**Exercise 1: Control Structures**

CREATE TABLE customer (

  customer\_id NUMBER PRIMARY KEY,

  name VARCHAR2(50),

  age NUMBER,

  balance NUMBER,

  IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE loan (

  loan\_id NUMBER PRIMARY KEY,

  customer\_id NUMBER,

  interest\_rate NUMBER,

  due\_date DATE,

  FOREIGN KEY (customer\_id) REFERENCES customer(customer\_id)

);

INSERT INTO customer VALUES (1, 'Tanima Bose', 65, 15000, 'FALSE');

INSERT INTO customer VALUES (2, 'Kalpana Basu', 45, 8000, 'FALSE');

INSERT INTO customer VALUES (3, 'Susmita Debnath', 70, 12000, 'FALSE');

INSERT INTO customer VALUES (4, 'Gopal Mitra', 30, 5000, 'FALSE');

INSERT INTO loan VALUES (101, 1, 10.5, SYSDATE + 15);

INSERT INTO loan VALUES (102, 2, 11.0, SYSDATE + 40);

INSERT INTO loan VALUES (103, 3, 12.0, SYSDATE + 5);

INSERT INTO loan VALUES (104, 4, 9.5, SYSDATE + 10);

COMMIT;

SELECT \* FROM customer;

SELECT \* FROM loan;

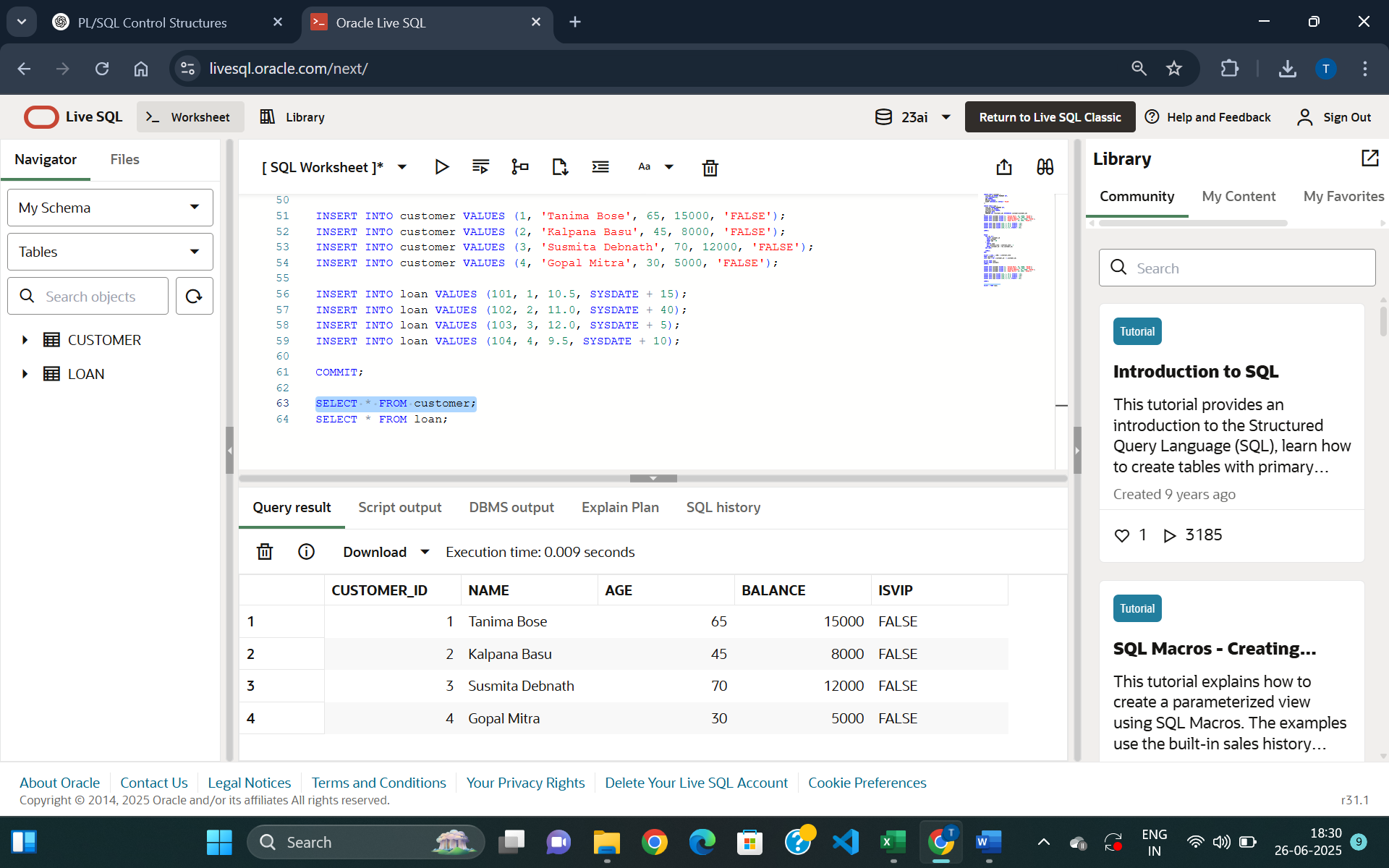
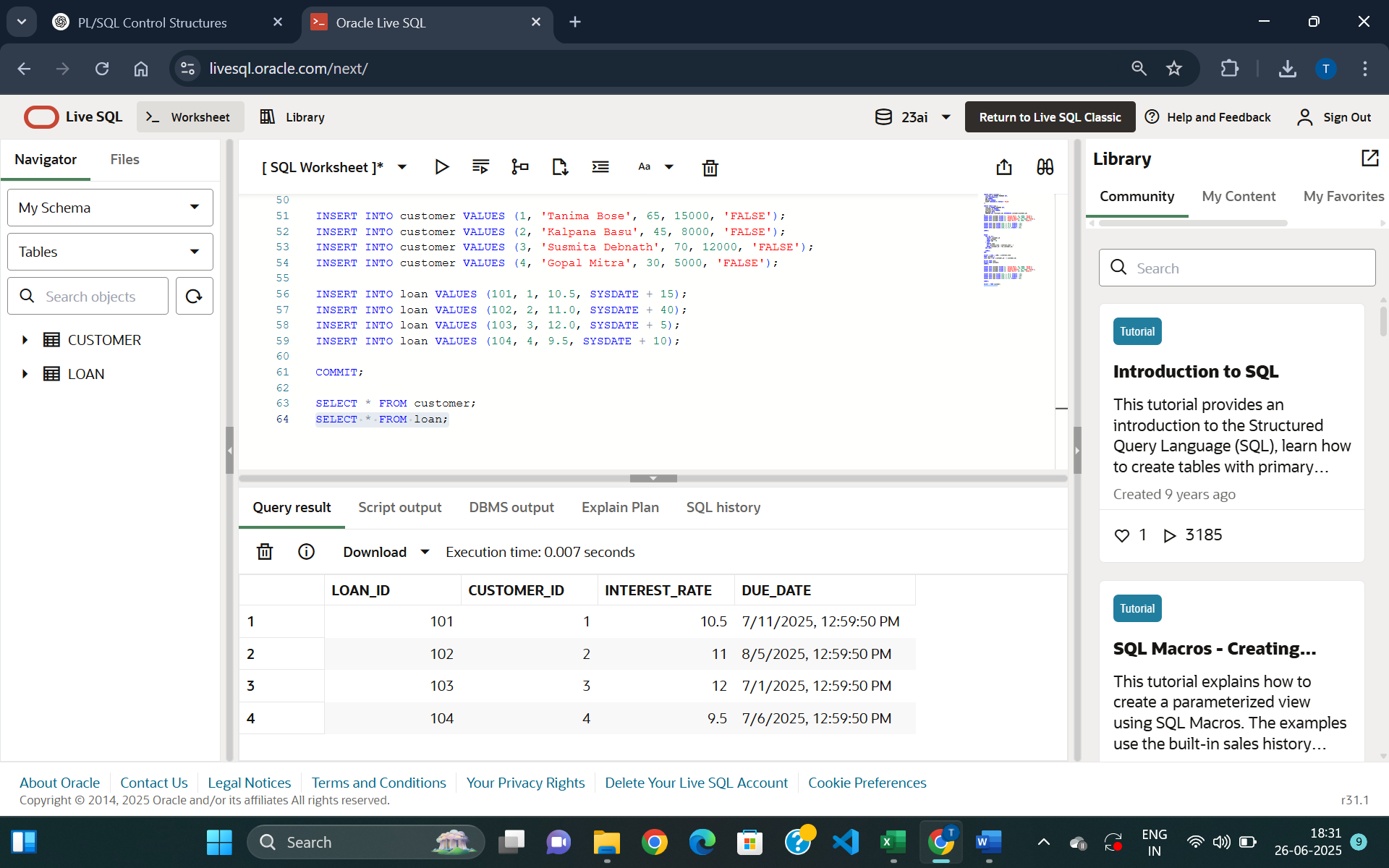


Figure 1: Customer Table Creation



*Figure 2: Loan Table Creation*

**Exercise 1:**

**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

--Scenario1

BEGIN

  FOR rec IN (

    SELECT customer\_id

    FROM customer

    WHERE age > 60

  ) LOOP

    UPDATE loan

    SET interest\_rate = interest\_rate - 1

    WHERE customer\_id = rec.customer\_id;

  END LOOP;

  COMMIT;

END;

SELECT c.name, c.AGE, l.interest\_rate

FROM customer c

JOIN loan l ON c.customer\_id = l.customer\_id;

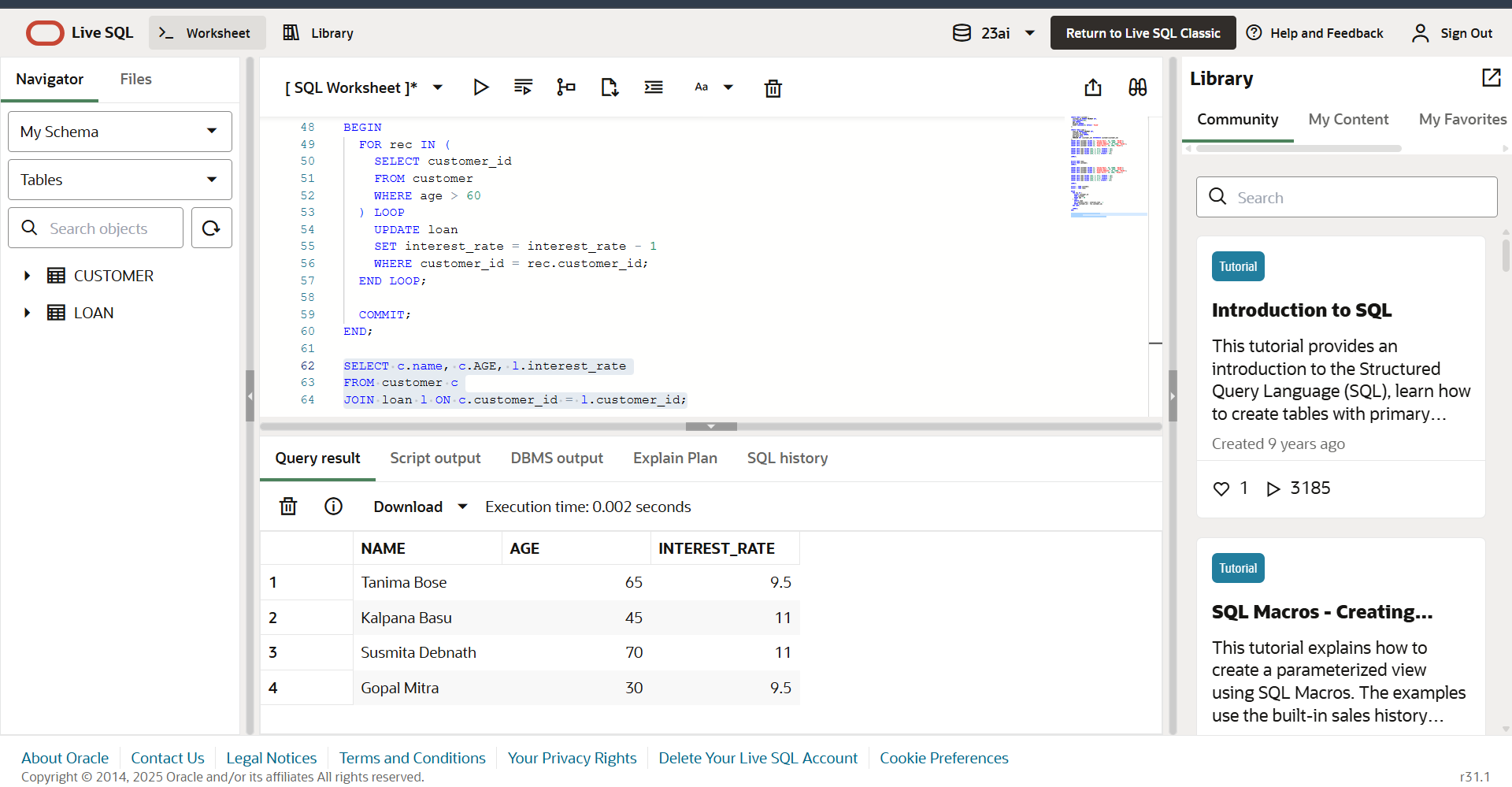


Figure 3: Applied a 1% discount to current loan interest rates for customers with age above 60

**Exercise 1:**

**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.

--Scenario2

BEGIN

  FOR rec IN (

    SELECT customer\_id

    FROM customer

    WHERE balance > 10000

  ) LOOP

    UPDATE customer

    SET IsVIP = 'TRUE'

    WHERE customer\_id = rec.customer\_id;

  END LOOP;

  COMMIT;

END;

SELECT customer\_id, name, balance, IsVIP

FROM customer;

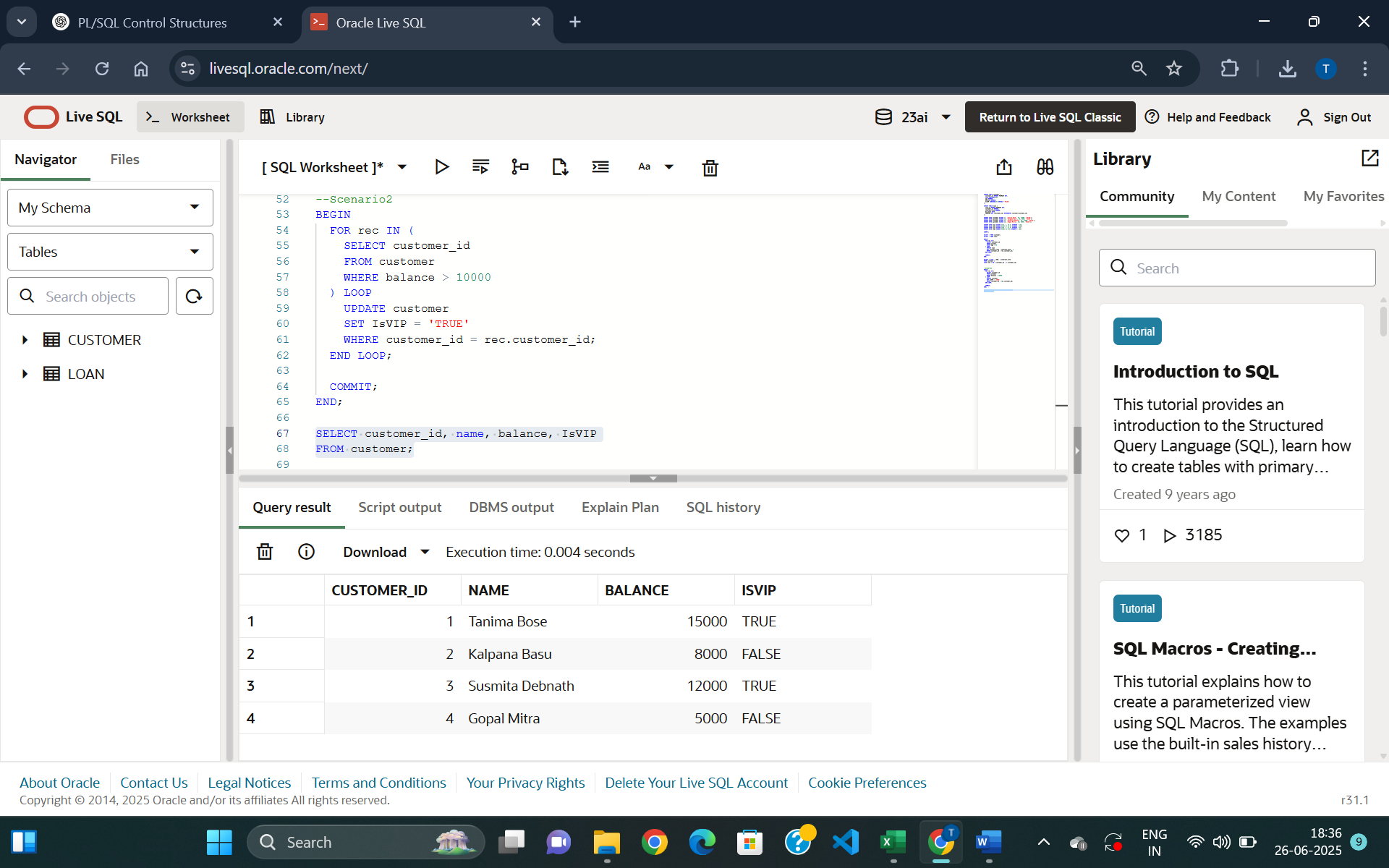


Figure 4: Gives VIP status to customers with balance over 10000

**Exercise 1:**

**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.

--Scenario3

SET SERVEROUTPUT ON;

BEGIN

  FOR rec IN (

    SELECT c.name, c.customer\_id, l.due\_date

    FROM customer c

    JOIN loan l ON c.customer\_id = l.customer\_id

    WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE(

      'Reminder: ' || rec.name ||

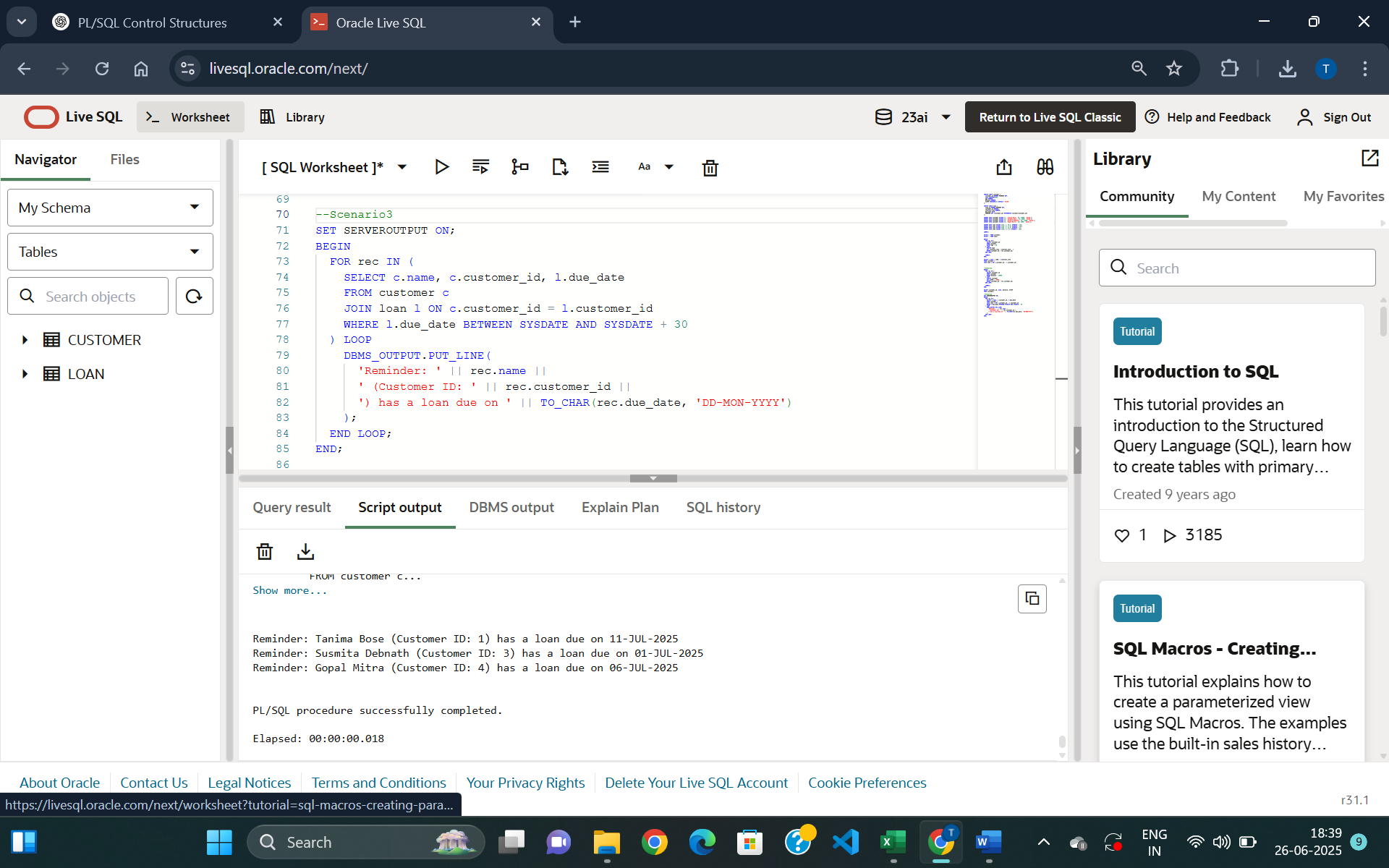
      ' (Customer ID: ' || rec.customer\_id ||

      ') has a loan due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY')

    );

  END LOOP;

END;



*Figure 5: Prints a reminder message for customers whose loan is due in next 30 days.*

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE TABLE savings\_account (

  account\_id NUMBER PRIMARY KEY,

  customer\_id NUMBER,

  balance NUMBER

);

INSERT INTO savings\_account VALUES (1, 101, 10000);

INSERT INTO savings\_account VALUES (2, 102, 20000);

INSERT INTO savings\_account VALUES (3, 103, 15000);

COMMIT;

SELECT \* FROM SAVINGS\_ACCOUNT;

--Scenario1

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR rec IN (SELECT account\_id, balance FROM savings\_account) LOOP

    UPDATE savings\_account

    SET balance = balance + (rec.balance \* 0.01)

    WHERE account\_id = rec.account\_id;

  END LOOP;

  COMMIT;

END;

EXEC ProcessMonthlyInterest;

SELECT \* FROM savings\_account;

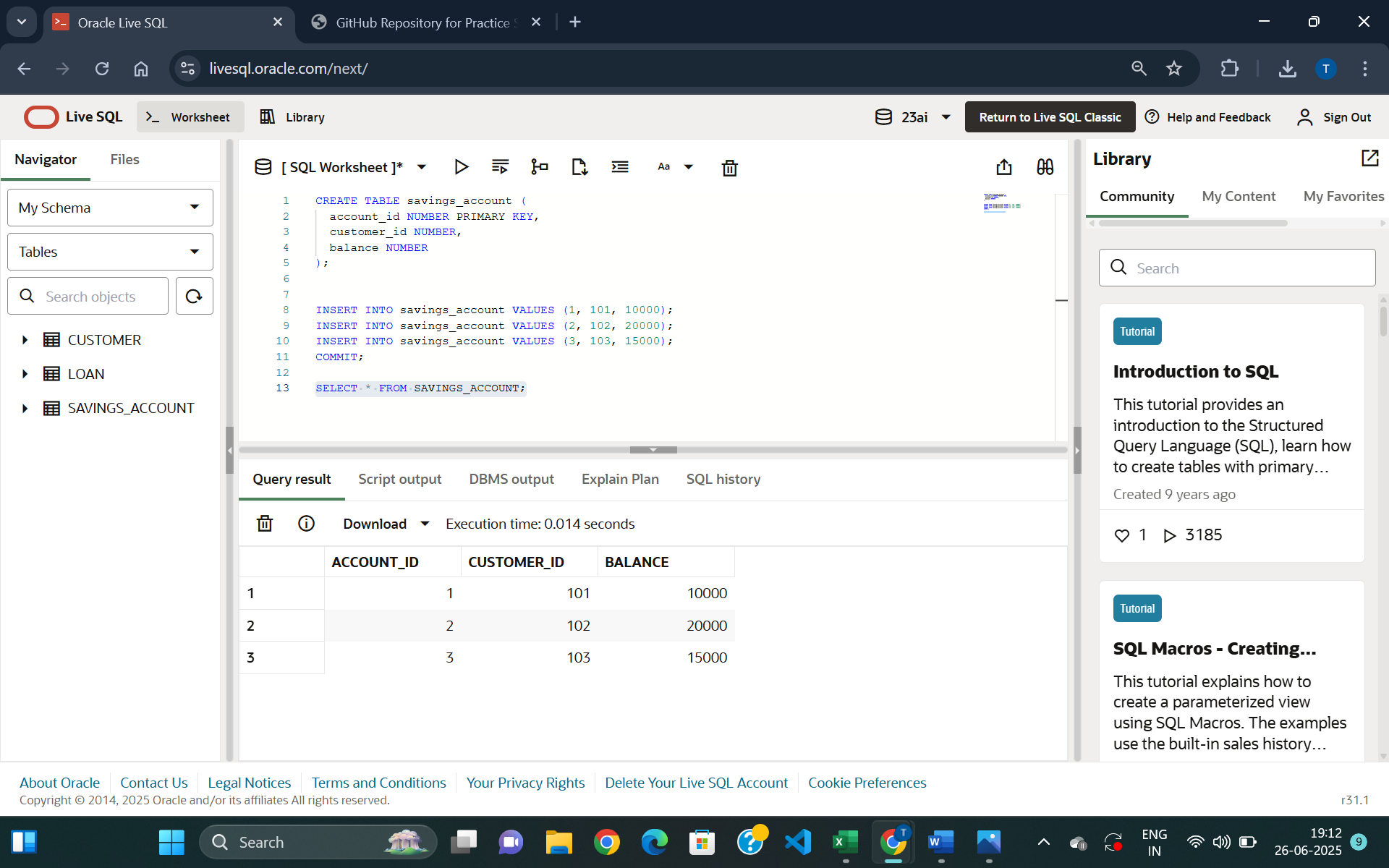


Figure 1: Table savings\_account

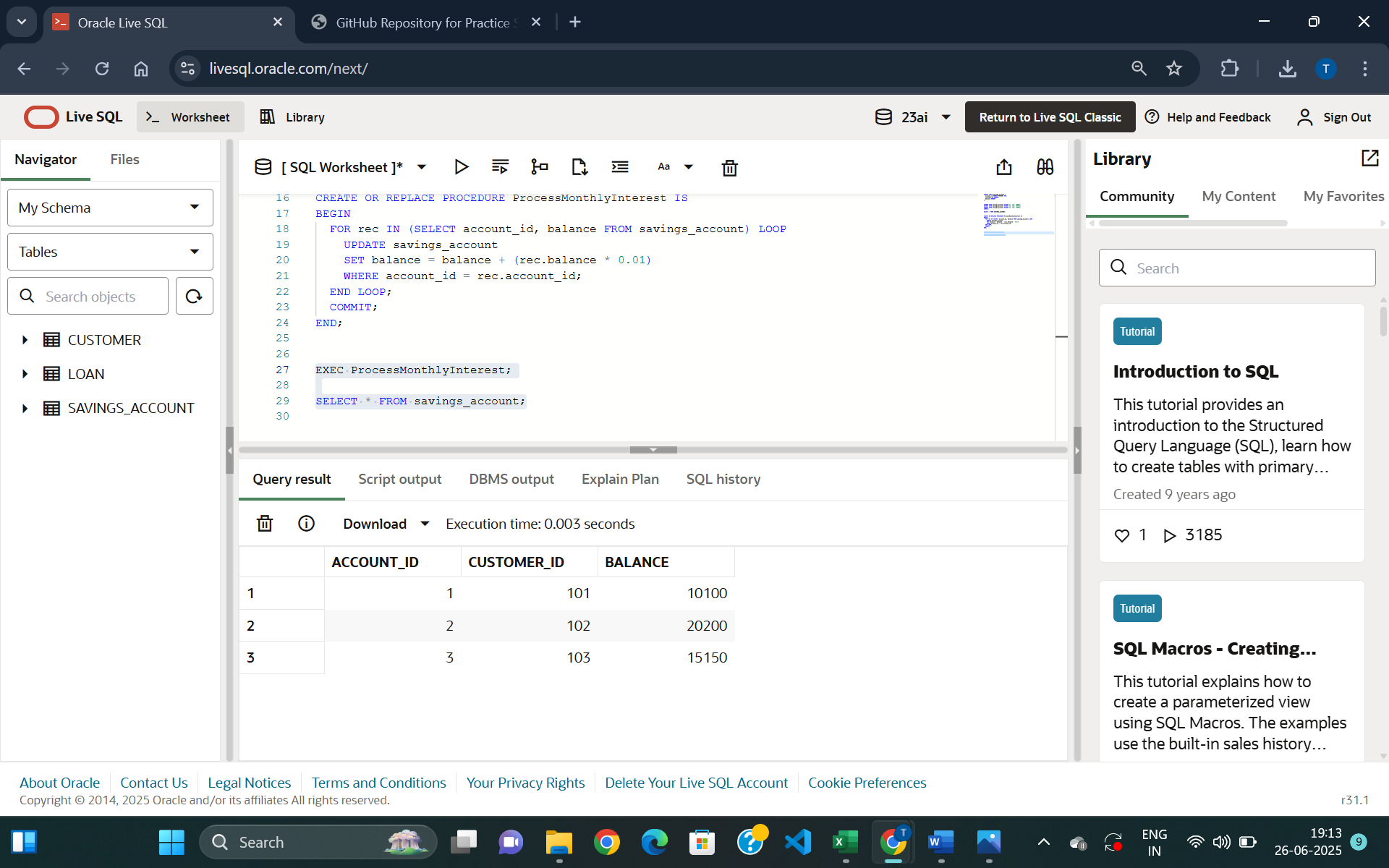


Figure 2: Updated balance of all savings accounts by applying an interest rate of 1% to the current balance

**Scenario 2:**

--Scenario2

CREATE TABLE employee (

  emp\_id NUMBER PRIMARY KEY,

  name VARCHAR2(50),

  department VARCHAR2(30),

  salary NUMBER

);

INSERT INTO employee VALUES (1, 'Nirupam Ghosh', 'IT', 50000);

INSERT INTO employee VALUES (2, 'Sumita Sen', 'HR', 40000);

INSERT INTO employee VALUES (3, 'Sandip Pal', 'IT', 60000);

INSERT INTO employee VALUES (4, 'Srideep Dey', 'Finance', 70000);

COMMIT;

SELECT \* FROM EMPLOYEE;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  dept\_name IN VARCHAR2,

  bonus\_percent IN NUMBER

) IS

BEGIN

  UPDATE employee

  SET salary = salary + (salary \* bonus\_percent / 100)

  WHERE department = dept\_name;

  COMMIT;

END;

EXEC UpdateEmployeeBonus('IT', 10);

SELECT \* FROM employee;

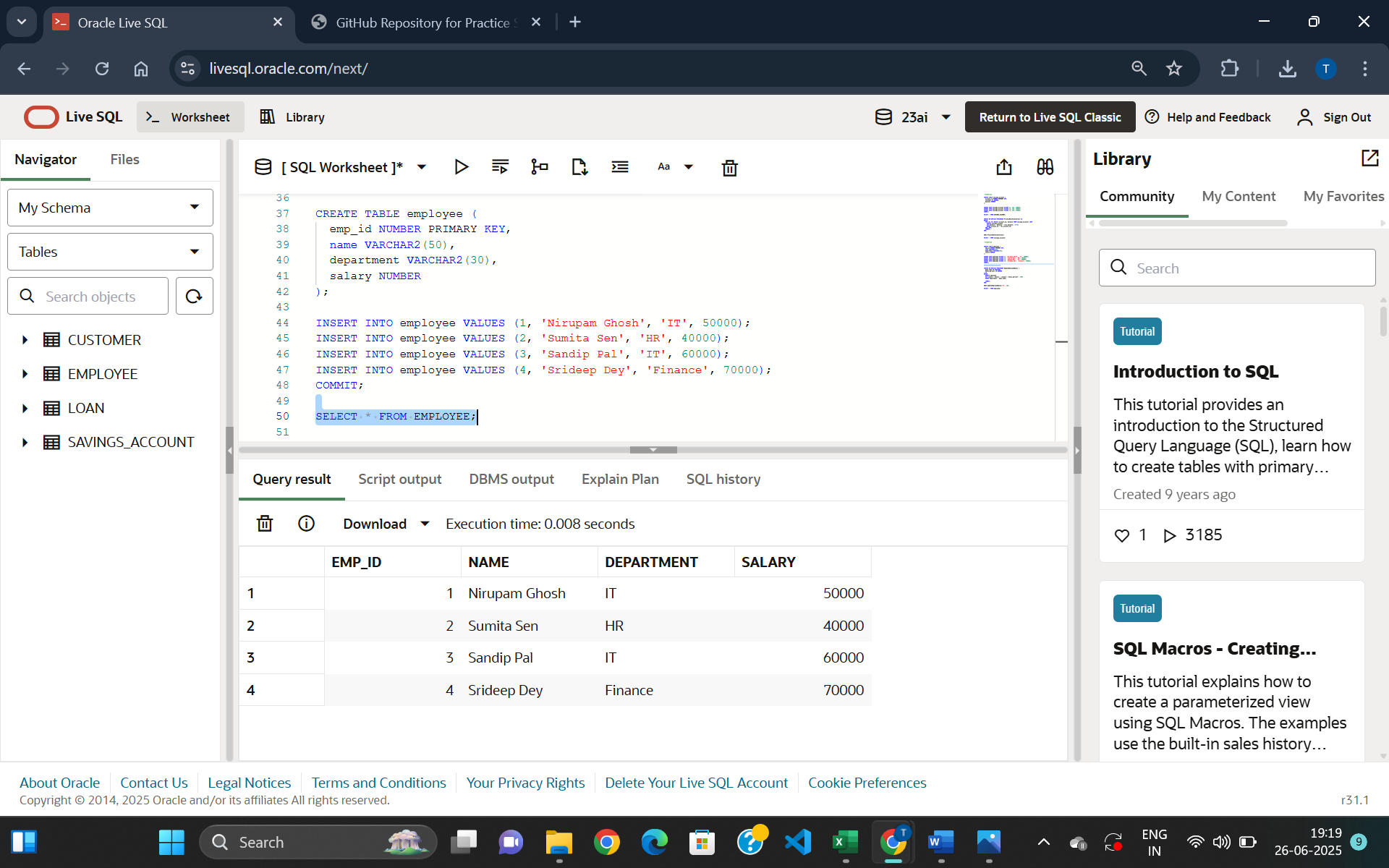
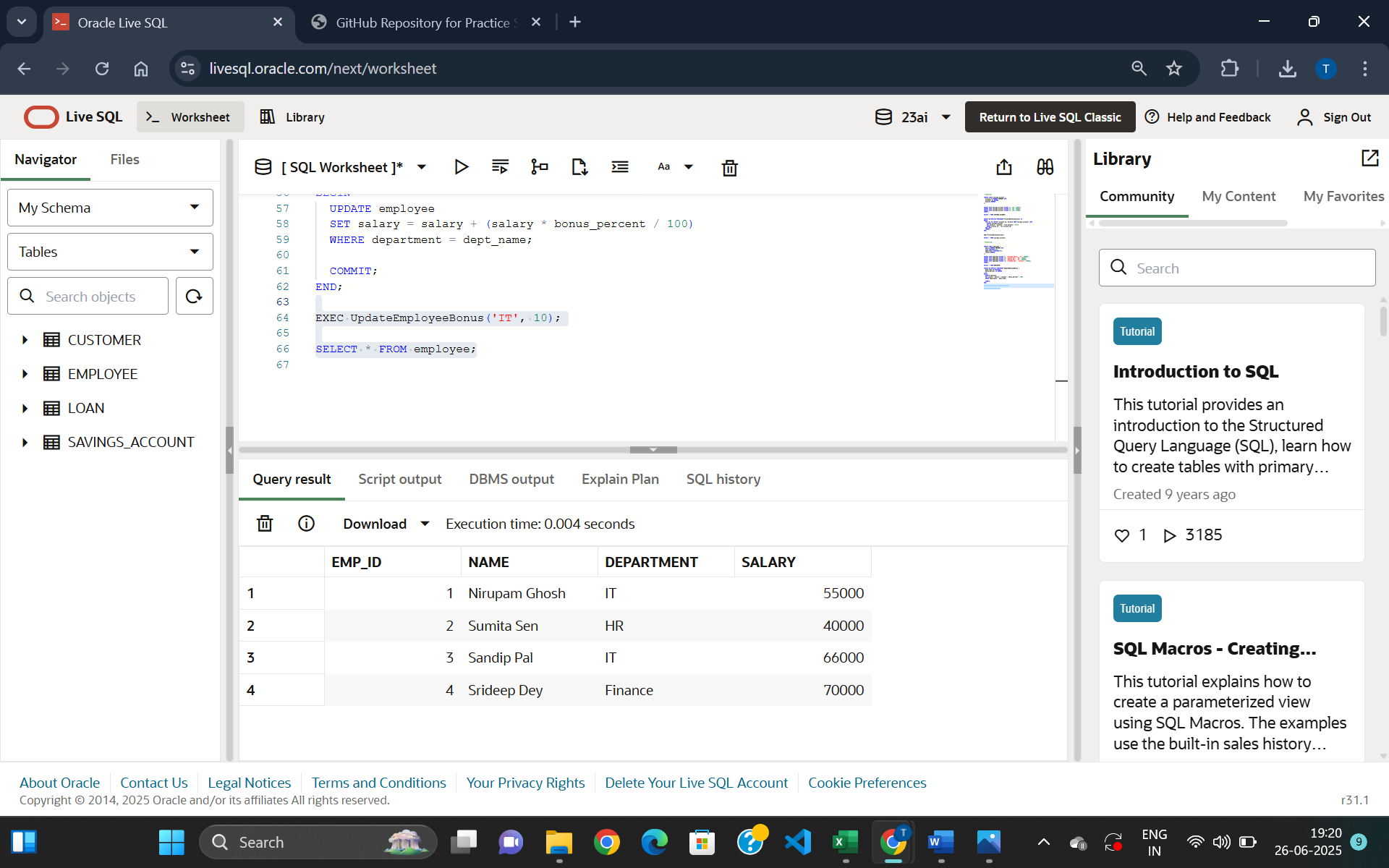


Figure 3: Table Employee Creation



*Figure 4: Updated salary of employees in ‘IT’ department by a bonus percentage*

**Scenario 3:**

--Scenario 3

CREATE TABLE bank\_account (

  account\_id NUMBER PRIMARY KEY,

  customer\_id NUMBER,

  balance NUMBER

);

INSERT INTO bank\_account VALUES (101, 201, 10000);

INSERT INTO bank\_account VALUES (102, 202, 8000);

INSERT INTO bank\_account VALUES (103, 203, 5000);

INSERT INTO bank\_account VALUES (104, 204, 7000);

COMMIT;

SELECT \* FROM bank\_account;

CREATE OR REPLACE PROCEDURE TransferFunds (

  from\_account IN NUMBER,

  to\_account IN NUMBER,

  amount IN NUMBER

) IS

  insufficient\_balance EXCEPTION;

BEGIN

  DECLARE

    source\_balance NUMBER;

  BEGIN

    SELECT balance INTO source\_balance FROM bank\_account WHERE account\_id = from\_account;

    IF source\_balance < amount THEN

      RAISE insufficient\_balance;

    END IF;

    UPDATE bank\_account

    SET balance = balance - amount

    WHERE account\_id = from\_account;

    UPDATE bank\_account

    SET balance = balance + amount

    WHERE account\_id = to\_account;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || amount || ' transferred from account ' || from\_account || ' to account ' || to\_account);

  EXCEPTION

    WHEN insufficient\_balance THEN

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

    WHEN NO\_DATA\_FOUND THEN

      DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Invalid account ID');

  END;

END;

SET SERVEROUTPUT ON;

EXEC TransferFunds(101, 102, 2000);

SELECT \* FROM bank\_account;

--Insufficient Balance

EXEC TransferFunds(103, 104, 6000);

--Invalid Account Id

EXEC TransferFunds(999, 102, 1000);

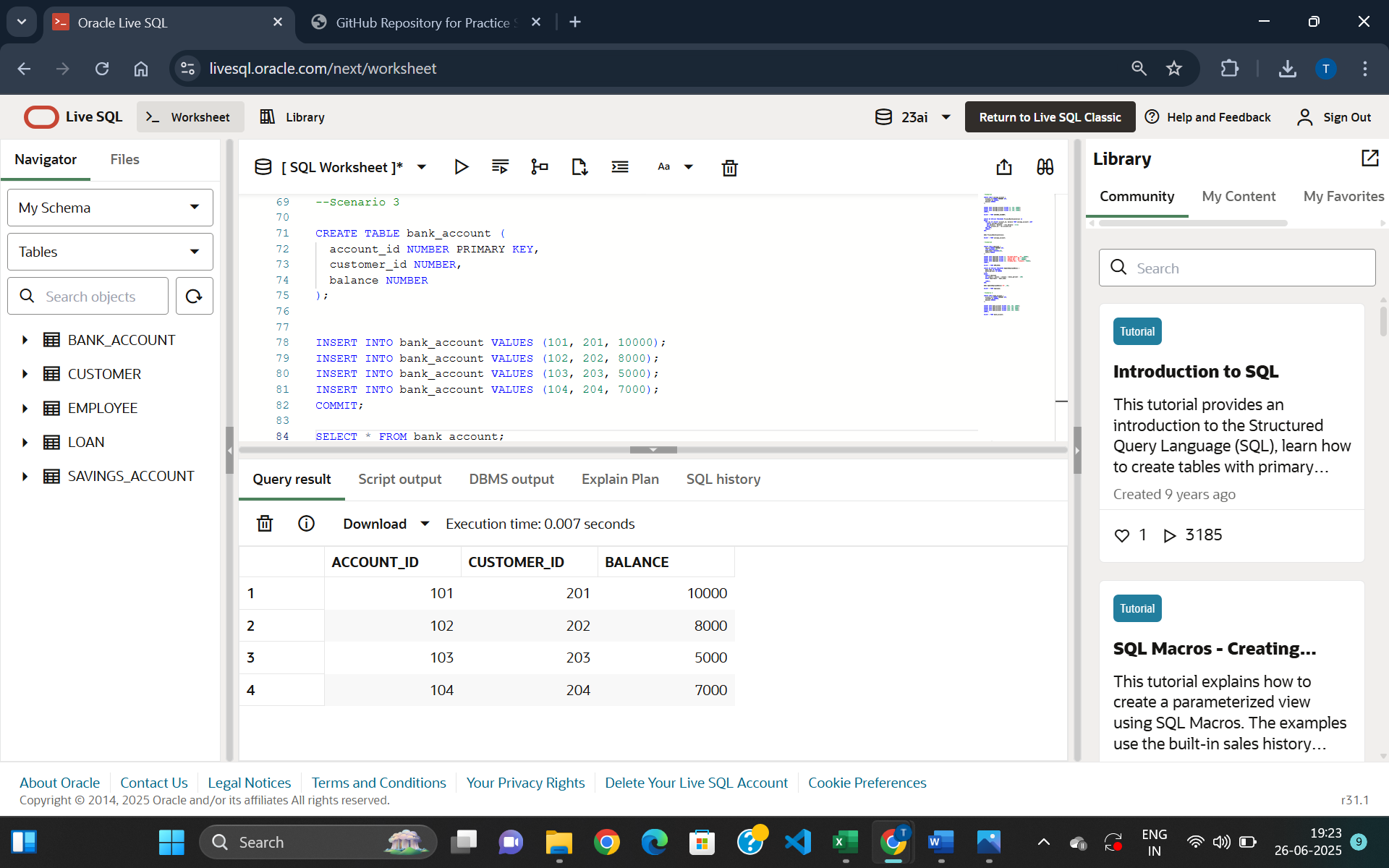
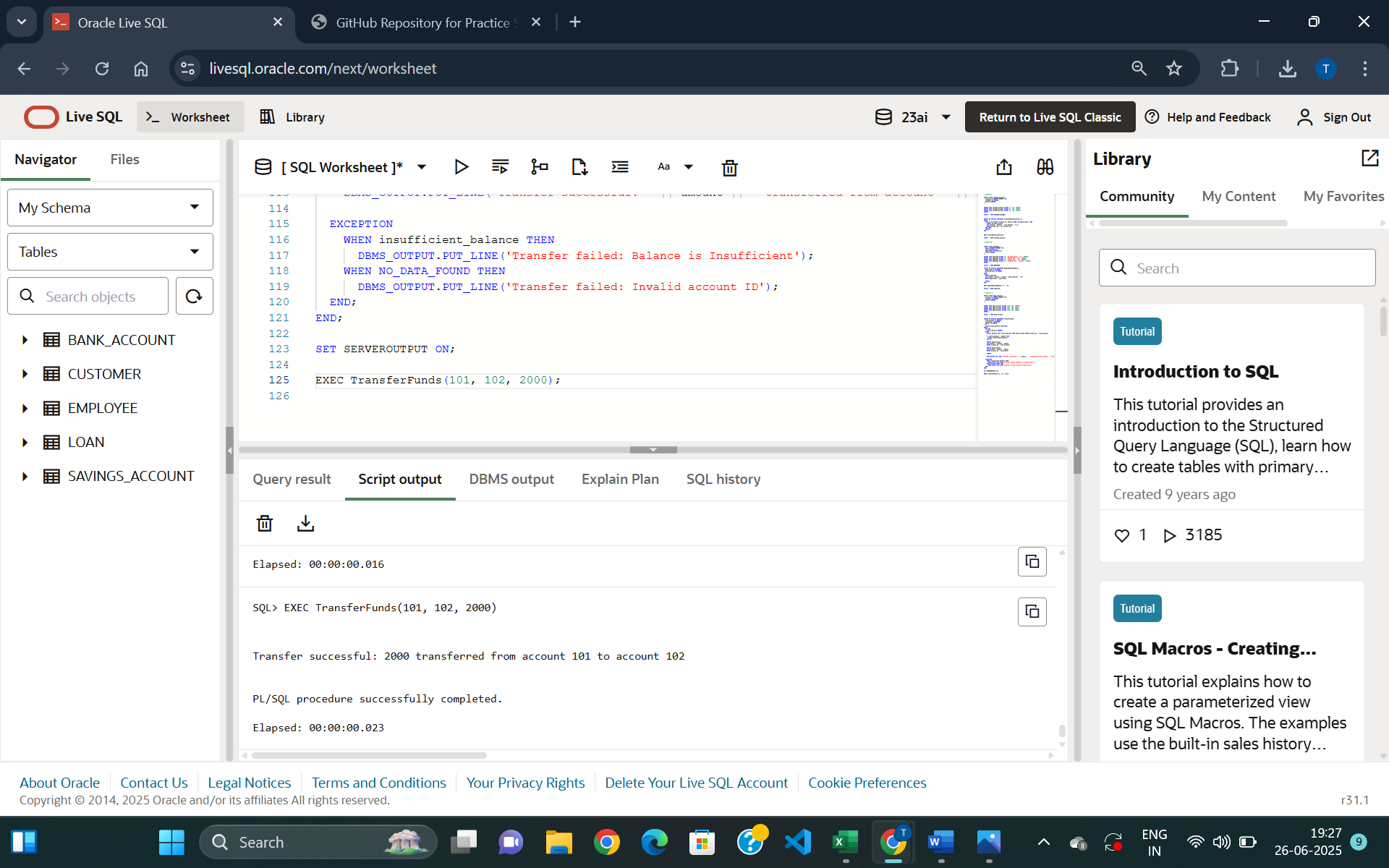


Figure 5: Table bank\_account creation

**Success Cases:**



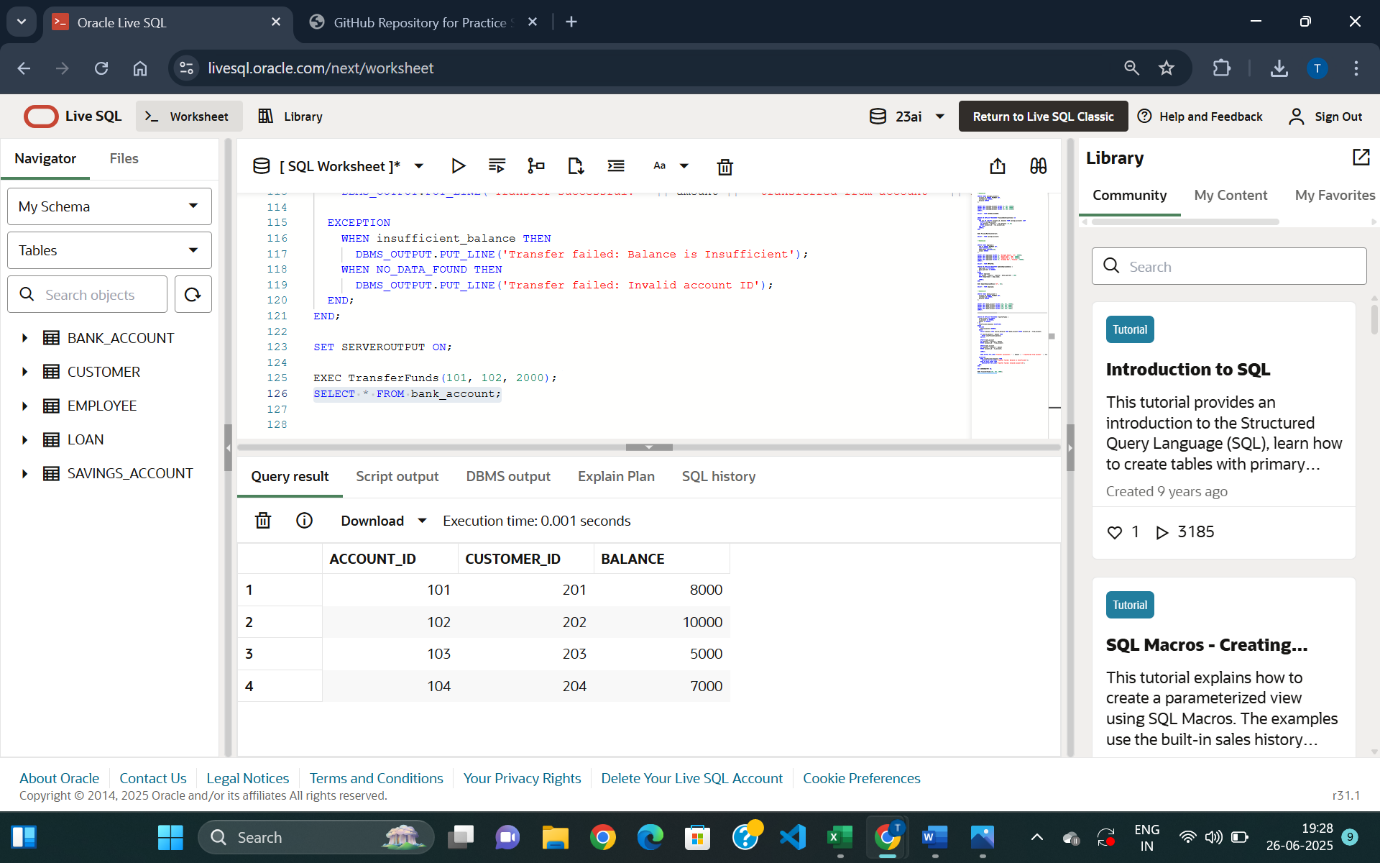


Figure 7: Transferred amount 2000 from account 101 to account 102

Figure 6: Stored Procedure executed

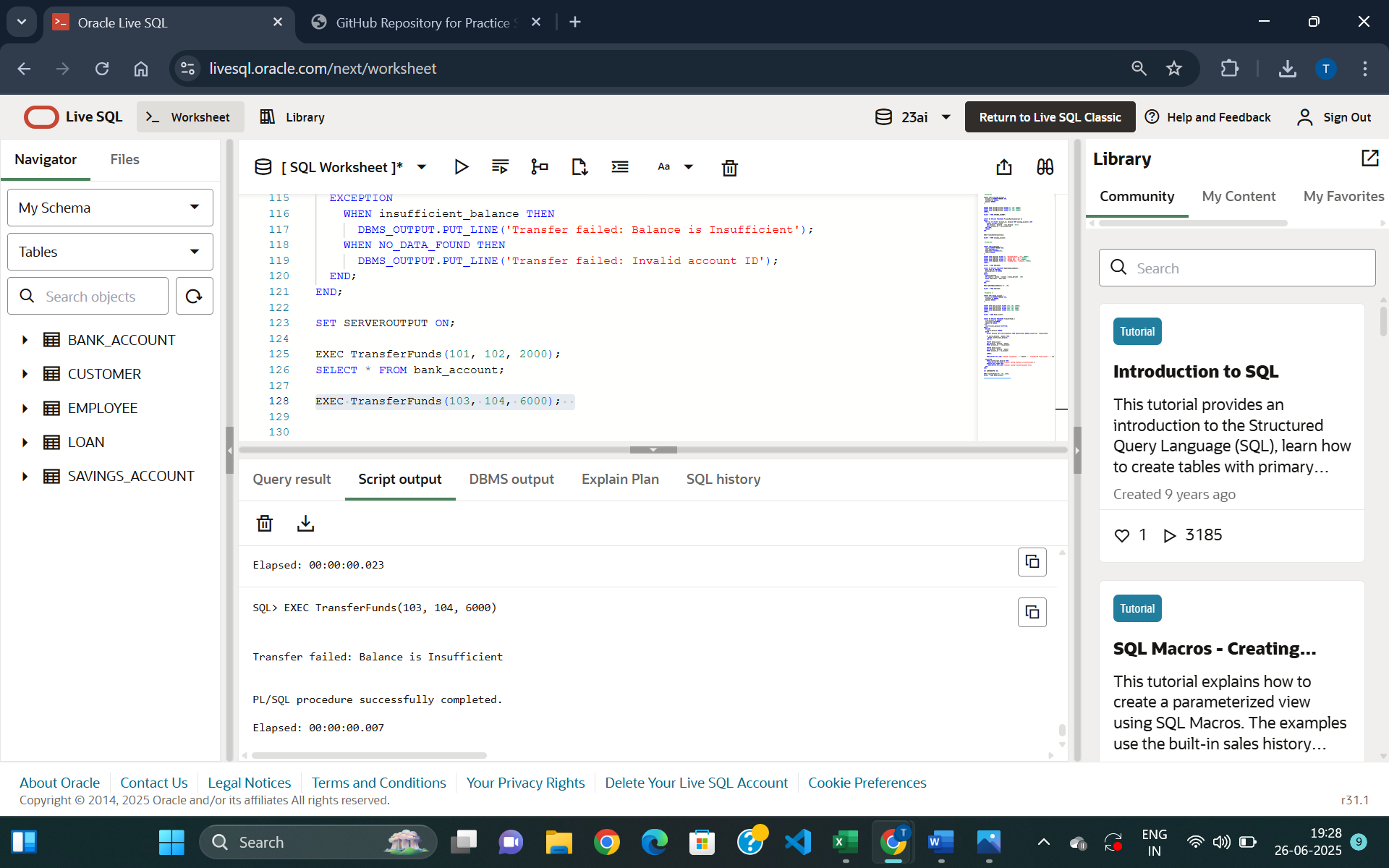
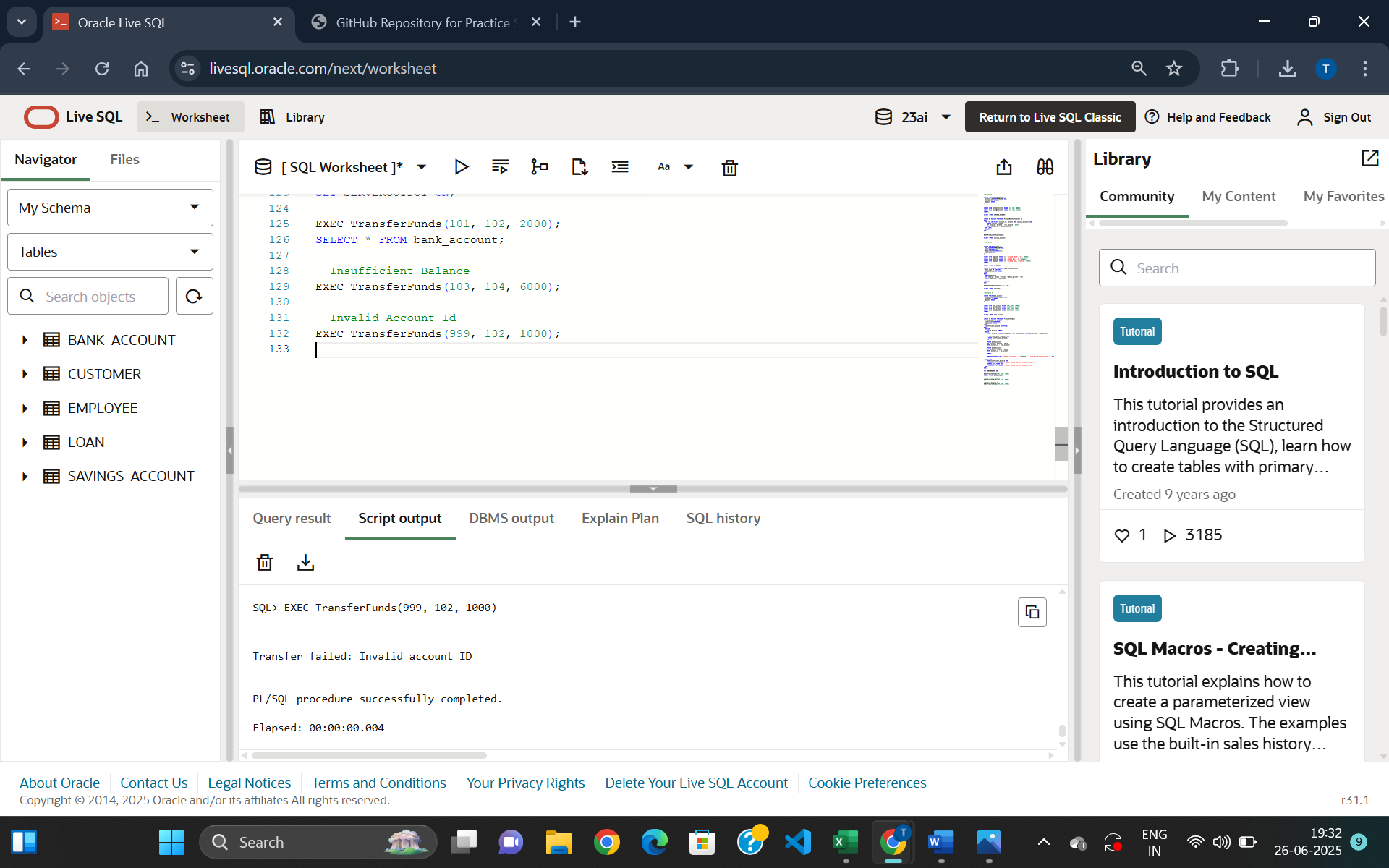
**Failure Cases:**

Figure 8:

Insufficient Balance Case since account 103 has balance less than 6000

*Figure 9:*

*Invalid Account ID case since account 999 does not exist*