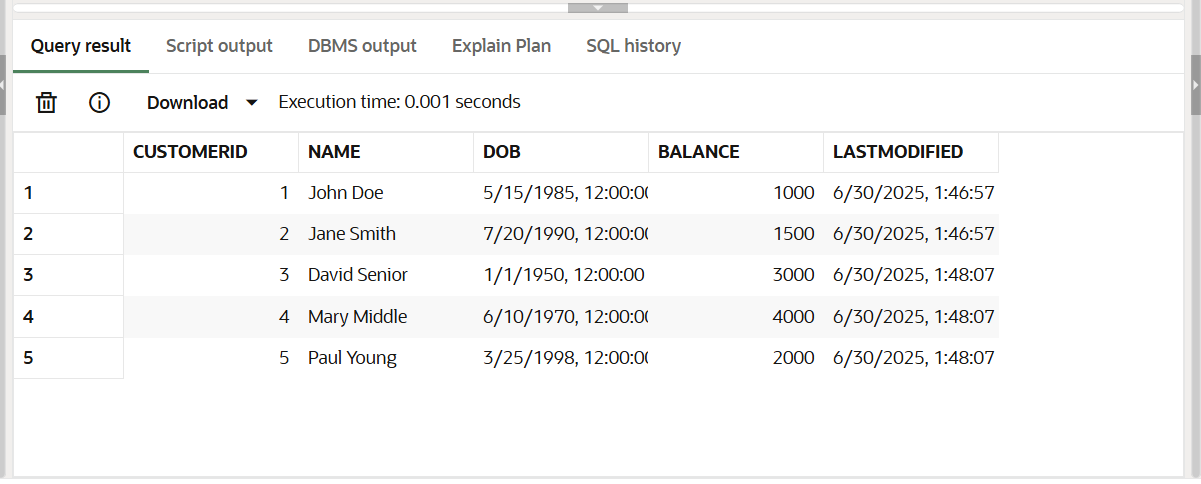
**Exercise 1: Control Structures**

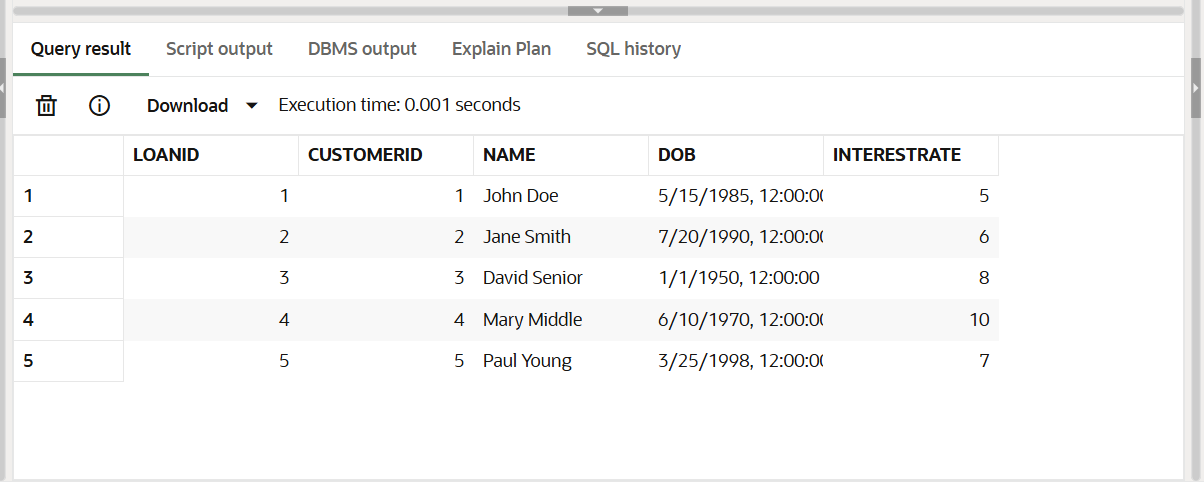
**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**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.

**CODE:**

SELECT \* FROM Customers;

SELECT \* FROM Loans;



**PL SQL CODE:**

BEGIN

FOR c IN (

SELECT l.LoanID

FROM Loans l

JOIN Customers cu ON l.CustomerID=cu.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE,cu.DOB)/12>60

)

LOOP

UPDATE Loans

SET InterestRate=InterestRate-1

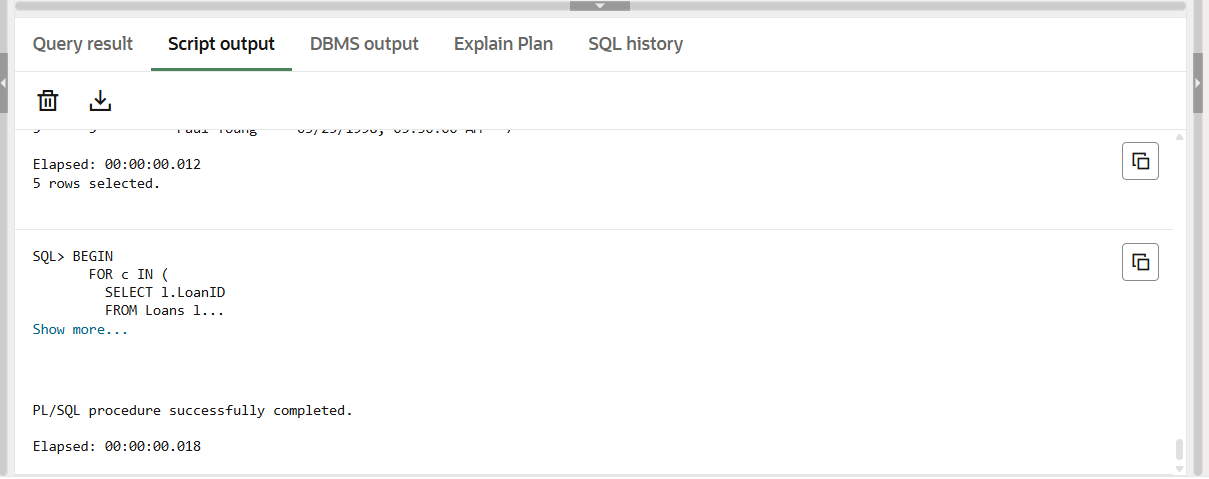
WHERE LoanID=c.LoanID;

END LOOP;

END;

/

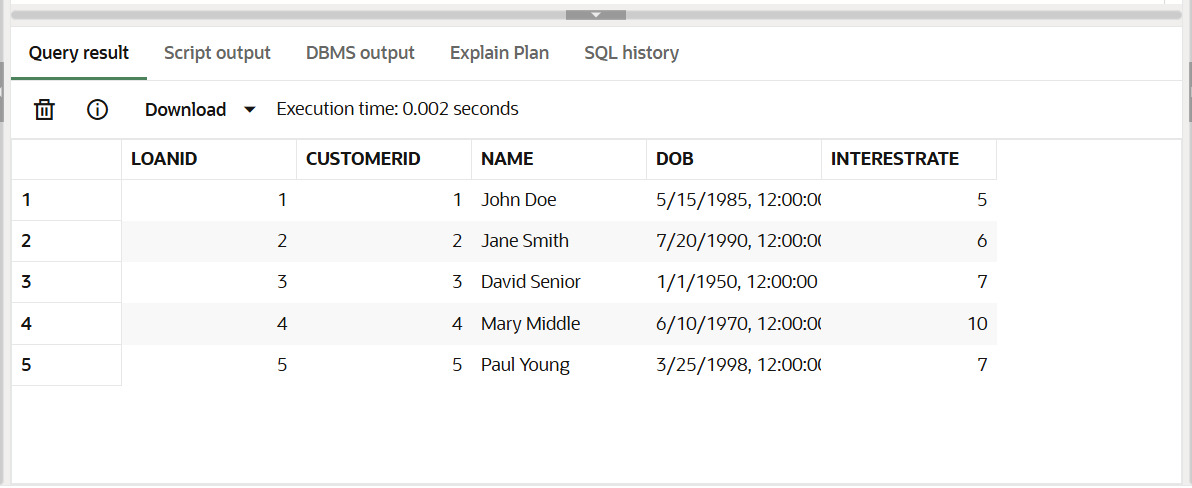
**OUTPUT:**



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

FROM Loans l

JOIN Customers c ON l.CustomerID=c.CustomerID;

**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.

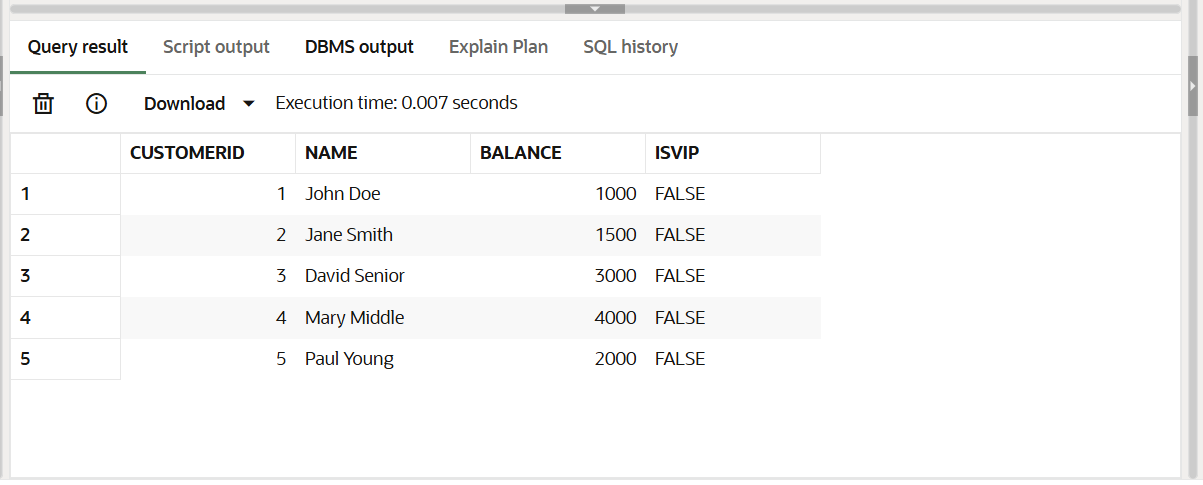
**CODE:**

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

UPDATE Customers SET IsVIP='FALSE';

COMMIT;

SELECT CustomerID,Name,Balance,IsVIP FROM Customers;

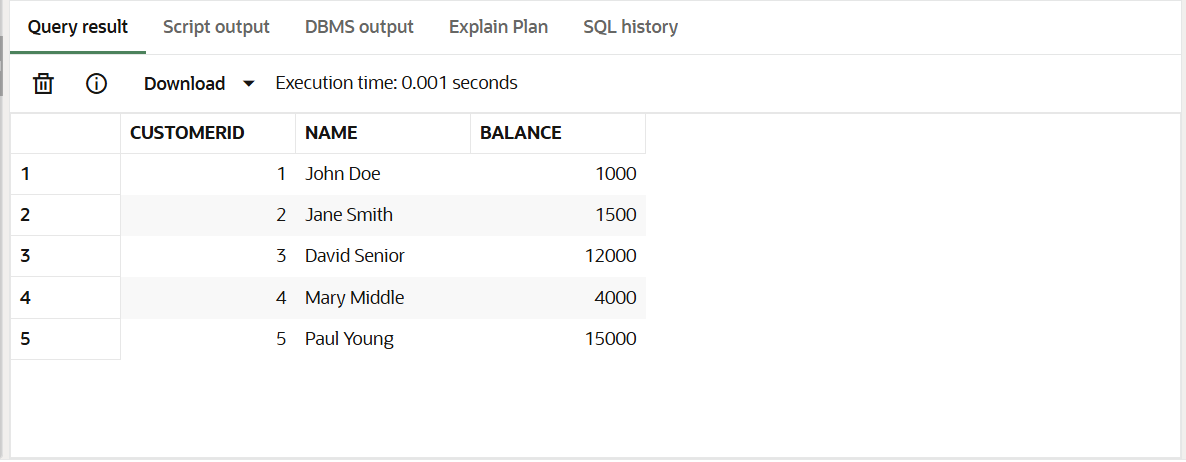
****

UPDATE Customers SET Balance = 12000 WHERE CustomerID = 3;

UPDATE Customers SET Balance = 15000 WHERE CustomerID = 5;

COMMIT;

SELECT CustomerID, Name, Balance FROM Customers;

****

**PLSQL CODE:**

BEGIN

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

LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

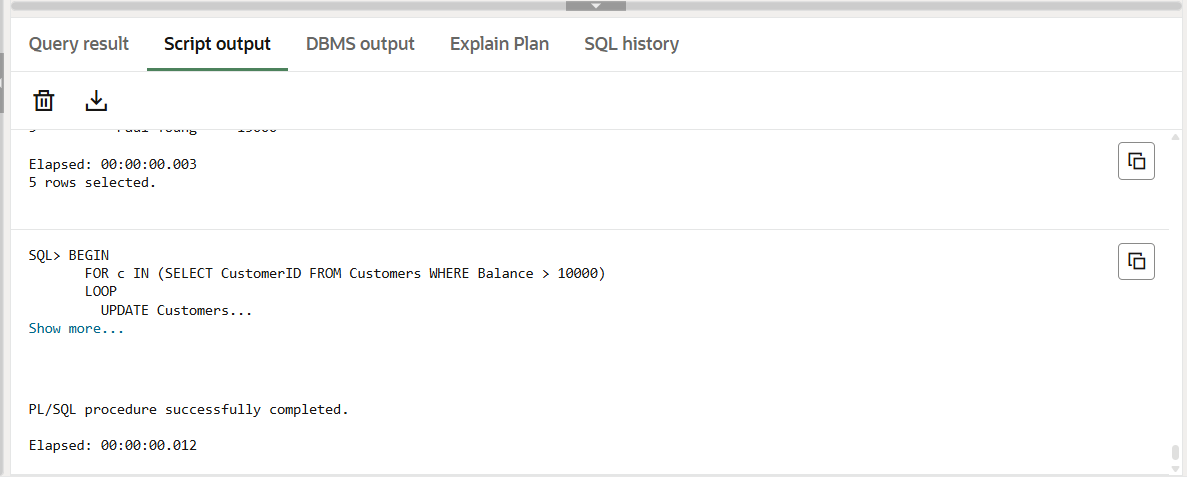
WHERE CustomerID = c.CustomerID;

END LOOP;

END;

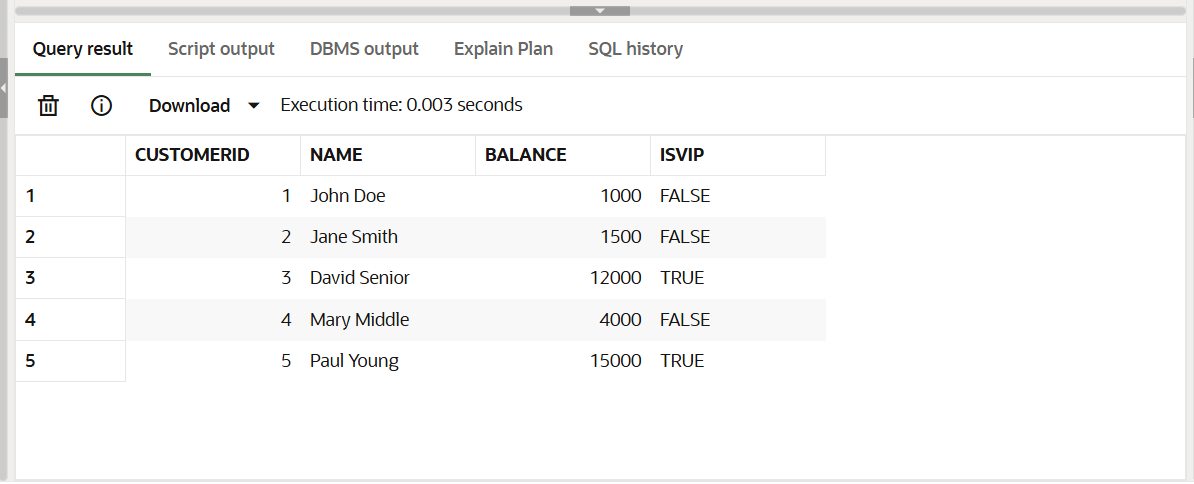
/

**OUTPUT:**

****

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

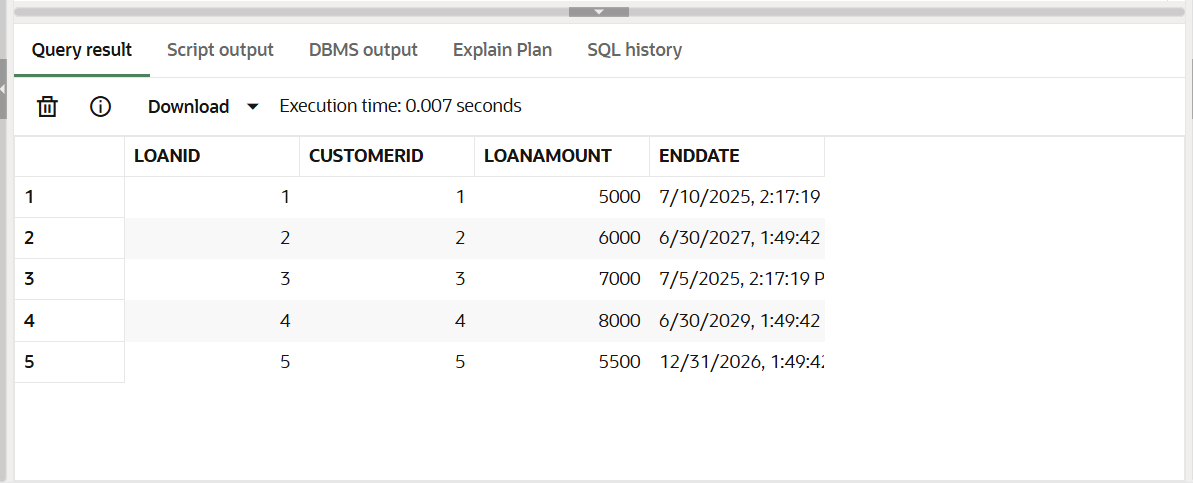
**OUTPUT:**

****

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

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**CODE:**

**SELECT LoanID, CustomerID, LoanAmount, EndDate FROM Loans;**

**PL SQL CODE:**

BEGIN

FOR l IN (

SELECT l.LoanID, l.CustomerID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || l.LoanID ||

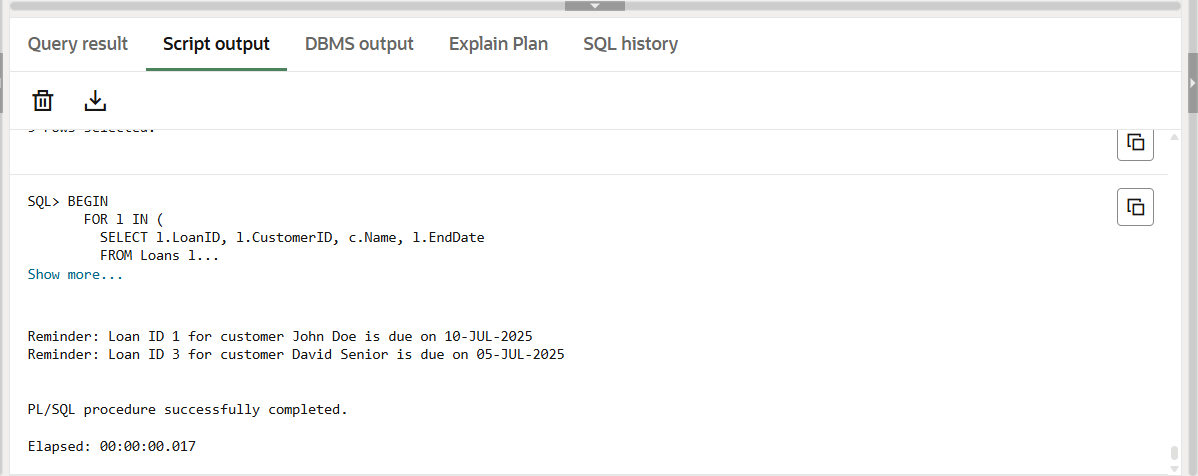
' for customer ' || l.Name ||

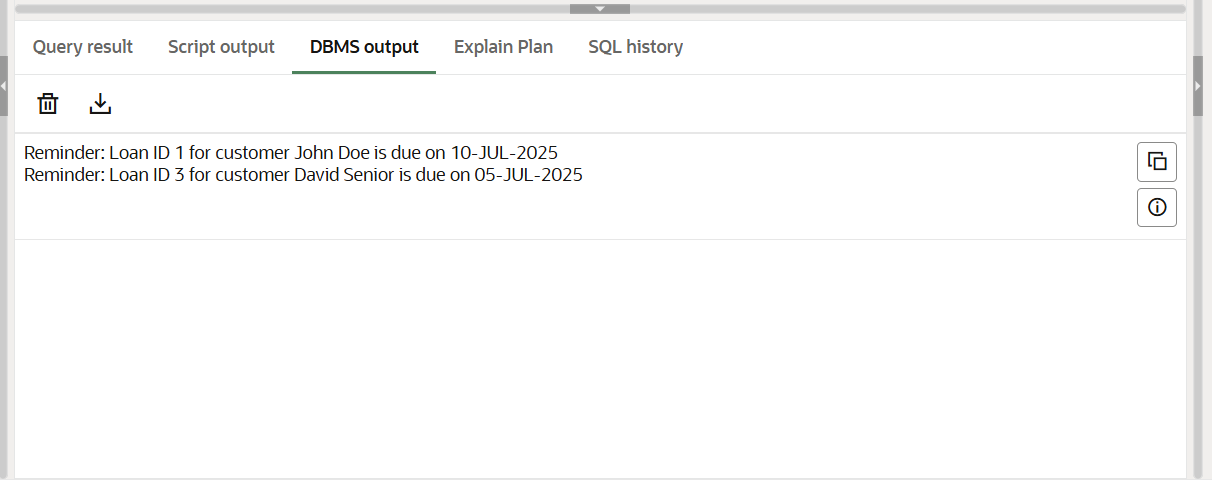
' is due on ' || TO\_CHAR(l.EndDate, 'DD-MON-YYYY'));

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.

**CODE:**

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

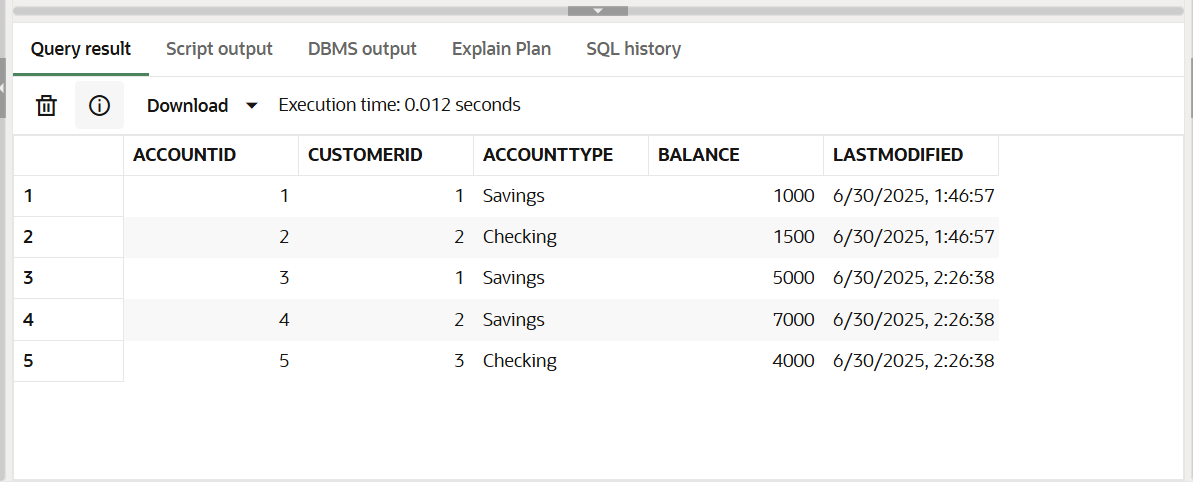
(3, 1, 'Savings', 5000, SYSDATE),

(4, 2, 'Savings', 7000, SYSDATE),

(5, 3, 'Checking', 4000, SYSDATE);

COMMIT;

SELECT \* FROM Accounts;



**PL SQL CODE:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR a IN (SELECT AccountID FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = a.AccountID;

END LOOP;

END;

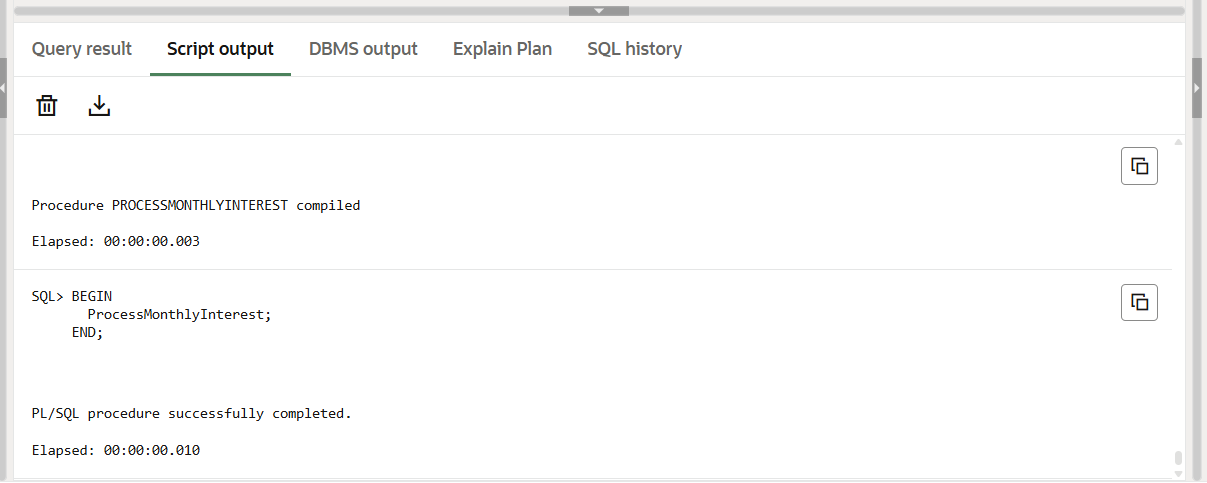
/

BEGIN

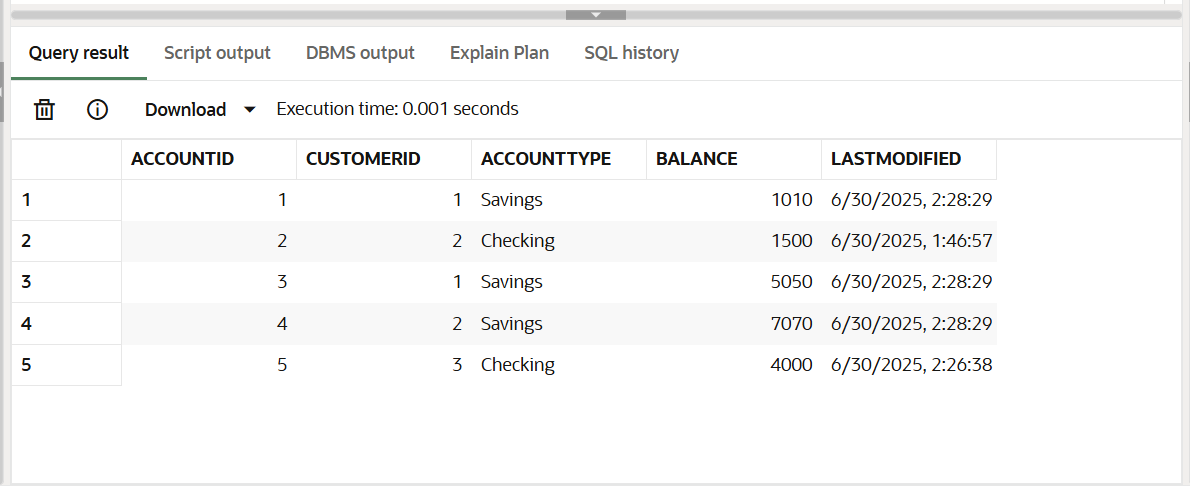
ProcessMonthlyInterest;

END;

/



SELECT \* FROM Accounts;



**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.

**CODE:**

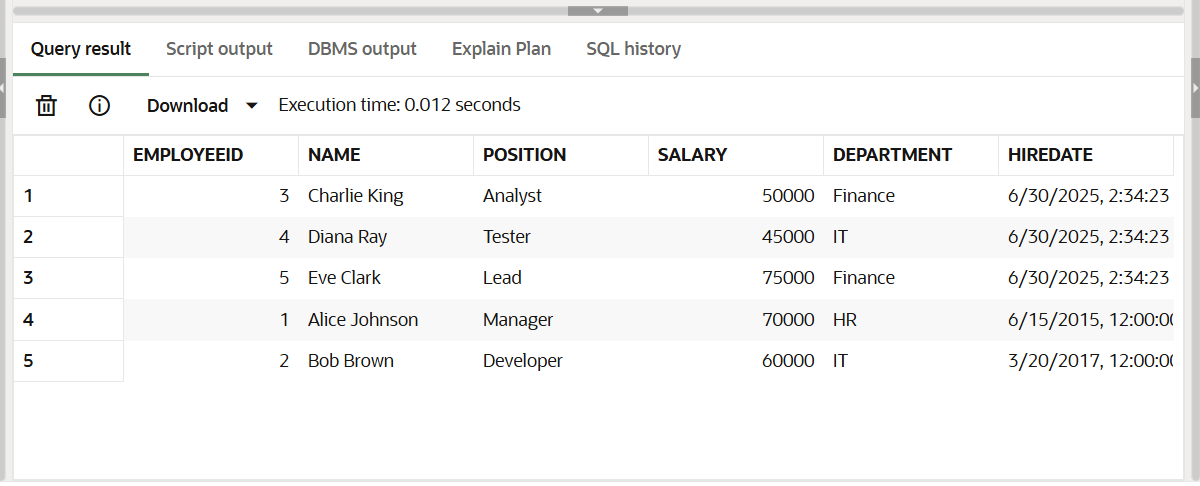
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES

(3, 'Charlie King', 'Analyst', 50000, 'Finance', SYSDATE),

(4, 'Diana Ray', 'Tester', 45000, 'IT', SYSDATE),

(5, 'Eve Clark', 'Lead', 75000, 'Finance', SYSDATE);

COMMIT;

SELECT \* FROM Employees;

**PL SQL CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

deptName IN VARCHAR2,

bonusPercent IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE Department = deptName;

END;

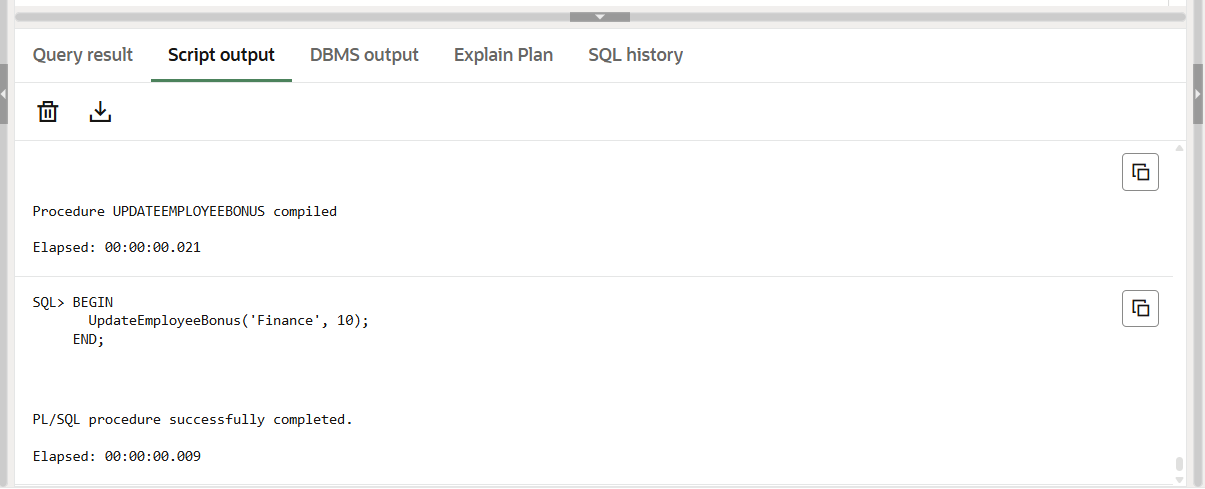
/

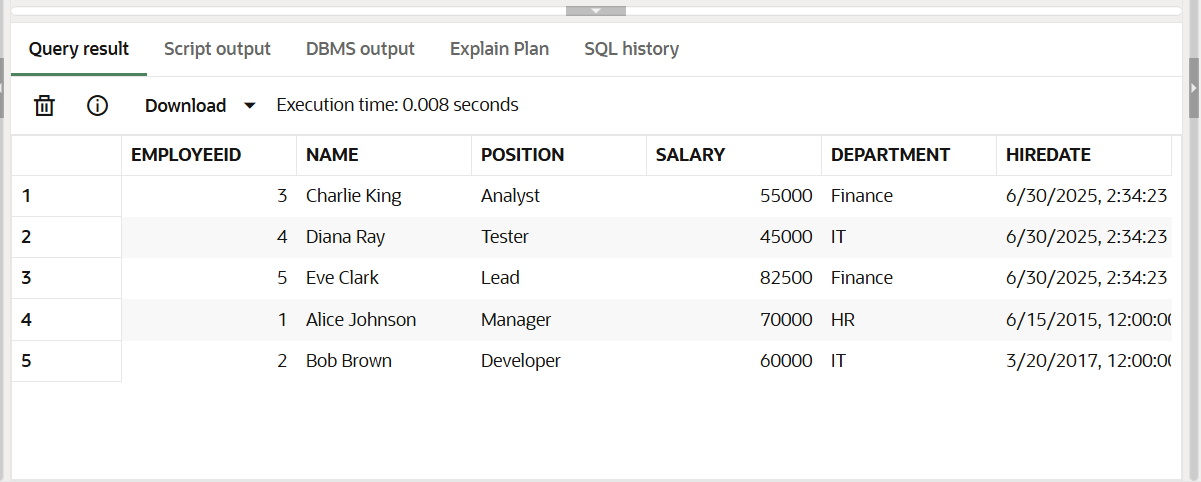
BEGIN

UpdateEmployeeBonus('Finance', 10);

END;

/

**OUTPUT:**

SELECT \* FROM Employees;

**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.

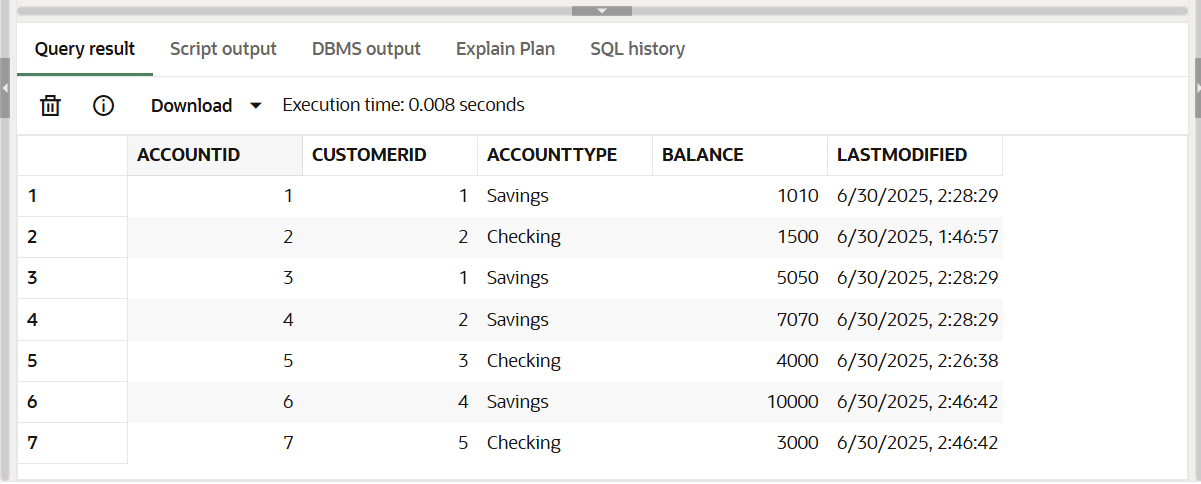
**CODE:**

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

(6, 4, 'Savings', 10000, SYSDATE),

(7, 5, 'Checking', 3000, SYSDATE);

COMMIT;

SELECT \* FROM Accounts;

**PL SQL CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds (

sourceID IN NUMBER,

targetID IN NUMBER,

amount IN NUMBER

) AS

insufficient\_balance EXCEPTION;

BEGIN

DECLARE

src\_balance NUMBER;

BEGIN

SELECT Balance INTO src\_balance FROM Accounts WHERE AccountID = sourceID;

IF src\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

END;

UPDATE Accounts

SET Balance = Balance - amount, LastModified = SYSDATE

WHERE AccountID = sourceID;

UPDATE Accounts

SET Balance = Balance + amount, LastModified = SYSDATE

WHERE AccountID = targetID;

EXCEPTION

WHEN insufficient\_balance THEN

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

END;

/

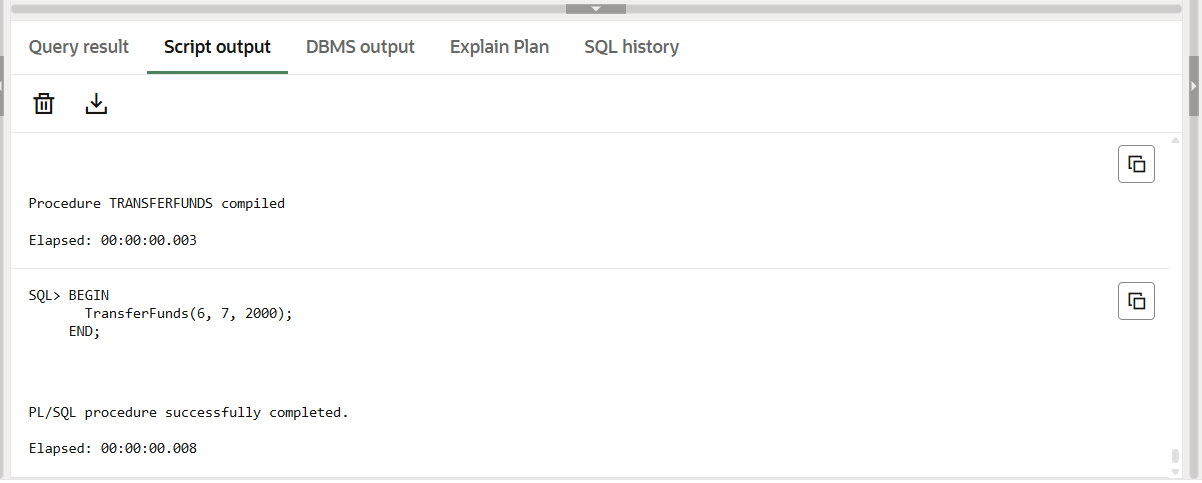
BEGIN

TransferFunds(6, 7, 2000);

END;

/

**OUTPUT:**



SELECT \* FROM Accounts;

