**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.

**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.

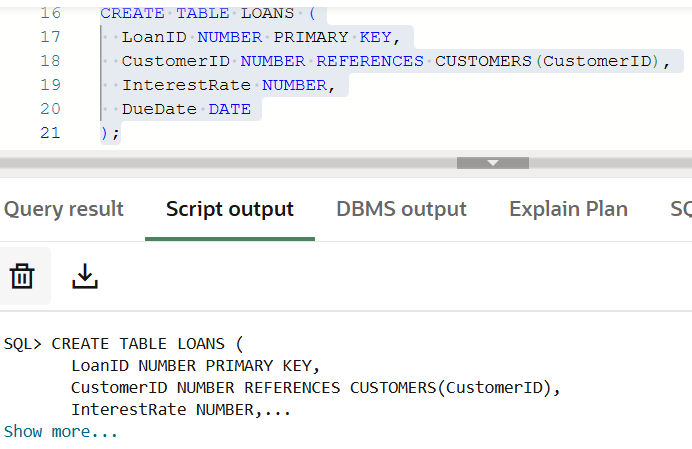
**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

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

**Objective:**

This document demonstrates how PL/SQL control structures such as loops and conditional logic can be used to solve real-world banking problems. The three scenarios simulate customer segmentation, loan discounting, and automated notifications.

**CODE:**

**Create CUSTOMERS Table and LOAN table:**

CREATE TABLE CUSTOMERS (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

CREATE TABLE LOANS (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES CUSTOMERS(CustomerID),

InterestRate NUMBER,

DueDate DATE

);

**INSERTION OF DATA:**

INSERT INTO CUSTOMERS VALUES (1, 'Niraja', 65, 12000, 'FALSE');

INSERT INTO CUSTOMERS VALUES (2, 'Dharshini', 45, 8000, 'FALSE');

INSERT INTO CUSTOMERS VALUES (3, 'Garima', 70, 15000, 'FALSE');

INSERT INTO CUSTOMERS VALUES (4, 'Bharathi', 59, 11000, 'FALSE');

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

INSERT INTO LOANS VALUES (102, 2, 12, SYSDATE + 40);

INSERT INTO LOANS VALUES (103, 3, 11, SYSDATE + 5);

INSERT INTO LOANS VALUES (104, 4, 13, SYSDATE + 25);

COMMIT;

**Scenario 1: Interest Rate Discount for Senior Citizens**

The bank wants to offer a 1% interest rate discount on all loans for customers who are above 60 years old.

BEGIN

FOR rec IN (SELECT CustomerID FROM CUSTOMERS WHERE Age > 60) LOOP

UPDATE LOANS

SET InterestRate = InterestRate - 1

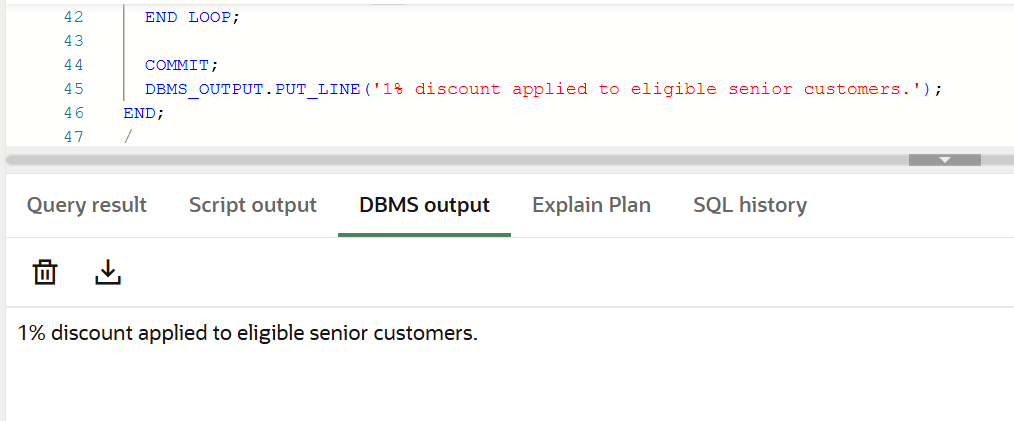
WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to eligible senior customers.');

END;

**OUTPUT**

**SCENARIO 2**: **Promote Customers to VIP Status Based on Balance**

Set IsVIP = 'TRUE' for all customers whose **balance > 10,000**.

**CODE:**

BEGIN

FOR rec IN (SELECT CustomerID FROM CUSTOMERS WHERE Balance > 10000) LOOP

UPDATE CUSTOMERS

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

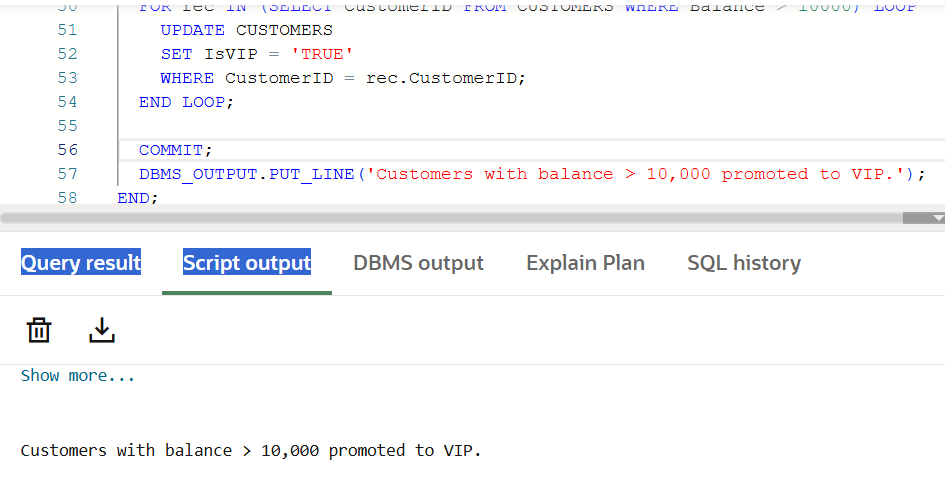
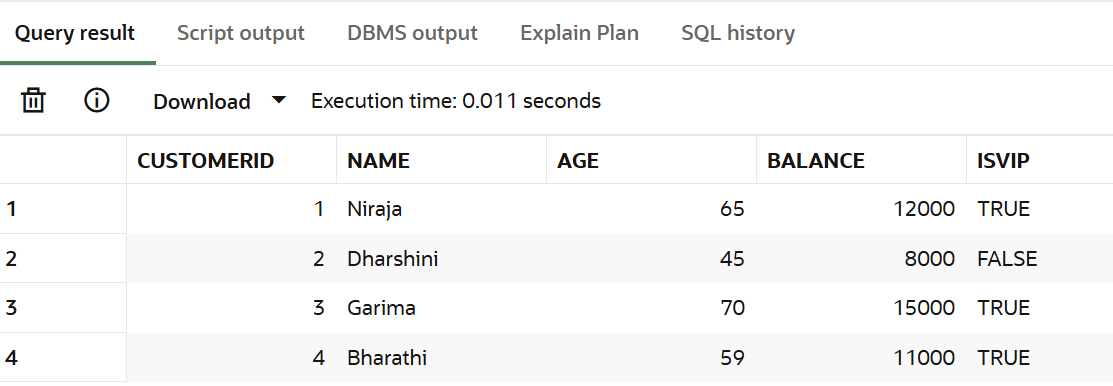
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Customers with balance > 10,000 promoted to VIP.');

END;

/

OUTPUT:

**SCENARIO 3: Send Reminders for Loans Due Within 30 Days**

**CODE:**

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 <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('📩 Reminder: Loan ' || rec.LoanID ||

' for customer ' || rec.Name ||

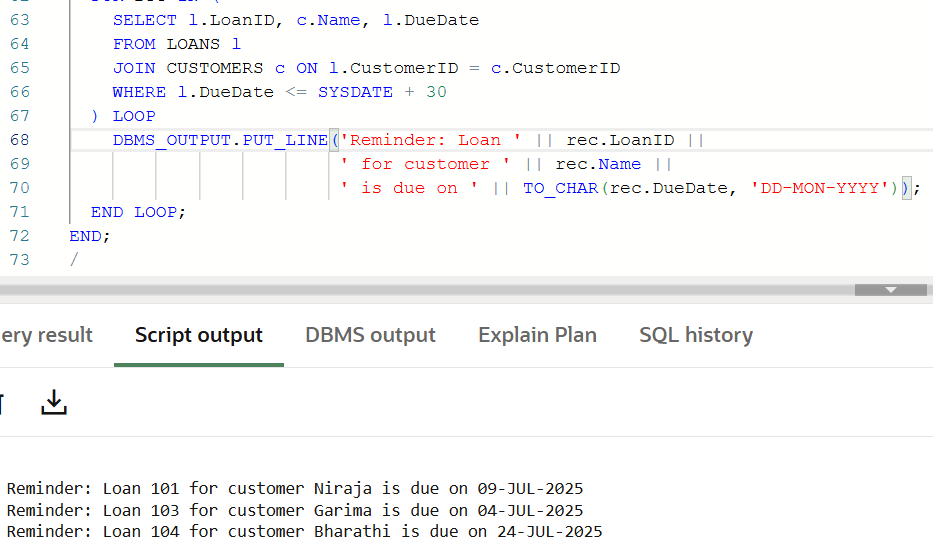
' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY'));

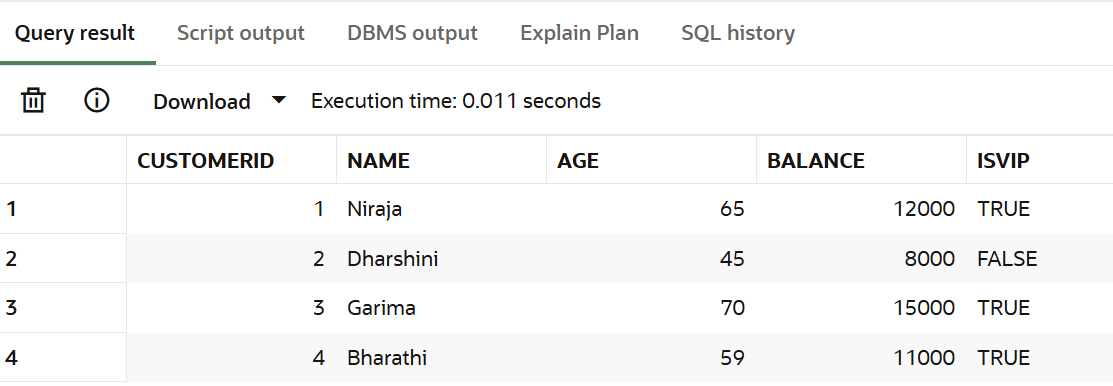
END LOOP;

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.

**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.

**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.

**Objective:**

The objective of this exercise is to design and implement stored procedures in PL/SQL to handle core banking operations. These include interest calculation for savings accounts, performance-based bonus updates for employees, and secure fund transfers between customer accounts. This helps in automating financial processes efficiently and accurately.

**Creation of Tables :**

CREATE TABLE ACCOUNTS (

AccountID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER,

AccountType VARCHAR2(20)

);

CREATE TABLE EMPLOYEES (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

**Insertion of data:**

INSERT INTO ACCOUNTS VALUES (1, 'Niraja', 50000, 'SAVINGS');

INSERT INTO ACCOUNTS VALUES (2, 'Dharshini', 40000, 'SAVINGS');

INSERT INTO ACCOUNTS VALUES (3, 'Garima', 30000, 'CURRENT');

INSERT INTO ACCOUNTS VALUES (4, 'Bharathi', 60000, 'SAVINGS');

INSERT INTO EMPLOYEES VALUES (101, 'Rahul', 'IT', 50000);

INSERT INTO EMPLOYEES VALUES (102, 'Priya', 'HR', 45000);

INSERT INTO EMPLOYEES VALUES (103, 'Amit', 'IT', 52000);

INSERT INTO EMPLOYEES VALUES (104, 'Neha', 'Finance',48000);

COMMIT**;**

**Scenario 1: Process Monthly Interest (1% for Savings Accounts)**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE ACCOUNTS

SET Balance = Balance + (Balance \* 0.01)

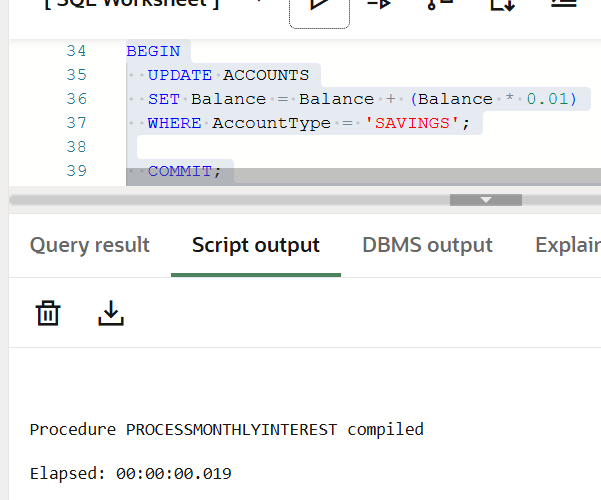
WHERE AccountType = 'SAVINGS';

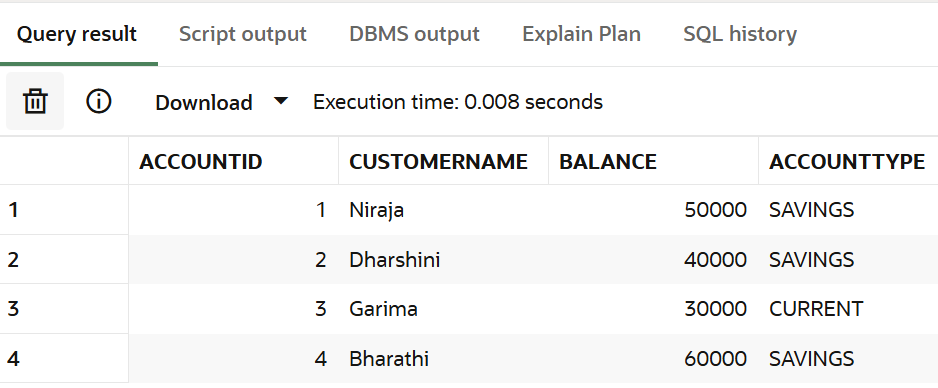
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% applied to all savings accounts.');

END;

**OUTPUT:**





**Scenario 2: Update Employee Bonus:**

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE EMPLOYEES

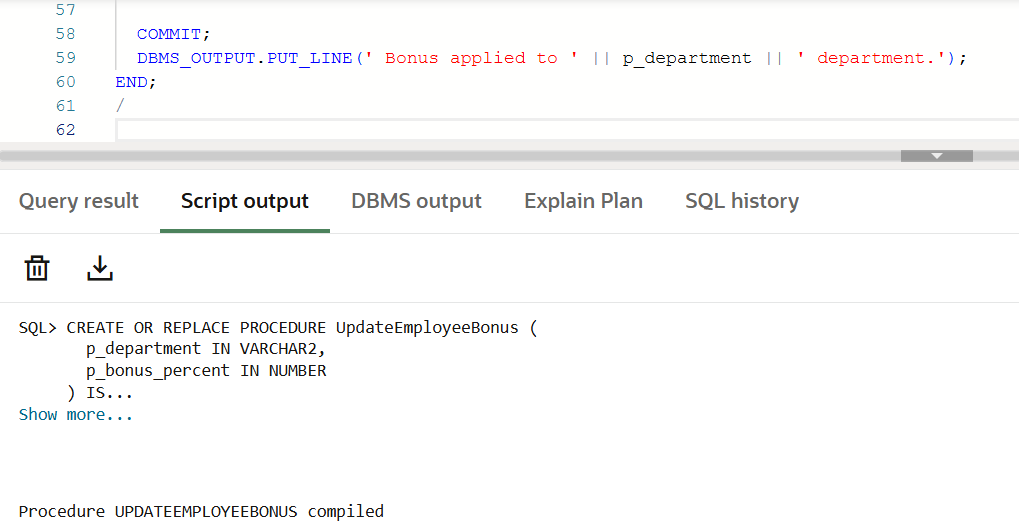
SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

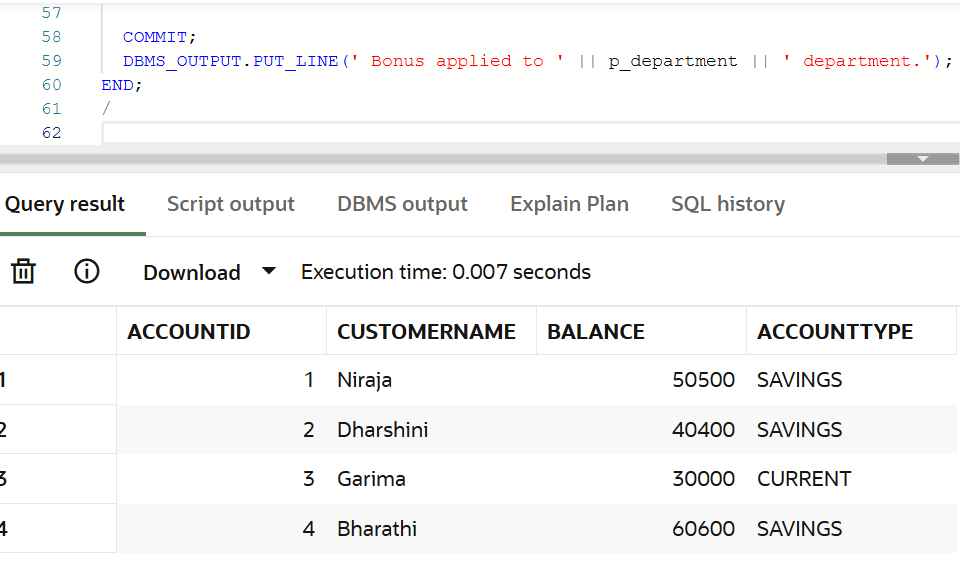
WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Bonus applied to ' || p\_department || ' department.');

END;





**Scenario 3: Transfer Funds Between Accounts**

**CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Get balance of sender

SELECT Balance INTO v\_balance

FROM ACCOUNTS

WHERE AccountID = p\_from\_account;

-- Check if sufficient balance

IF v\_balance < p\_amount THEN

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

ELSE

-- Deduct from sender

UPDATE ACCOUNTS

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

-- Add to receiver

UPDATE ACCOUNTS

SET Balance = Balance + p\_amount

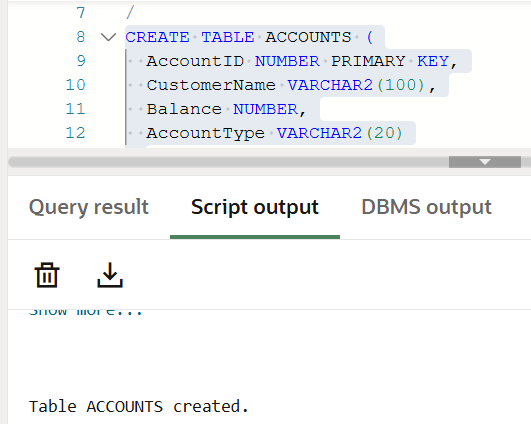
WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' ₹' || p\_amount || ' transferred from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

END IF;

END;

**OUTPUT:**

