Name: INIGASHREE N S

College: Kongu Engineering College

**PLSQL\_EXERCISES**

# Exercise 1: Control Structures

## ****1. Understanding Control Structures in PL/SQL****

PL/SQL (Procedural Language/SQL) supports **control structures** such as **loops, conditional checks**, and **procedure execution**. These allow developers to implement logic like updating customer records, applying discounts, or automating reminders.

In a **banking system**, control structures can be used to:

* Process customers based on conditions (e.g., age, balance)
* Apply operations in bulk (e.g., discounts, reminders)
* Automate periodic tasks (e.g., interest application, alerts)

**2. Table Setup**

Set Up Tables and Insert Sample Data

🡪Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10,2),

IsVIP CHAR(1)

);

🡪Loans Table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

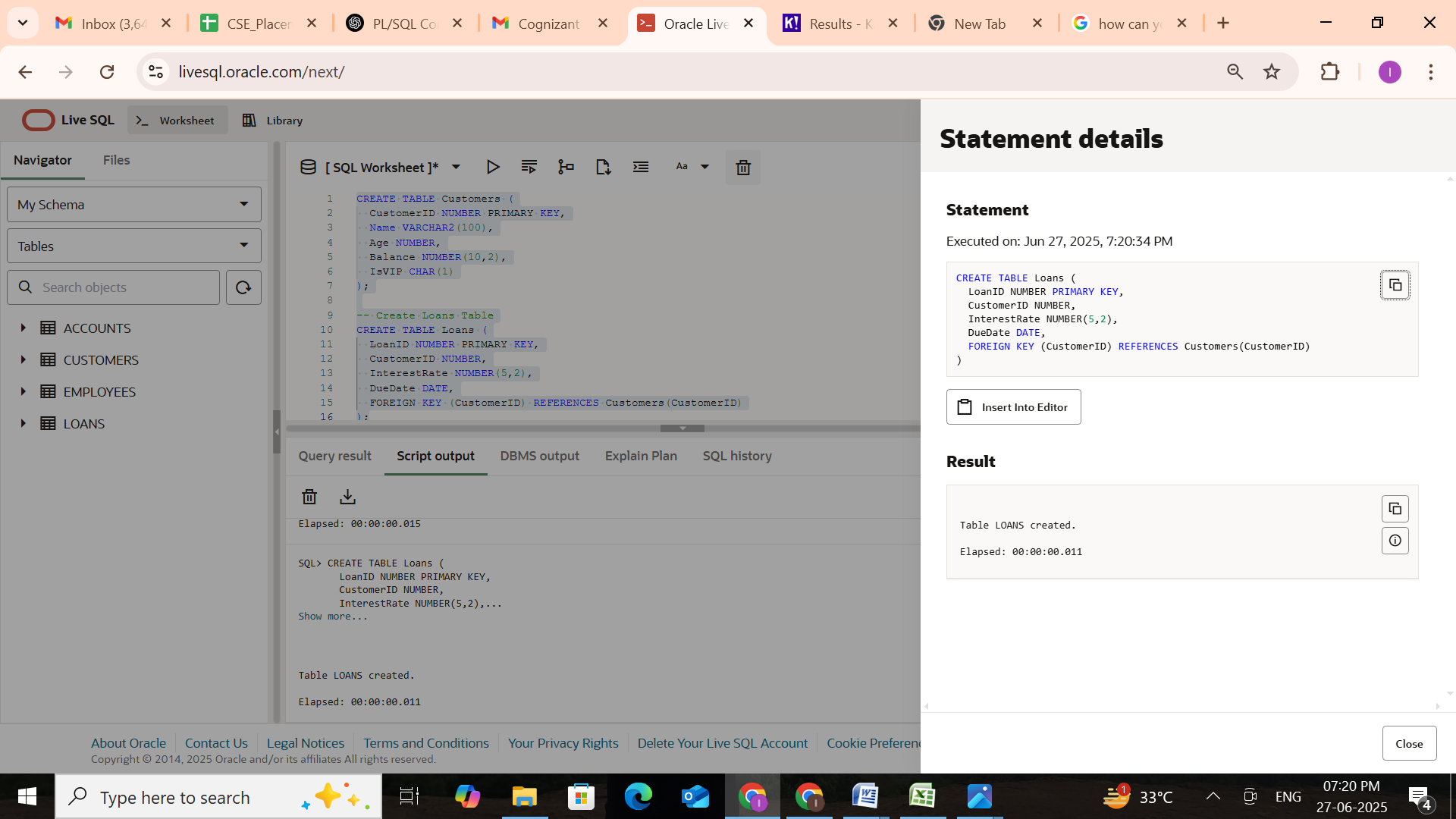
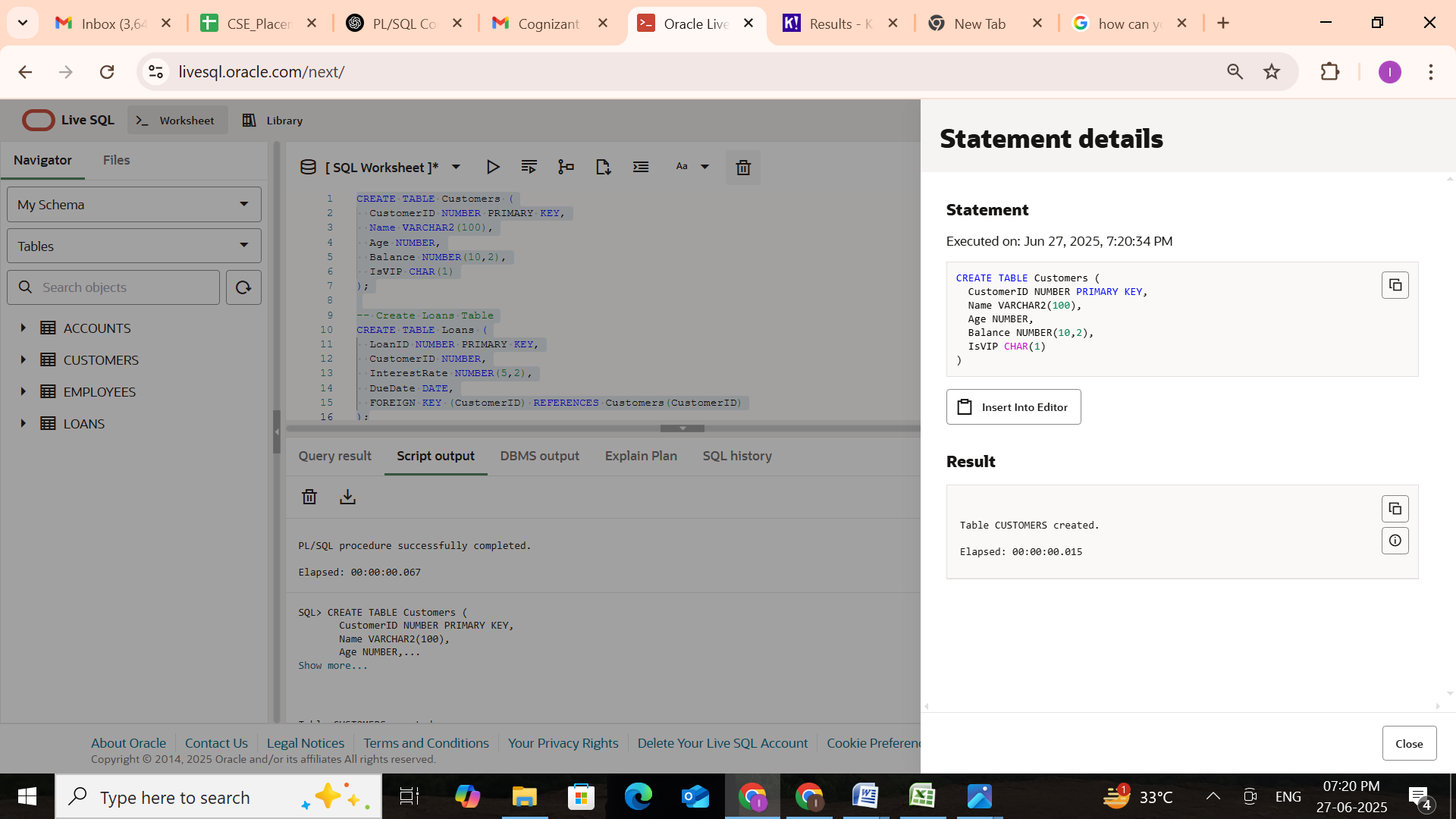
CustomerID NUMBER,

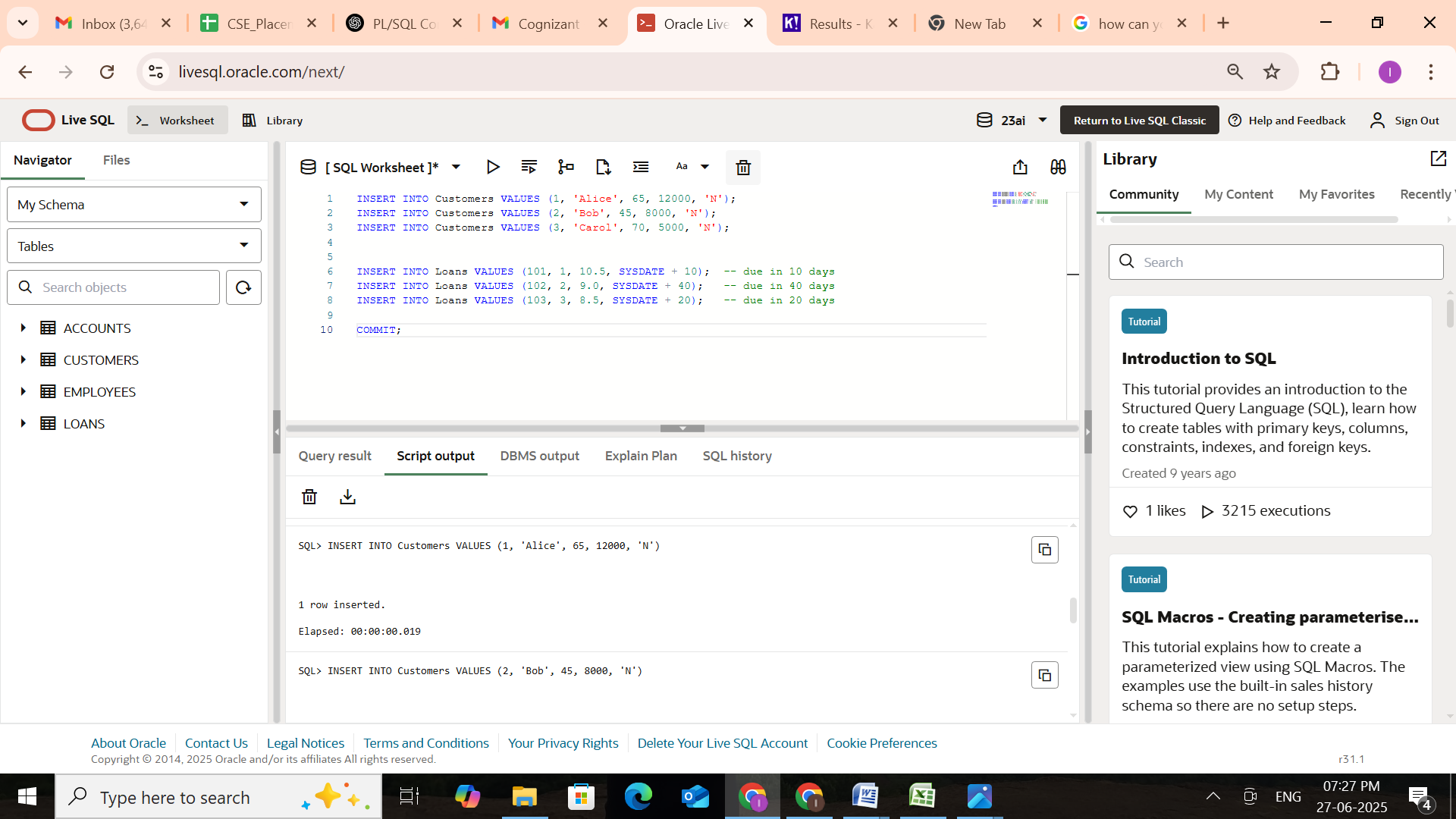
InterestRate NUMBER(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);



****

**3.Implementation**

**Scenario 1 – Apply 1% Discount for Age > 60**

BEGIN

  FOR cust IN (SELECT CustomerID FROM Customers WHERE Age > 60) LOOP

    UPDATE Loans

    SET InterestRate = InterestRate - 1

    WHERE CustomerID = cust.CustomerID;

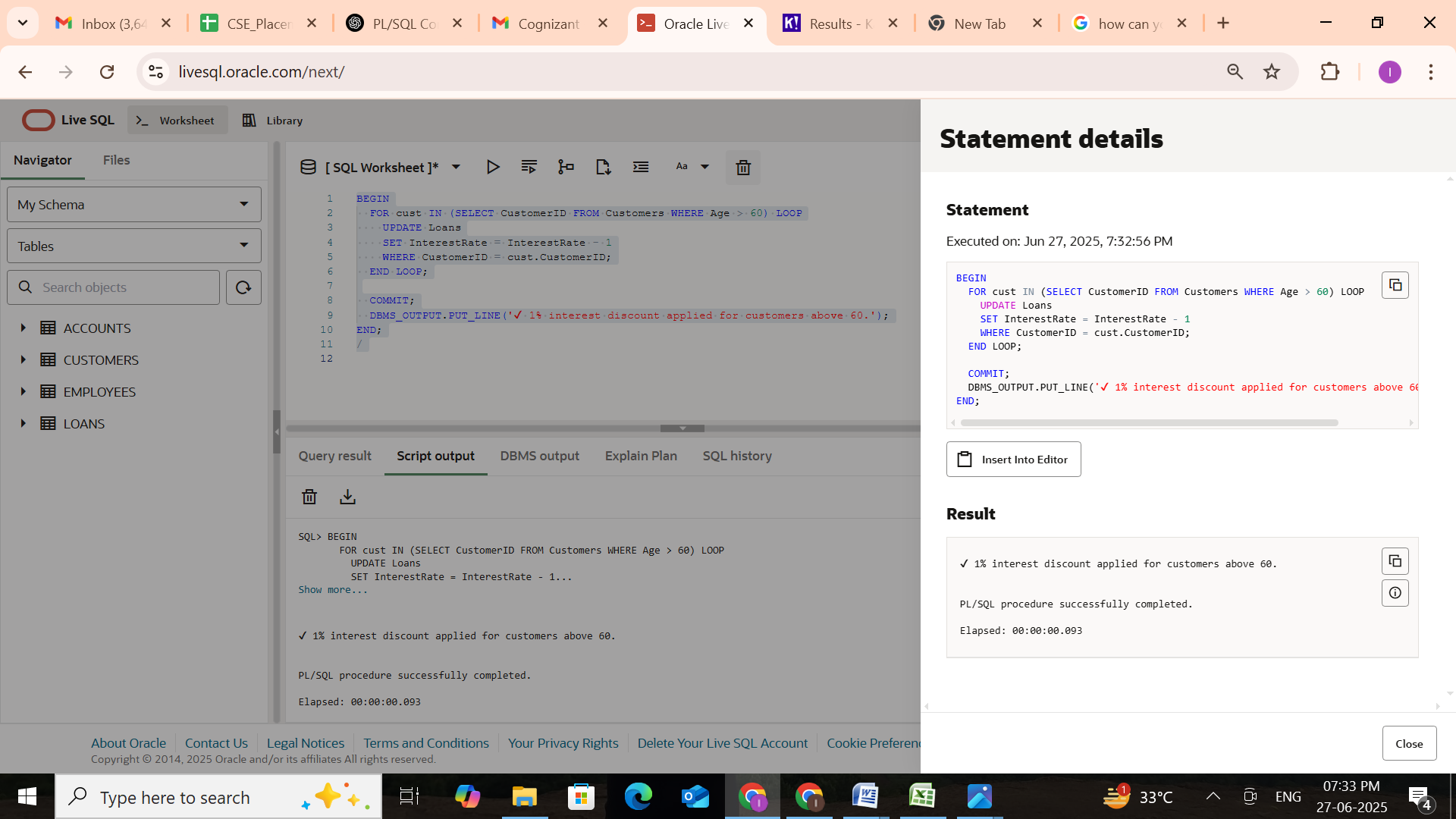
  END LOOP;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('✔ 1% interest discount applied for customers above 60.');

END;

/



**Scenario 2 – Promote to VIP (Balance > 10000)**

BEGIN

  FOR cust IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

    UPDATE Customers

    SET IsVIP = 'Y'

    WHERE CustomerID = cust.CustomerID;

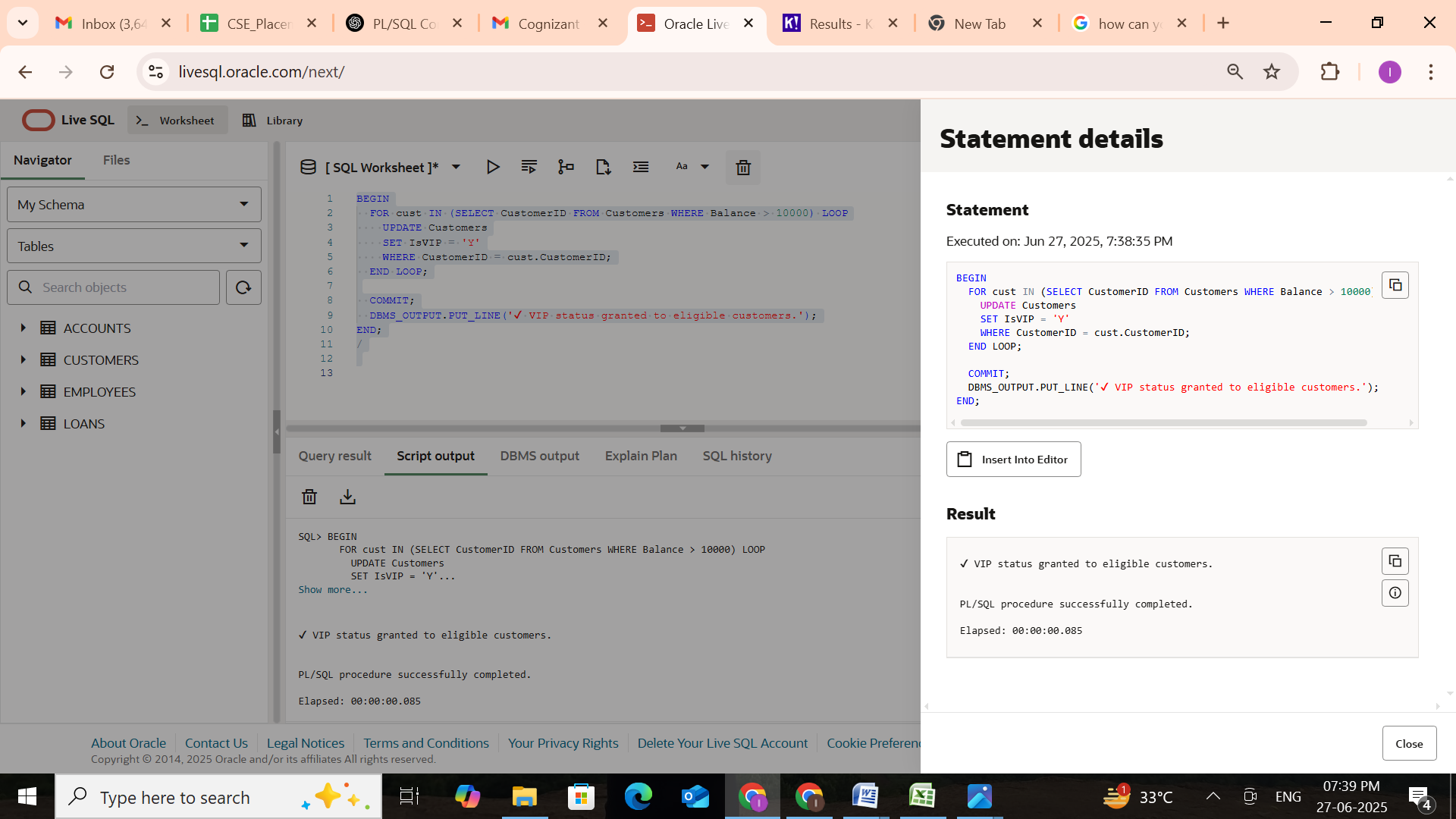
  END LOOP;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('✔ VIP status granted to eligible customers.');

END;

/



**Scenario 3 – Print Loan Reminders for Next 30 Days**

BEGIN

  FOR loan\_rec IN (

    SELECT l.LoanID, l.DueDate, c.Name

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE('📢 Reminder: Loan ID ' || loan\_rec.LoanID ||

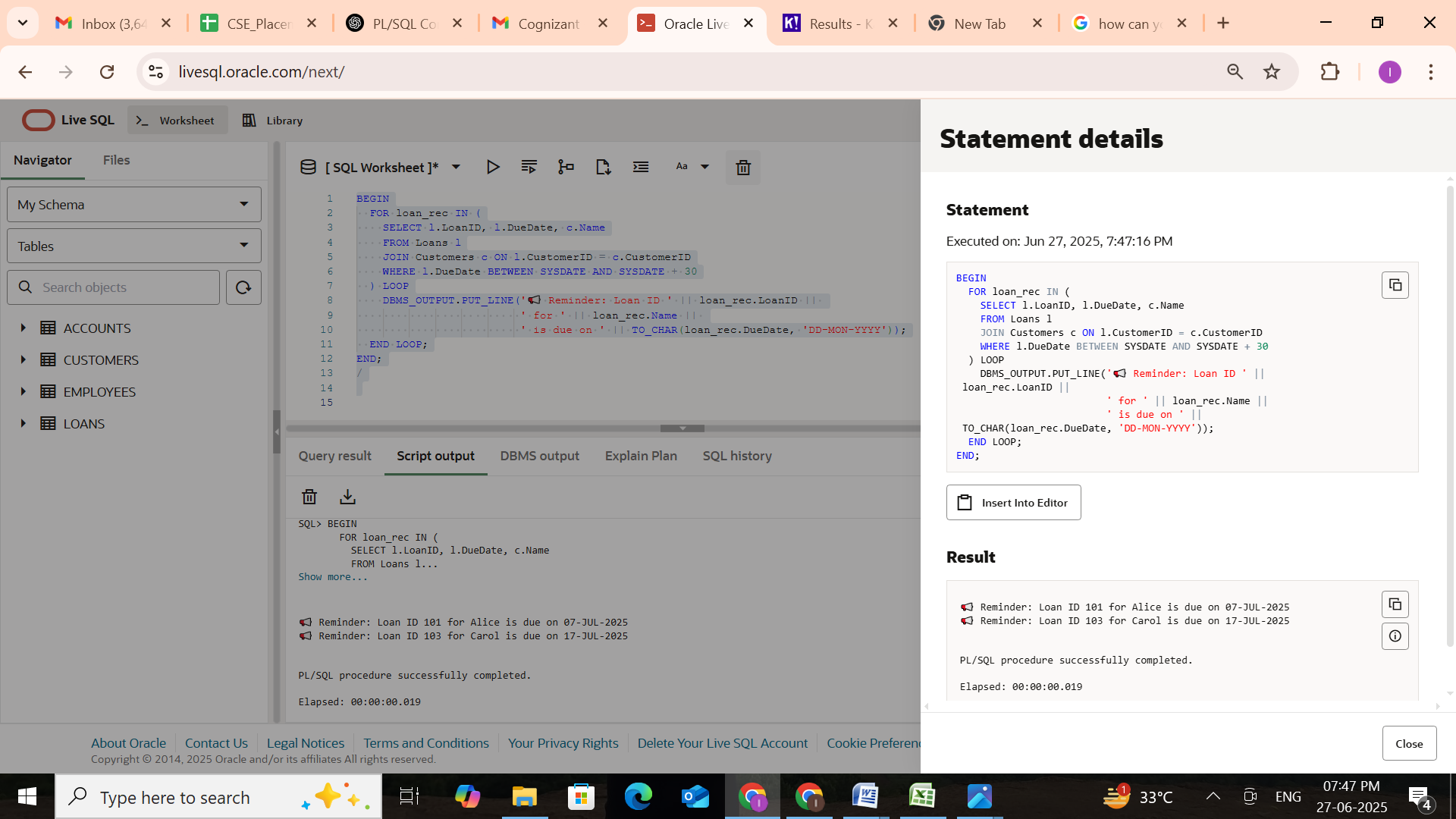
                         ' for ' || loan\_rec.Name ||

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

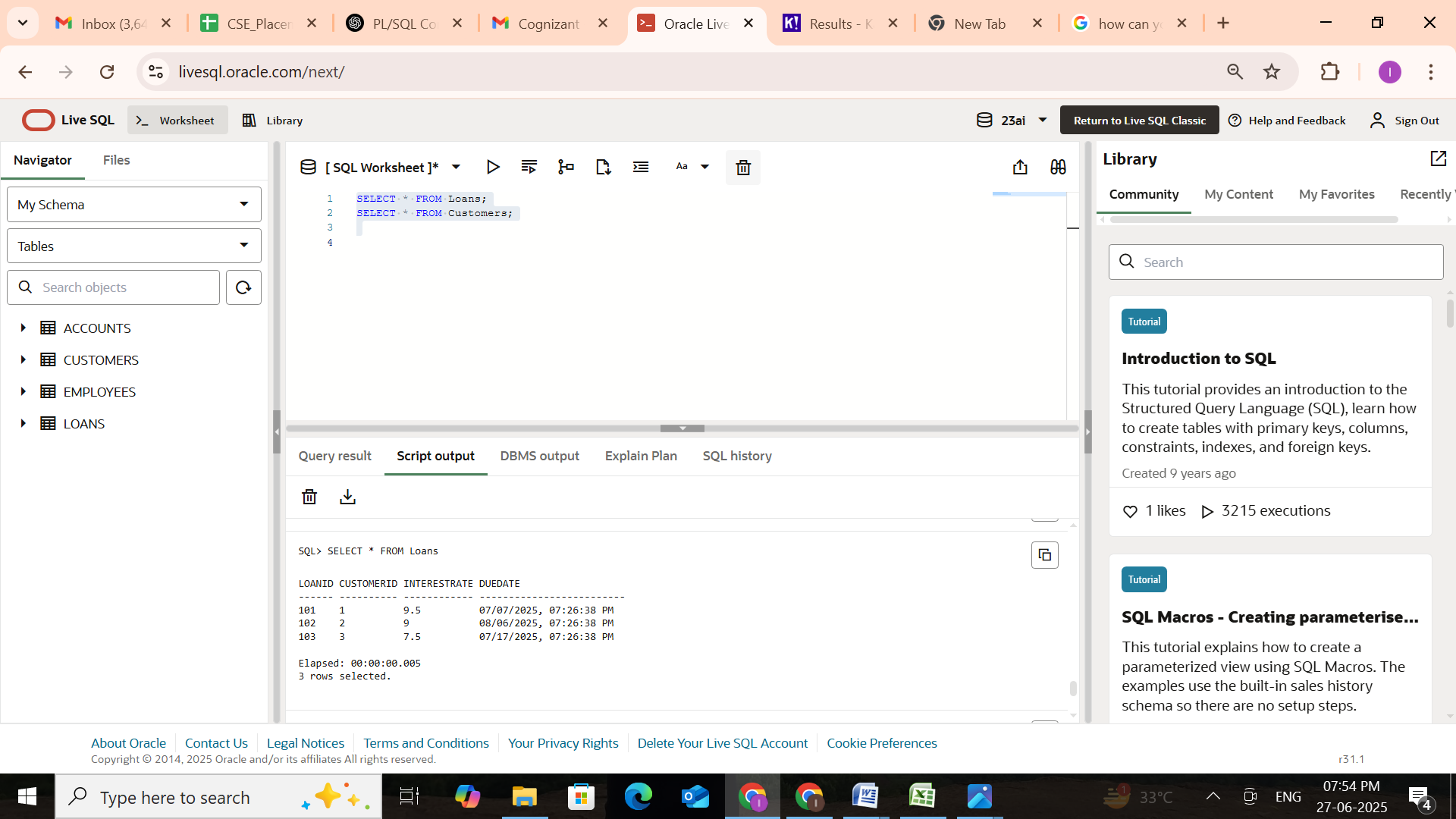
  END LOOP;

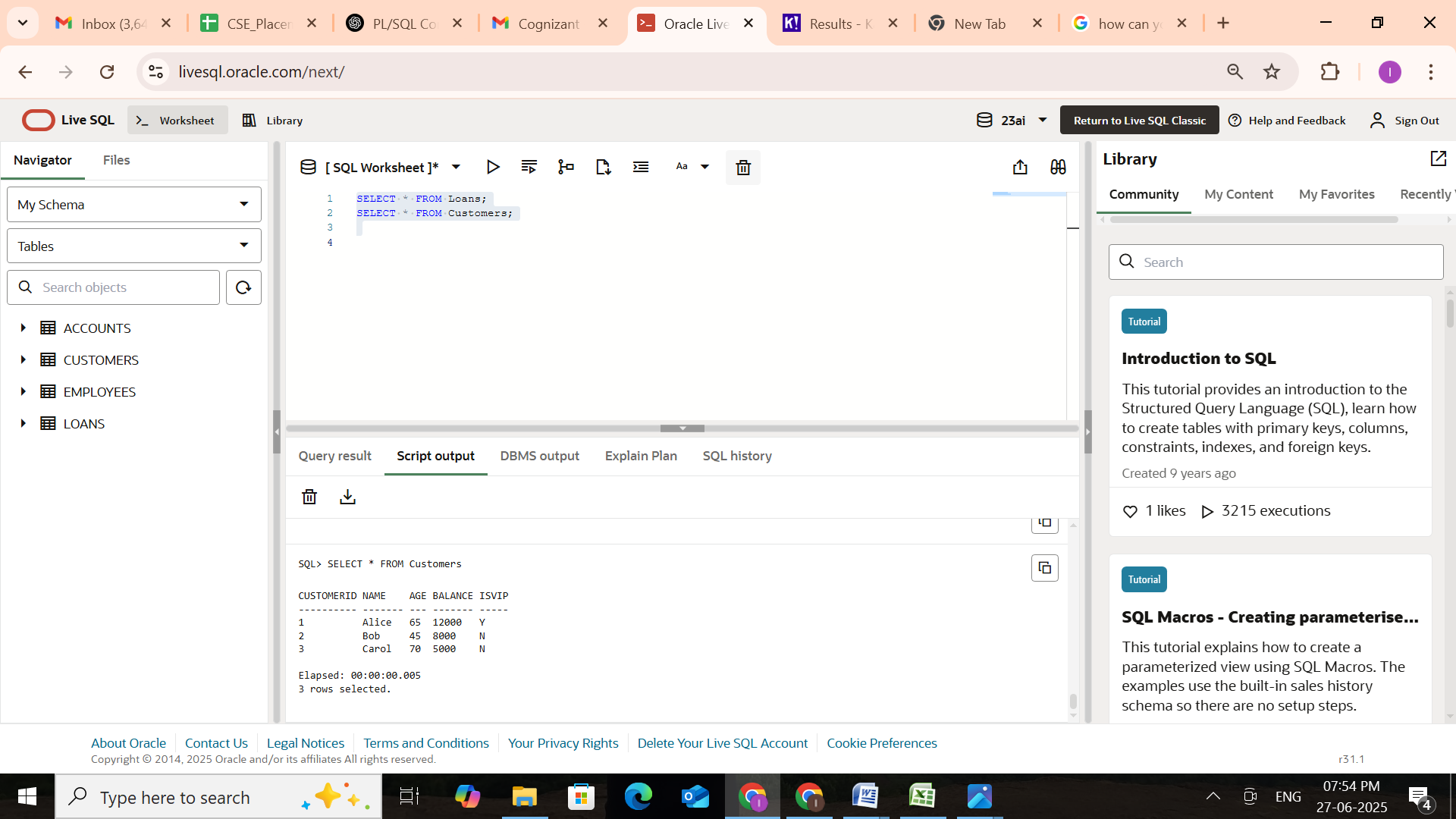
END;

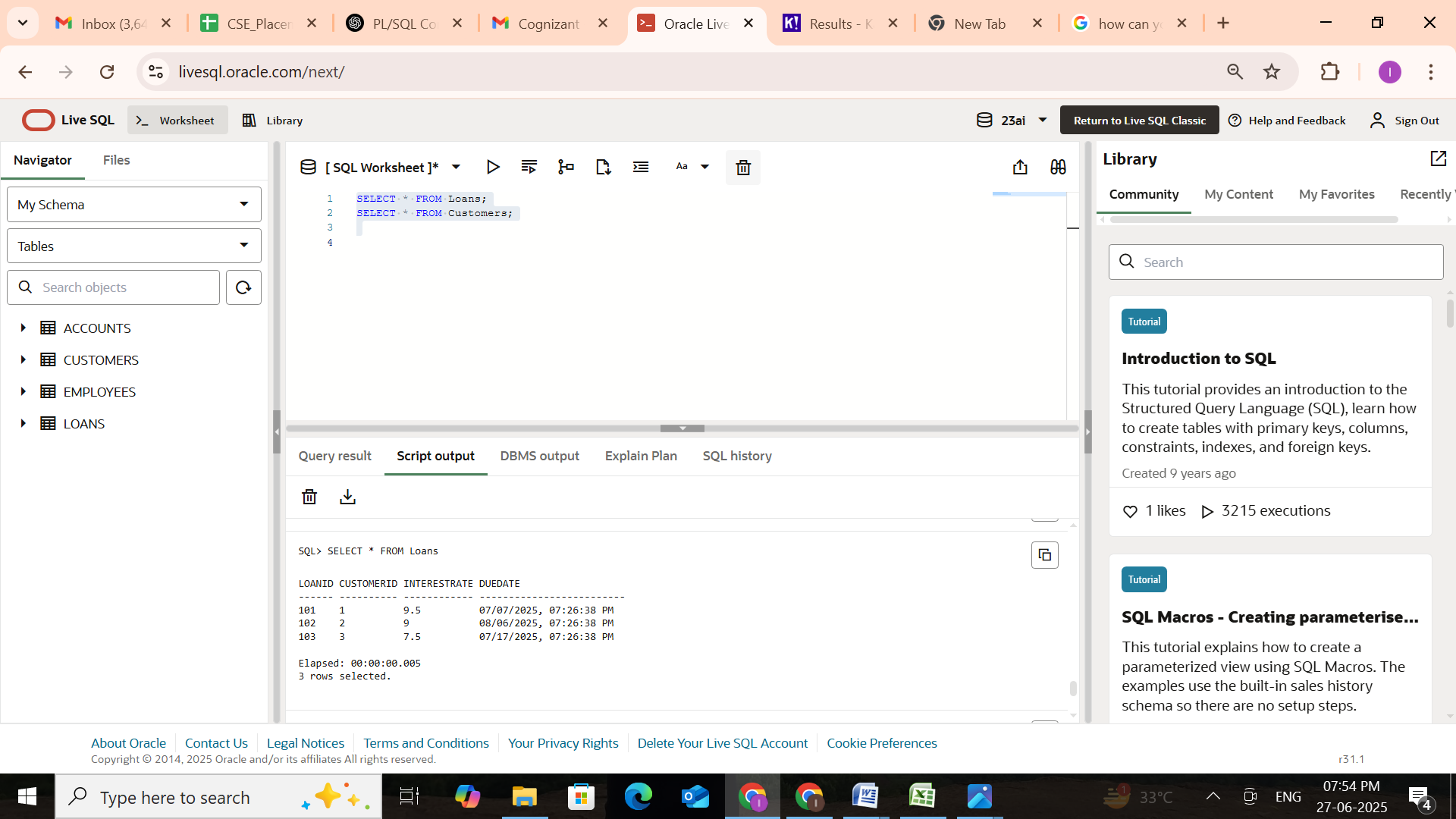
/



**Final Result:**

****





**Exercise 3: Stored Procedures**

## ****1. Understanding Stored Procedures in PL/SQL****

PL/SQL supports **stored procedures**, which are named blocks of code that can be saved and reused. These procedures allow developers to **encapsulate business logic** and execute complex tasks in a single call.

In a **banking system**, stored procedures are highly valuable for:

* **Automating repetitive operations** (e.g., interest calculation)
* **Applying business rules** (e.g., performance-based bonuses)
* **Handling transactional tasks securely** (e.g., fund transfers)

They improve **code reusability**, **security**, and **performance** in database-driven applications.

**2. Table Setup:**

Set Up Tables and Insert Sample Data

🡪Accounts Table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER(10, 2)

);

🡪Employees Table

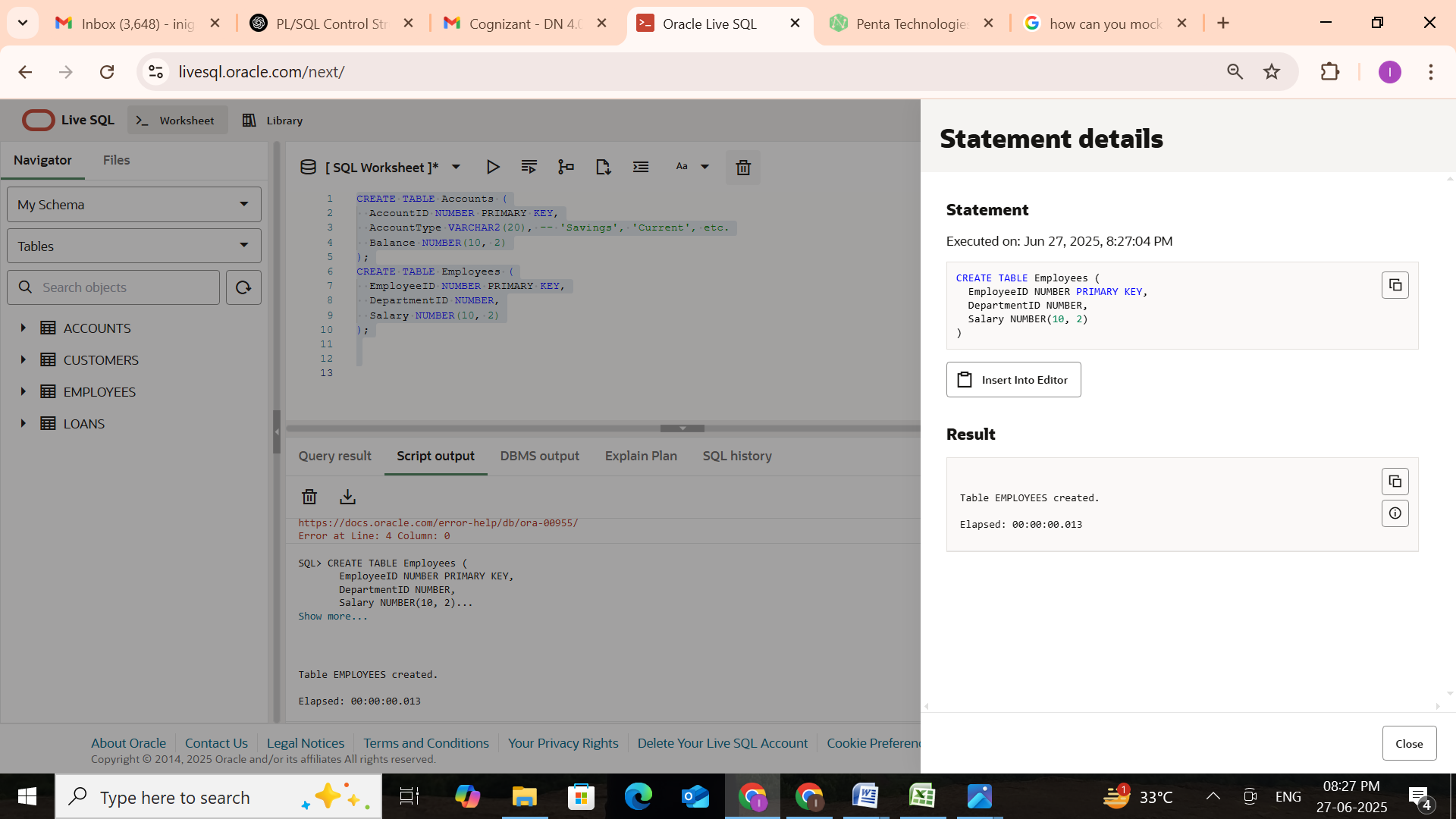
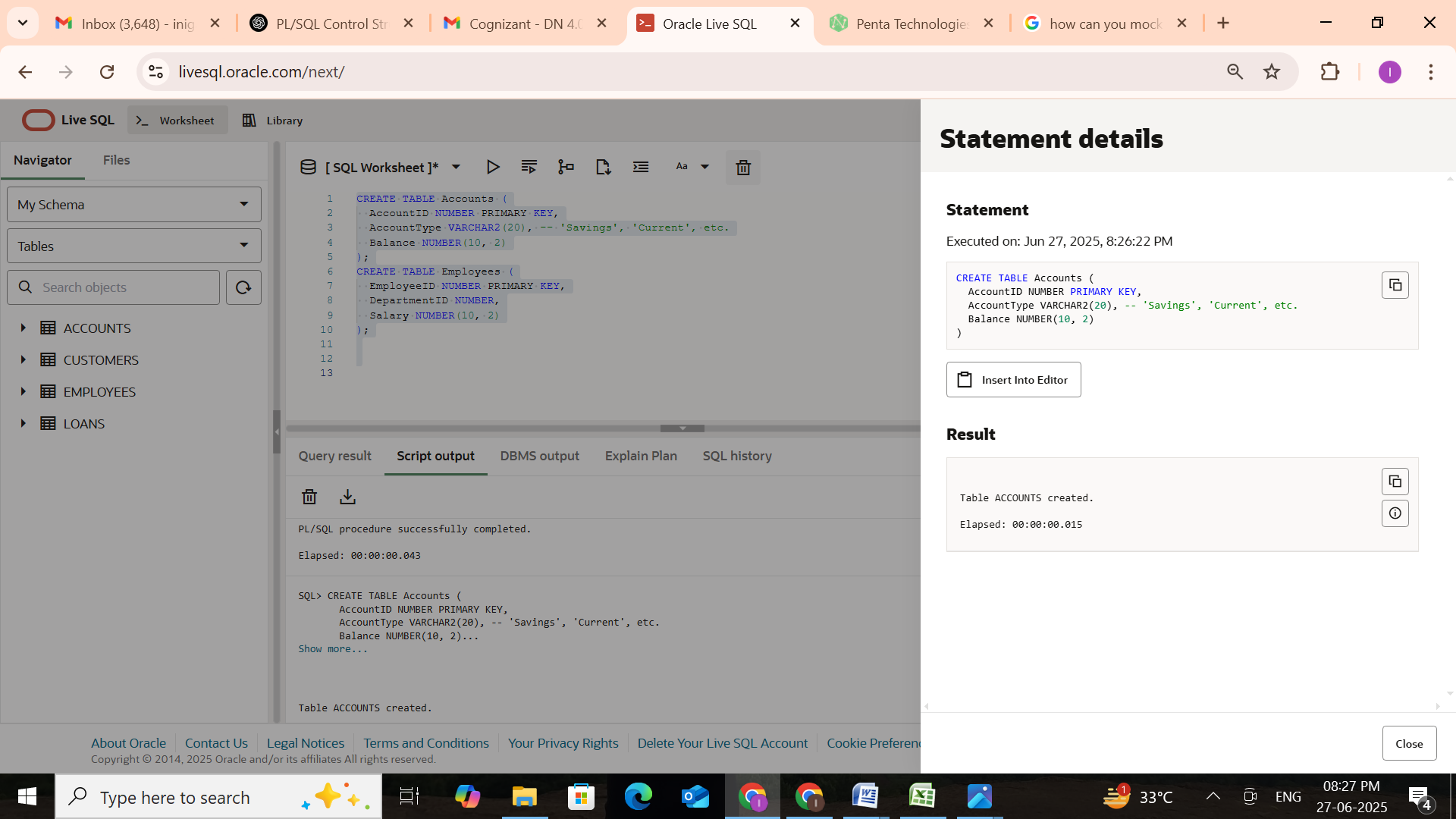
CREATE TABLE Employees (

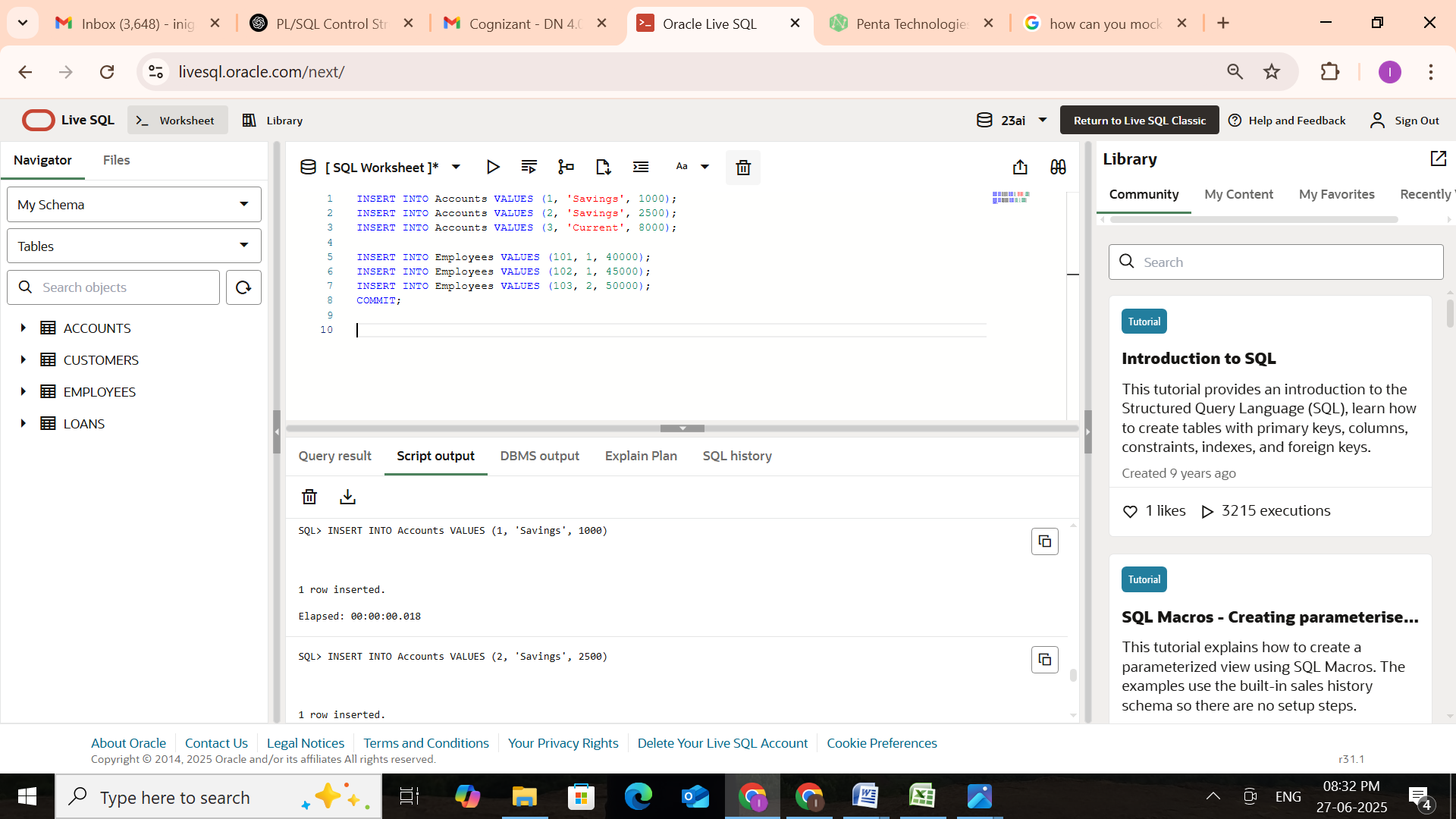
EmployeeID NUMBER PRIMARY KEY,

DepartmentID NUMBER,

Salary NUMBER(10, 2)

);

****

****

**3. Implementation:**

**Scenario 1: Process Monthly Interest for Savings Accounts**

Step 1: Create Stored Procedure and Apply **1% interest** to all **savings accounts.**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

  UPDATE Accounts

  SET Balance = Balance + (Balance \* 0.01)

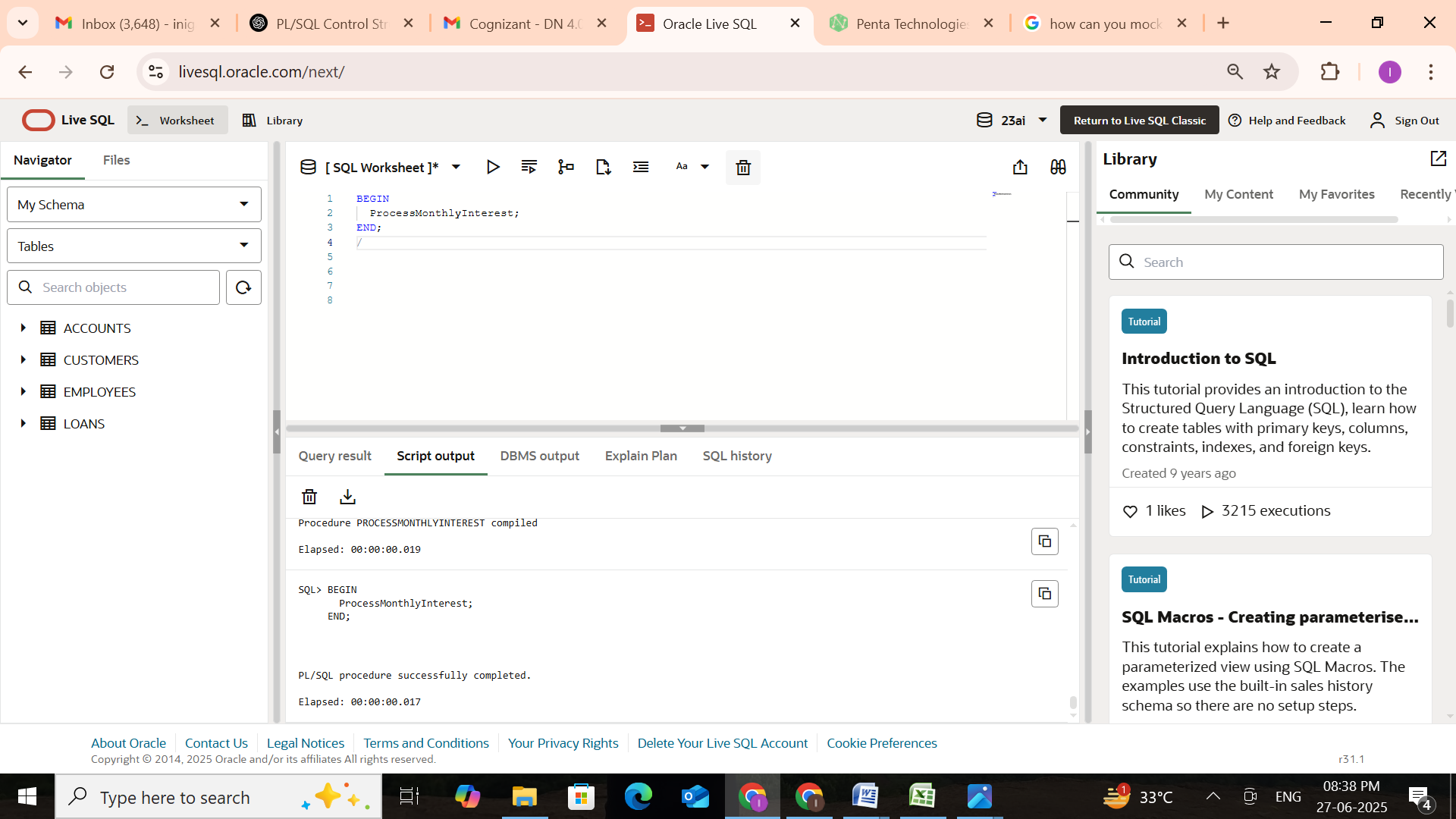
  WHERE AccountType = 'Savings';

  COMMIT;

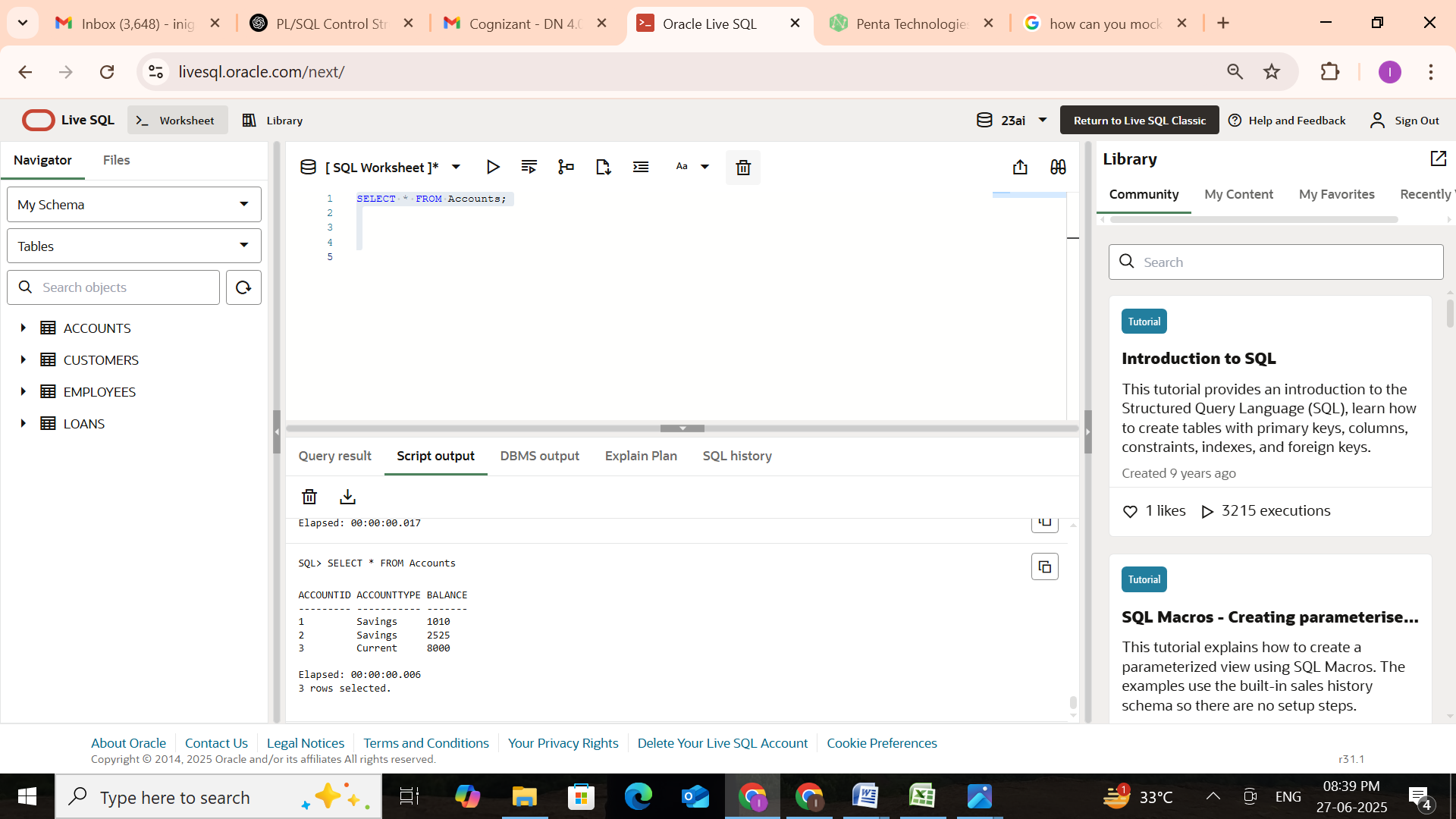
END;

/

Step 2: Execute Procedure



Step 3: Check Result



## Scenario 2: Update Employee Bonus by Department

Step 1: Create Stored Procedure and Update **salaries** by adding a **bonus percentage** for a specified **department**.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  dept\_id IN NUMBER,

  bonus\_percent IN NUMBER

) AS

BEGIN

  UPDATE Employees

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

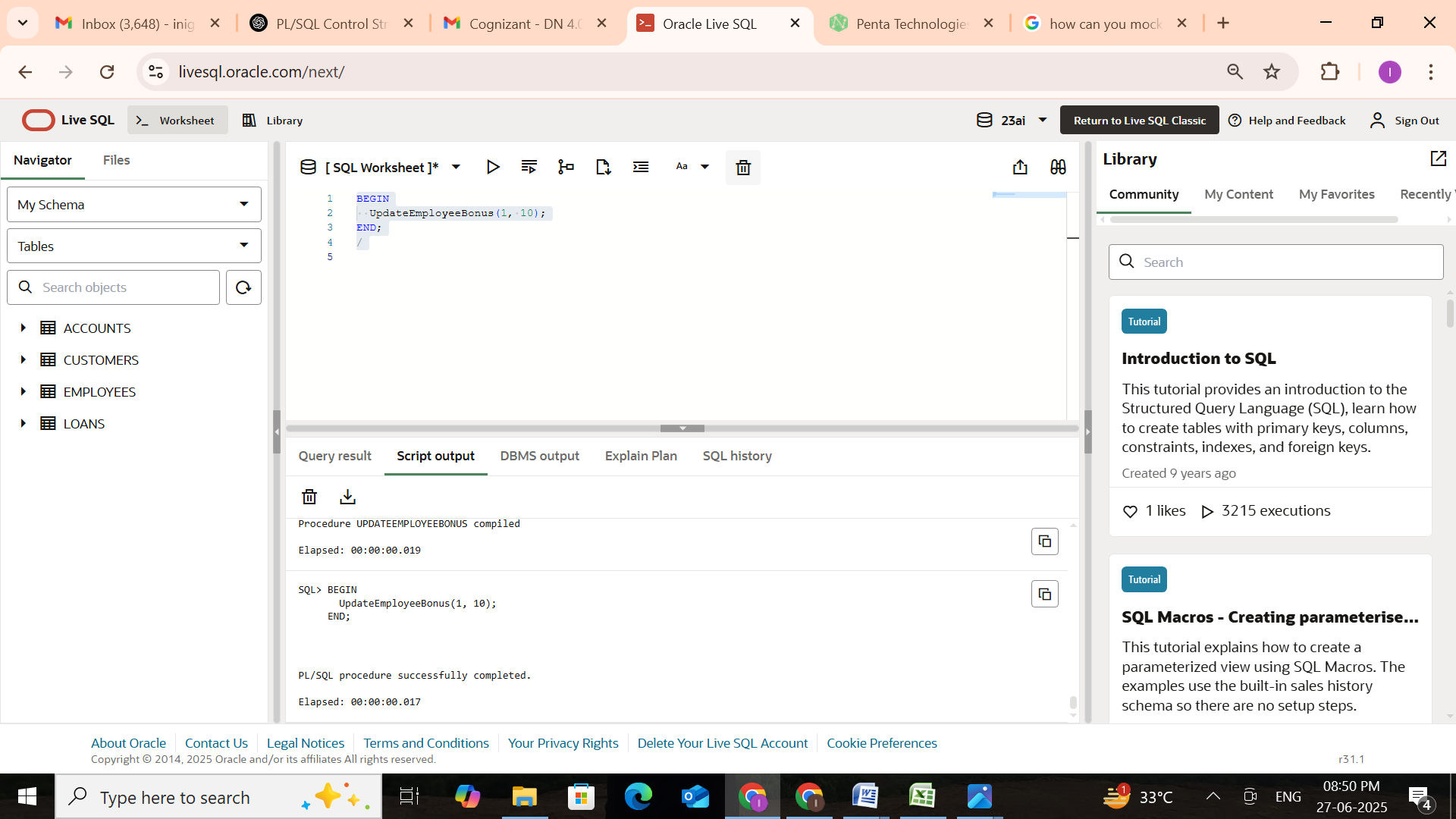
  WHERE DepartmentID = dept\_id;

  COMMIT;

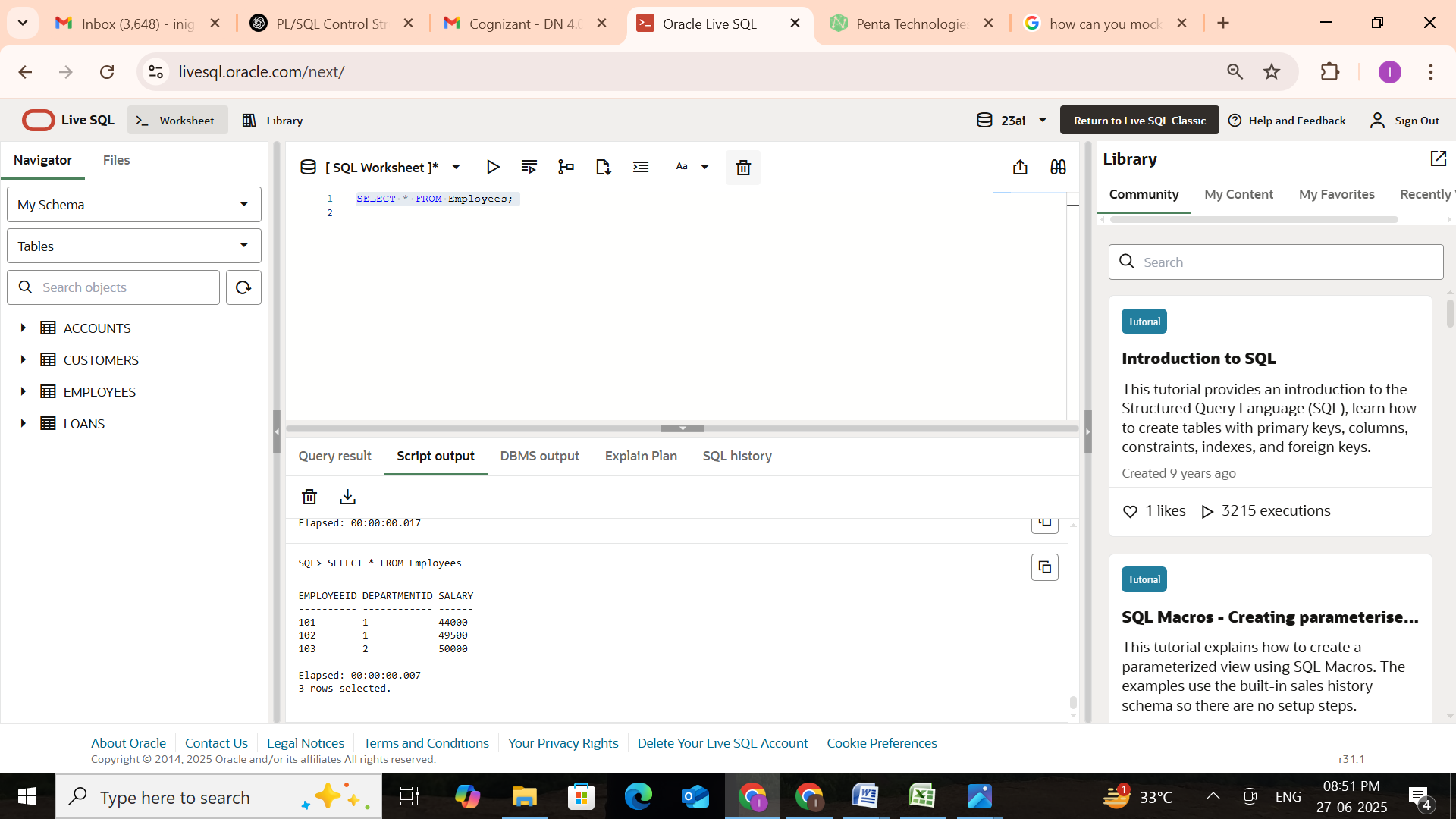
END;

/

Step 2: Execute Procedure (e.g., 10% Bonus for Dept 1)



Step 3: Check Result



**Scenario 3: Transfer Funds Between Accounts**

Transfer funds between two accounts **only if sufficient balance exists** in the source account.

Step 1: Create Stored Procedure

CREATE OR REPLACE PROCEDURE TransferFunds (

  from\_acc IN NUMBER,

  to\_acc IN NUMBER,

  amount IN NUMBER

) AS

  insufficient\_balance EXCEPTION;

BEGIN

  DECLARE

    current\_balance NUMBER;

  BEGIN

    -- Get current balance of source account

    SELECT Balance INTO current\_balance FROM Accounts WHERE AccountID = from\_acc;

    IF current\_balance < amount THEN

      RAISE insufficient\_balance;

    END IF;

    -- Debit from source account

    UPDATE Accounts

    SET Balance = Balance - amount

    WHERE AccountID = from\_acc;

    -- Credit to target account

    UPDATE Accounts

    SET Balance = Balance + amount

    WHERE AccountID = to\_acc;

    COMMIT;

  END;

EXCEPTION

  WHEN insufficient\_balance THEN

    ROLLBACK;

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

  WHEN OTHERS THEN

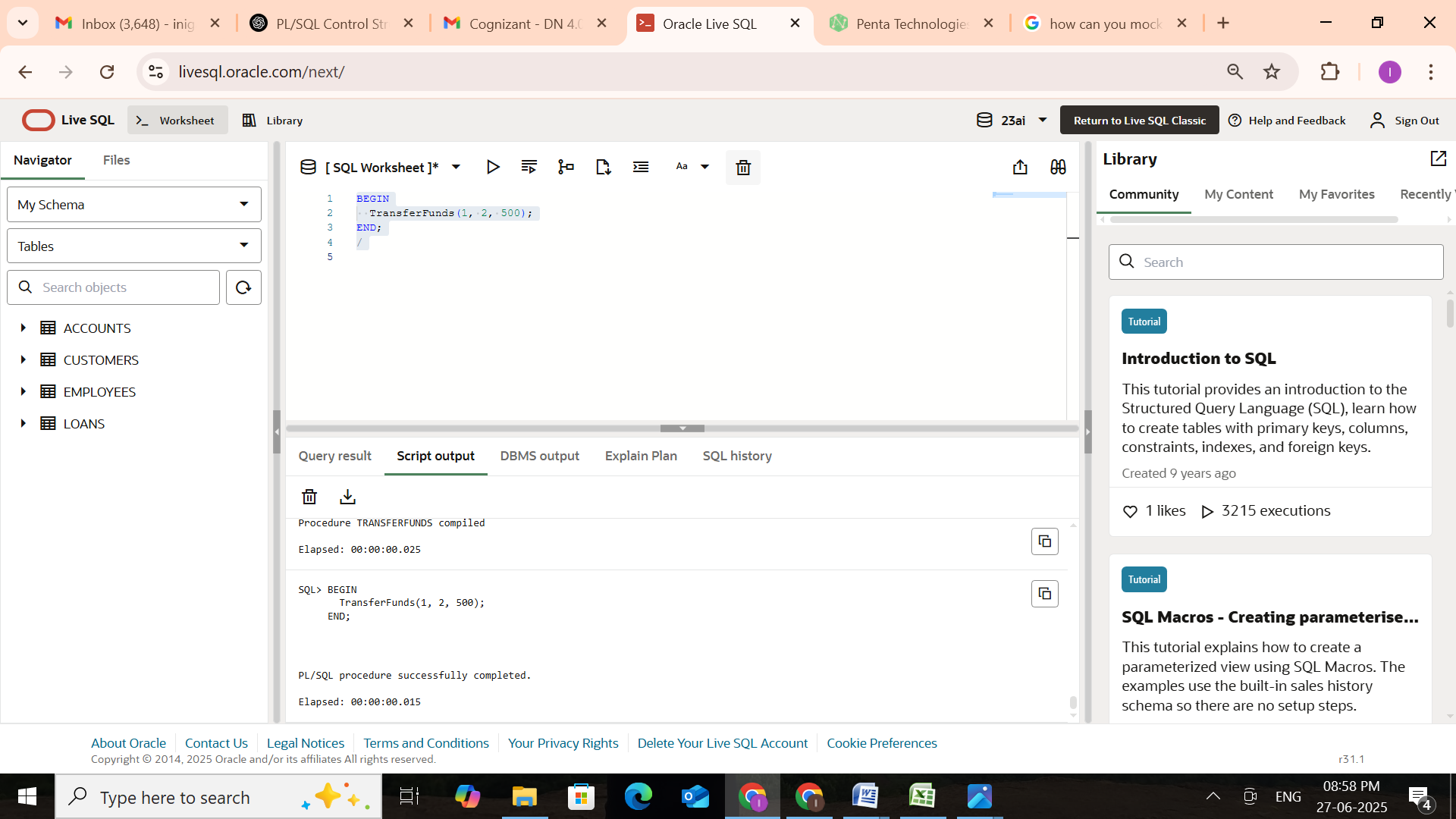
    ROLLBACK;

    DBMS\_OUTPUT.PUT\_LINE('❌ Unexpected error occurred during transfer.');

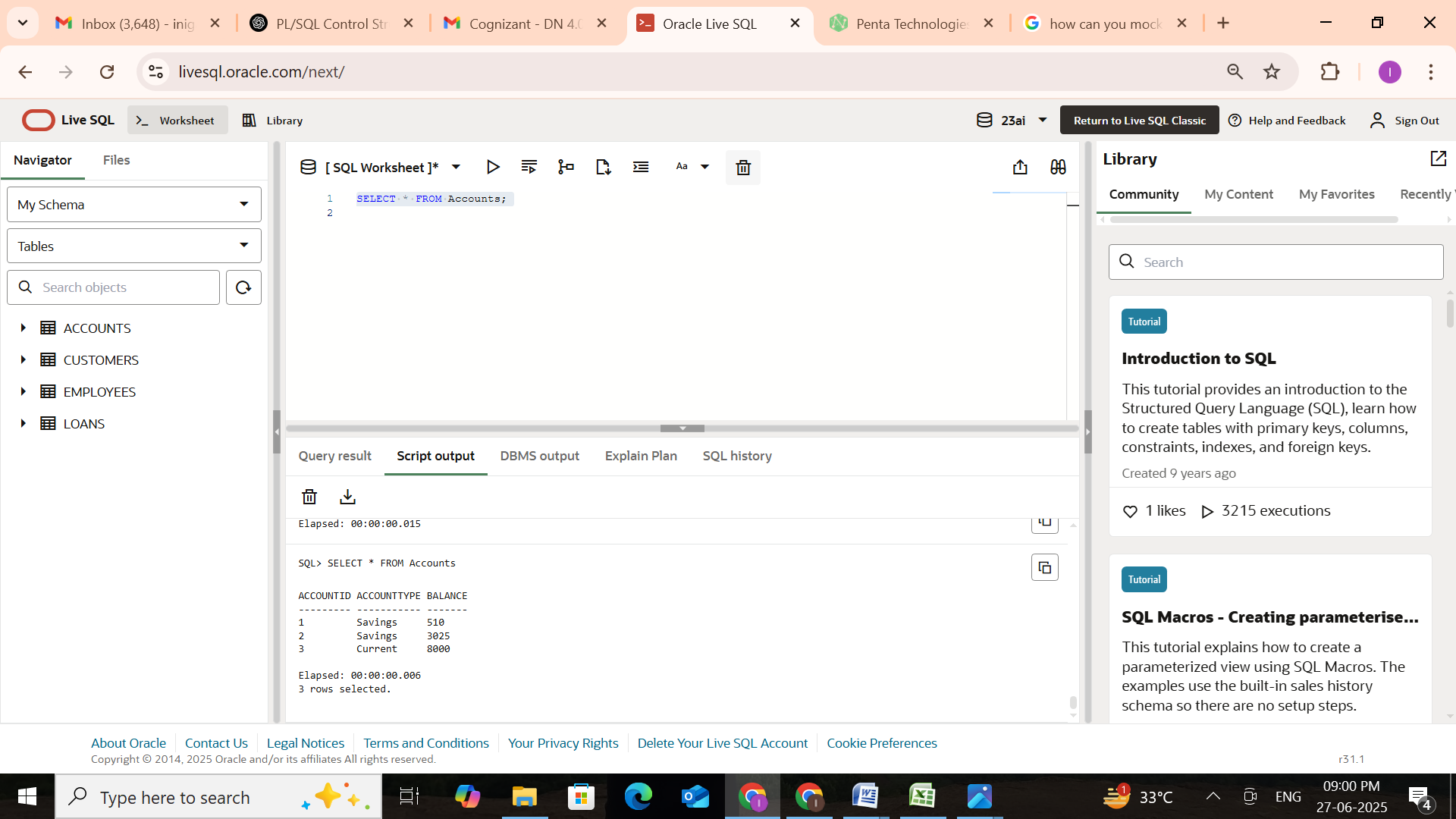
END;

/

Step 2: Execute Fund Transfer

****

Step 3: Check Balances

****