**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:**

BEGIN

  FOR rec IN (

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

           TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB)/12) AS Age

    FROM Customers c

    JOIN Loans l ON c.CustomerID = l.CustomerID

  ) LOOP

    IF rec.Age > 60 THEN

      UPDATE Loans

      SET InterestRate = InterestRate - 1

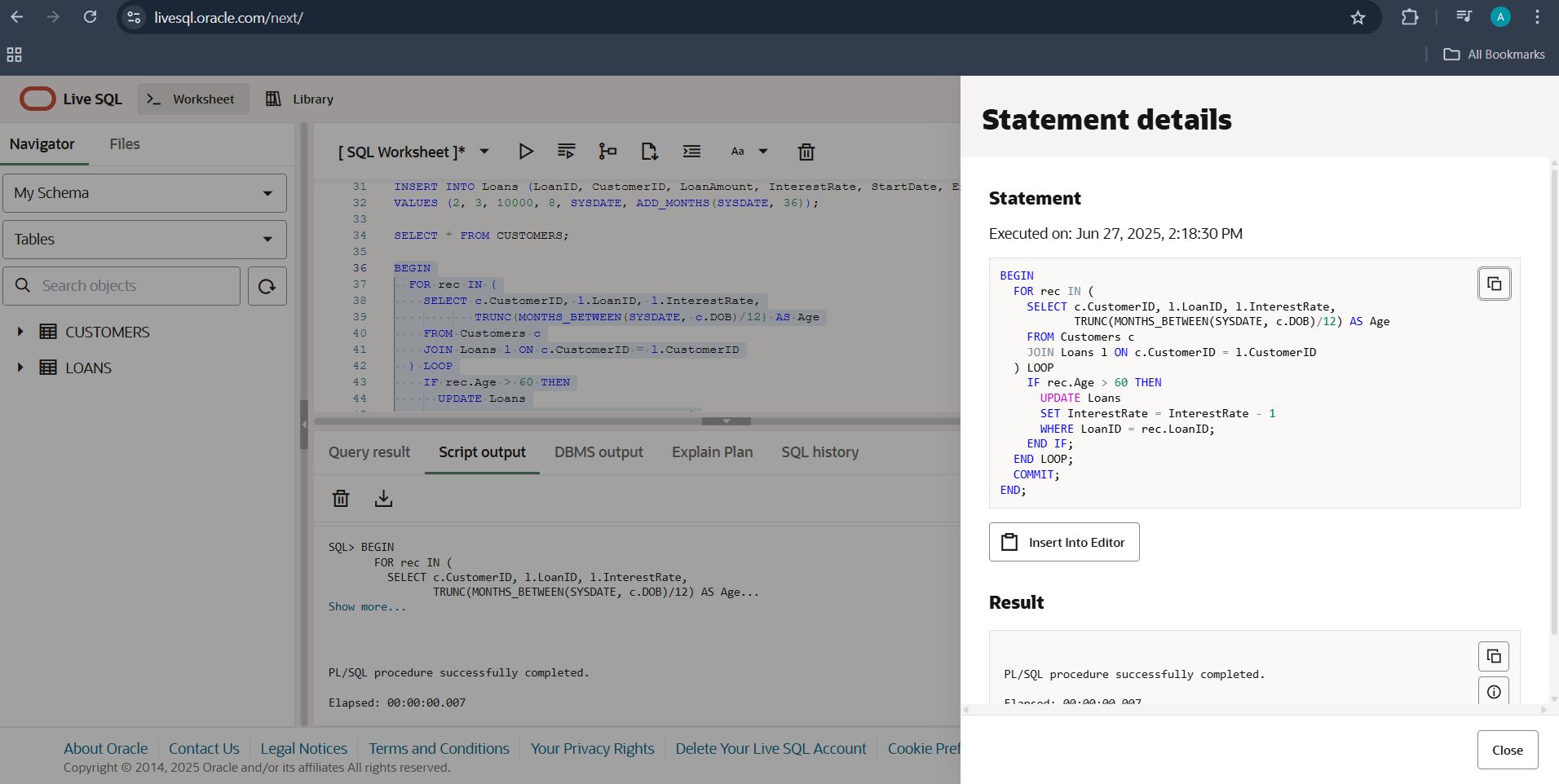
      WHERE LoanID = rec.LoanID;

    END IF;

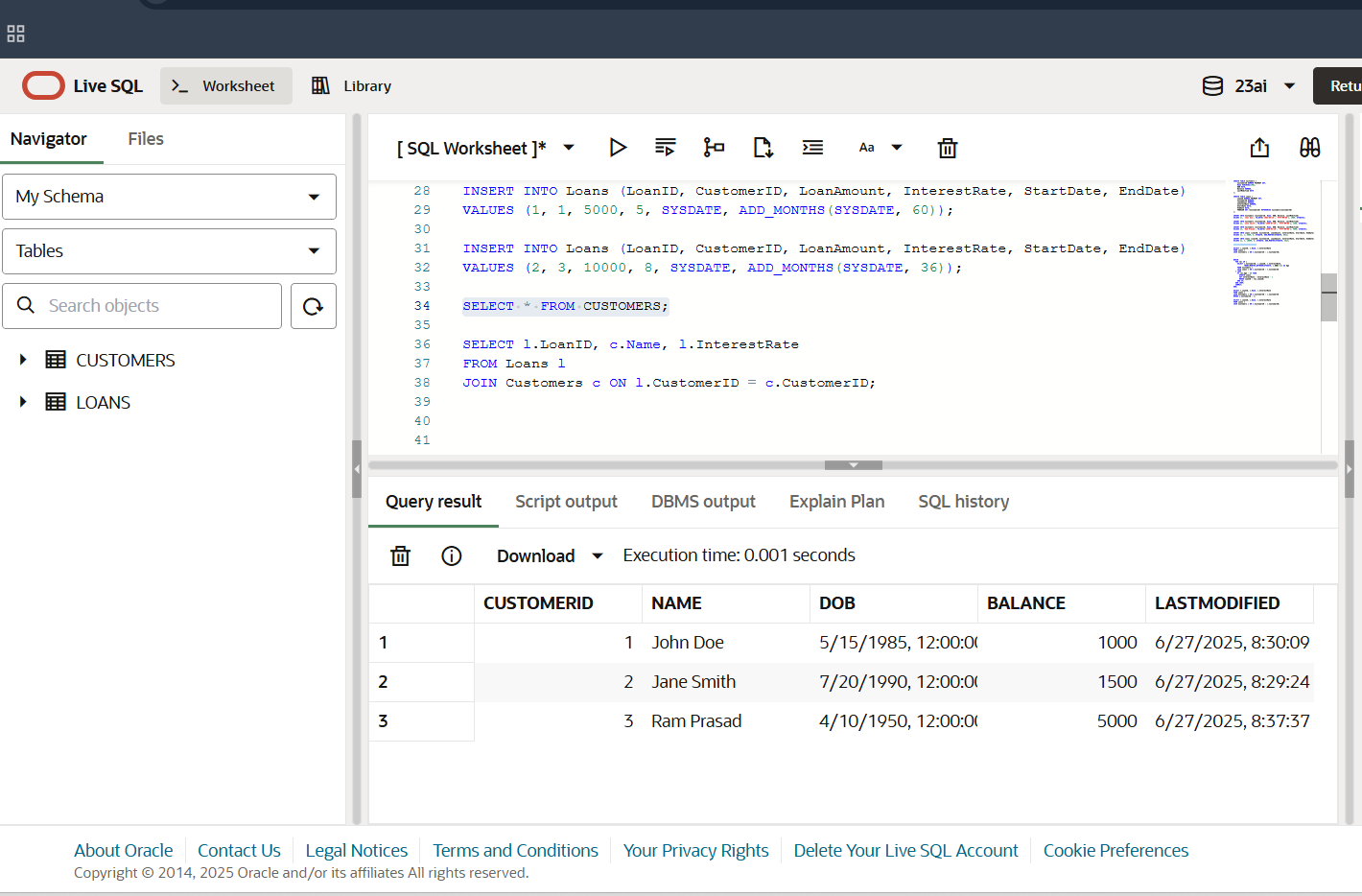
  END LOOP;

  COMMIT;

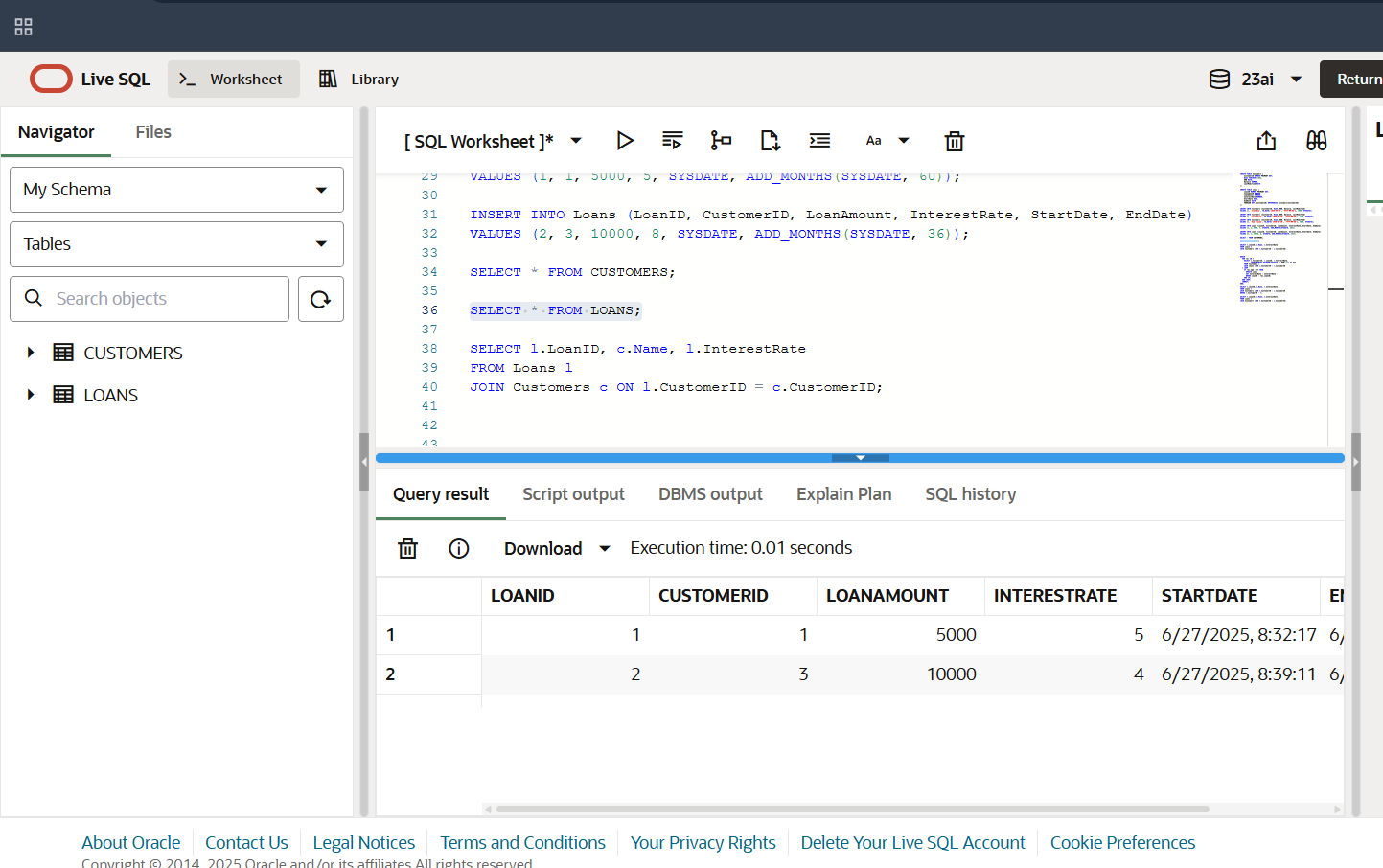
END;



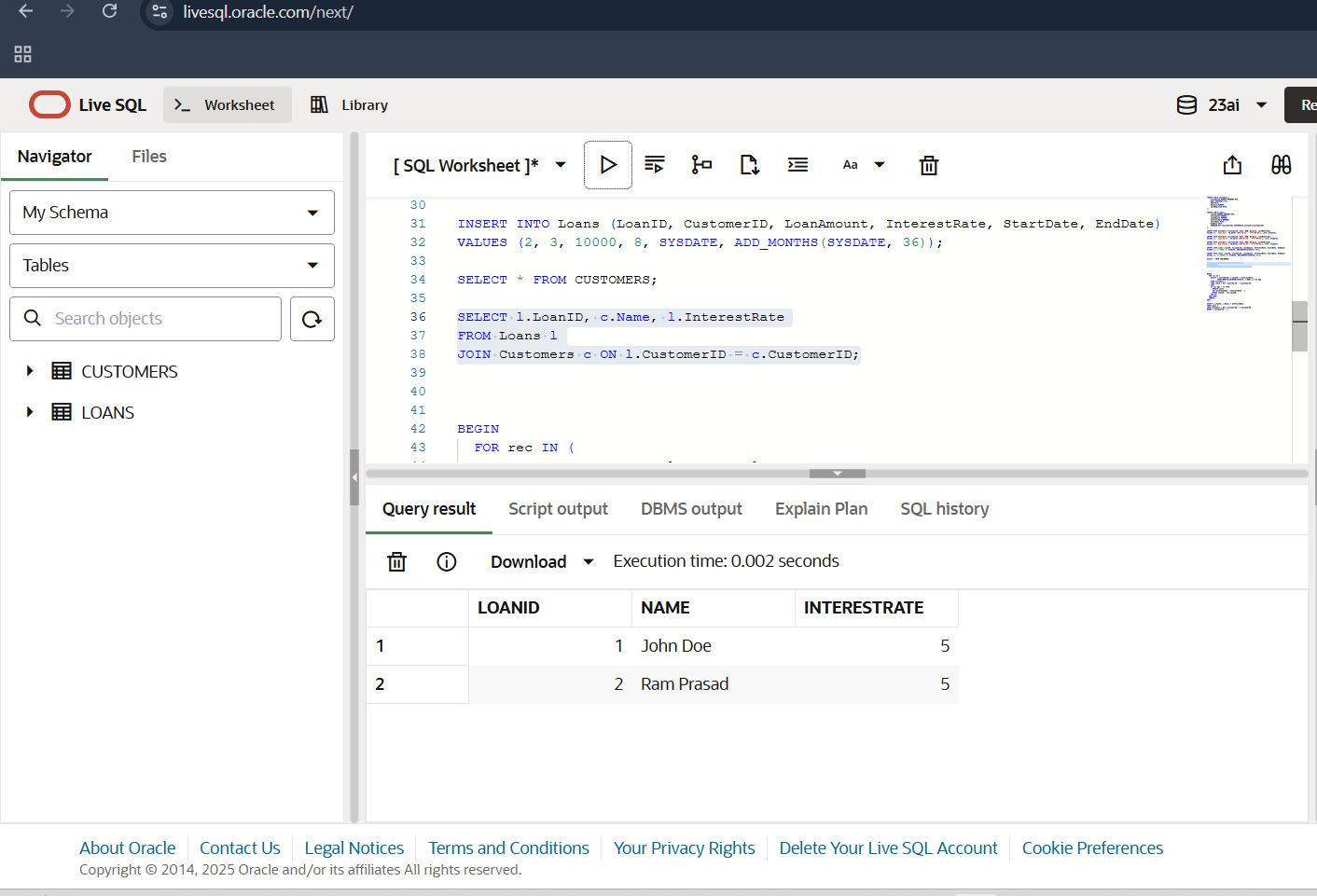
**CUSTOMERS TABLE:**

****

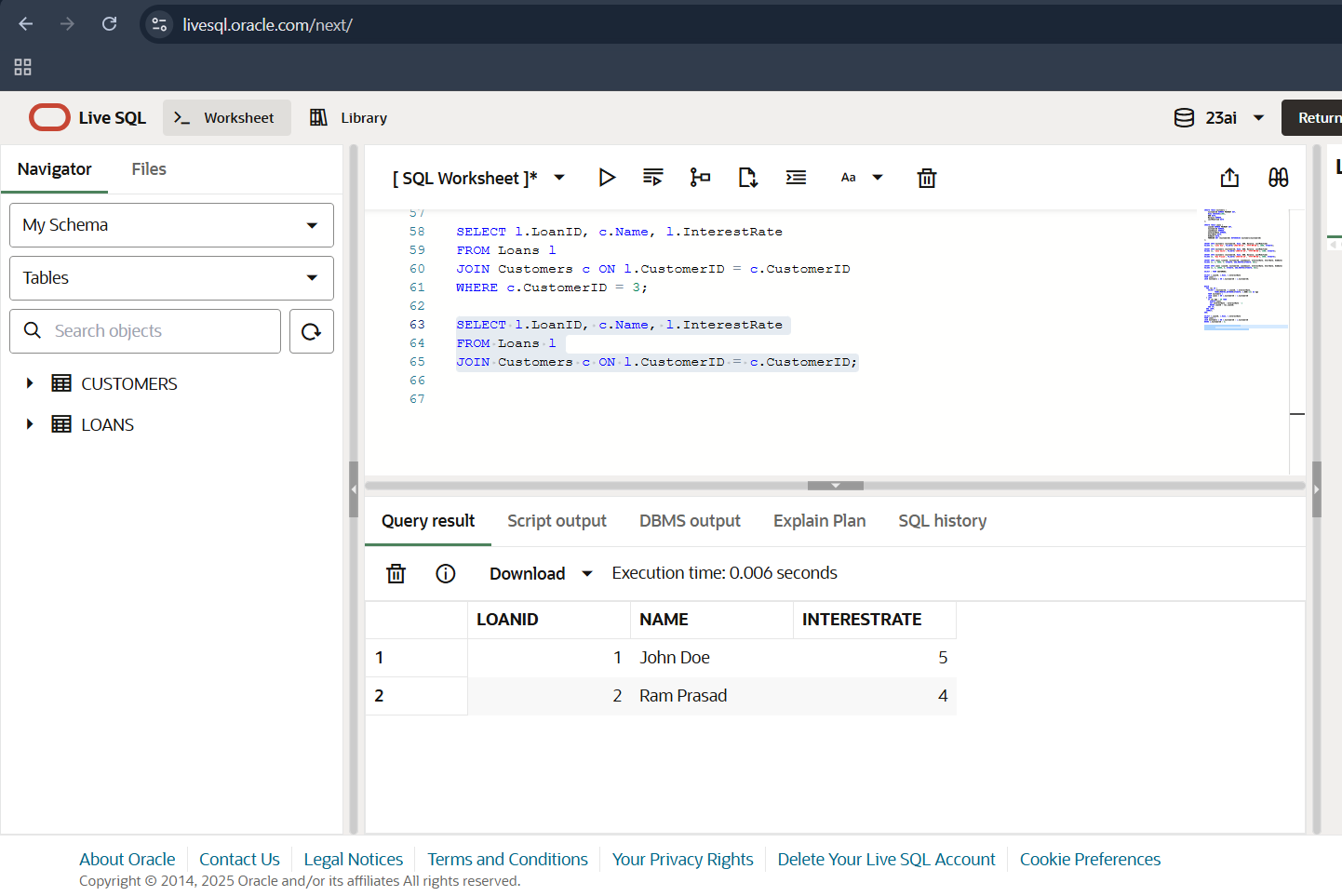
**LOANS TABLE:**

****

**OUTPUT BEFORE APPLYING PL/SQL CODE:**

****

**OUTPUT AFTER APPLYING PL/SQL CODE:**

****

**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.

**ALTER TABLE CUSTOMERS:**

ALTER TABLE Customers

ADD IsVIP VARCHAR2(5) DEFAULT 'FALSE';

**PL/SQL Code:**

BEGIN

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

    IF rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

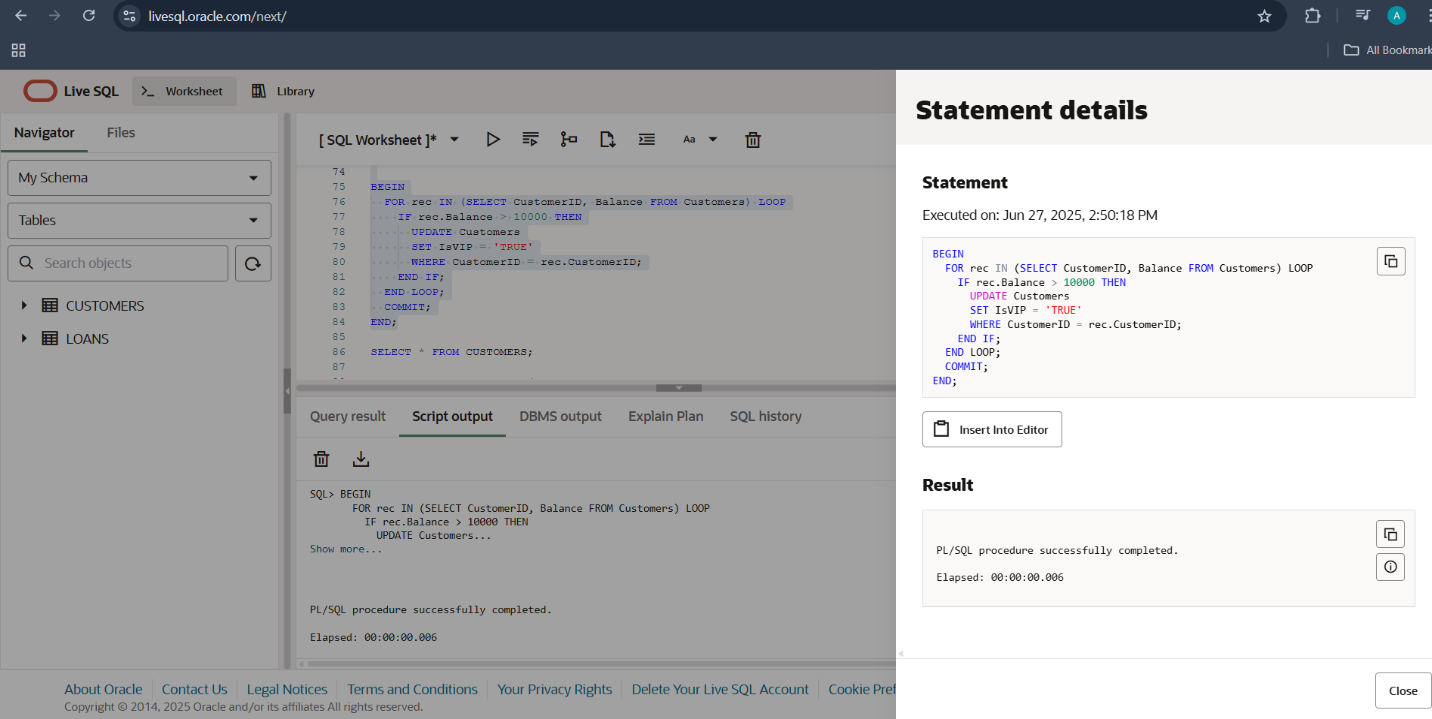
      WHERE CustomerID = rec.CustomerID;

    END IF;

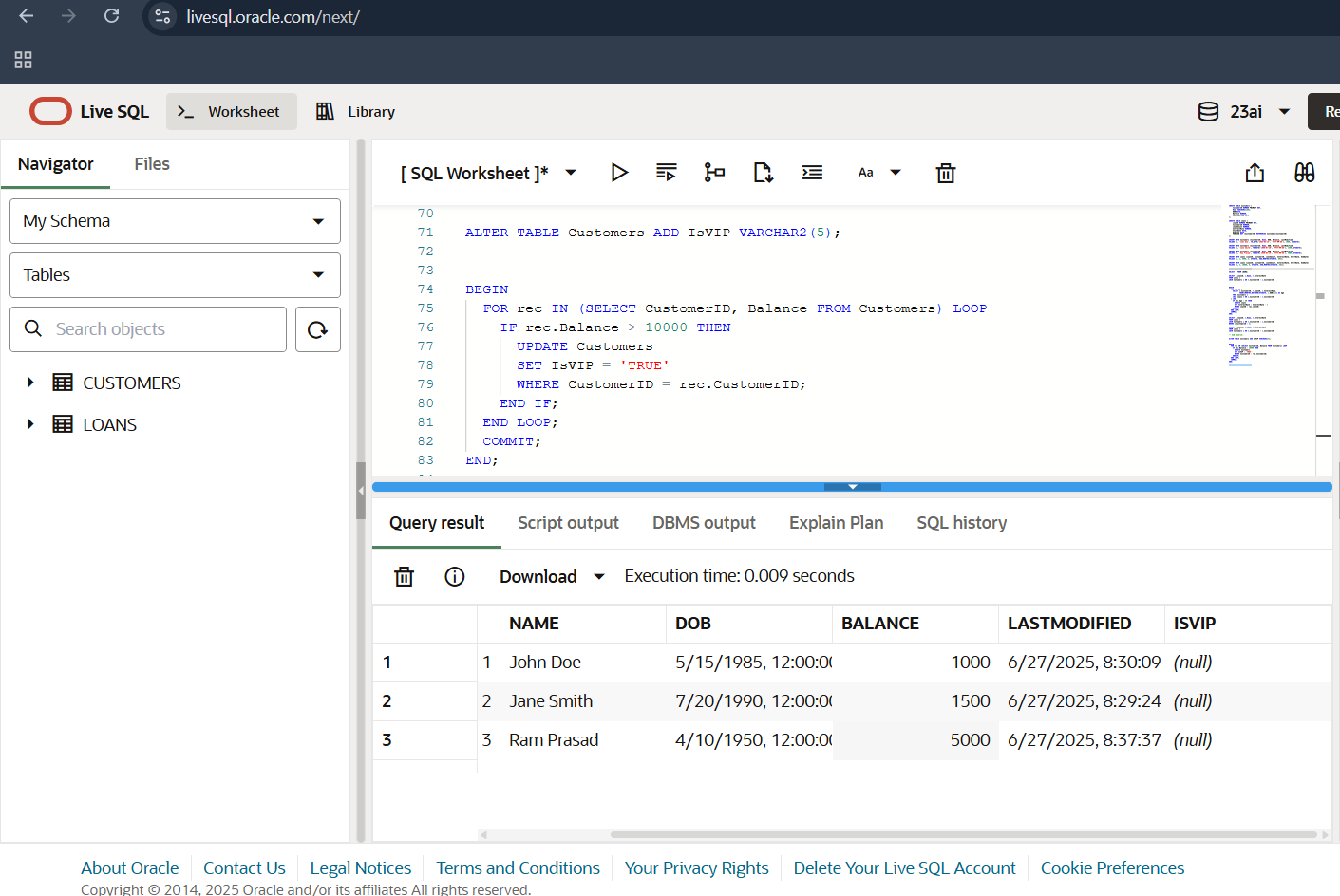
  END LOOP;

  COMMIT;

END;

****

**OUTPUT AFTER APPLYING PL/SQL:**

****

**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

**INSERT IN LOANS:**

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (3, 2, 8000, 7, SYSDATE, SYSDATE + 10);

**PL/SQL Code:**

SET SERVEROUTPUT ON;

BEGIN

  FOR rec IN (

    SELECT c.Name, l.EndDate

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

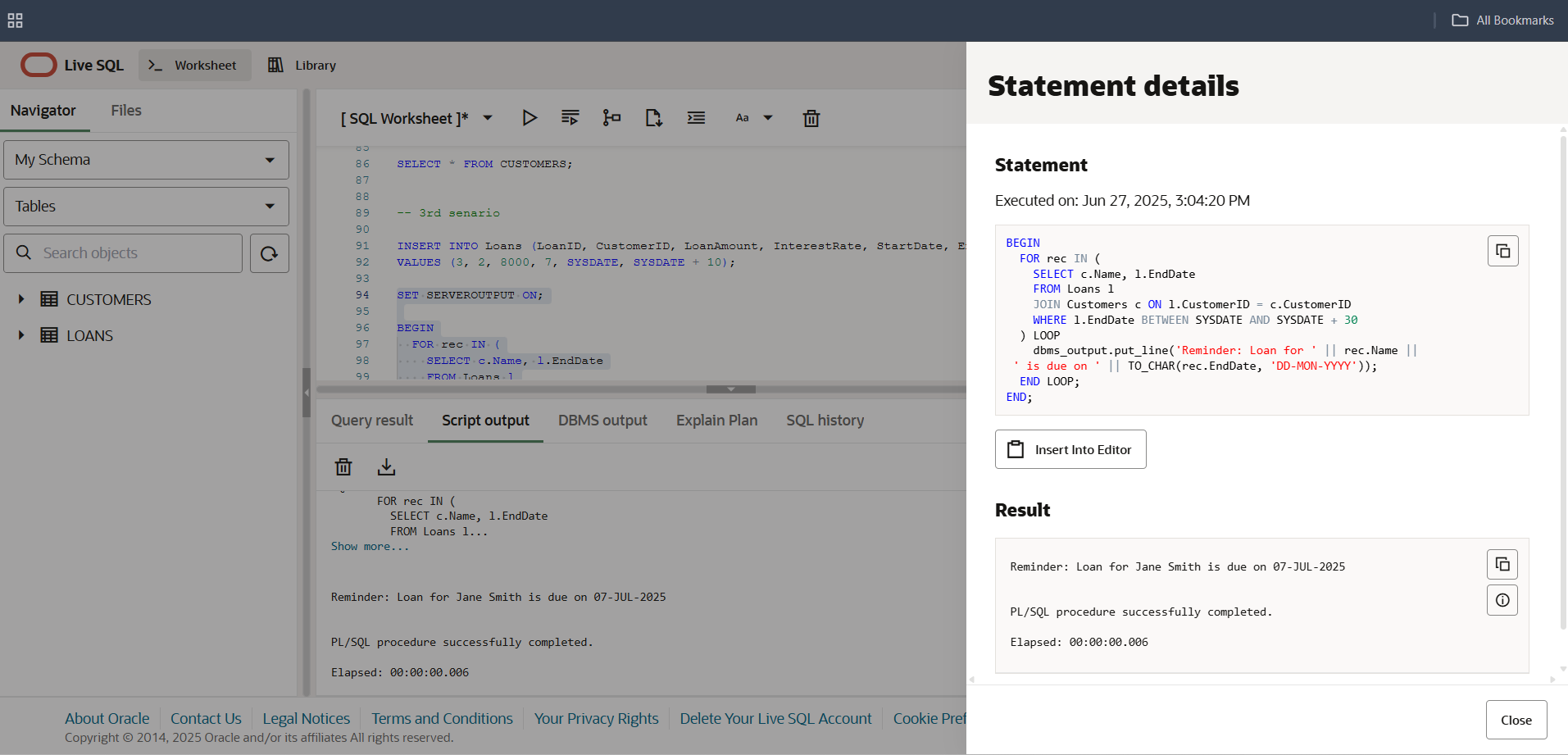
  ) LOOP

    dbms\_output.put\_line('Reminder: Loan for ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

  END LOOP;

END;

**OUTPUT:**

****

**Exercise 2: Error Handling**

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

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**PL/SQL Code:**

EXEC SafeTransferFunds(1, 2, 500);

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

  p\_from\_account IN NUMBER,

  p\_to\_account   IN NUMBER,

  p\_amount       IN NUMBER

)

IS

  v\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_balance

  FROM Accounts

  WHERE AccountID = p\_from\_account;

  IF v\_balance < p\_amount THEN

    RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');

  END IF;

  UPDATE Accounts

  SET Balance = Balance - p\_amount

  WHERE AccountID = p\_from\_account;

  UPDATE Accounts

  SET Balance = Balance + p\_amount

  WHERE AccountID = p\_to\_account;

  COMMIT;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    ROLLBACK;

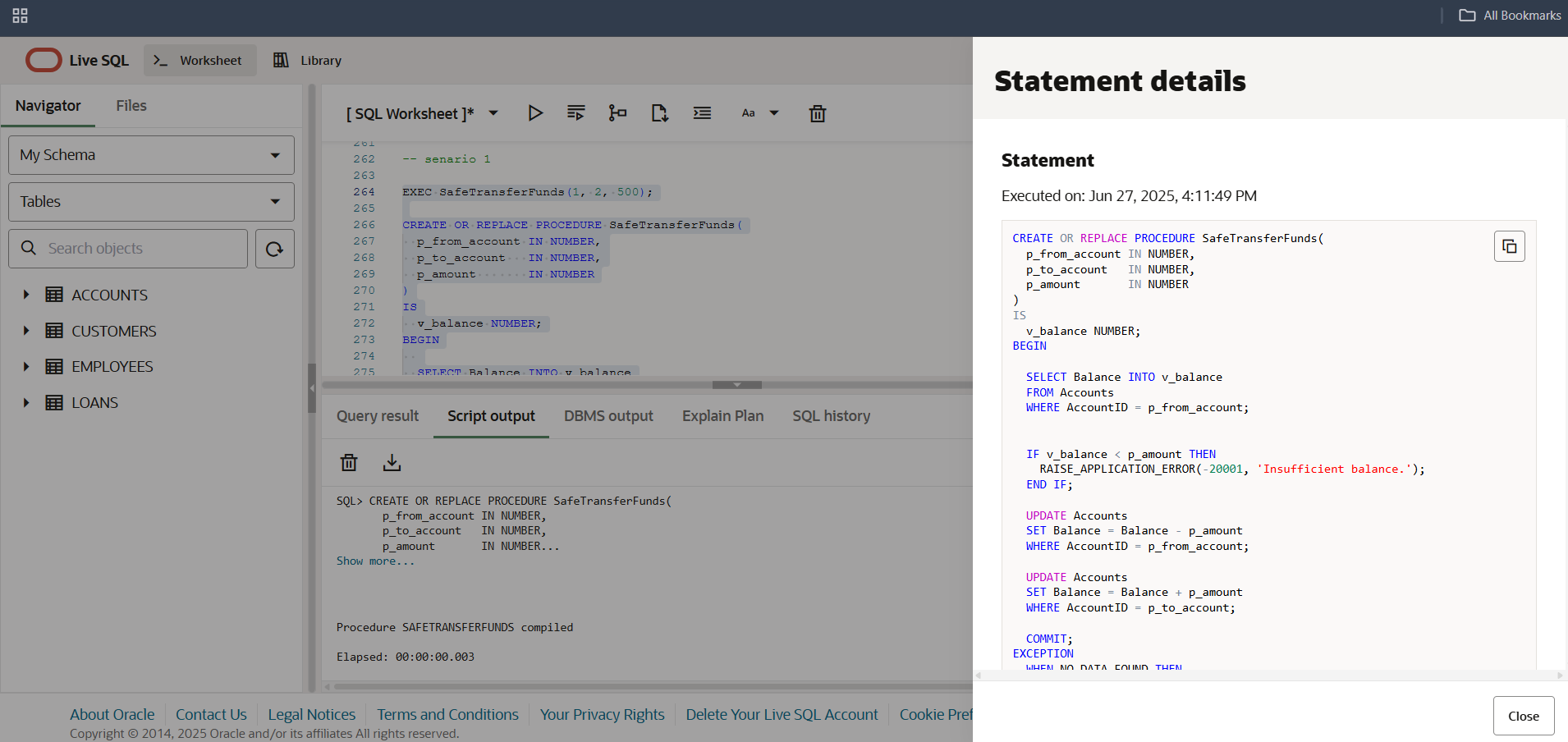
    DBMS\_OUTPUT.PUT\_LINE('Error: One of the accounts does not exist.');

  WHEN OTHERS THEN

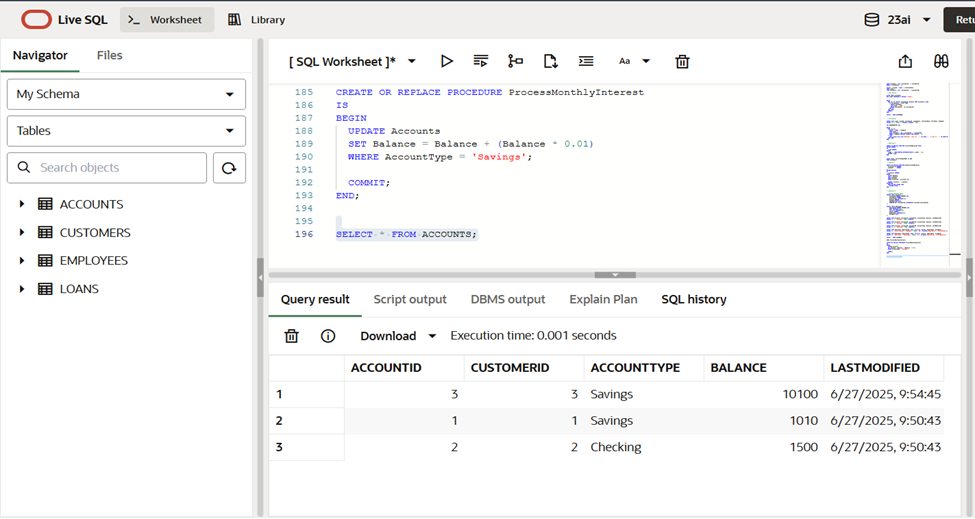
    ROLLBACK;

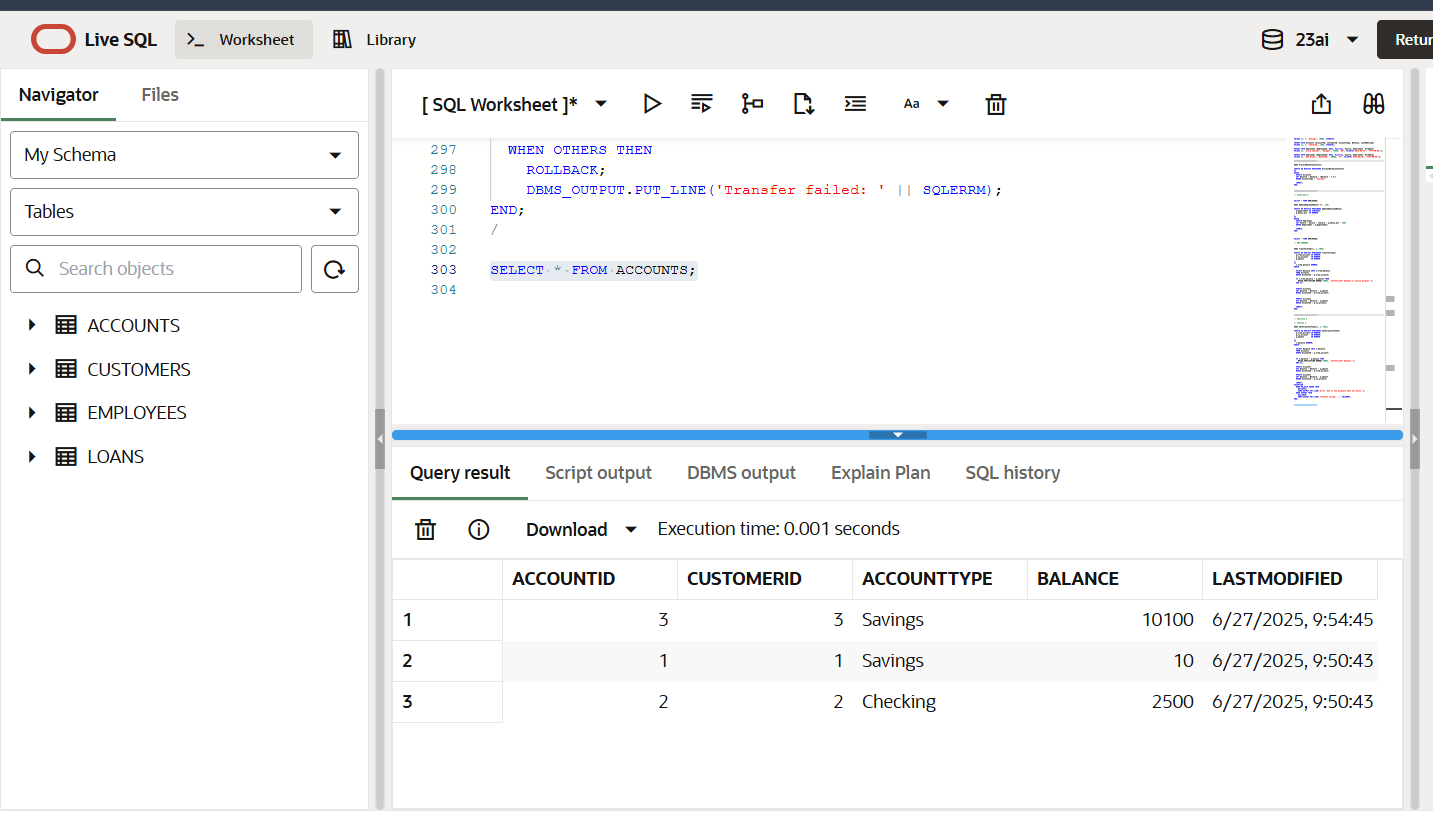
    DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

****

**OUTPUT BEFORE:**

****

**OUTPUT AFTER: **

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

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

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

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**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:**

EXEC ProcessMonthlyInterest;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

  UPDATE Accounts

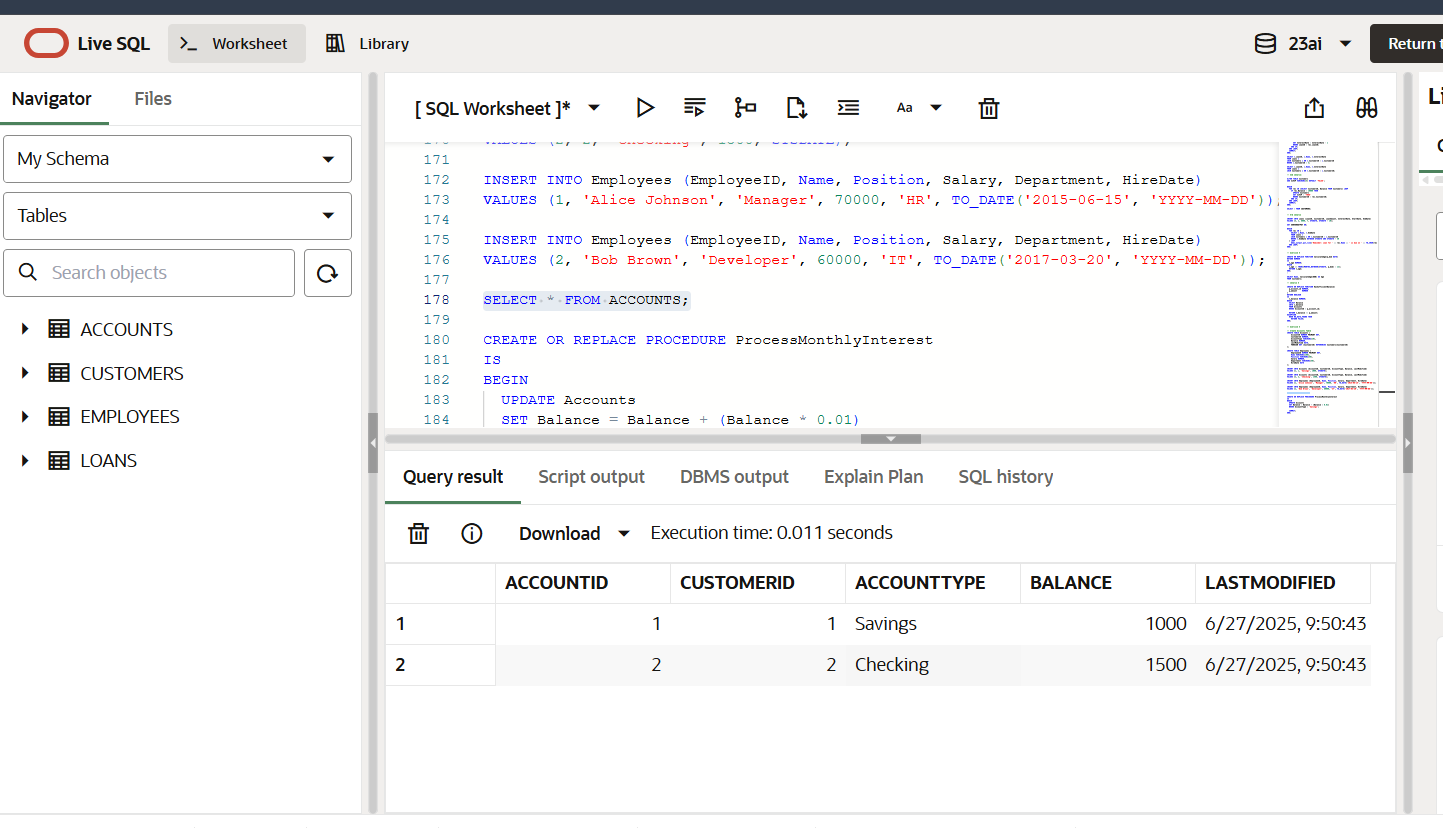
  SET Balance = Balance + (Balance \* 0.01)

  WHERE AccountType = 'Savings';

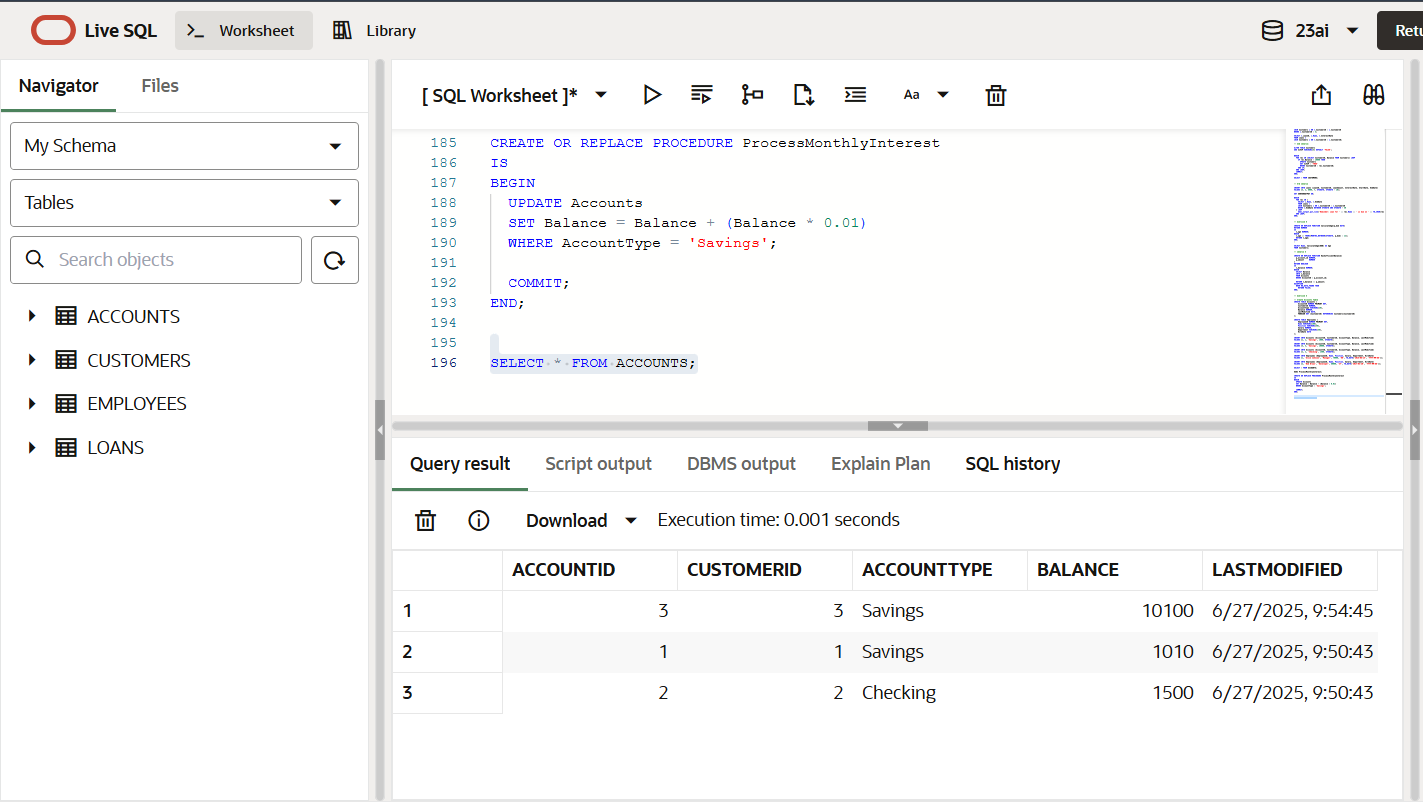
  COMMIT;

END;

**OUTPUT BEFORE:**

****

**OUTPUT AFTER:**

****

**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:**

EXEC UpdateEmployeeBonus('IT', 10);

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

  p\_department IN VARCHAR2,

  p\_bonus\_pct  IN NUMBER

)

IS

BEGIN

  UPDATE Employees

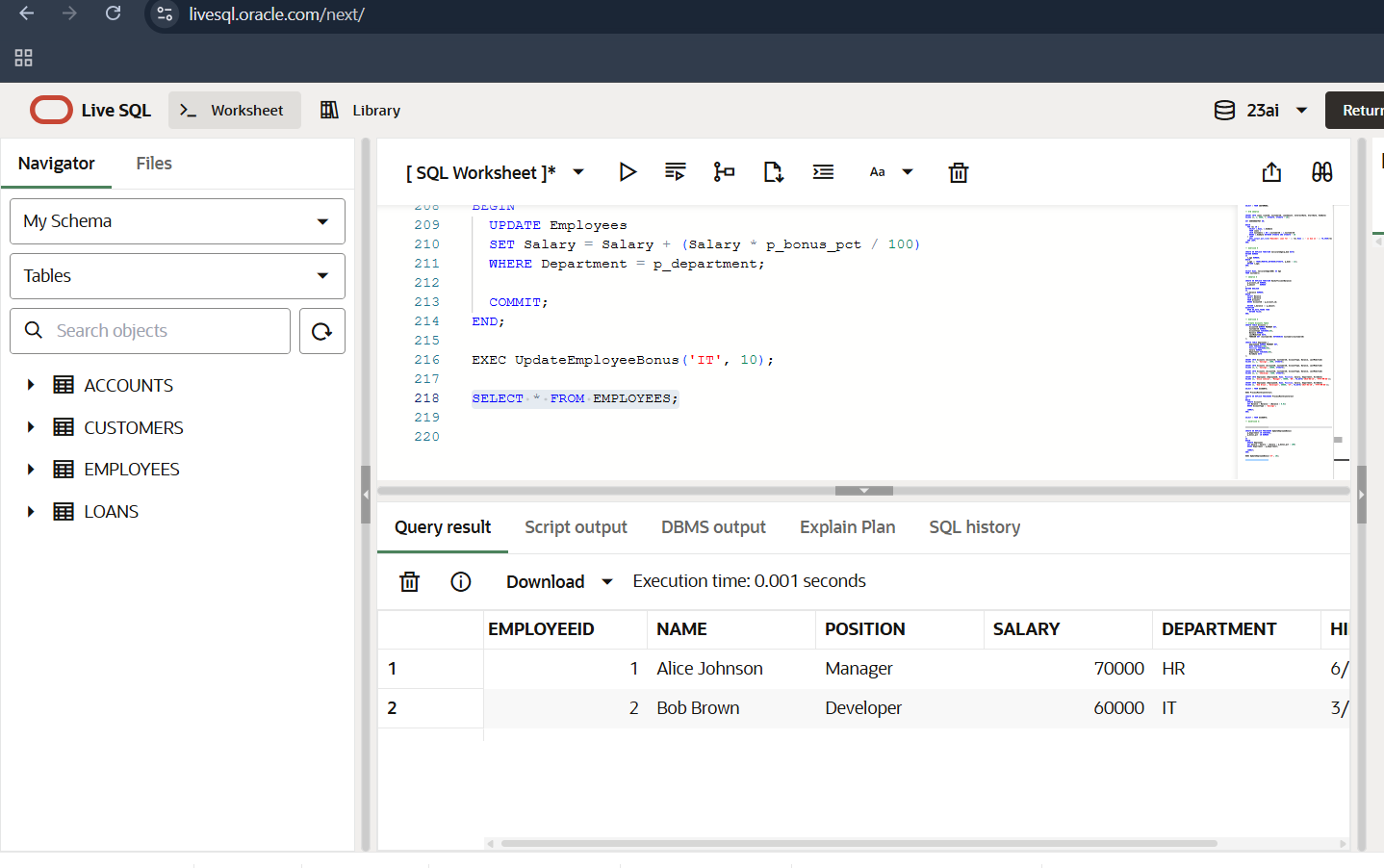
  SET Salary = Salary + (Salary \* p\_bonus\_pct / 100)

  WHERE Department = p\_department;

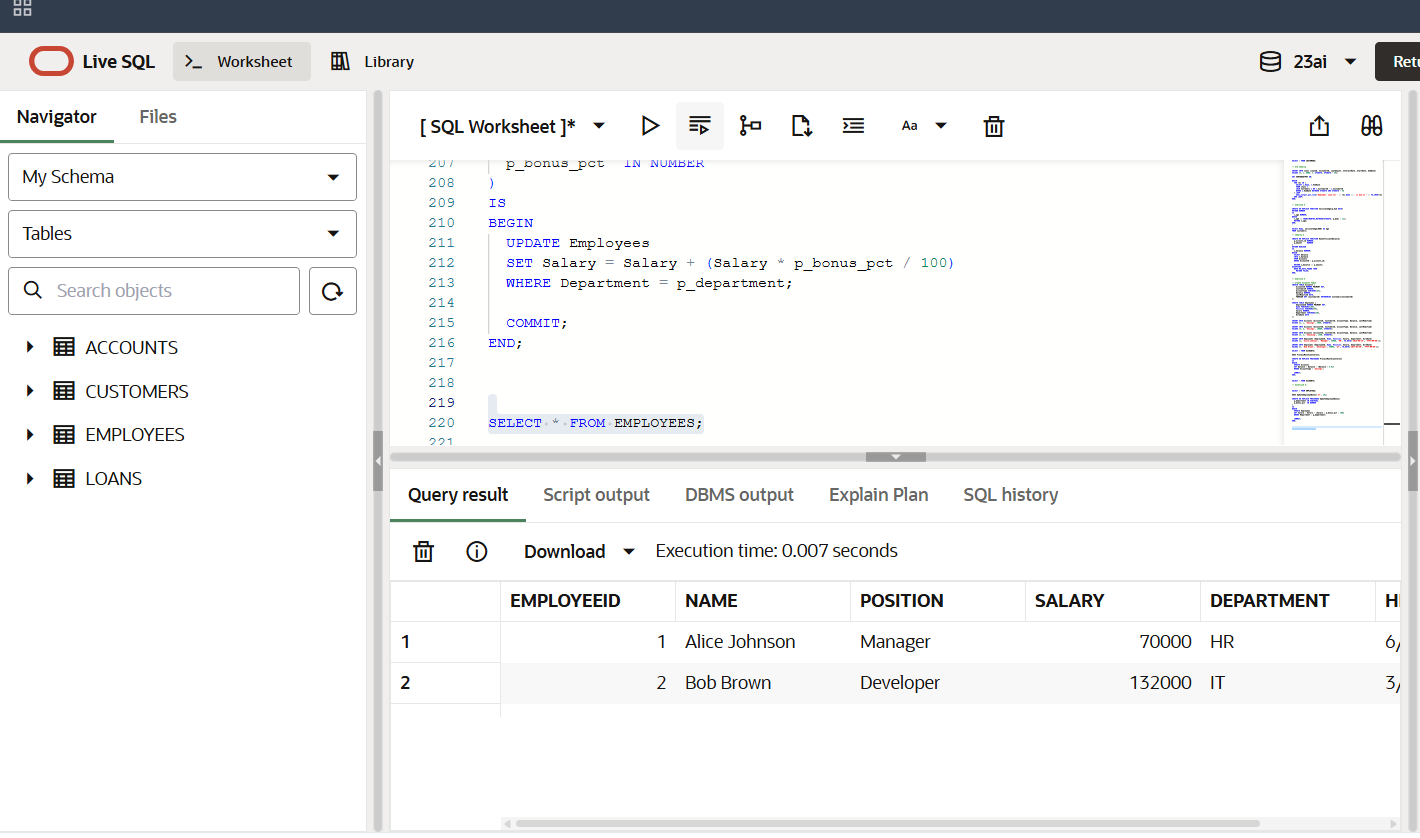
  COMMIT;

END;

**OUTPUT BEFORE:**

****

**OUTPUT AFTER:**

****

**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:**

EXEC TransferFunds(1, 2, 500);

CREATE OR REPLACE PROCEDURE TransferFunds(

  p\_from\_account IN NUMBER,

  p\_to\_account   IN NUMBER,

  p\_amount       IN NUMBER

)

IS

  v\_from\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_from\_balance

  FROM Accounts

  WHERE AccountID = p\_from\_account;

  IF v\_from\_balance < p\_amount THEN

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

  END IF;

  UPDATE Accounts

  SET Balance = Balance - p\_amount

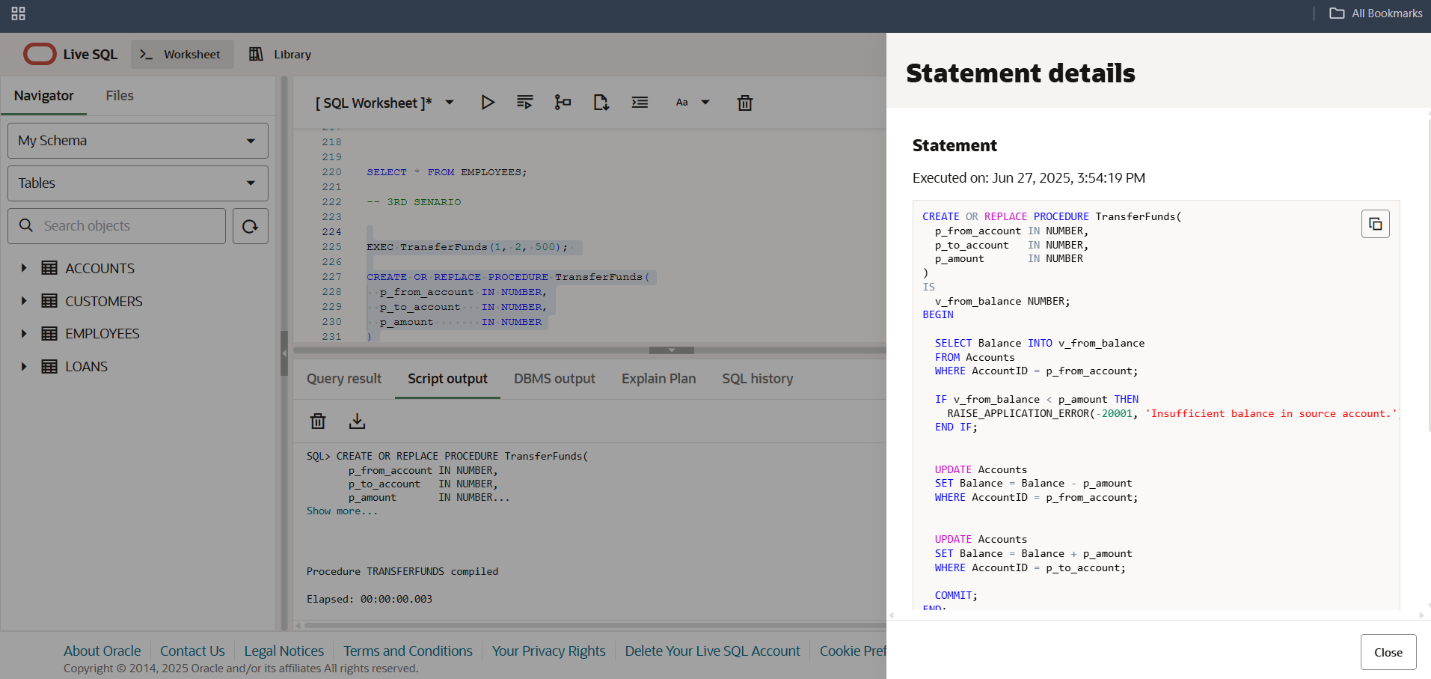
  WHERE AccountID = p\_from\_account;

  UPDATE Accounts

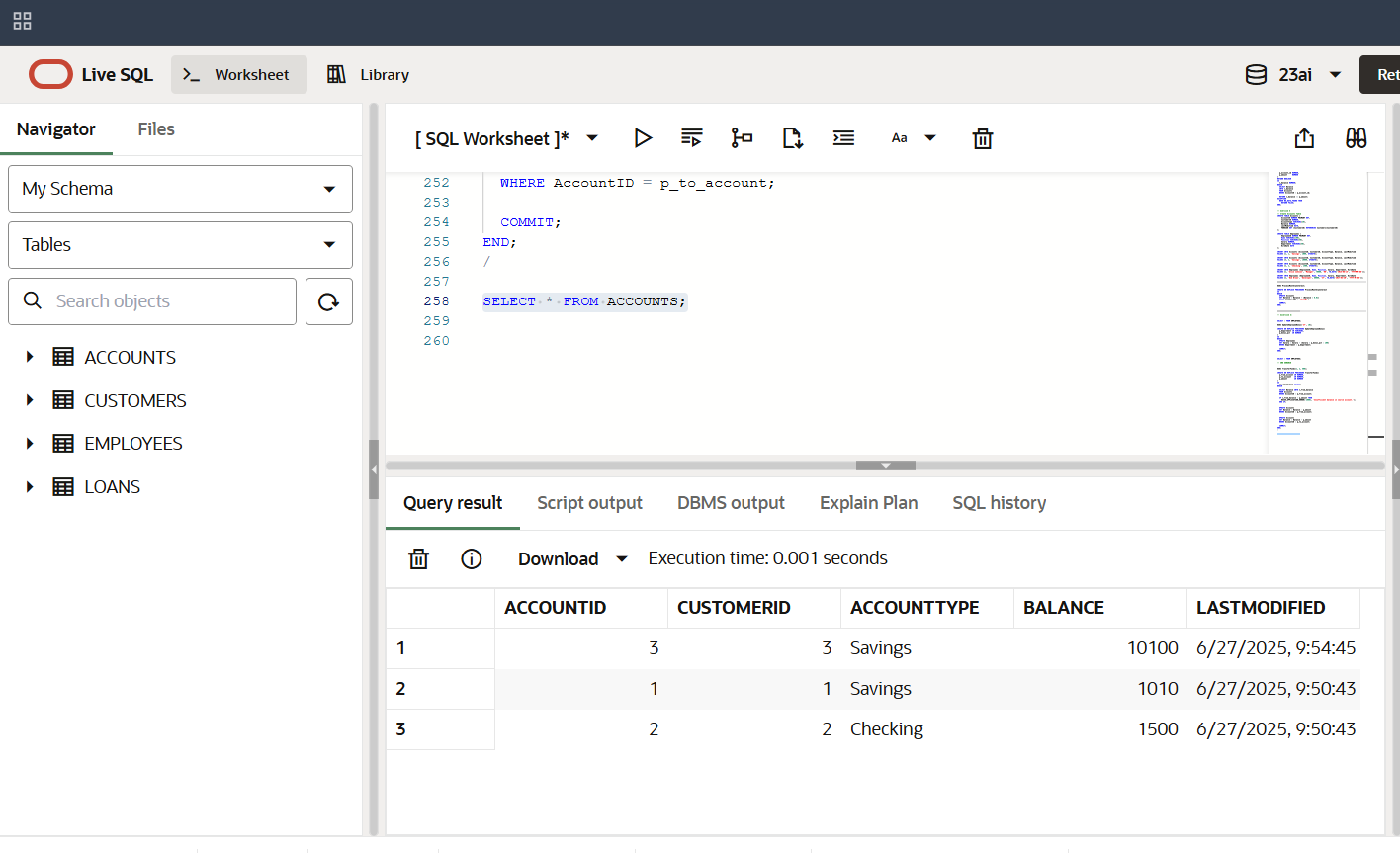
  SET Balance = Balance + p\_amount

  WHERE AccountID = p\_to\_account;

COMMIT;

END; ****

**OUTPUT :**

****

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**PL/SQL Code:** CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER

IS

  v\_age NUMBER;

BEGIN

  v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

  RETURN v\_age;

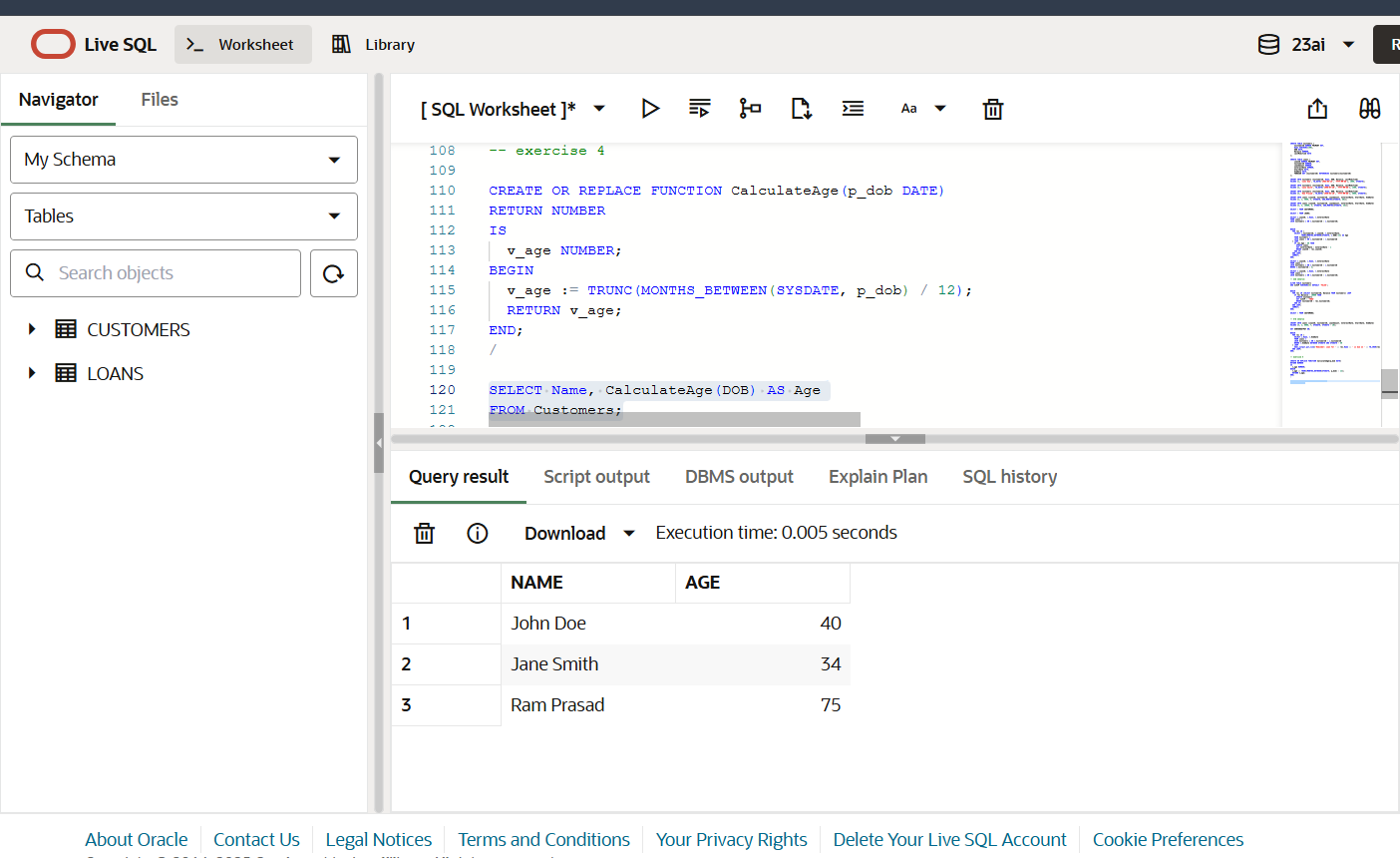
END;

/

SELECT Name, CalculateAge(DOB) AS Age

FROM Customers;

**OUTPUT:**



**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

**PL/SQL Code:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

  p\_account\_id NUMBER,

  p\_amount     NUMBER

)

RETURN BOOLEAN

IS

  v\_balance NUMBER;

BEGIN

  SELECT Balance

  INTO v\_balance

  FROM Accounts

  WHERE AccountID = p\_account\_id;

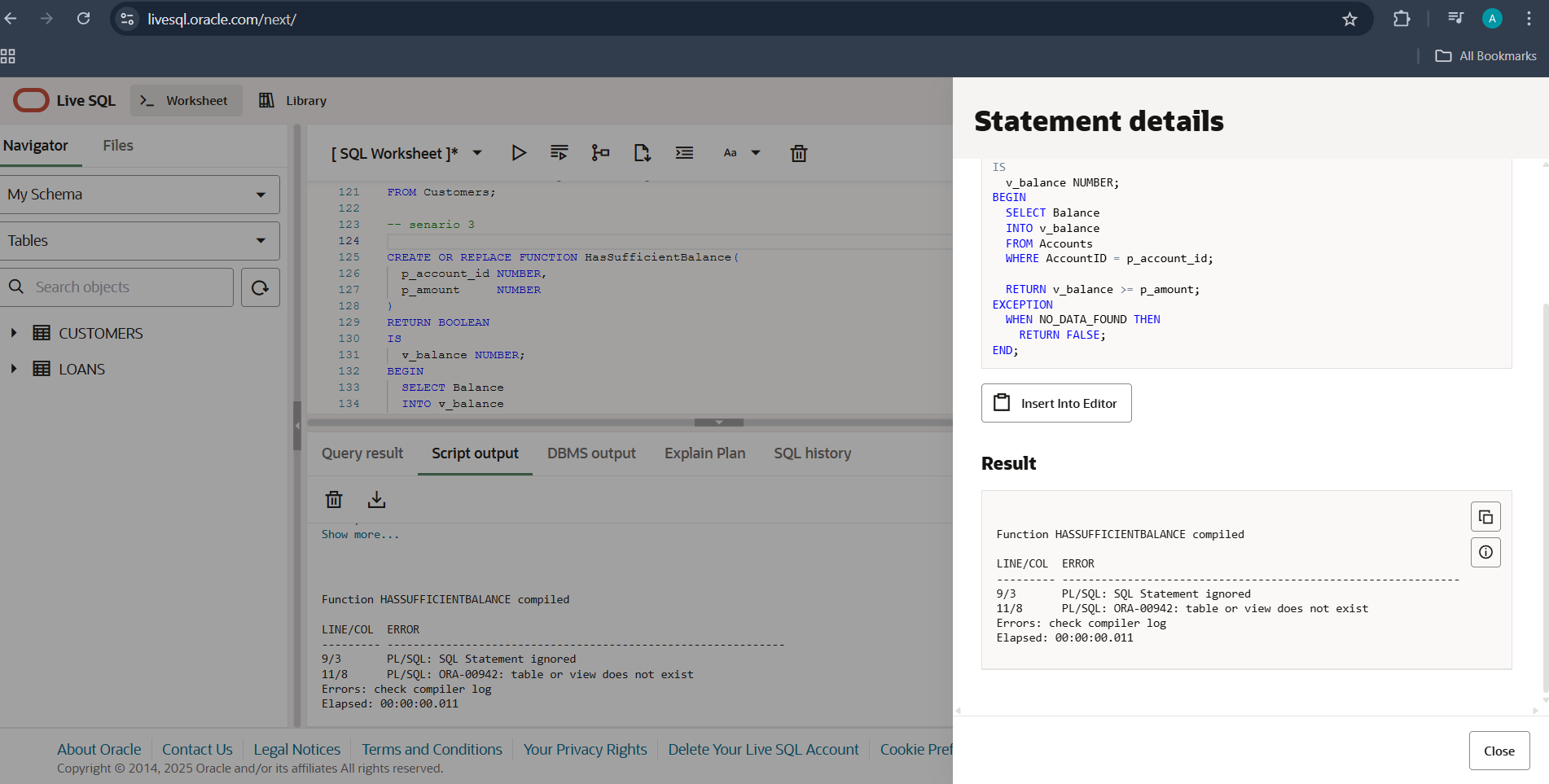
  RETURN v\_balance >= p\_amount;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    RETURN FALSE;

END;

****

**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

**Scenario 3:** Update the interest rate for all loans based on a new policy.

* + **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Schema to be Created**

*CREATE TABLE Customers (*

*CustomerID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*DOB DATE,*

*Balance NUMBER,*

*LastModified DATE*

*);*

*CREATE TABLE Accounts (*

*AccountID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*AccountType VARCHAR2(20),*

*Balance NUMBER,*

*LastModified DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Transactions (*

*TransactionID NUMBER PRIMARY KEY,*

*AccountID NUMBER,*

*TransactionDate DATE,*

*Amount NUMBER,*

*TransactionType VARCHAR2(10),*

*FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)*

*);*

*CREATE TABLE Loans (*

*LoanID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*LoanAmount NUMBER,*

*InterestRate NUMBER,*

*StartDate DATE,*

*EndDate DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Employees (*

*EmployeeID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*Position VARCHAR2(50),*

*Salary NUMBER,*

*Department VARCHAR2(50),*

*HireDate DATE*

*);*

**Example Scripts for Sample Data Insertion**

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

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

*INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)*

*VALUES (1, 1, 'Savings', 1000, SYSDATE);*

*INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)*

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

*INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)*

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

*INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)*

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

*INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)*

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*