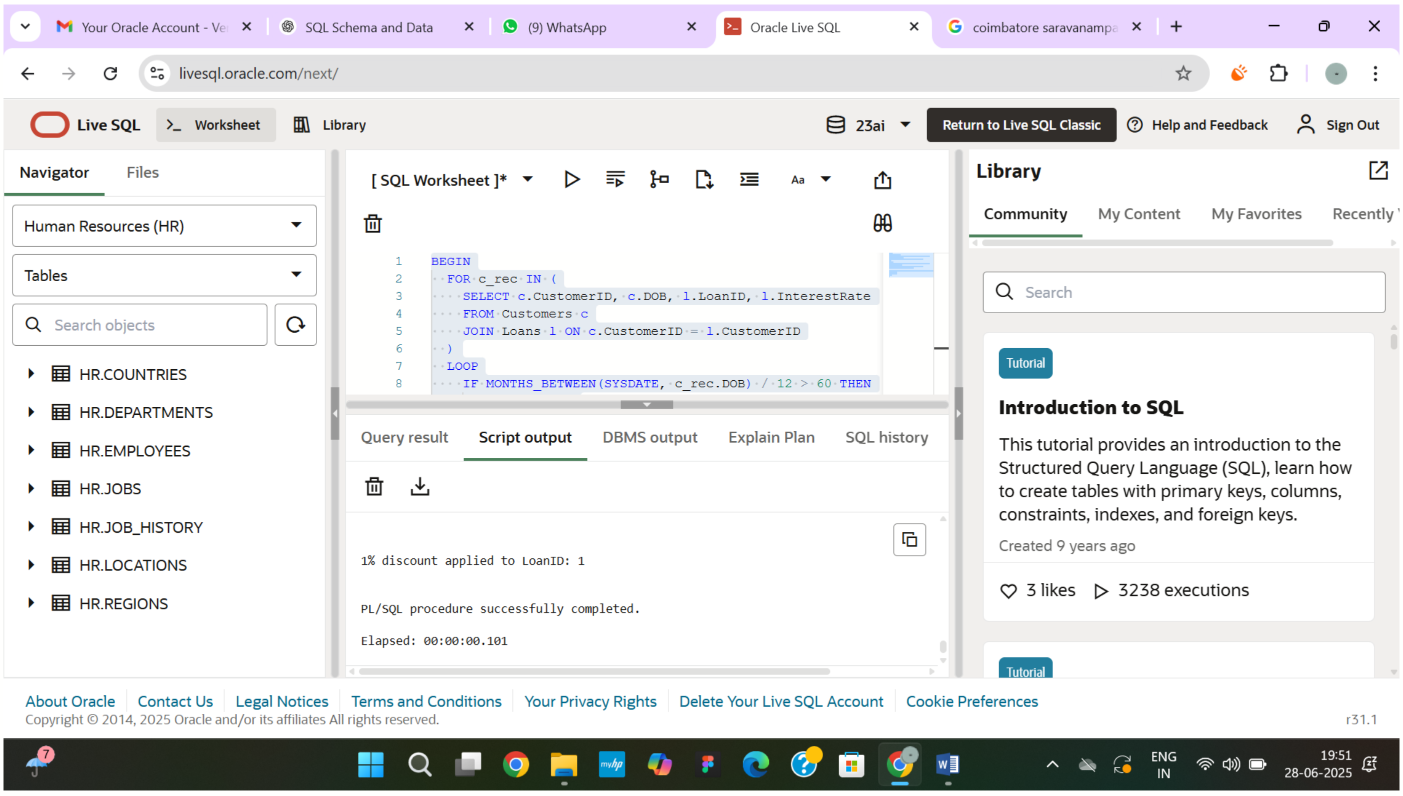
PL/SQL EXERCISES

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:



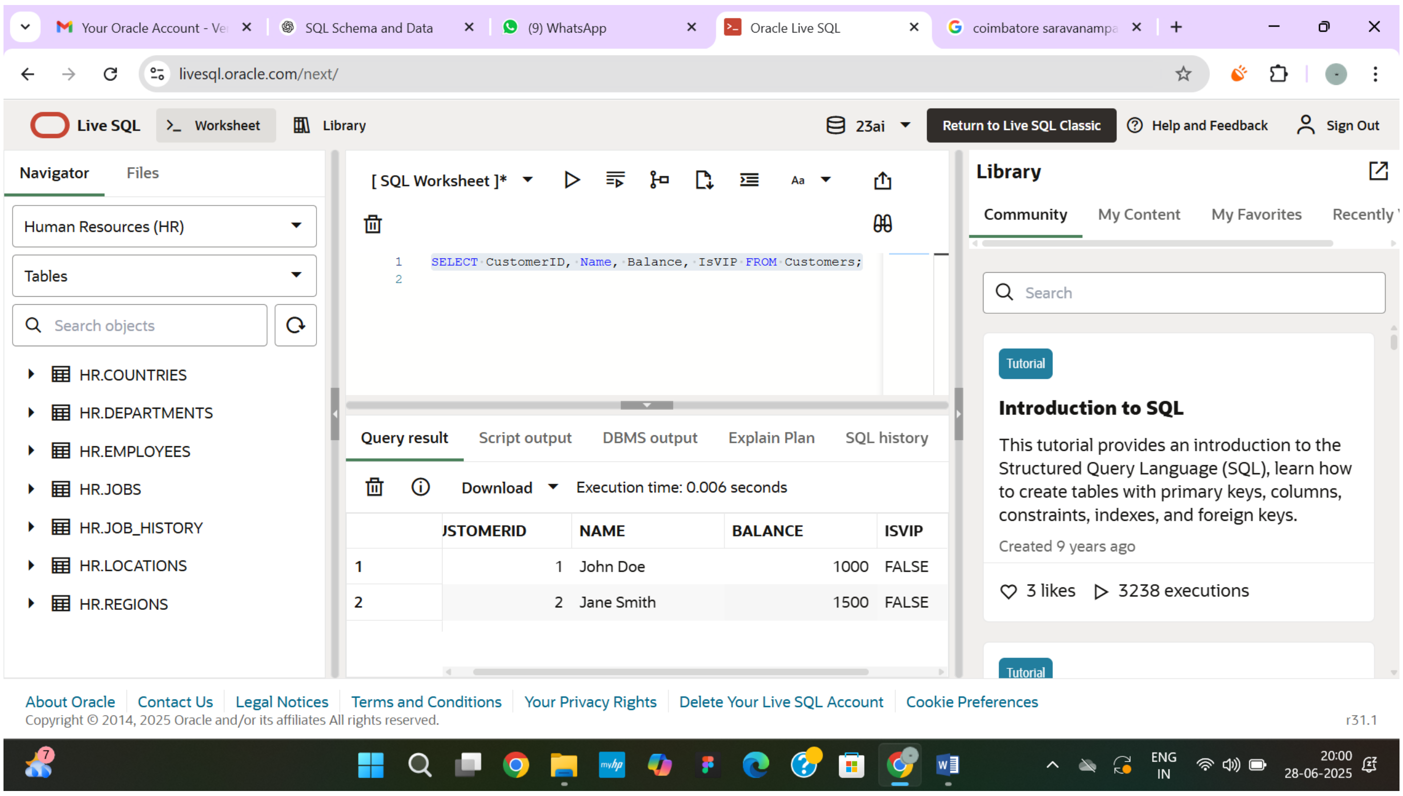
```
BEGIN
  FOR c_rec IN (
    SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate
    FROM Customers c
    JOIN Loans l ON c.CustomerID = l.CustomerID
  )
  LOOP
    IF MONTHS_BETWEEN(SYSDATE, c_rec.DOB) / 12 > 60 THEN
```

```
1% discount applied to LoanID: 1

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.101
```

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:



```
SELECT CustomerID, Name, Balance, IsVIP FROM Customers;
```

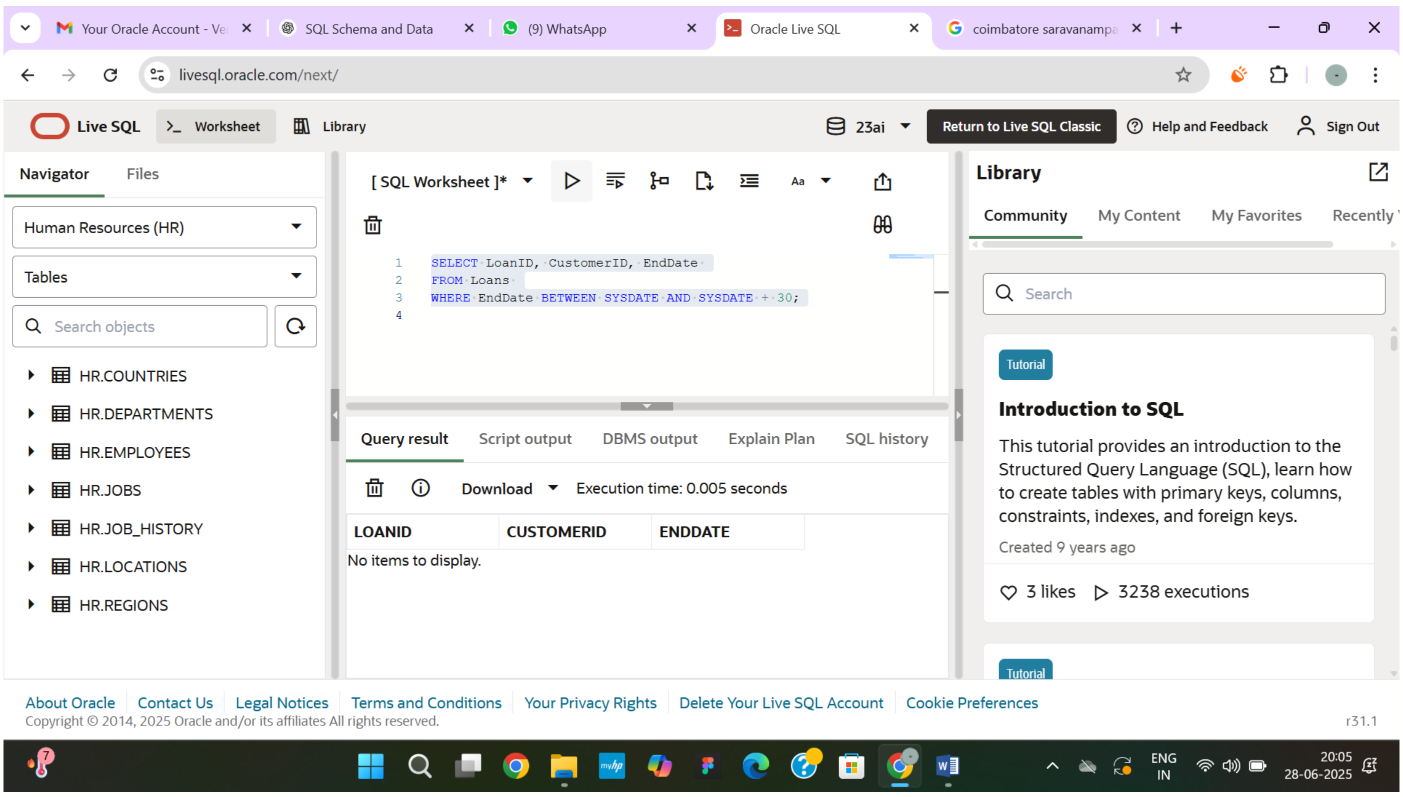
Execution time: 0.006 seconds

	JSTOMERID	NAME	BALANCE	ISVIP
1	1	John Doe	1000	FALSE
2	2	Jane Smith	1500	FALSE

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:



```
SELECT LoanID, CustomerID, EndDate
FROM Loans
WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;
```

Execution time: 0.005 seconds

LOANID	CUSTOMERID	ENDDATE

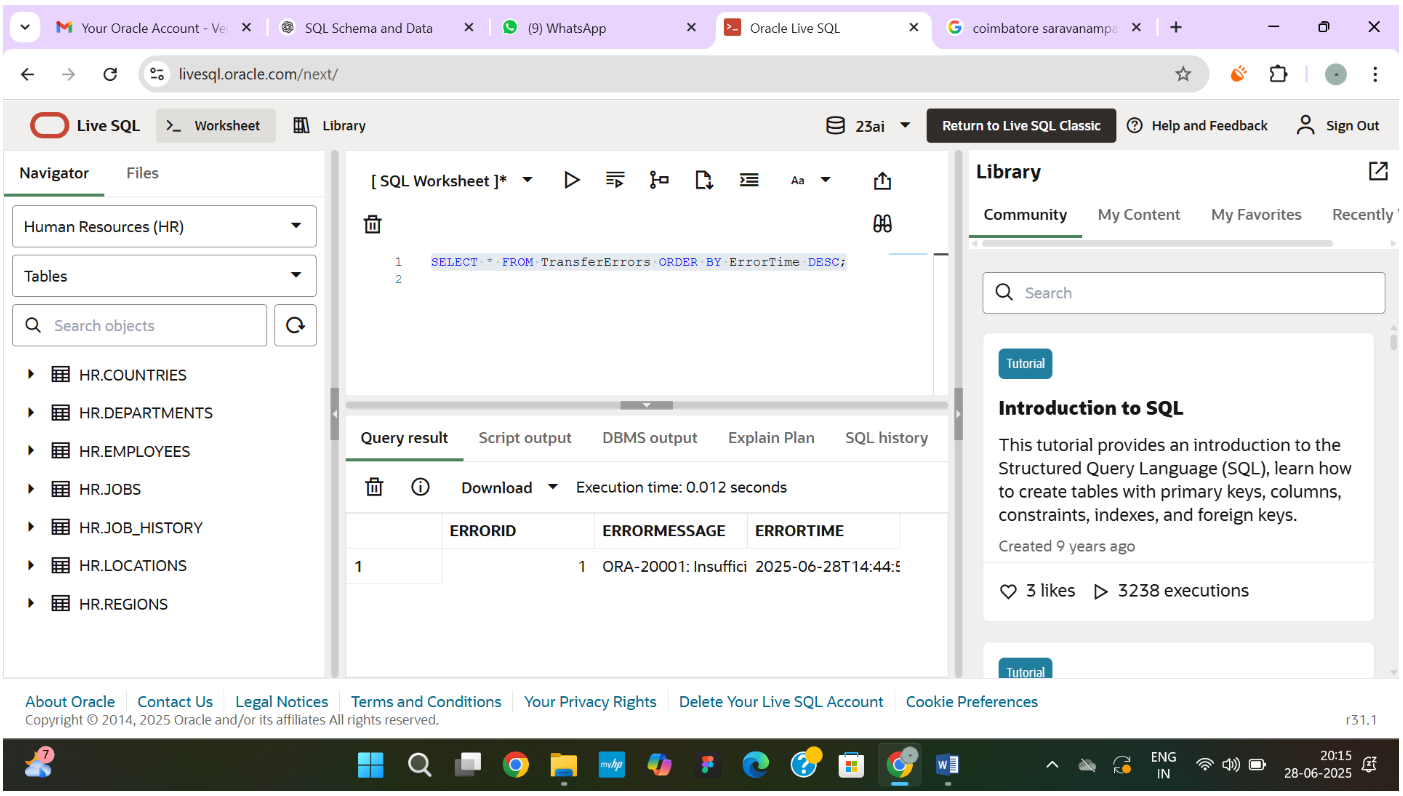
No items to display.

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:



```
SELECT * FROM TransferErrors ORDER BY ErrorTime DESC;
```

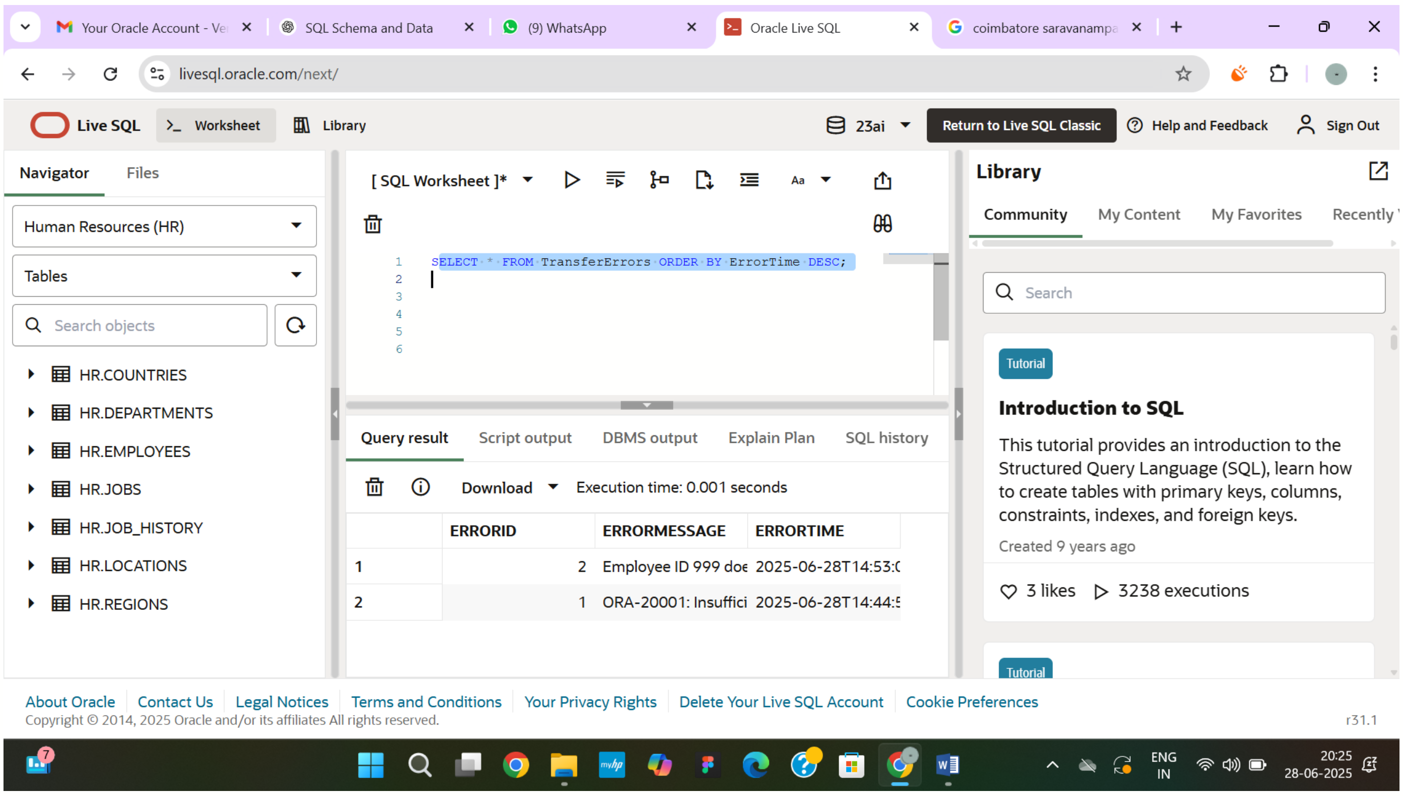
Execution time: 0.012 seconds

	ERRORID	ERRORMESSAGE	ERRORTIME
1	1	ORA-20001: Insuffici	2025-06-28T14:44:5

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:



```
SELECT * FROM TransferErrors ORDER BY ErrorTime DESC;
```

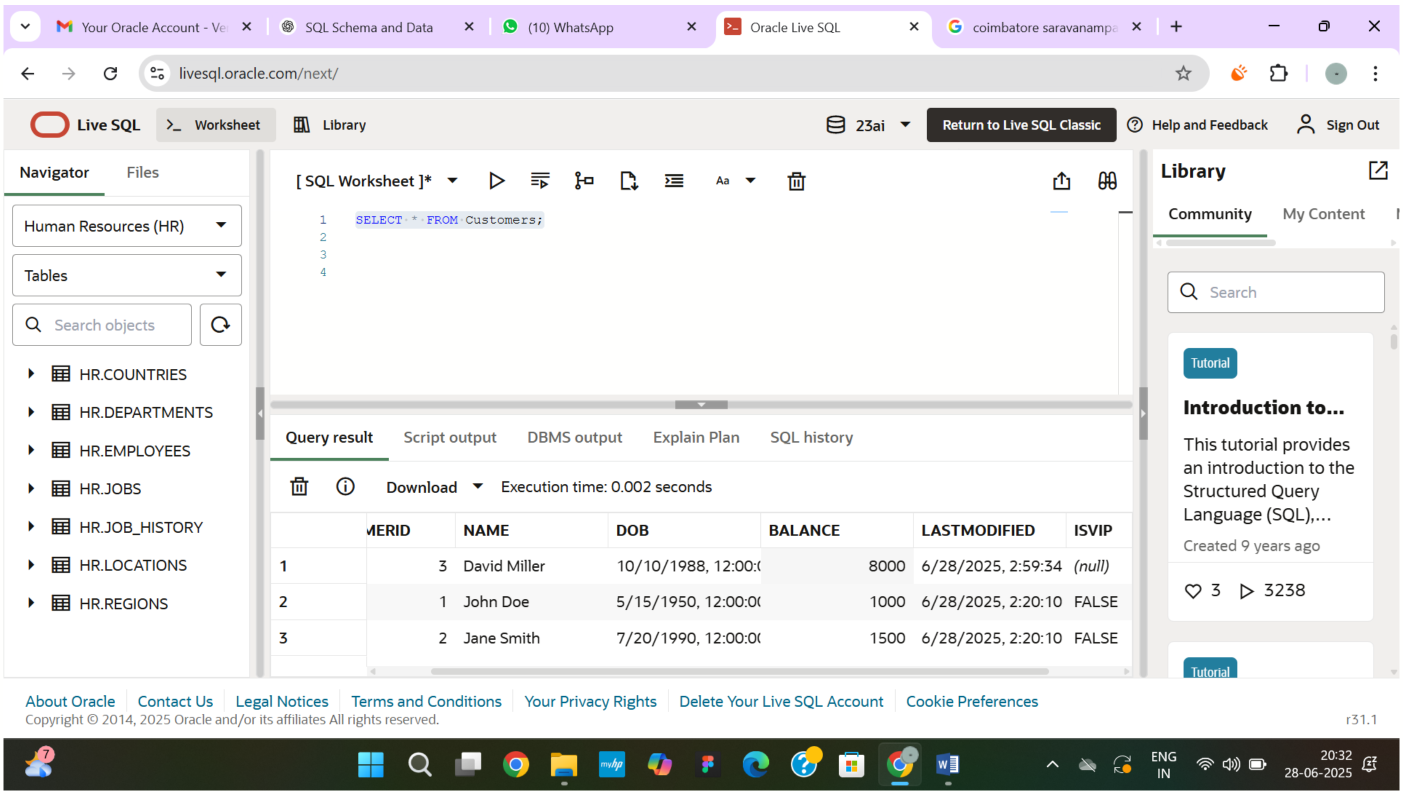
Execution time: 0.001 seconds

	ERRORID	ERRORMESSAGE	ERRORTIME
1	2	Employee ID 999 doe	2025-06-28T14:53:0
2	1	ORA-20001: Insuffici	2025-06-28T14:44:5

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:



```
SELECT * FROM Customers;
```

Execution time: 0.002 seconds

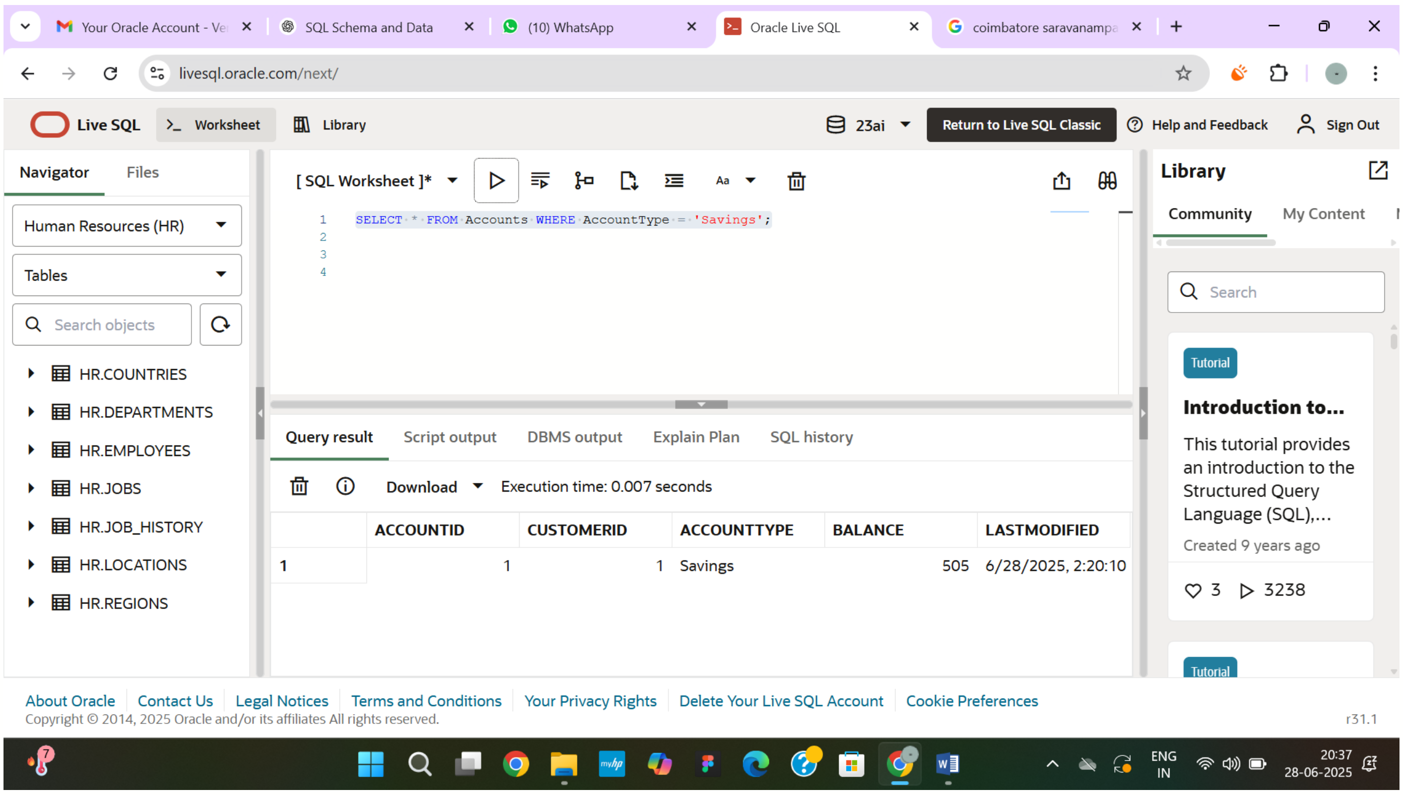
	MERID	NAME	DOB	BALANCE	LASTMODIFIED	ISVIP
1	3	David Miller	10/10/1988, 12:00:(	8000	6/28/2025, 2:59:34	(null)
2	1	John Doe	5/15/1950, 12:00:0(	1000	6/28/2025, 2:20:10	FALSE
3	2	Jane Smith	7/20/1990, 12:00:0(	1500	6/28/2025, 2:20:10	FALSE

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:



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

Execution time: 0.007 seconds

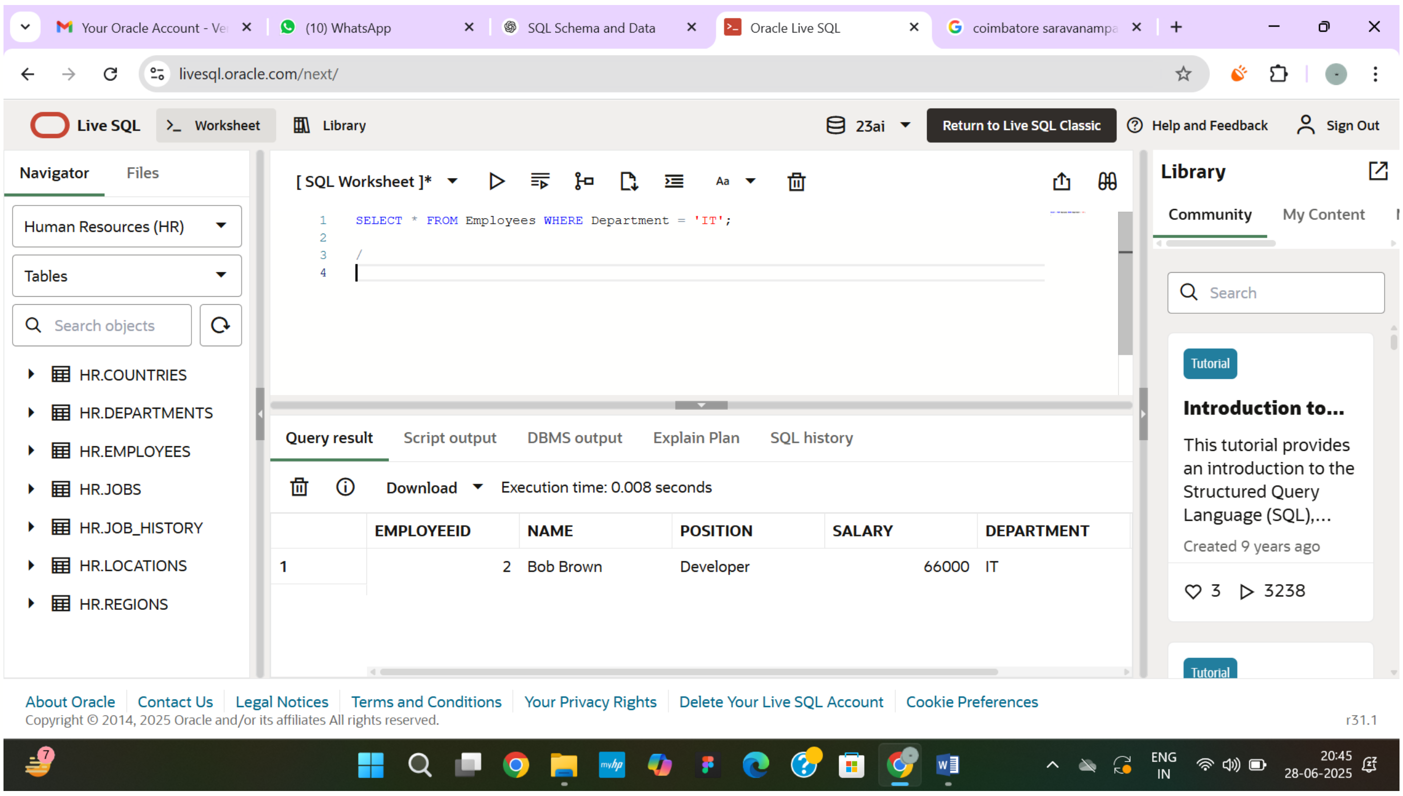
	ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	1	1	Savings	505	6/28/2025, 2:20:10

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:



```
SELECT * FROM Employees WHERE Department = 'IT';
/
```

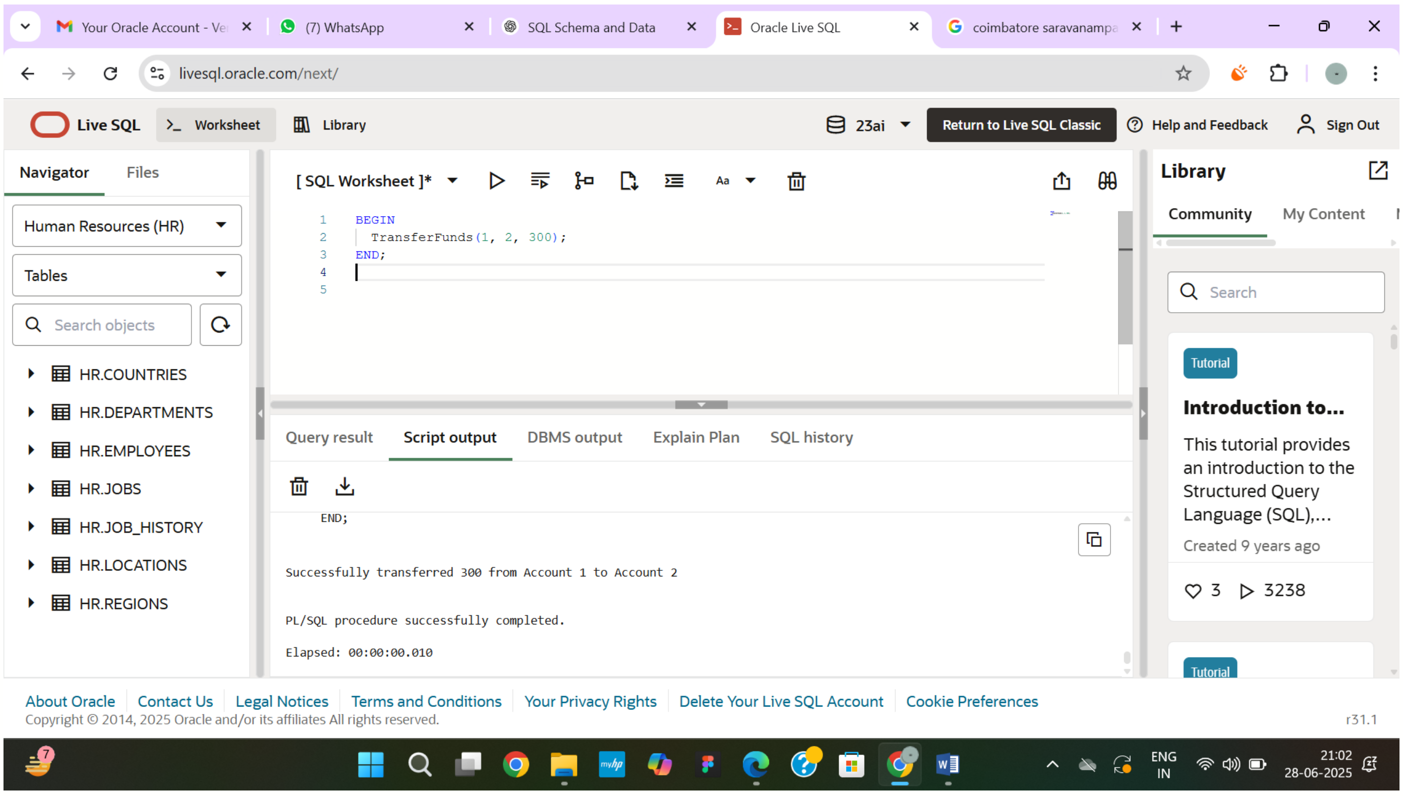
Execution time: 0.008 seconds

	EMPLOYEEID	NAME	POSITION	SALARY	DEPARTMENT
1	2	Bob Brown	Developer	66000	IT

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:



```
BEGIN
  TransferFunds(1, 2, 300);
END;
```

```
END;

Successfully transferred 300 from Account 1 to Account 2

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```