**PL/SQL EXERCISES**

**Exercise 1: Control Structures**

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

BEGIN

  FOR c\_rec IN (

    SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

    FROM Customers c

    JOIN Loans l ON c.CustomerID = l.CustomerID

  )

  LOOP

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

      UPDATE Loans

      SET InterestRate = c\_rec.InterestRate - 1

      WHERE LoanID = c\_rec.LoanID;

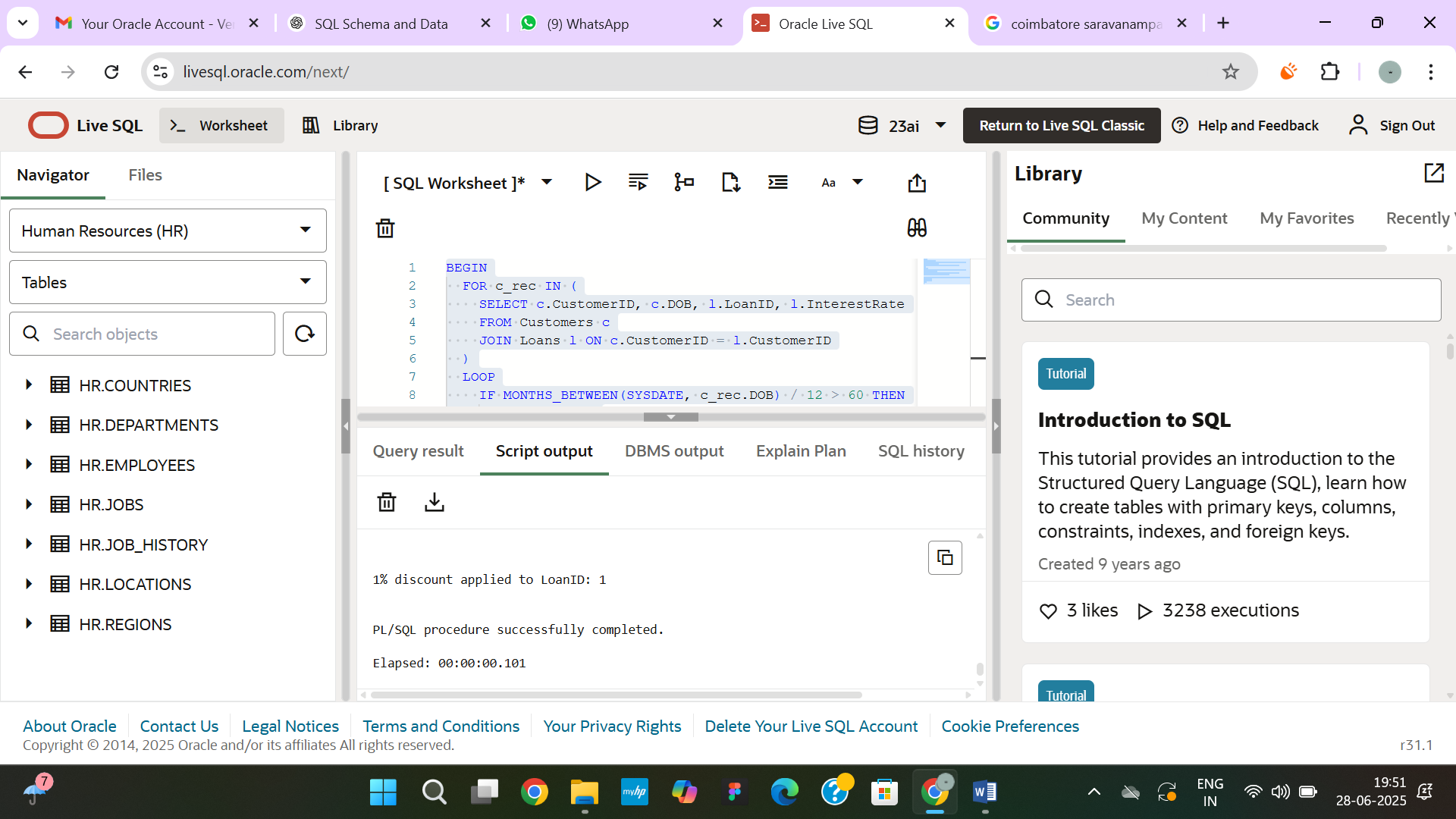
      DBMS\_OUTPUT.PUT\_LINE('1% discount applied to LoanID: ' || c\_rec.LoanID);

    END IF;

  END LOOP;

END;

**Output:**



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

BEGIN

  FOR c\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

    IF c\_rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = c\_rec.CustomerID;

      DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || c\_rec.CustomerID || ' promoted to VIP.');

    ELSE

      UPDATE Customers

      SET IsVIP = 'FALSE'

      WHERE CustomerID = c\_rec.CustomerID;

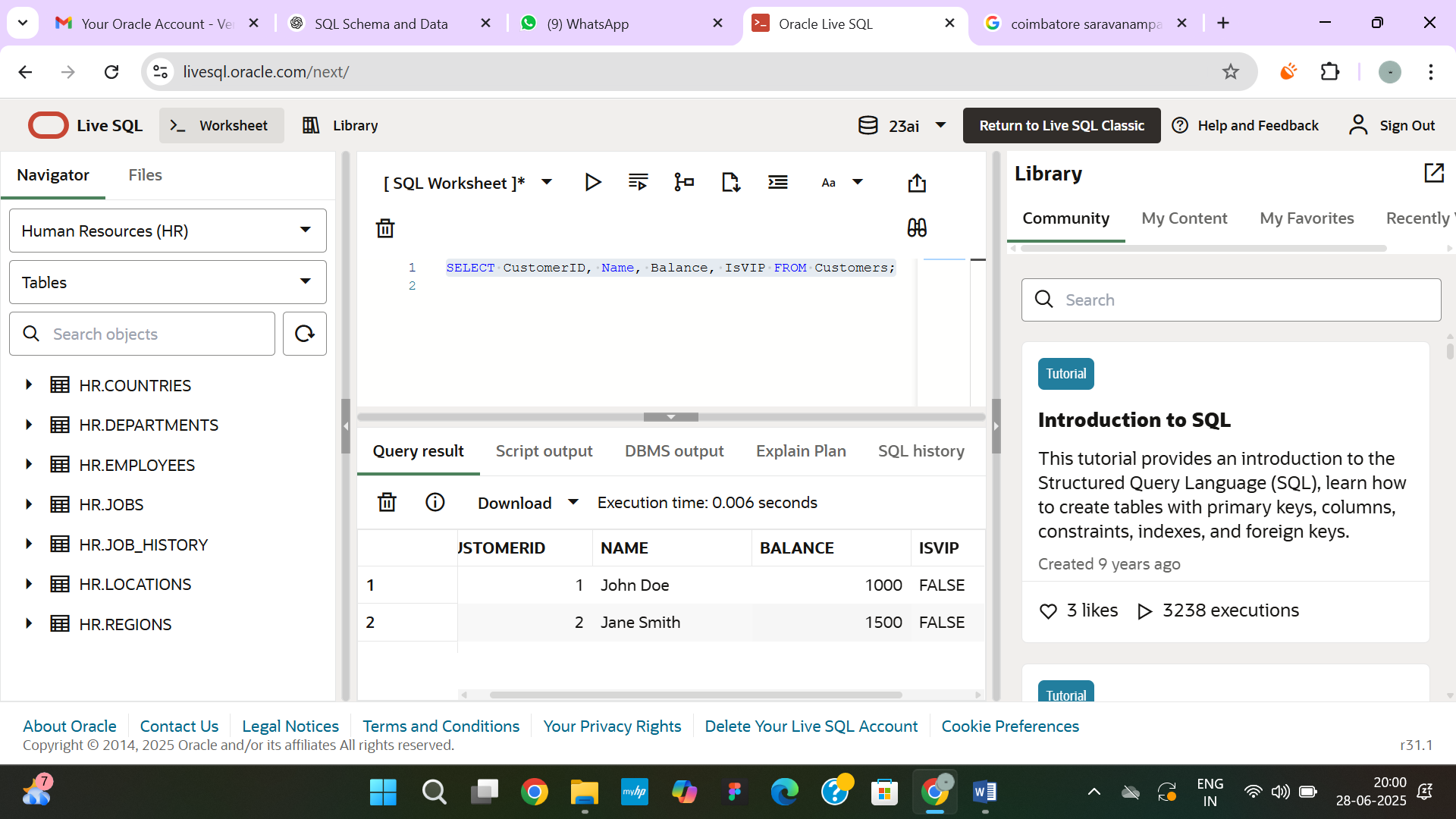
    END IF;

  END LOOP;

END;

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**OUTPUT:**



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

BEGIN

  FOR loan\_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: Loan ID ' || loan\_rec.LoanID ||

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

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

  END LOOP;

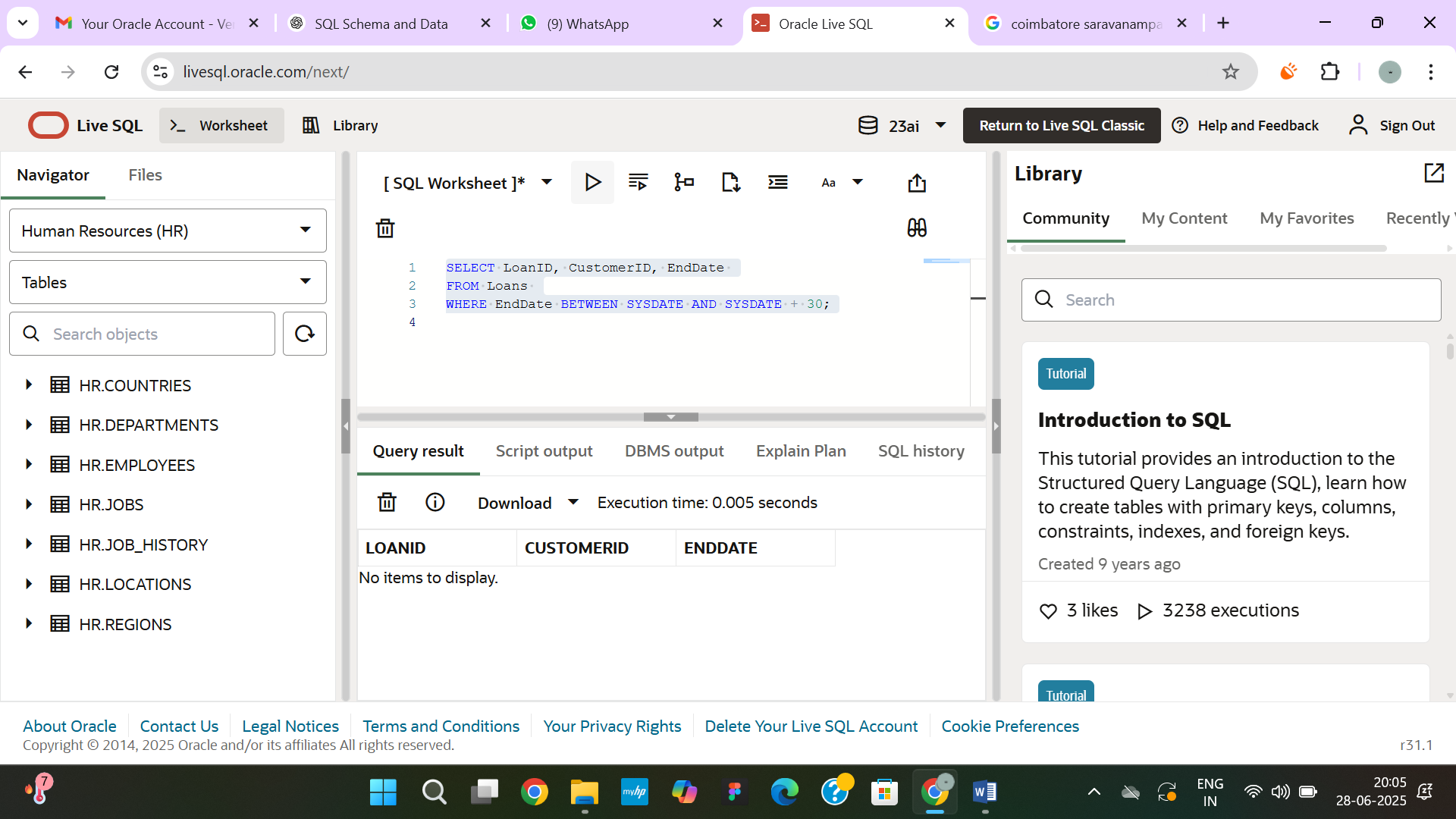
END;

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

**OUTPUT:**



**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

    p\_FromAccountID IN NUMBER,

    p\_ToAccountID IN NUMBER,

    p\_Amount IN NUMBER

) IS

    v\_FromBalance NUMBER;

    v\_ErrorMsg VARCHAR2(4000);

BEGIN

    -- Lock the source account and get the balance

    SELECT Balance INTO v\_FromBalance

    FROM Accounts

    WHERE AccountID = p\_FromAccountID

    FOR UPDATE;

    -- Check for sufficient funds

    IF v\_FromBalance < p\_Amount THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

    END IF;

    -- Deduct from source account

    UPDATE Accounts

    SET Balance = Balance - p\_Amount

    WHERE AccountID = p\_FromAccountID;

    -- Add to destination account

    UPDATE Accounts

    SET Balance = Balance + p\_Amount

    WHERE AccountID = p\_ToAccountID;

    COMMIT;

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

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        v\_ErrorMsg := SQLERRM;

        INSERT INTO TransferErrors (ErrorMessage)

        VALUES (v\_ErrorMsg);

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed. Error logged: ' || v\_ErrorMsg);

END;

BEGIN

  SafeTransferFunds(1, 2, 500);

END;

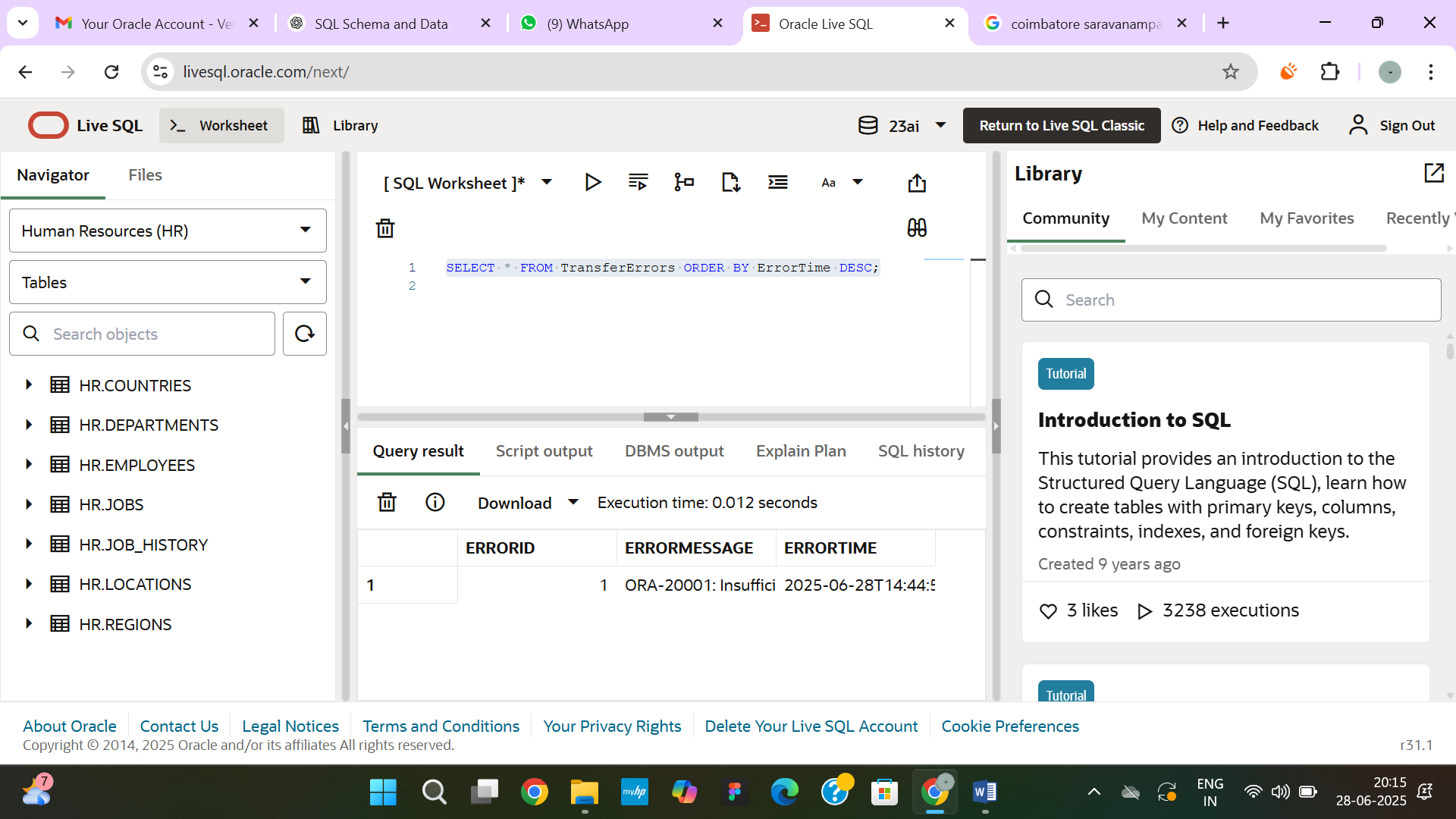
BEGIN

  SafeTransferFunds(1, 2, 999999);

END;

SELECT \* FROM TransferErrors ORDER BY ErrorTime DESC;

**OUTPUT:**

****

**Scenario 2:** Manage errors when updating employee salaries.

CREATE OR REPLACE PROCEDURE UpdateSalary(

    p\_EmployeeID IN NUMBER,

    p\_Percent IN NUMBER

) IS

    v\_Salary Employees.Salary%TYPE;

    v\_ErrorMessage VARCHAR2(4000);

BEGIN

    -- Try to fetch employee salary to verify existence

    SELECT Salary INTO v\_Salary

    FROM Employees

    WHERE EmployeeID = p\_EmployeeID

    FOR UPDATE;

    -- Update salary by given percentage

    UPDATE Employees

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

    WHERE EmployeeID = p\_EmployeeID;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID: ' || p\_EmployeeID);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        v\_ErrorMessage := 'Employee ID ' || p\_EmployeeID || ' does not exist.';

        INSERT INTO TransferErrors (ErrorMessage) VALUES (v\_ErrorMessage);

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_ErrorMessage);

    WHEN OTHERS THEN

        ROLLBACK;

        v\_ErrorMessage := 'Unexpected error for Employee ID ' || p\_EmployeeID || ': ' || SQLERRM;

        INSERT INTO TransferErrors (ErrorMessage) VALUES (v\_ErrorMessage);

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_ErrorMessage);

END;

BEGIN

  UpdateSalary(1, 10);

END;

BEGIN

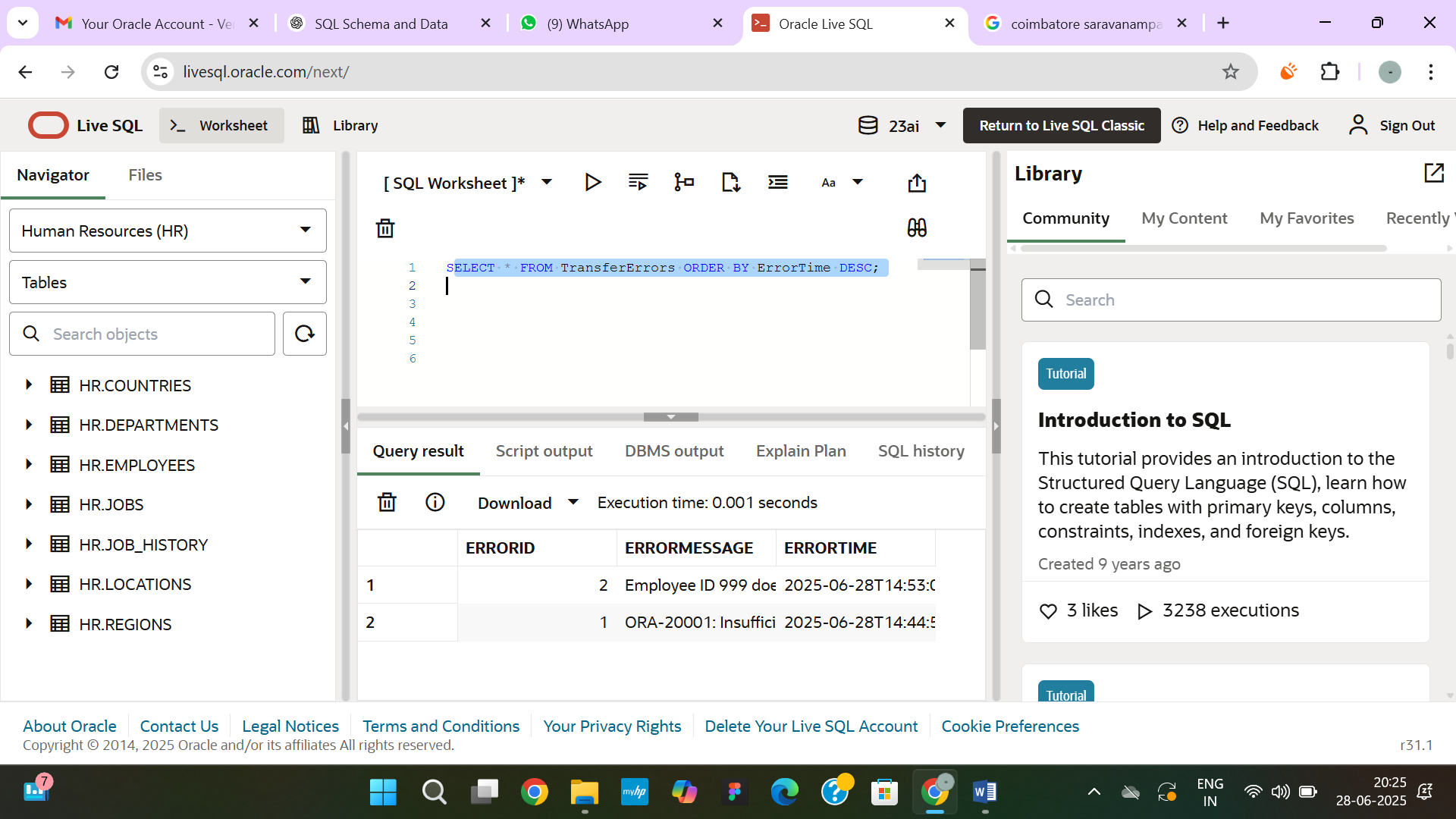
  UpdateSalary(999, 10);

END;

SELECT \* FROM Employees WHERE EmployeeID = 1;

SELECT \* FROM TransferErrors ORDER BY ErrorTime DESC;

OUTPUT:



**Scenario 3:** Ensure data integrity when adding a new customer.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

    p\_CustomerID IN NUMBER,

    p\_Name IN VARCHAR2,

    p\_DOB IN DATE,

    p\_Balance IN NUMBER

) IS

    v\_ErrorMessage VARCHAR2(4000);

BEGIN

    -- Attempt to insert the new customer

    INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

    VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Customer added successfully: ID ' || p\_CustomerID);

EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

        v\_ErrorMessage := 'Customer ID ' || p\_CustomerID || ' already exists. Insertion prevented.';

        INSERT INTO TransferErrors (ErrorMessage) VALUES (v\_ErrorMessage);

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_ErrorMessage);

    WHEN OTHERS THEN

        ROLLBACK;

        v\_ErrorMessage := 'Unexpected error while adding customer ID ' || p\_CustomerID || ': ' || SQLERRM;

        INSERT INTO TransferErrors (ErrorMessage) VALUES (v\_ErrorMessage);

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_ErrorMessage);

END;

BEGIN

  AddNewCustomer(3, 'David Miller', TO\_DATE('1988-10-10', 'YYYY-MM-DD'), 8000);

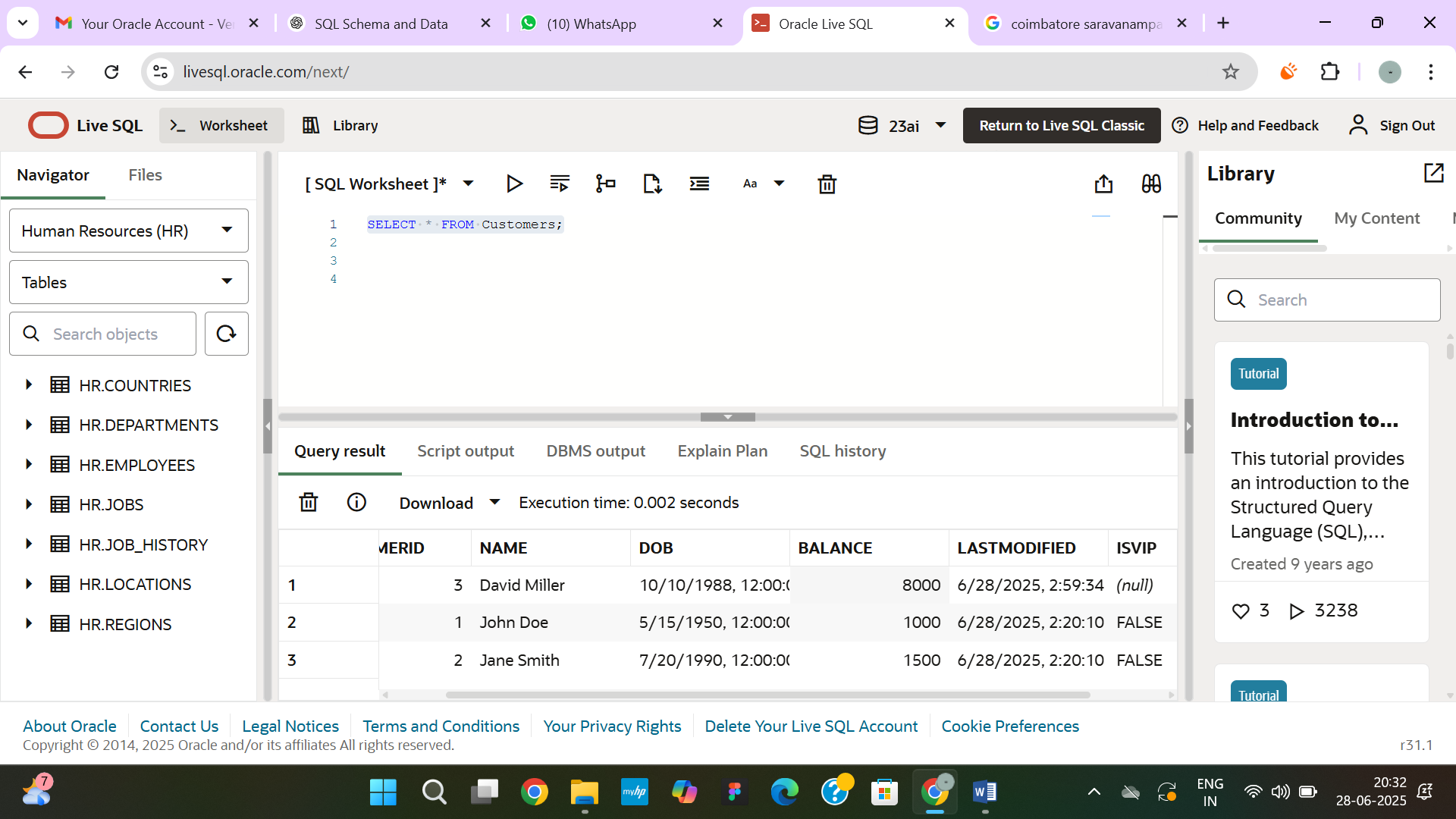
END;

BEGIN

  AddNewCustomer(3, 'Duplicate Entry', TO\_DATE('1990-01-01', 'YYYY-MM-DD'), 9000);

END;

**OUTPUT:**



**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR acc\_rec IN (

    SELECT AccountID, Balance

    FROM Accounts

    WHERE AccountType = 'Savings'

    FOR UPDATE

  ) LOOP

    -- Apply 1% interest

    UPDATE Accounts

    SET Balance = acc\_rec.Balance + (acc\_rec.Balance \* 0.01)

    WHERE AccountID = acc\_rec.AccountID;

    DBMS\_OUTPUT.PUT\_LINE('1% interest applied to Account ID: ' || acc\_rec.AccountID);

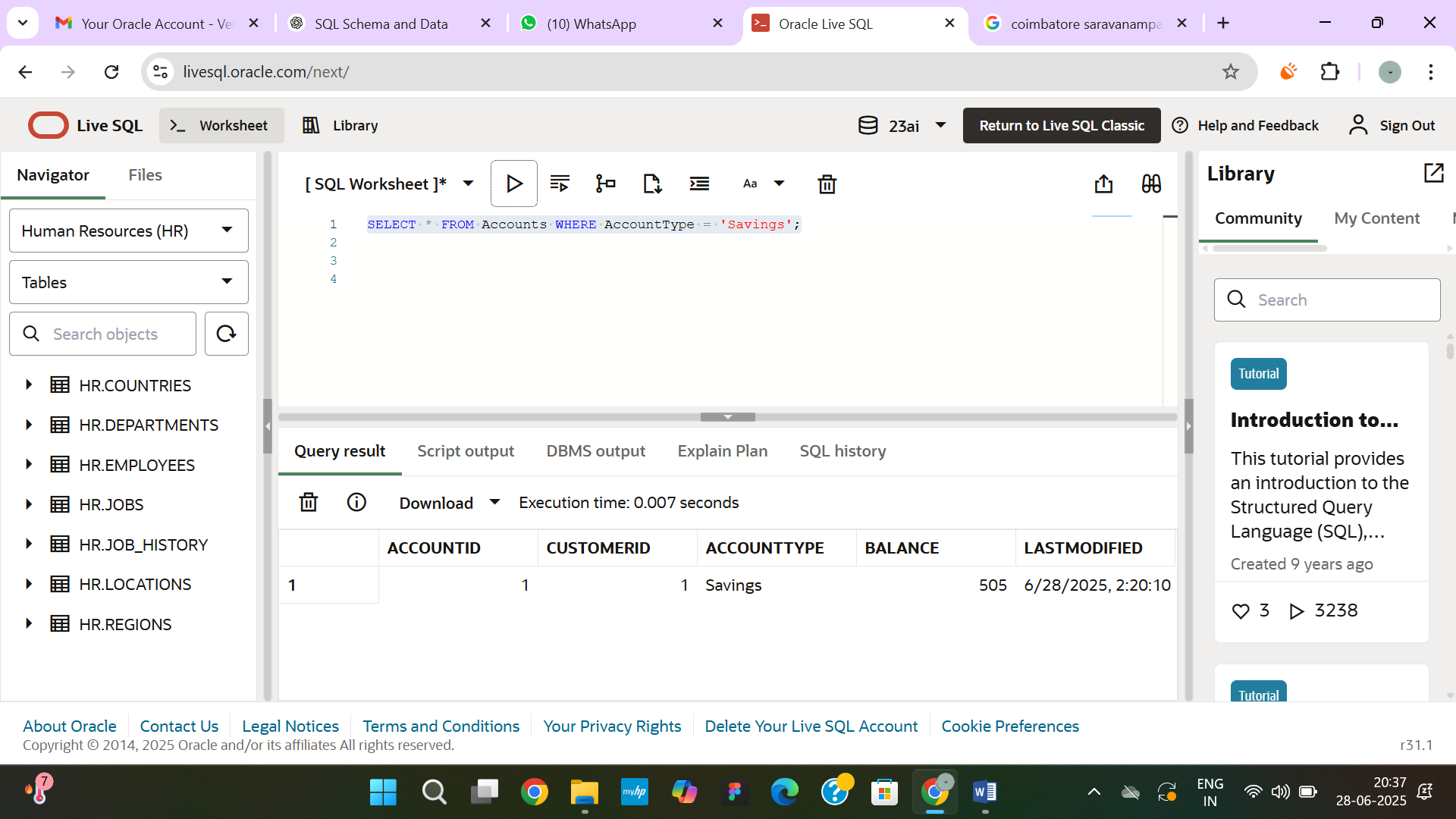
  END LOOP;

  COMMIT;

END;

SELECT \* FROM Accounts WHERE AccountType = 'Savings';

**OUTPUT:**



**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    p\_Department IN VARCHAR2,

    p\_BonusPercent IN NUMBER

) IS

BEGIN

  FOR emp\_rec IN (

    SELECT EmployeeID, Salary

    FROM Employees

    WHERE Department = p\_Department

    FOR UPDATE

  ) LOOP

    -- Update salary with bonus

    UPDATE Employees

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

    WHERE EmployeeID = emp\_rec.EmployeeID;

    DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp\_rec.EmployeeID);

  END LOOP;

  COMMIT;

END;

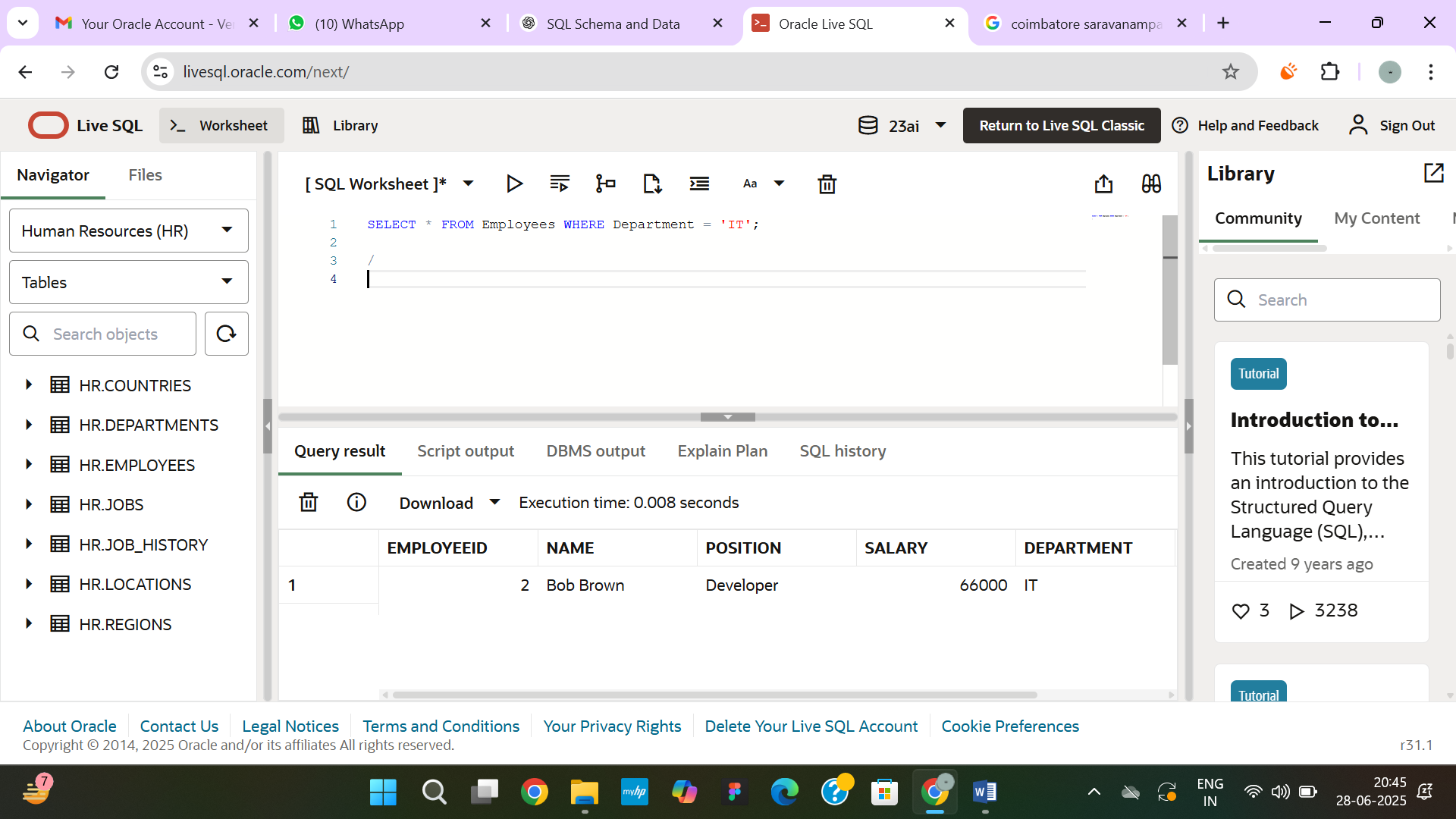
BEGIN

  UpdateEmployeeBonus('IT', 10);

END;

SELECT \* FROM Employees WHERE Department = 'IT';

**OUTPUT:**

****

**Scenario 3:** Customers should be able to transfer funds between their accounts.

CREATE OR REPLACE PROCEDURE TransferFunds(

    p\_FromAccountID IN NUMBER,

    p\_ToAccountID IN NUMBER,

    p\_Amount IN NUMBER

) IS

    v\_FromBalance NUMBER;

BEGIN

    IF p\_FromAccountID = p\_ToAccountID THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Source and destination accounts must be different.');

    END IF;

    SELECT Balance INTO v\_FromBalance

    FROM Accounts

    WHERE AccountID = p\_FromAccountID

    FOR UPDATE;

    IF v\_FromBalance < p\_Amount THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds in the source account.');

    END IF;

    UPDATE Accounts

    SET Balance = Balance - p\_Amount

    WHERE AccountID = p\_FromAccountID;

    UPDATE Accounts

    SET Balance = Balance + p\_Amount

    WHERE AccountID = p\_ToAccountID;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || p\_Amount || ' from Account ' || p\_FromAccountID || ' to Account ' || p\_ToAccountID);

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

BEGIN

  TransferFunds(1, 2, 300);

END;

**OUTPUT:**

