Cognizant - DN 4.0 Deep Skilling Java FSE

Week 02 – PL/SQL PROGRAMMING

Superset ID: 6386074

Name: A Sri Pranav

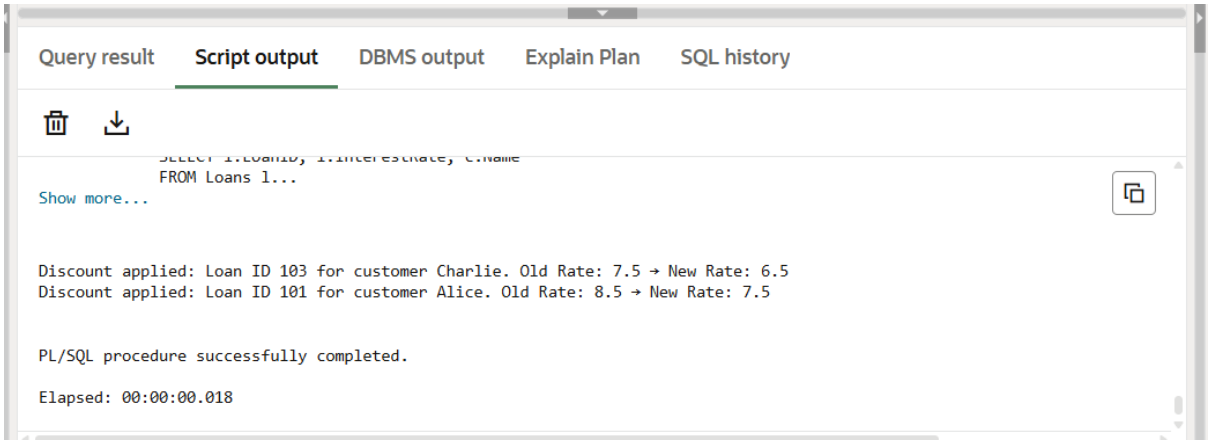
Exercise 1: Control Structures

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

```
BEGIN
  FOR rec IN (
    SELECT l.LoanID, l.InterestRate, c.Name
    FROM Loans l
    JOIN Customers c ON l.CustomerID = c.CustomerID
    WHERE c.Age > 60
  )
  LOOP
    UPDATE Loans
    SET InterestRate = InterestRate - 1
    WHERE LoanID = rec.LoanID;

    DBMS_OUTPUT.PUT_LINE('Discount applied: Loan ID ' || rec.LoanID ||
                ' for customer ' || rec.Name ||
                '. Old Rate: ' || rec.InterestRate ||
                ' → New Rate: ' || (rec.InterestRate - 1));
  END LOOP;
END;
/
```

Output:

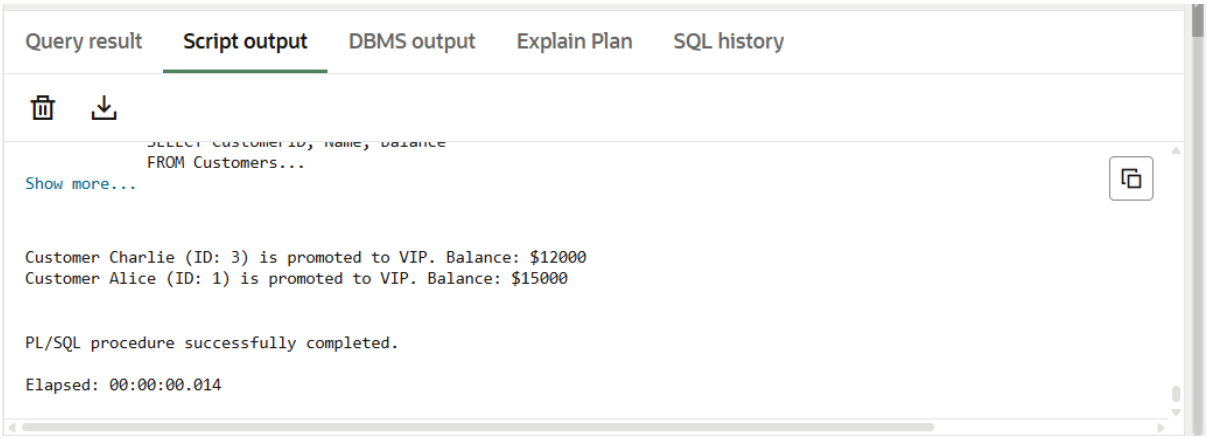


Scenario 2: A customer can be promoted to VIP status based on their balance.

```
BEGIN
  FOR rec IN (
    SELECT CustomerID, Name, Balance
    FROM Customers
    WHERE Balance > 10000
  )
  LOOP
    UPDATE Customers
    SET IsVIP = 'Y'
    WHERE CustomerID = rec.CustomerID;

    -- Output message
    DBMS_OUTPUT.PUT_LINE('Customer ' || rec.Name ||
                ' (ID: ' || rec.CustomerID ||
                ') is promoted to VIP. Balance: $' || rec.Balance);
  END LOOP;
END;
/
```

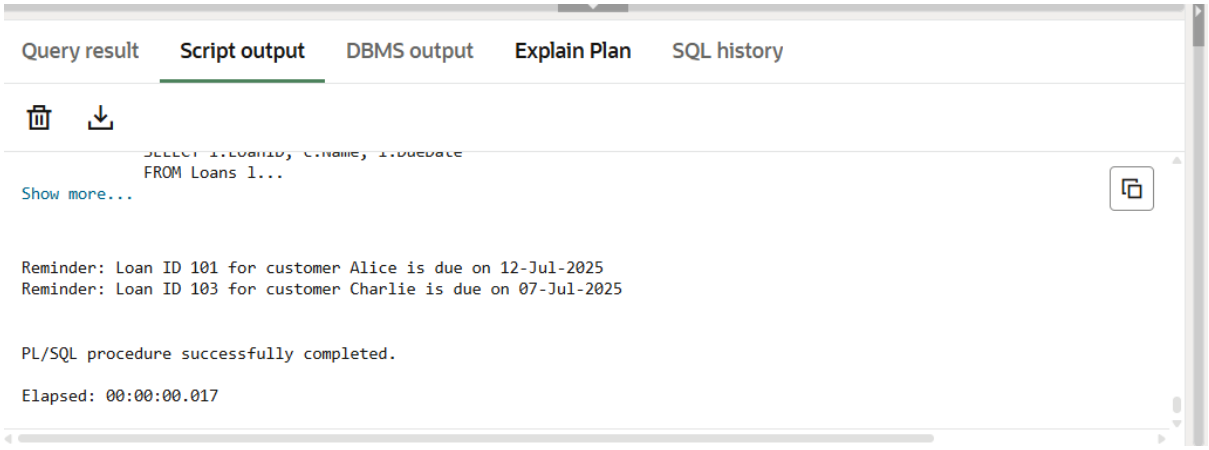
Output:



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

```
BEGIN
  FOR rec IN (
    SELECT l.LoanID, c.Name, l.DueDate
    FROM Loans l
    JOIN Customers c ON l.CustomerID = c.CustomerID
    WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || rec.LoanID ||
                  ' for customer ' || rec.Name ||
                  ' is due on ' || TO_CHAR(rec.DueDate, 'DD-Mon-YYYY'));
  END LOOP;
END;
/
```

Output:



Exercise 2: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
   UPDATE Accounts
   SET Balance = Balance + (Balance * 0.01)
   WHERE AccountType = 'Savings';
END;
/
BEGIN
   ProcessMonthlyInterest;
END;
/
SELECT * FROM Accounts;
```

Output:

Query result | Script output | DBMS output | Explain Plan | SQL history

Download ▾ Execution time: 0.006 seconds

	ACCOUNTID	ACCOUNTTYPE	BALANCE
1	102	Current	2000
2	103	Savings	1515
3	101	Savings	1010

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

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
   p_DepartmentID IN NUMBER,
   p_BonusPercent IN NUMBER
) IS
BEGIN
   UPDATE Employees
   SET Salary = Salary + (Salary * p_BonusPercent / 100)
   WHERE DepartmentID = p_DepartmentID;
END;
/
BEGIN
   UpdateEmployeeBonus(10, 10); -- 10% bonus to dept 10
END;
/
SELECT * FROM Employees;
```

Output:

Query result | Script output | DBMS output | Explain Plan | SQL history

Download ▾ Execution time: 0.003 seconds

	EMPLOYEEID	DEPARTMENTID	SALARY
1	1	10	33000
2	2	20	35000
3	3	10	44000

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

```
CREATE OR REPLACE PROCEDURE TransferFunds (
```

```
    p_FromAccountID IN NUMBER,
    p_ToAccountID IN NUMBER,
    p_Amount IN NUMBER
) IS
    v_FromBalance NUMBER;
BEGIN
    SELECT Balance INTO v_FromBalance
    FROM Accounts
    WHERE AccountID = p_FromAccountID;

    IF v_FromBalance >= p_Amount THEN
        UPDATE Accounts
        SET Balance = Balance - p_Amount
        WHERE AccountID = p_FromAccountID;

        UPDATE Accounts
        SET Balance = Balance + p_Amount
        WHERE AccountID = p_ToAccountID;
    ELSE
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds for transfer');
    END IF;
END;
/
BEGIN
    TransferFunds(101, 102, 500);
END;
/
SELECT * FROM Accounts;
```

Output:

Query result | Script output | DBMS output | Explain Plan | SQL history

Download Execution time: 0.005 seconds

	ACCOUNTID	ACCOUNTTYPE	BALANCE
1	102	Current	2500
2	103	Savings	1515
3	101	Savings	510