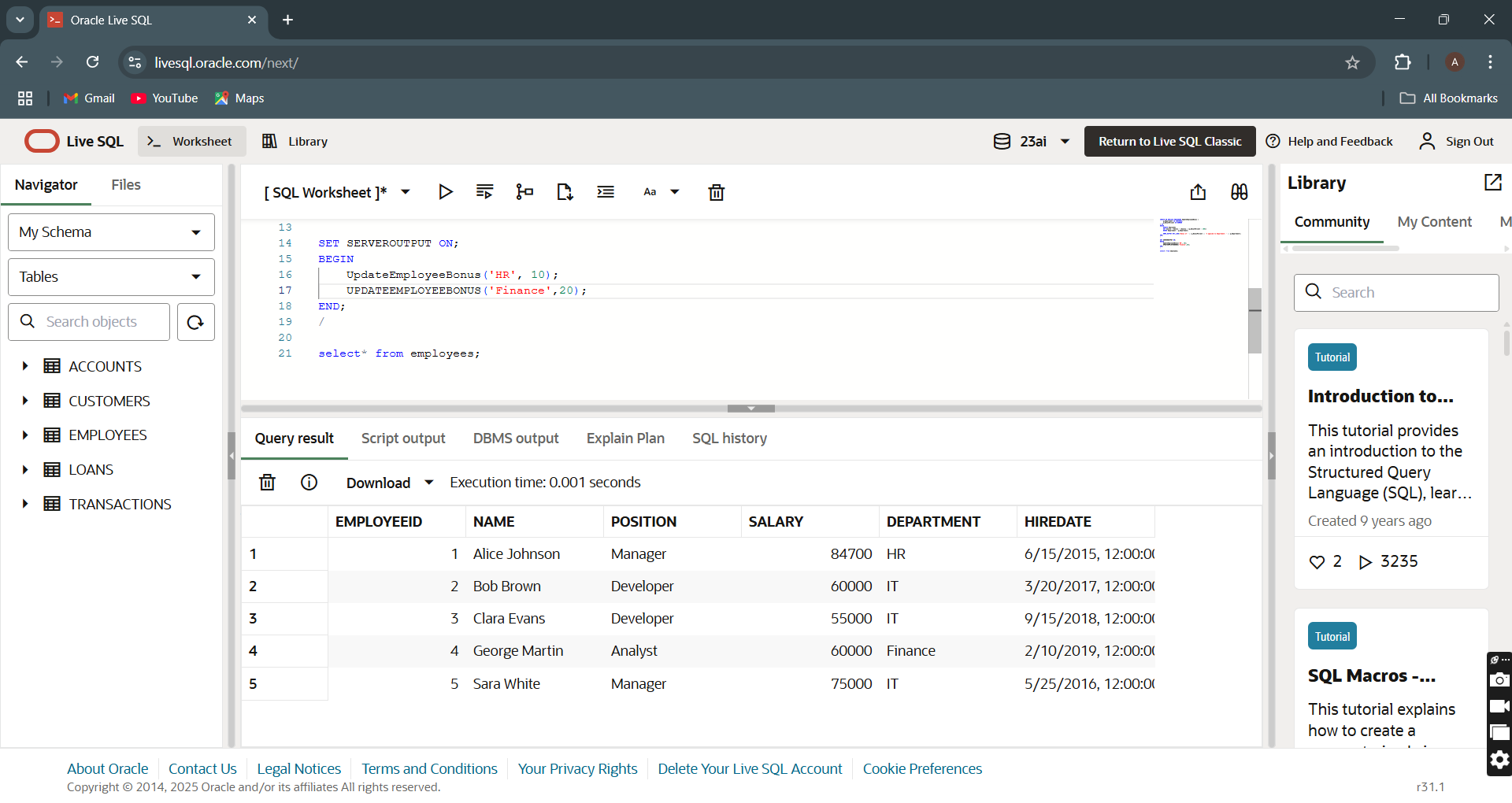
END;

/

select\* from employees;

**OUTPUT:**

****

****

**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.

**PL/SQL CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_FromAccount IN NUMBER,

    p\_ToAccount IN NUMBER,

    p\_Amount IN NUMBER

) IS

    v\_Balance NUMBER;

BEGIN

    SELECT Balance INTO v\_Balance

    FROM Accounts

    WHERE AccountID = p\_FromAccount;

    IF v\_Balance >= p\_Amount THEN

        UPDATE Accounts

        SET Balance = Balance - p\_Amount

        WHERE AccountID = p\_FromAccount;

        UPDATE Accounts

        SET Balance = Balance + p\_Amount

        WHERE AccountID = p\_ToAccount;

        DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_Amount || ' completed from AccountID ' || p\_FromAccount || ' to AccountID ' || p\_ToAccount);

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in AccountID ' || p\_FromAccount);

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: One or both account IDs are invalid.');

END;

/

SET SERVEROUTPUT ON;

BEGIN

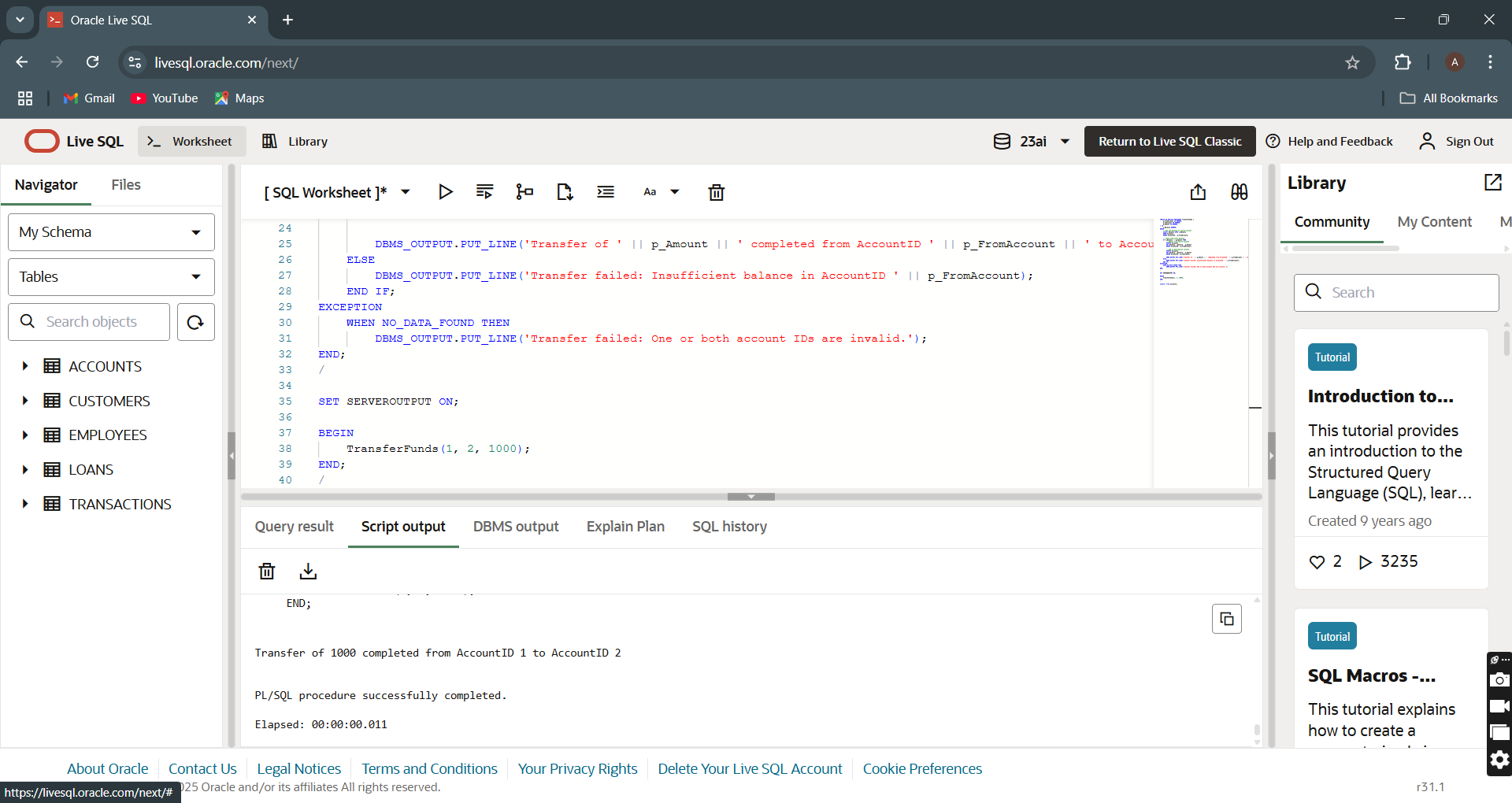
    TransferFunds(1, 2, 1000);

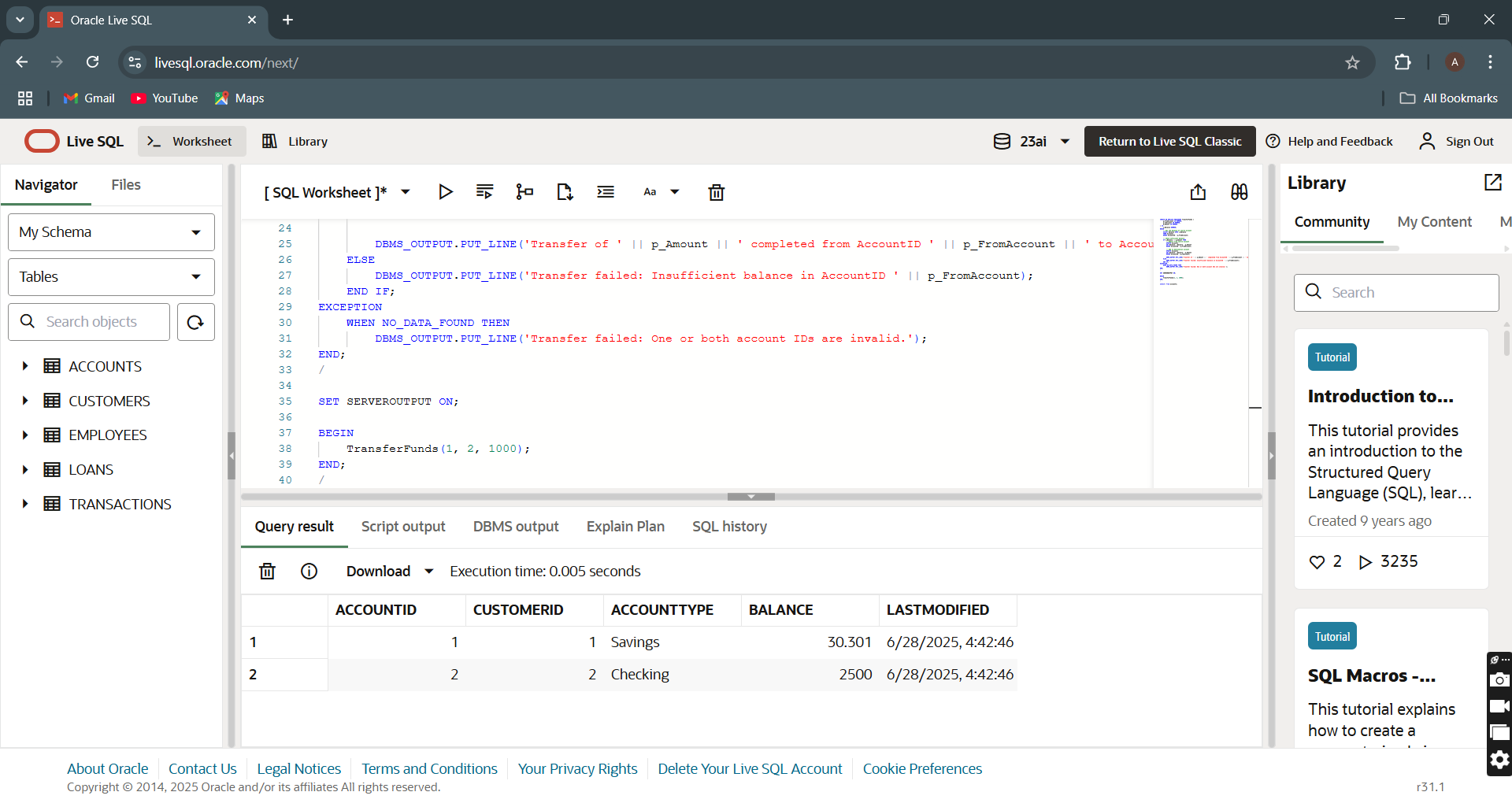
END;

/

select\* from accounts;

**OUTPUT:**

****

****