**PL/SQL programming**

**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:**

CREATE TABLE CUSTOMERS (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

LoanInterestRate NUMBER(5,2)

);

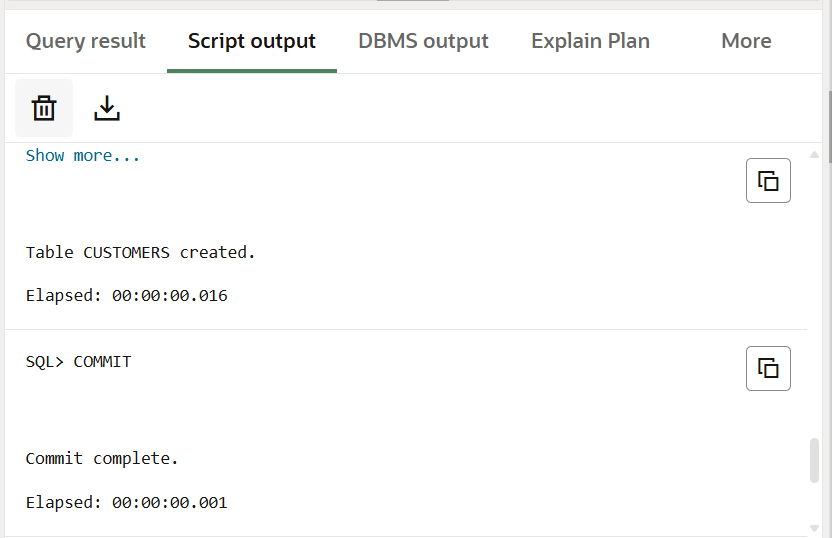
INSERT INTO CUSTOMERS VALUES (1, 'Ravi', 65, 10.50);

INSERT INTO CUSTOMERS VALUES (2, 'Meena', 45, 11.00);

INSERT INTO CUSTOMERS VALUES (3, 'Sundar', 70, 9.75);

INSERT INTO CUSTOMERS VALUES (4, 'Anjali', 58, 12.25);

COMMIT;



BEGIN

FOR cust IN (SELECT CustomerID, Age, LoanInterestRate FROM CUSTOMERS) LOOP

IF cust.Age > 60 THEN

UPDATE CUSTOMERS

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = cust.CustomerID;

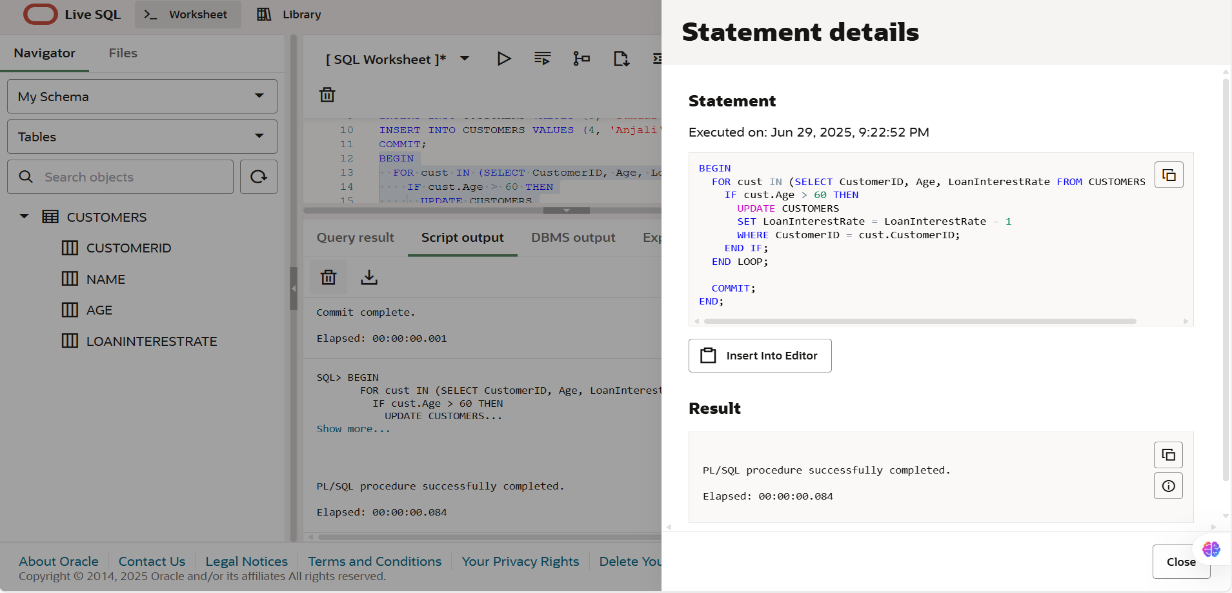
END IF;

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.**

**Code:**

CREATE TABLE CUSTOMERS (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Age NUMBER,

    Balance NUMBER(10,2),

    IsVIP CHAR(1) DEFAULT 'N'

);

INSERT INTO CUSTOMERS VALUES (1, 'Ravi', 65, 9500, 'N');

INSERT INTO CUSTOMERS VALUES (2, 'Meena', 45, 12000, 'N');

INSERT INTO CUSTOMERS VALUES (3, 'Sundar', 70, 15000, 'N');

INSERT INTO CUSTOMERS VALUES (4, 'Anjali', 30, 8000, 'N');

COMMIT;

BEGIN

  FOR cust IN (SELECT CustomerID, Balance FROM CUSTOMERS) LOOP

    IF cust.Balance > 10000 THEN

      UPDATE CUSTOMERS

      SET IsVIP = 'Y'

      WHERE CustomerID = cust.CustomerID;

    END IF;

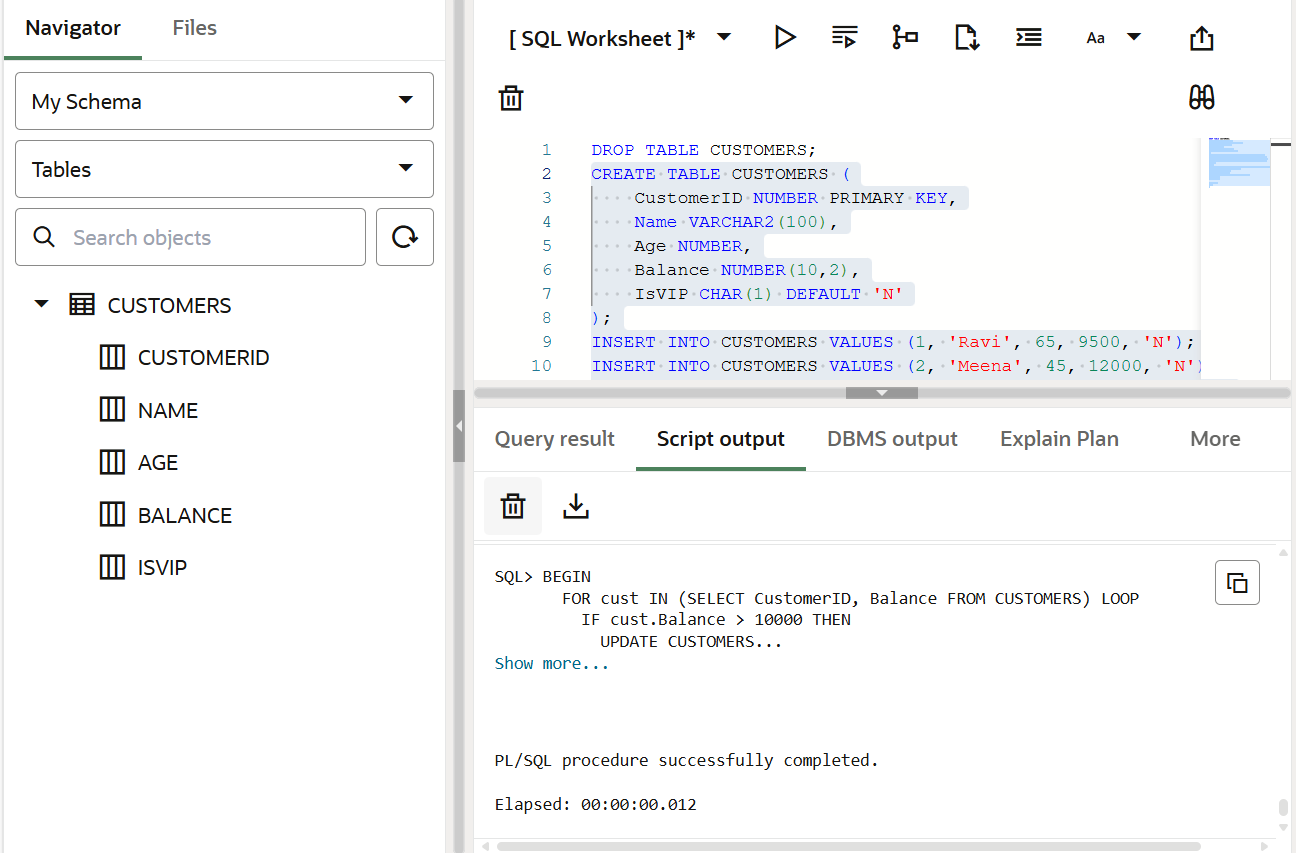
  END LOOP;

  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.**

CREATE TABLE CUSTOMERS (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Age NUMBER,

    Balance NUMBER(10,2),

    IsVIP CHAR(1) DEFAULT 'N',

    LoanDueDate DATE

);

-- Today's date assumed as SYSDATE

INSERT INTO CUSTOMERS VALUES (1, 'Ravi', 65, 9500, 'N', SYSDATE + 10);   -- Due in 10 days

INSERT INTO CUSTOMERS VALUES (2, 'Meena', 45, 12000, 'Y', SYSDATE + 35); -- Due in 35 days

INSERT INTO CUSTOMERS VALUES (3, 'Sundar', 70, 15000, 'Y', SYSDATE + 5); -- Due in 5 days

INSERT INTO CUSTOMERS VALUES (4, 'Anjali', 30, 8000, 'N', SYSDATE - 2);  -- Already overdue

COMMIT;

BEGIN

  FOR cust IN (

    SELECT Name, LoanDueDate

    FROM CUSTOMERS

    WHERE LoanDueDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

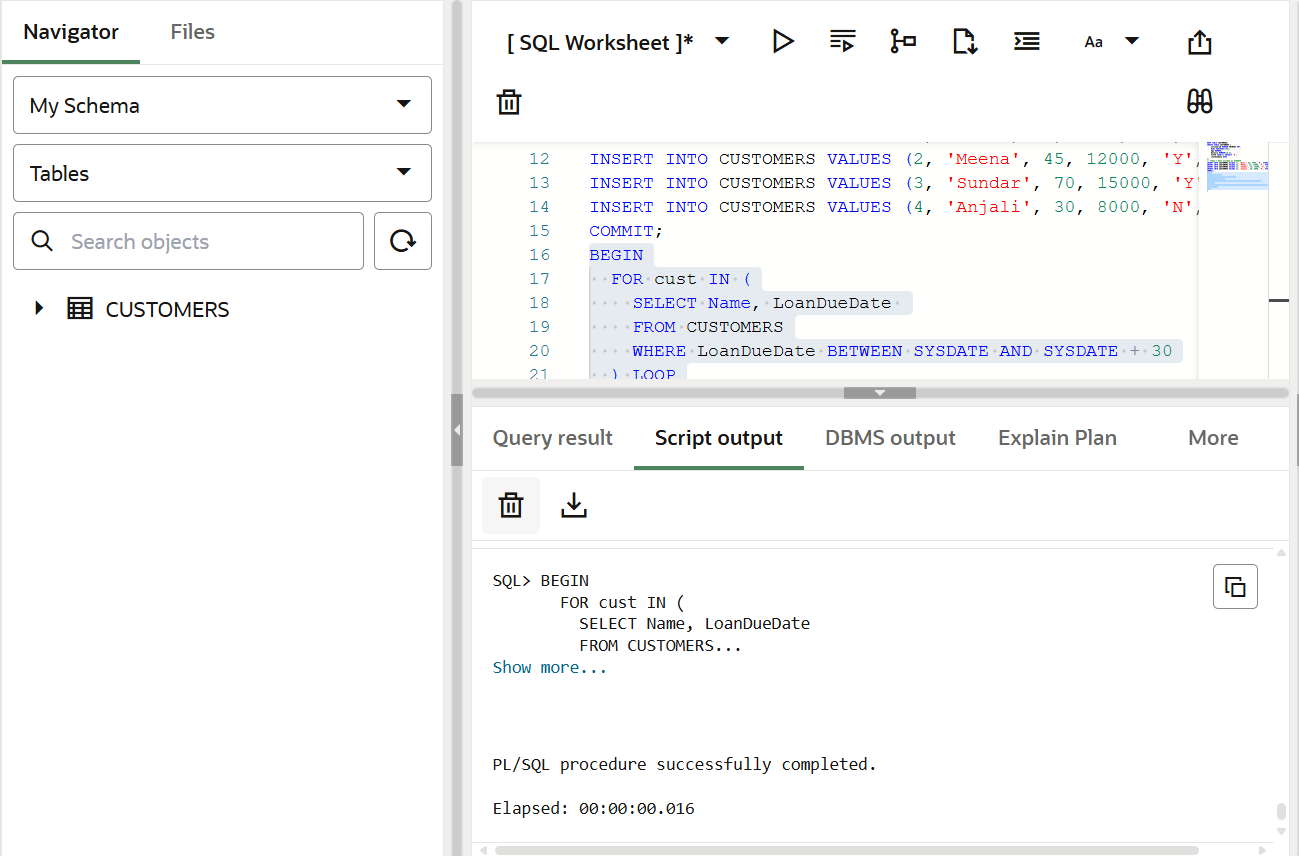
    DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || cust.Name || '''s loan is due on ' || TO\_CHAR(cust.LoanDueDate, 'DD-Mon-YYYY'));

  END LOOP;

END;

/

**Output:**



**Exercise 3: Stored Procedures**

**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**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,

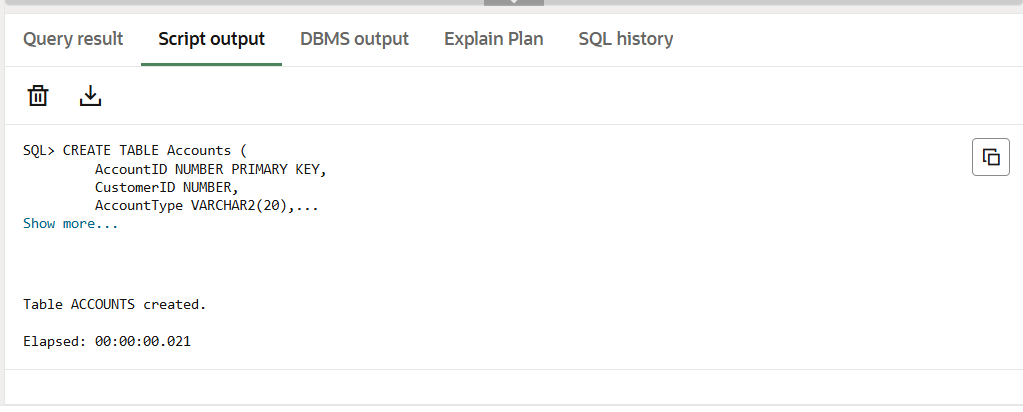
AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);



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

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 5000, SYSDATE);

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

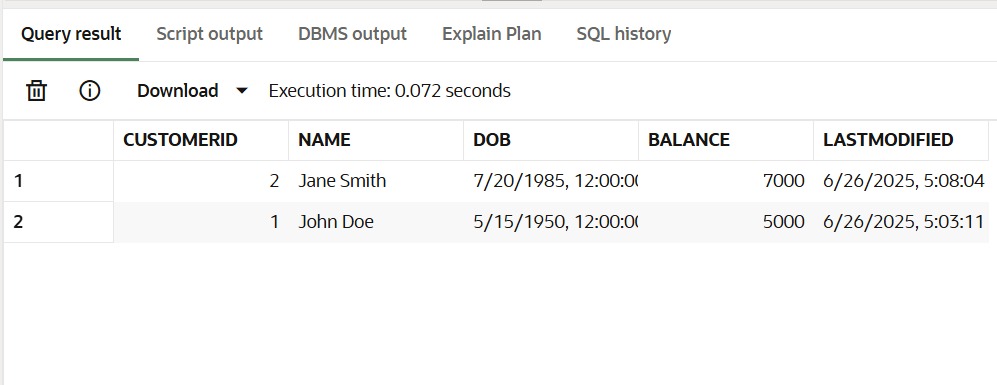
VALUES (2, 'Jane Smith', TO\_DATE('1985-07-20', 'YYYY-MM-DD'), 7000, SYSDATE);

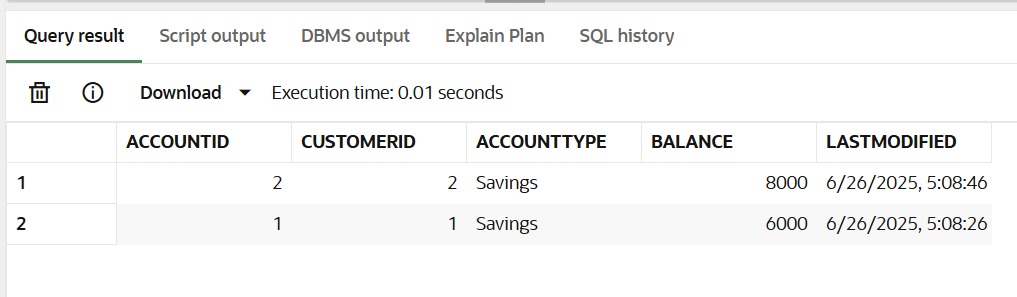
INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 6000, SYSDATE);

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

VALUES (2, 2, 'Savings', 8000, SYSDATE);





CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

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

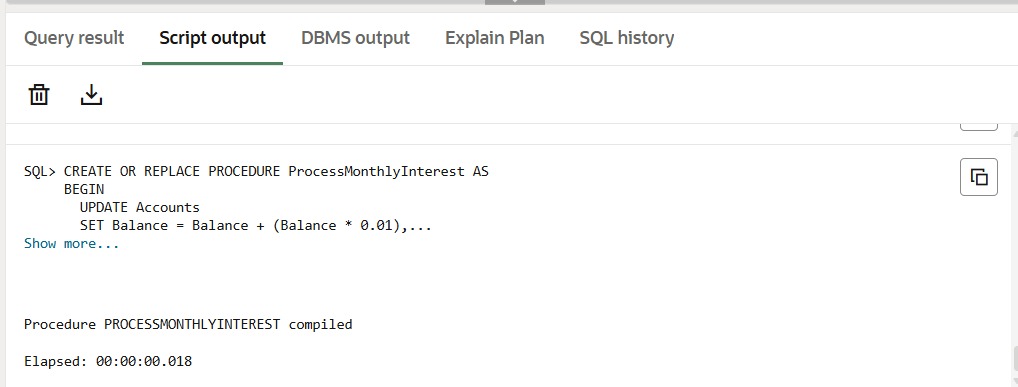
LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

END;

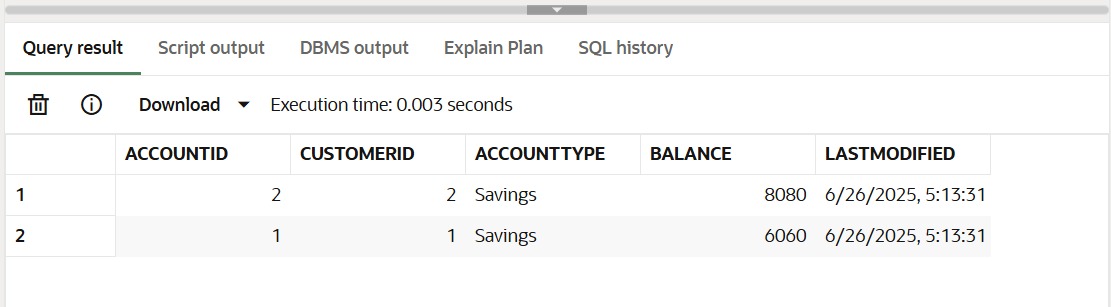
/



EXEC ProcessMonthlyInterest;

SELECT \* FROM Accounts;

**Output:**



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

* 1. **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** CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

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

VALUES (1, 'Alice', 'Manager', 60000, 'HR', TO\_DATE('2018-06-01', 'YYYY-MM-DD'));

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

VALUES (2, 'Bob', 'Developer', 50000, 'IT', TO\_DATE('2019-09-01', 'YYYY-MM-DD'));

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

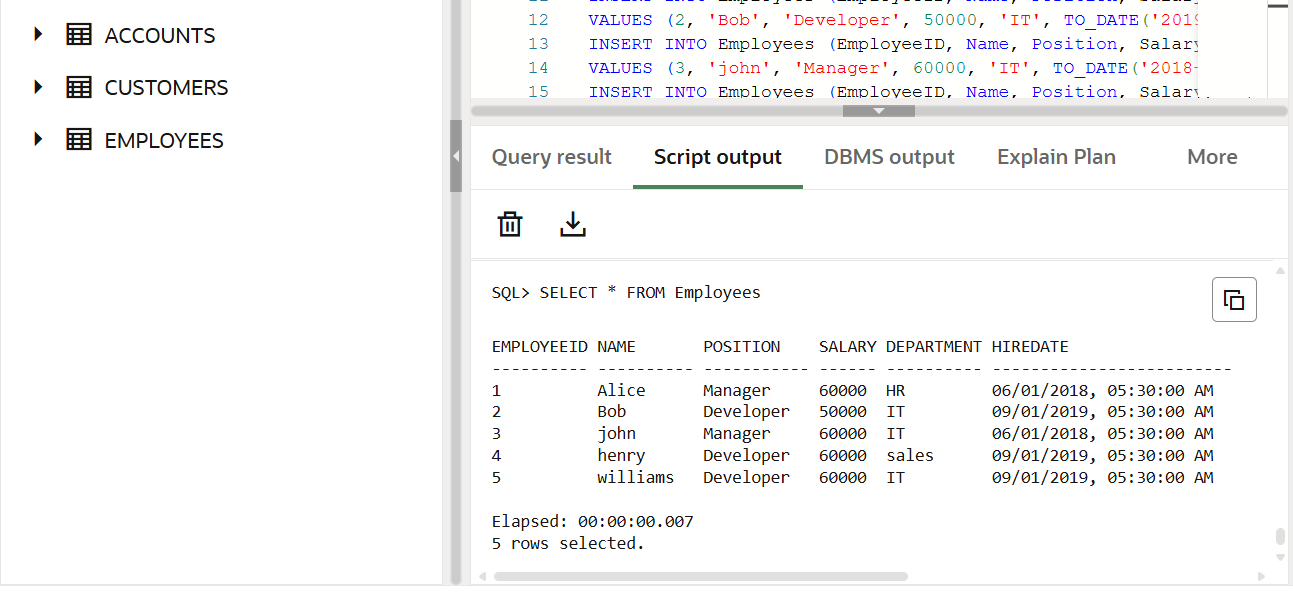
VALUES (3, 'john', 'Manager', 60000, 'IT', TO\_DATE('2018-06-01', 'YYYY-MM-DD'));

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

VALUES (4, 'henry', 'Developer', 60000, 'sales', TO\_DATE('2019-09-01', 'YYYY-MM-DD'));

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

VALUES (5, 'williams', 'Developer', 60000, 'IT', TO\_DATE('2019-09-01', 'YYYY-MM-DD'));



CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

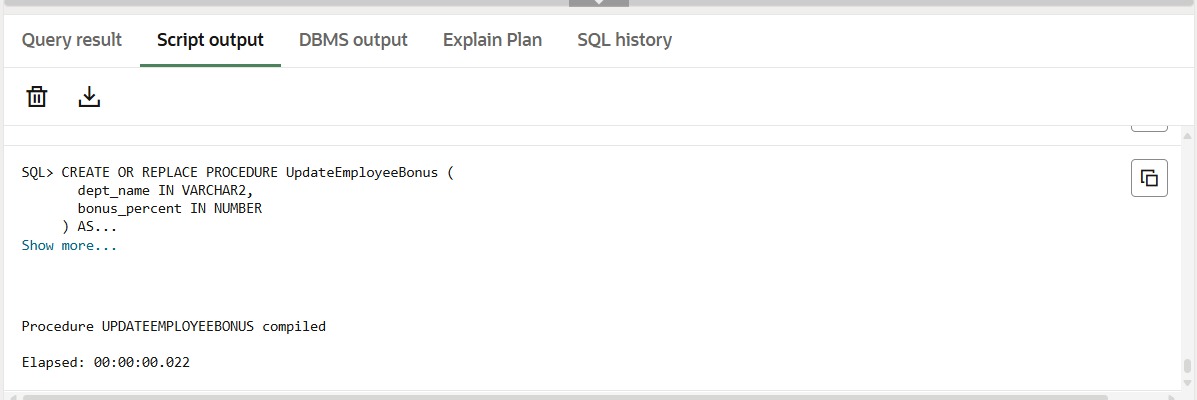
SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE Department = dept\_name;

COMMIT;

END;

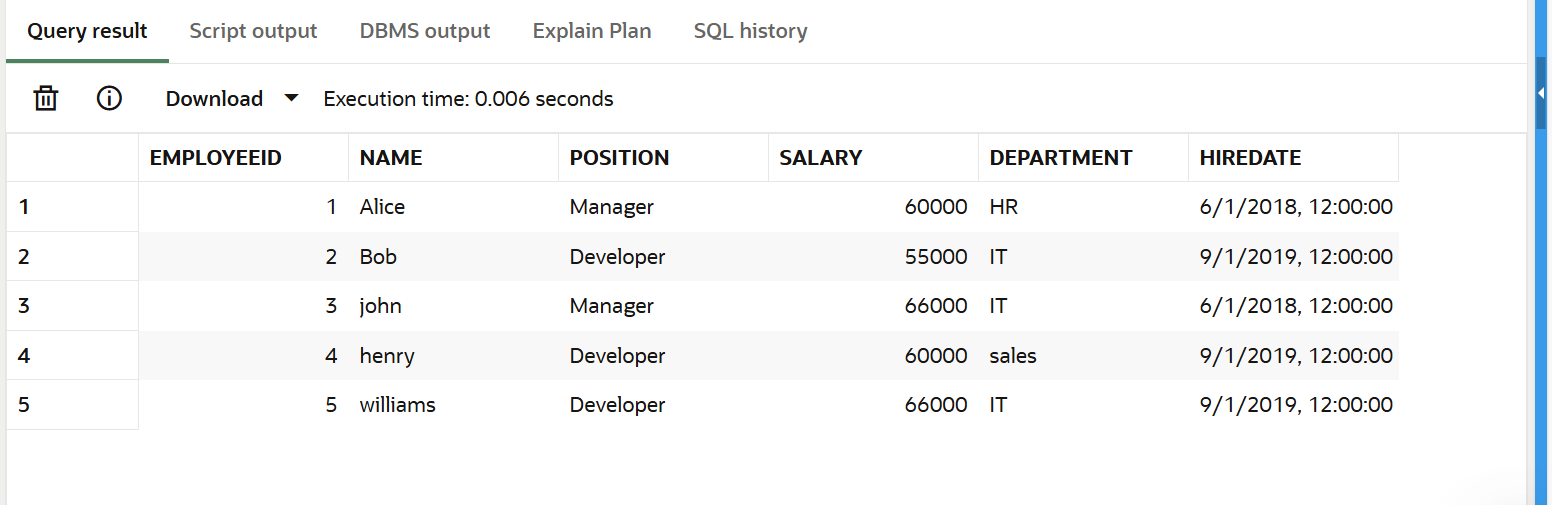
/



EXEC UpdateEmployeeBonus('IT', 10);

SELECT \* FROM Employees;

**Output:**



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

* 1. **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:**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

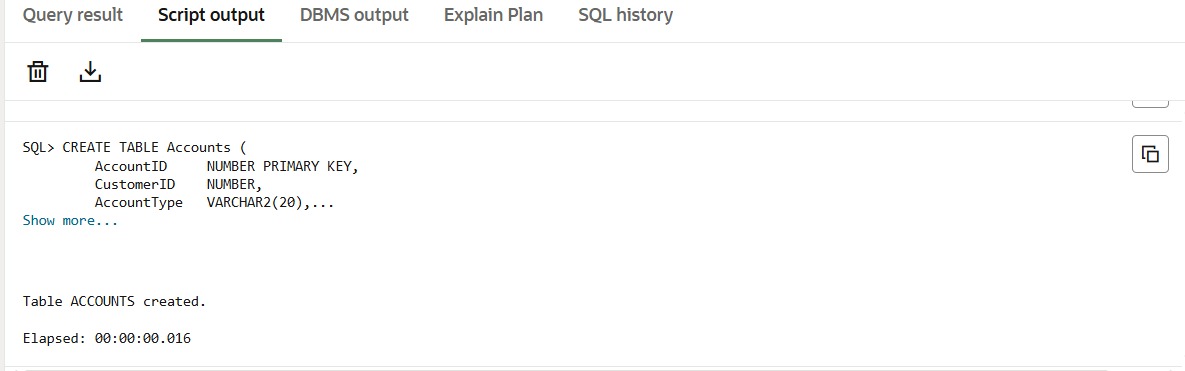
AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);



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

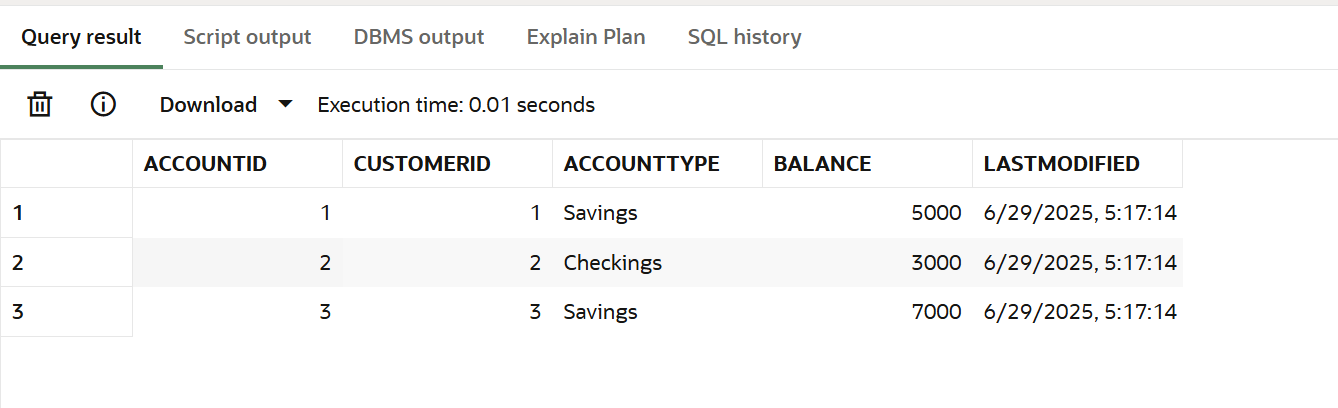
VALUES (1, 1, 'Savings', 5000, SYSDATE);

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

VALUES (2, 2, 'Checkings', 3000, SYSDATE);

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

VALUES (3, 3, 'Savings', 7000, SYSDATE);



CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account\_id IN NUMBER,

to\_account\_id IN NUMBER,

transfer\_amount IN NUMBER

) AS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account\_id;

IF from\_balance >= transfer\_amount THEN

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - transfer\_amount,

LastModified = SYSDATE

WHERE AccountID = from\_account\_id;

UPDATE Accounts

SET Balance = Balance + transfer\_amount,

LastModified = SYSDATE

WHERE AccountID = to\_account\_id;

COMMIT;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in Account ID ' || from\_account\_id);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid account ID');

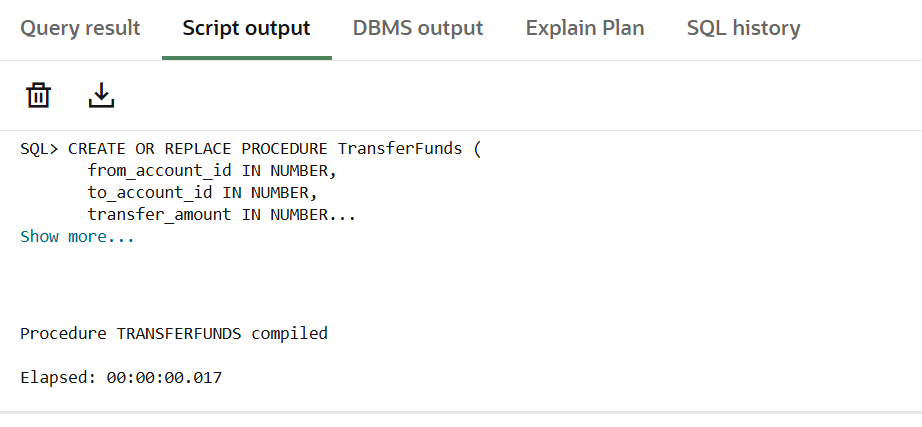
WHEN OTHERS THEN

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

ROLLBACK;

END;

/



EXEC TransferFunds(1, 2, 1000);

SELECT \* FROM Accounts;

**Output:**

