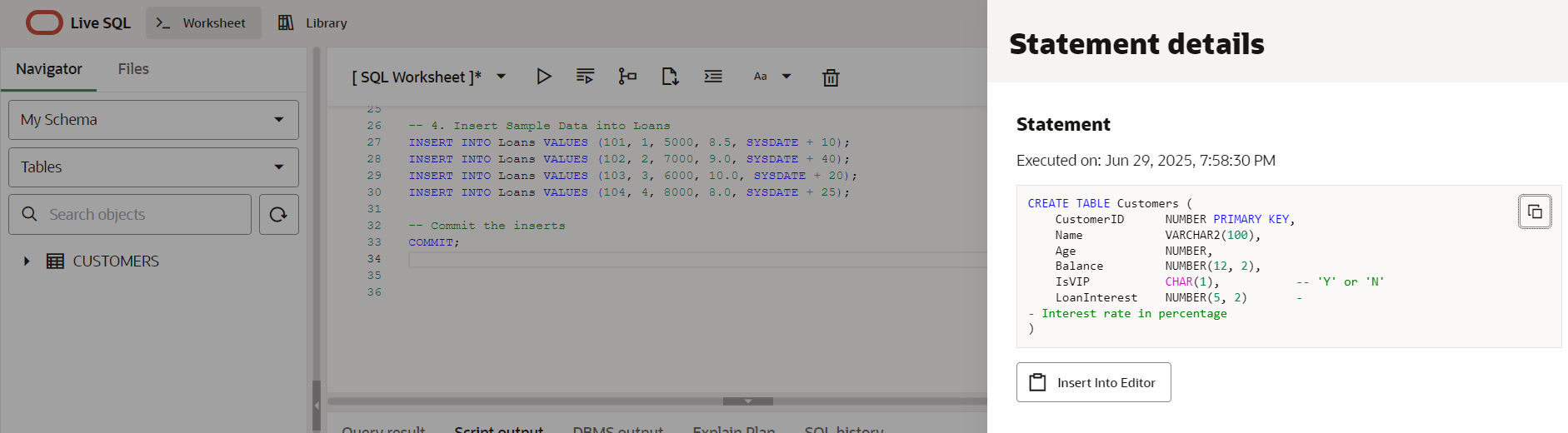
**PL/SQL programming Mandatory Hands-on**

**Exercise 1: Control Structures**

**DATABASE CREATION:**

****

**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:**

BEGIN

FOR cust IN (SELECT CustomerID, LoanInterest, Age FROM Customers) LOOP

IF cust.Age > 60 THEN

UPDATE Customers

SET LoanInterest = LoanInterest - 1

WHERE CustomerID = cust.CustomerID;

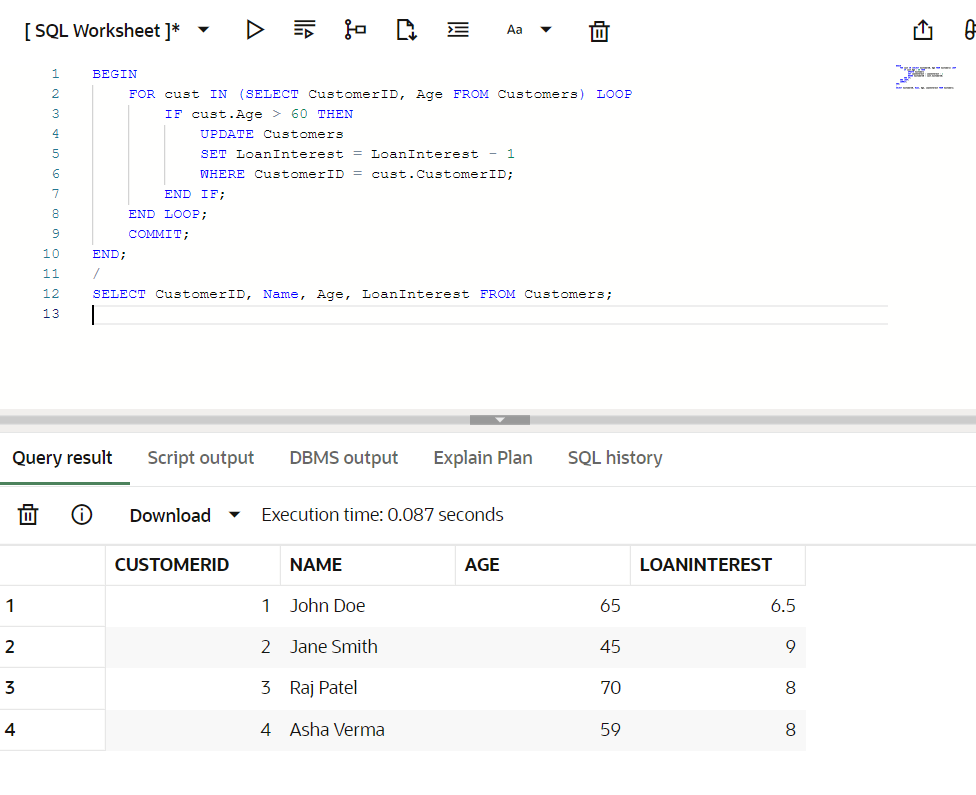
END IF;

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:**

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

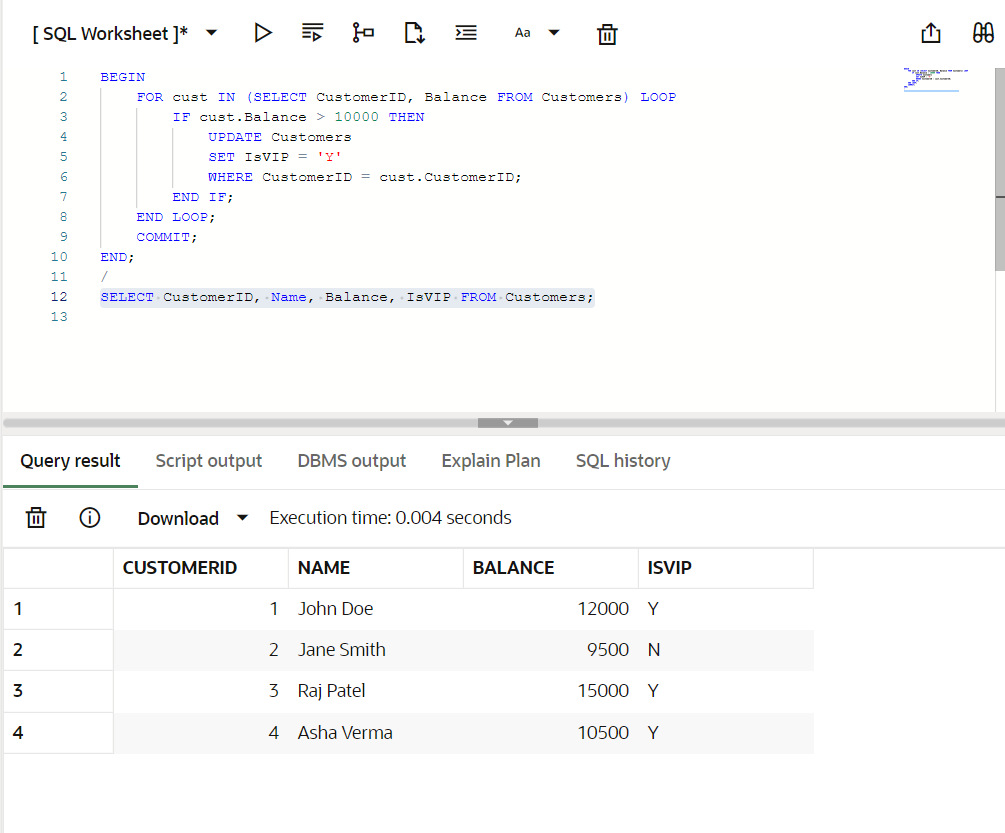
END IF;

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.

**Code:**

SET SERVEROUTPUT ON;

BEGIN

FOR loan IN (

SELECT l.LoanID, c.Name, l.DueDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate <= SYSDATE + 30

) LOOP

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

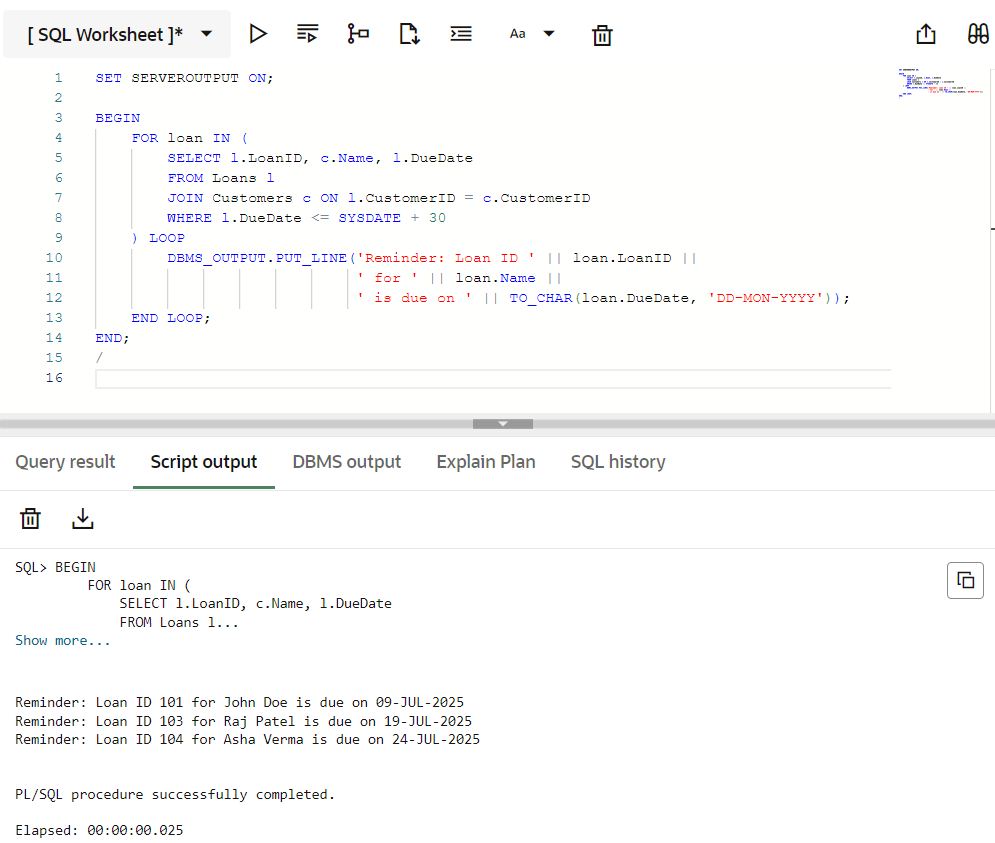
' for ' || loan.Name ||

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

END LOOP;

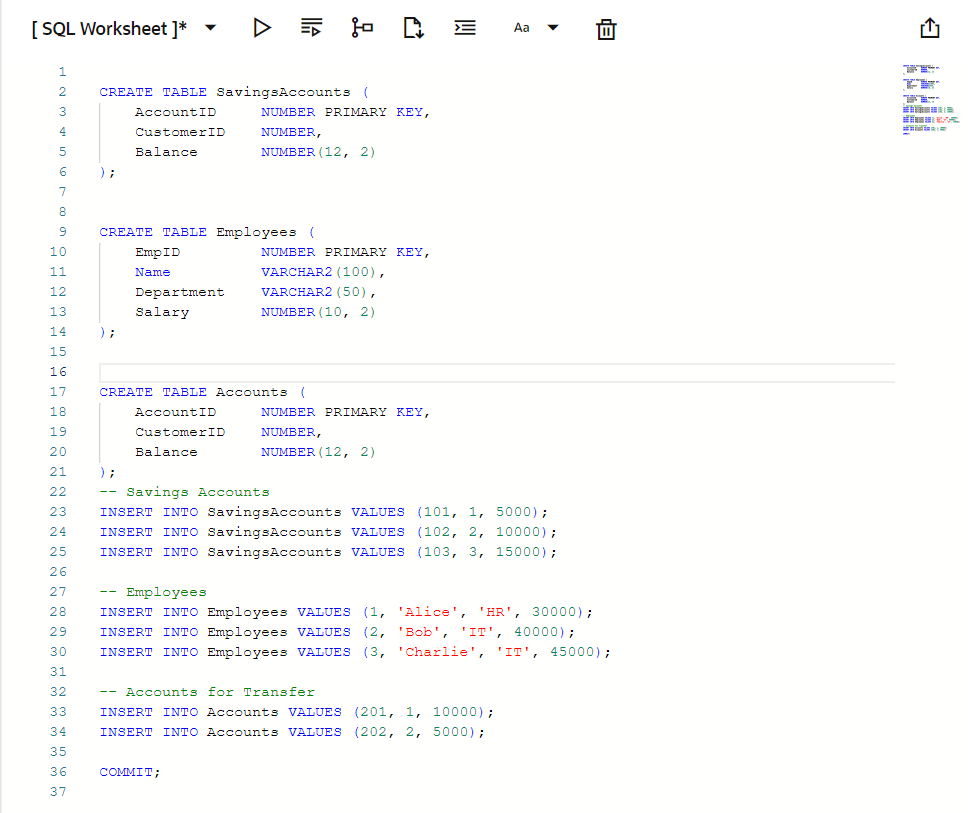
END;

**Output:**



**Exercise 3: Stored Procedures**

**DATABASE CREATION**



**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 OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

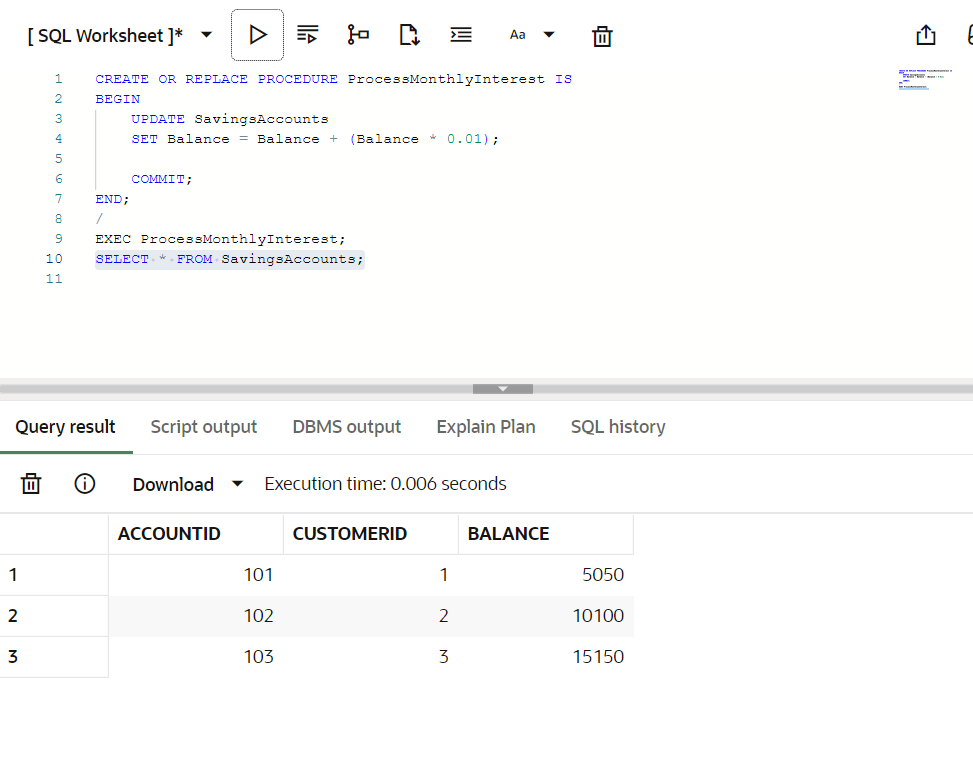
SET Balance = Balance + (Balance \* 0.01);

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.

**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

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

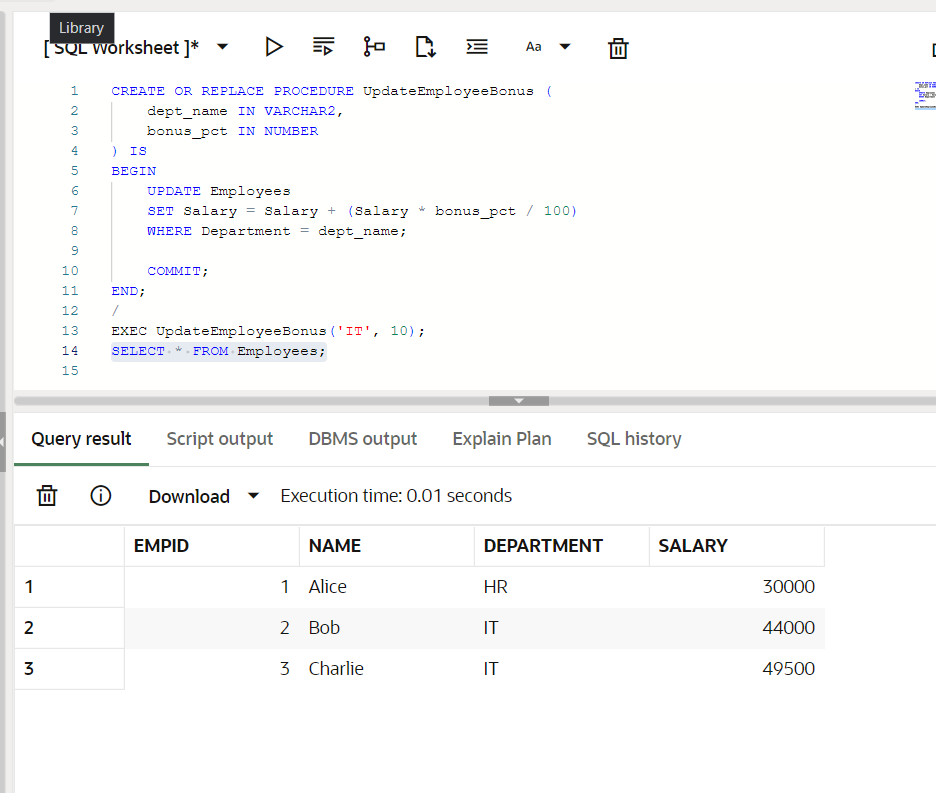
WHERE Department = dept\_name;

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.

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acct IN NUMBER,

to\_acct IN NUMBER,

amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_acct;

IF from\_balance < amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_acct;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_acct;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

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

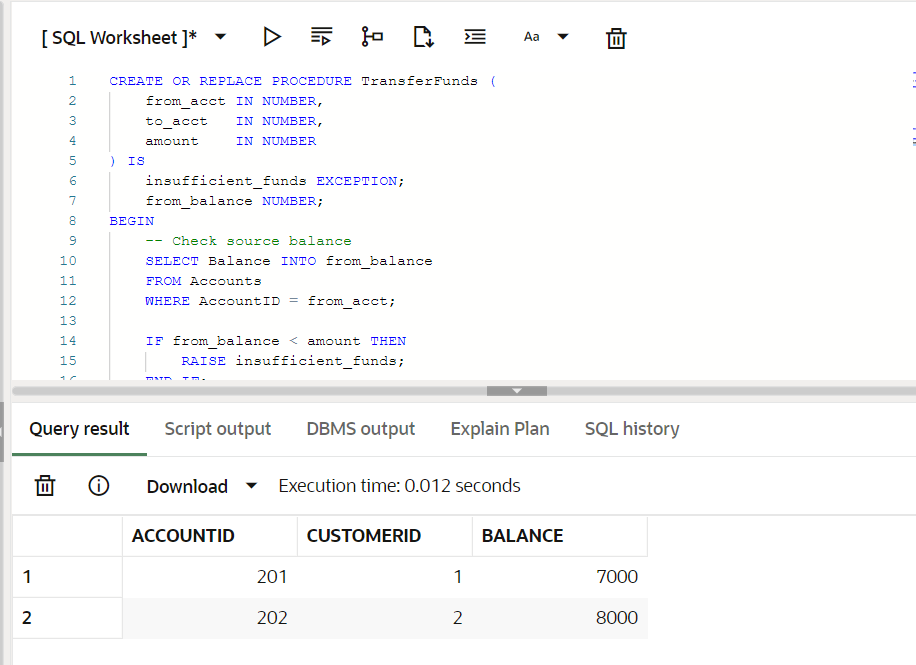
WHEN OTHERS THEN

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

END;

/

**Output:**

****