**Cognizant Digital Nurture 4.0**

**Name: Siri Chandana Chittipolu**

**Email:** [**sirichittipolu11@gmail.com**](mailto:sirichittipolu11@gmail.com)

**Superset ID: 6386277**

**Mandatory Hands-On Exercises**

**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.

**Solution:**

**Step 1- Creating the Customers and Loans table:**

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

balance NUMBER(10, 2),

isvip VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE Loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

interest\_rate NUMBER(5, 2),

due\_date DATE,

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

);

**Step 2 : Inserting Rows :**

* Inserting into Customers:

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000.50, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 9500.00, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 20000.00, 'FALSE');

INSERT INTO Customers VALUES (4, 'Diana', 58, 11000.00, 'FALSE');

* Inserting into Loans:

INSERT INTO Loans VALUES (101, 1, 10.5, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, 12.0, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, 11.5, SYSDATE + 20);

INSERT INTO Loans VALUES (104, 4, 13.0, SYSDATE + 5);

**Step 3: Applying 1% Discount to Interest for Customers Over 60 :**

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (SELECT customer\_id, age, name FROM Customers) LOOP

IF cust.age > 60 THEN

UPDATE Loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = cust.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to: ' || cust.name);

ELSE

DBMS\_OUTPUT.PUT\_LINE('No discount for: ' || cust.name);

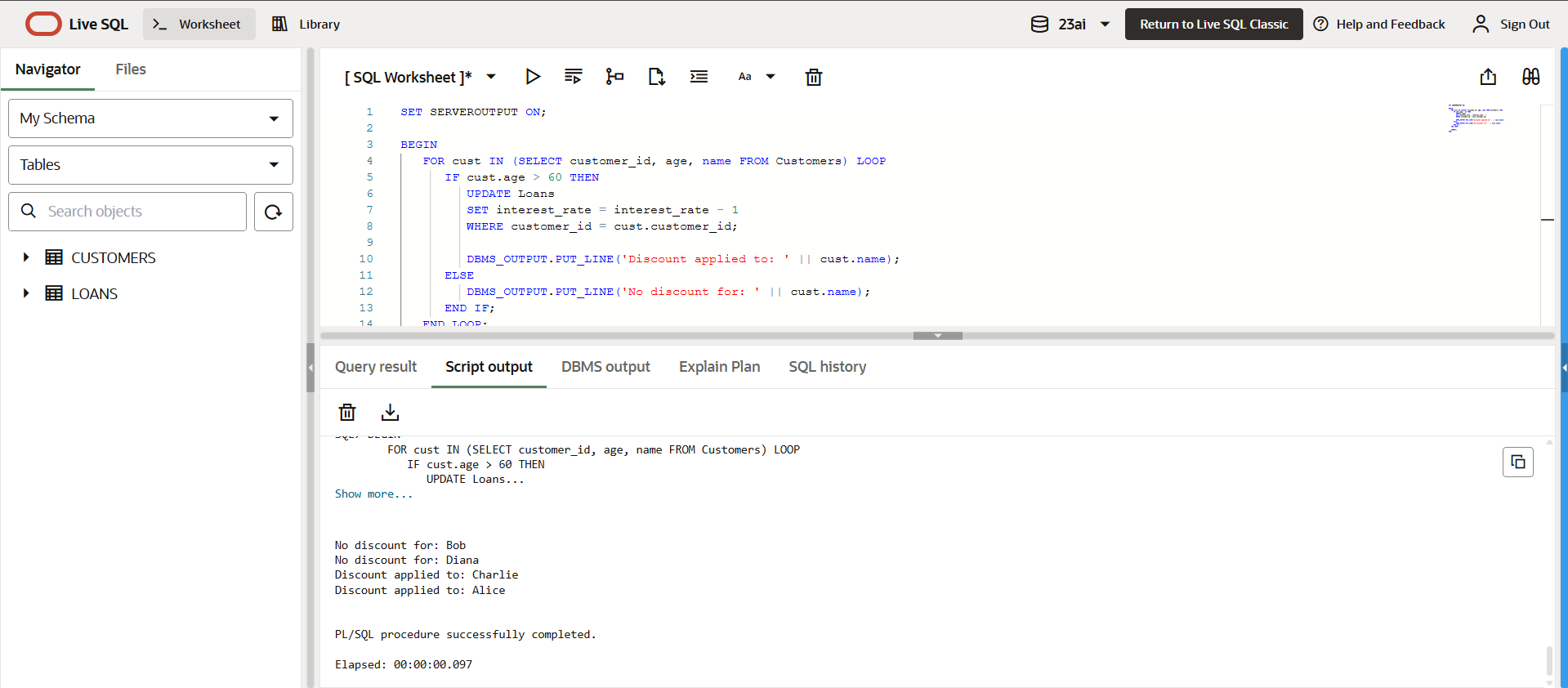
END IF;

END LOOP;

COMMIT;

END;

**Output :**

****

**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.

**Solution:**

* Set IsVIP = TRUE if Balance > 10000

BEGIN

FOR cust IN (SELECT customer\_id, balance, name FROM Customers) LOOP

IF cust.balance > 10000 THEN

UPDATE Customers

SET isvip = 'TRUE'

WHERE customer\_id = cust.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Promoted to VIP: ' || cust.name);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Not eligible for VIP: ' || cust.name);

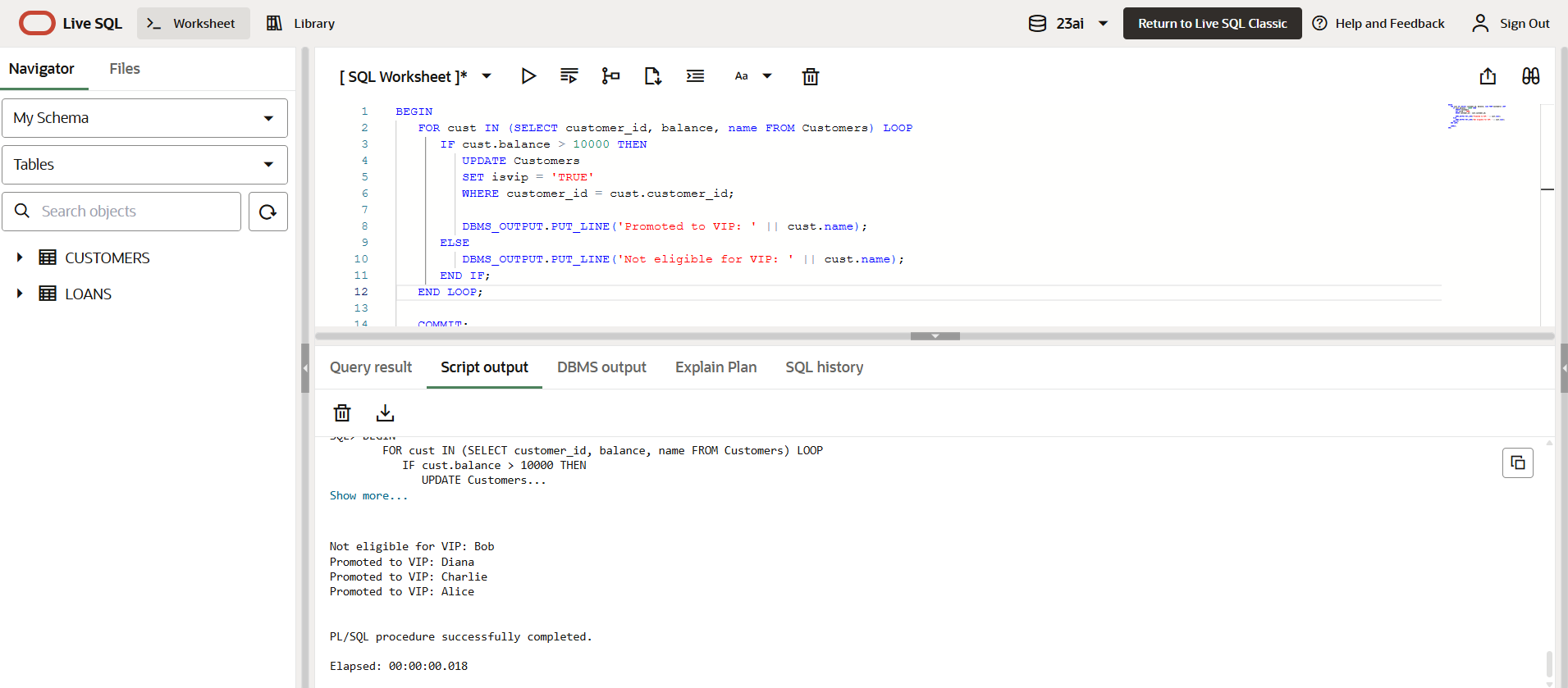
END IF;

END LOOP;

COMMIT;

END;

**Output:**

****

**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.

**Solution :**

* Sending Reminders for Loans Due in Next 30 Days

BEGIN

FOR loan\_rec IN (

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

FROM Loans l

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

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

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.name ||

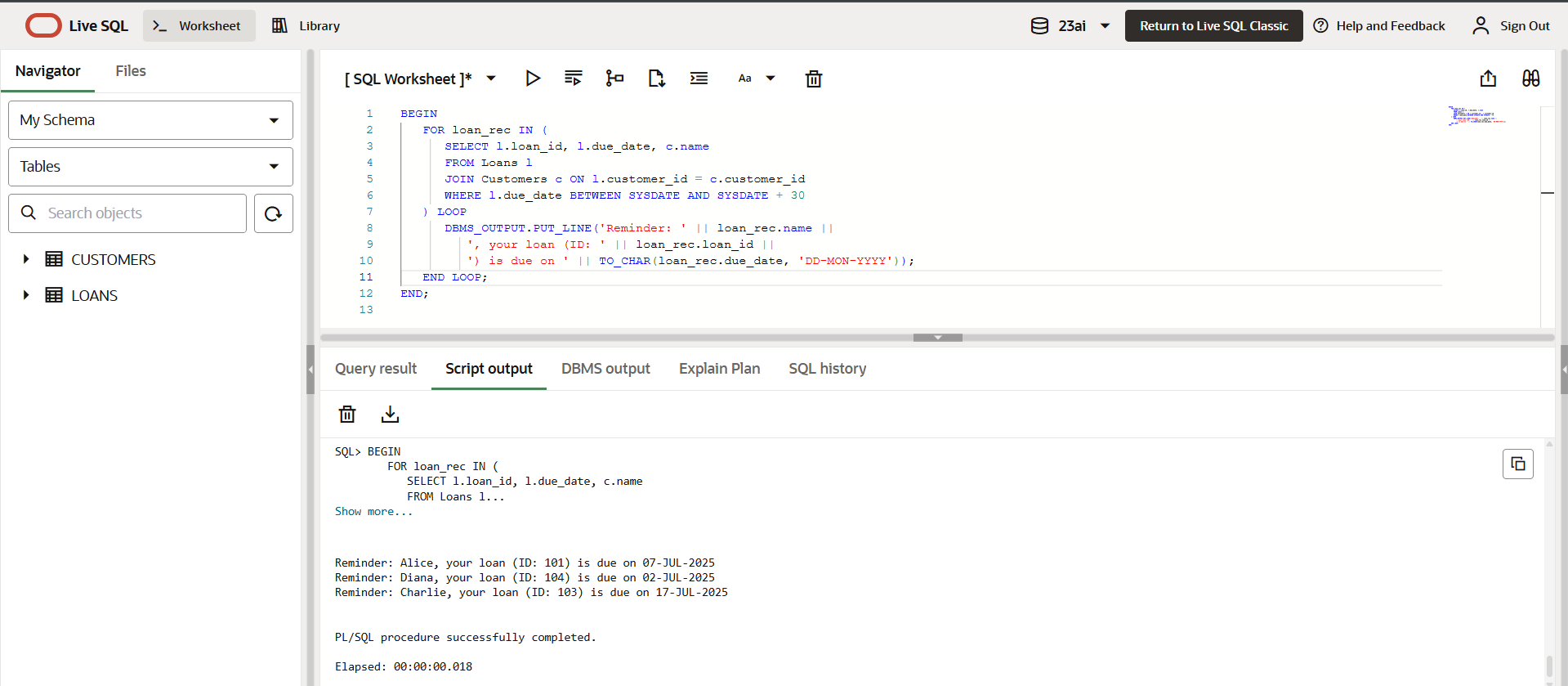
', your loan (ID: ' || loan\_rec.loan\_id ||

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

END LOOP;

END;

**Output:**

****

**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.

**Solution :**

**Step 1: Create Tables:**

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20),

balance NUMBER(10, 2)

);

CREATE TABLE Employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

department VARCHAR2(50),

salary NUMBER(10, 2)

);**Step 2: Inserting rows :**

* Inserting rows into Accounts

INSERT INTO Accounts VALUES (1, 101, 'Savings', 10000.00);

INSERT INTO Accounts VALUES (2, 102, 'Current', 5000.00);

INSERT INTO Accounts VALUES (3, 103, 'Savings', 8000.00);

INSERT INTO Accounts VALUES (4, 104, 'Savings', 2000.00);

* Inserting rows into Employees

INSERT INTO Employees VALUES (1, 'Alice', 'Sales', 50000);

INSERT INTO Employees VALUES (2, 'Bob', 'Sales', 45000);

INSERT INTO Employees VALUES (3, 'Charlie', 'HR', 40000);

**Step 3 : Applying 1% interest to all Savings accounts:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT account\_id, balance FROM Accounts WHERE account\_type = 'Savings') LOOP

UPDATE Accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_id = acc.account\_id;

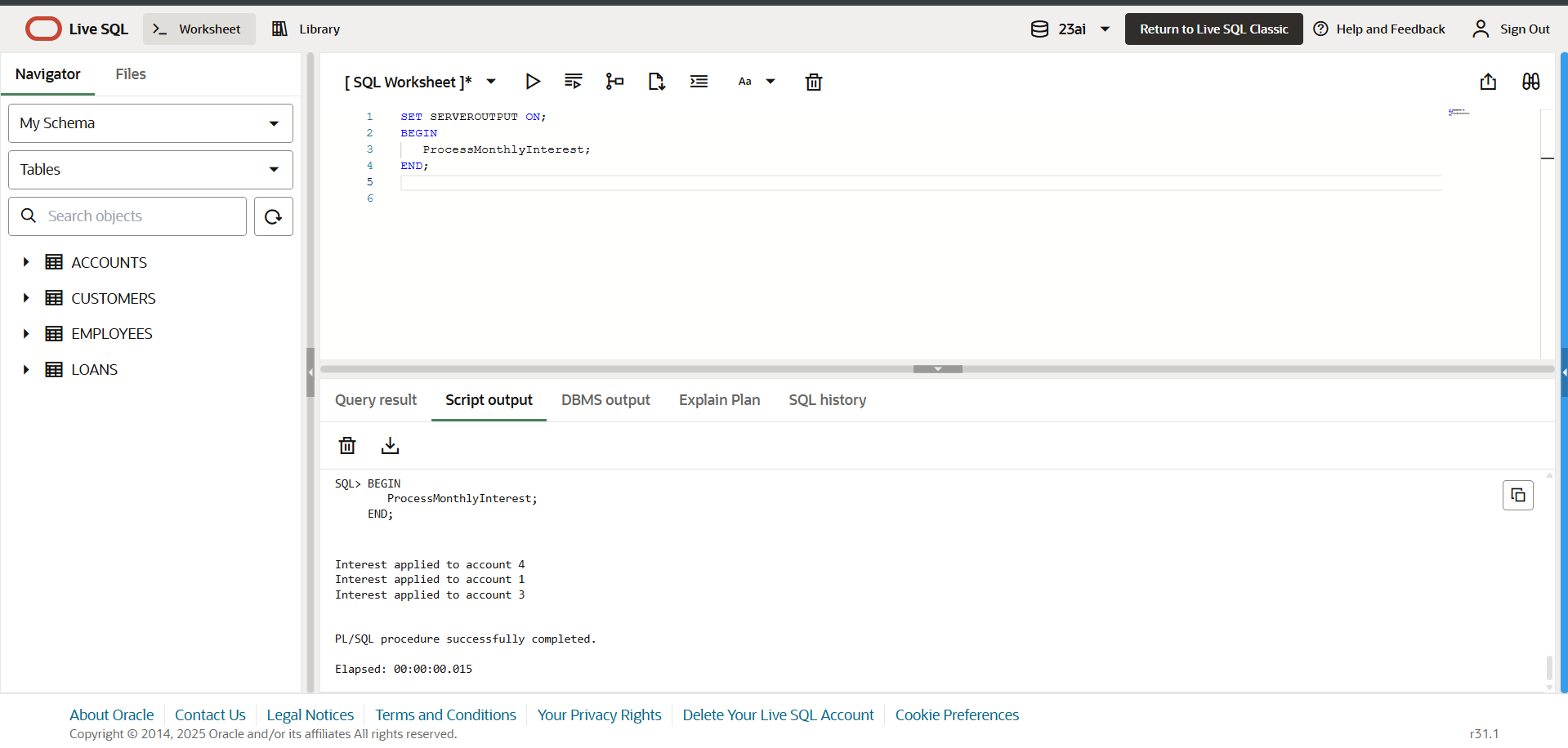
DBMS\_OUTPUT.PUT\_LINE('Interest applied to account ' || acc.account\_id);

END LOOP;

COMMIT;

END;

**Output:**



**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.

**Solution :**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

FOR emp IN (SELECT emp\_id, name, salary FROM Employees WHERE department = dept\_name) LOOP

UPDATE Employees

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

WHERE emp\_id = emp.emp\_id;

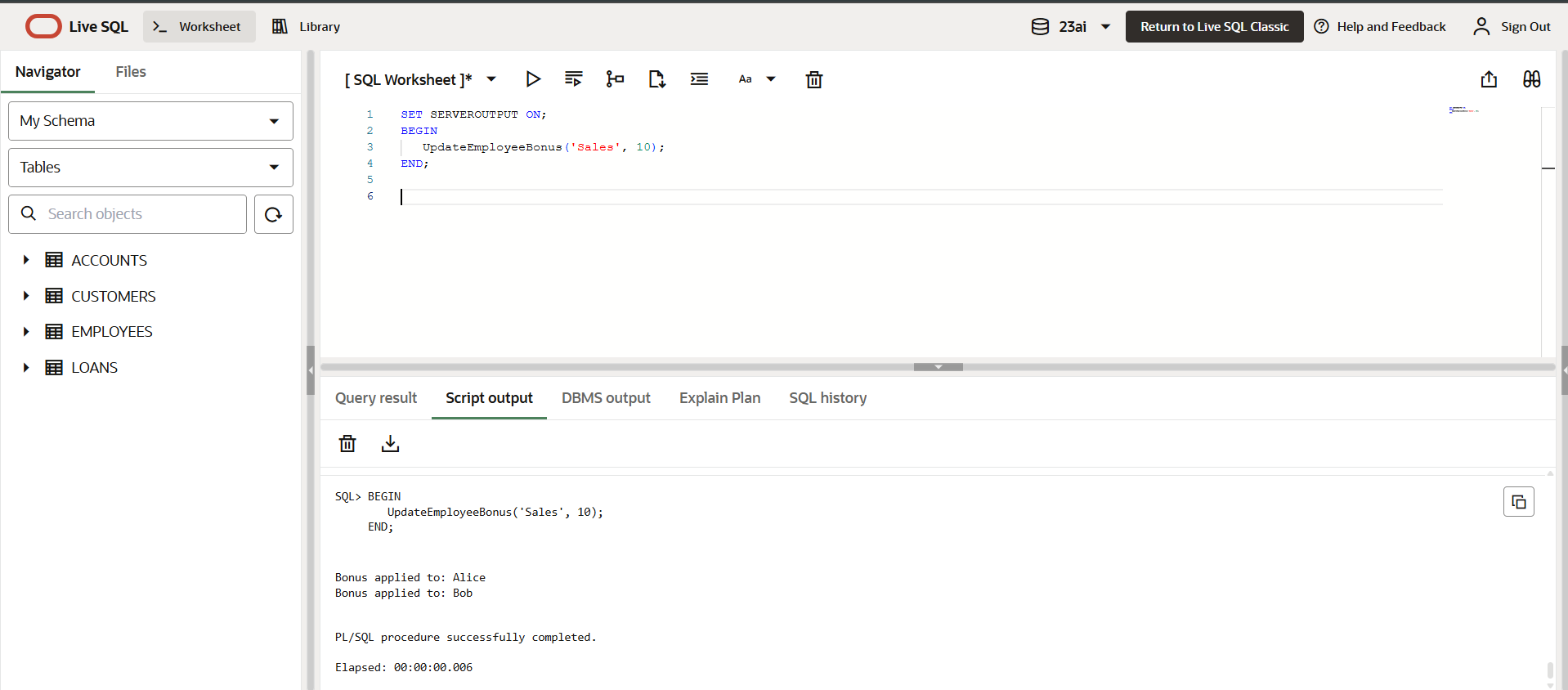
DBMS\_OUTPUT.PUT\_LINE('Bonus applied to: ' || emp.name);

END LOOP;

COMMIT;

END;

**Output :**

****

**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.

**Solution :**

CREATE OR REPLACE PROCEDURE TransferFunds (

   from\_account IN NUMBER,

   to\_account   IN NUMBER,

   amount       IN NUMBER

) IS

   from\_balance NUMBER;

BEGIN

   SELECT balance INTO from\_balance

   FROM Accounts

   WHERE account\_id = from\_account

   FOR UPDATE;

   IF from\_balance < amount THEN

      DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in account ' || from\_account);

      RETURN;

   END IF;

   UPDATE Accounts

   SET balance = balance - amount

   WHERE account\_id = from\_account;

   UPDATE Accounts

   SET balance = balance + amount

   WHERE account\_id = to\_account;

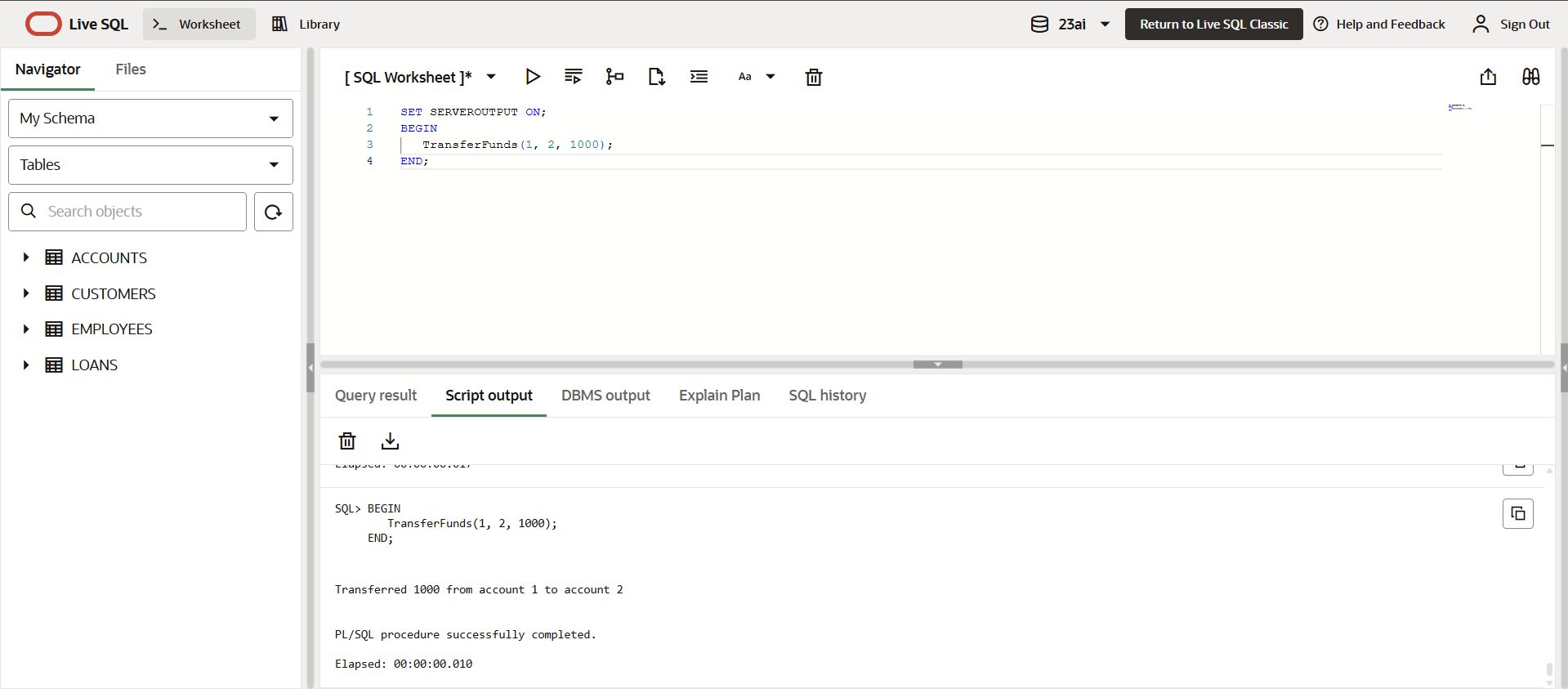
   DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount ||

      ' from account ' || from\_account || ' to account ' || to\_account);

   COMMIT;

END;

**Output:**

****