**Schema Architecture:**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE,

    IsVIP VARCHAR2(5)

);

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

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

VALUES (1, 'John Doe', TO\_DATE('1955-05-15', 'YYYY-MM-DD'), 1000, SYSDATE, 'FALSE');

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 15000, SYSDATE, 'FALSE');

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

VALUES (1, 1, 'Savings', 1000, SYSDATE);

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

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 7000, 6.5, SYSDATE, ADD\_MONTHS(SYSDATE, 20));

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

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

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

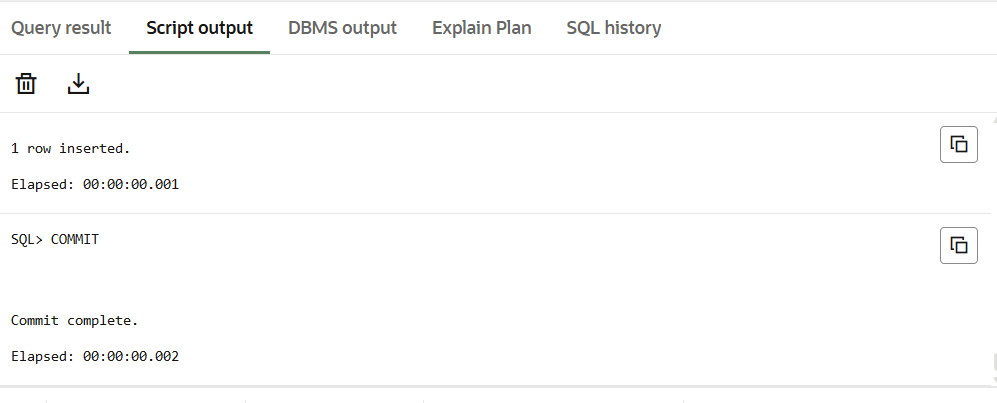
VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (3, 1, 3000, 4.5, SYSDATE, SYSDATE + 15);

COMMIT;

**OUTPUT:**

****

**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:**

SET SERVEROUTPUT ON;

BEGIN

  FOR cust IN (

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

    FROM Customers c

    JOIN Loans l ON c.CustomerID = l.CustomerID

  ) LOOP

    IF MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12 > 60 THEN

      UPDATE Loans

      SET InterestRate = InterestRate - 1

      WHERE LoanID = cust.LoanID;

      DBMS\_OUTPUT.PUT\_LINE('1% interest discount applied to Loan ID ' || cust.LoanID);

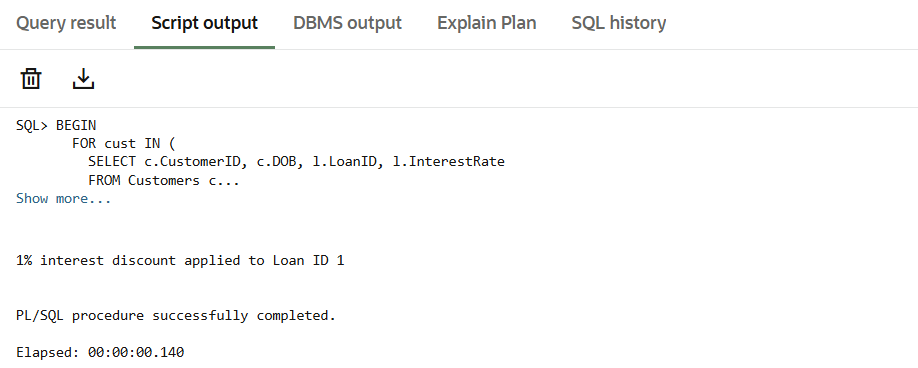
    END IF;

  END LOOP;

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:**

SET SERVEROUTPUT ON;

BEGIN

  FOR cust IN (

    SELECT CustomerID, Balance

    FROM Customers

    WHERE Balance > 10000

  ) LOOP

    UPDATE Customers

    SET IsVIP = 'TRUE',

        LastModified = SYSDATE

    WHERE CustomerID = cust.CustomerID;

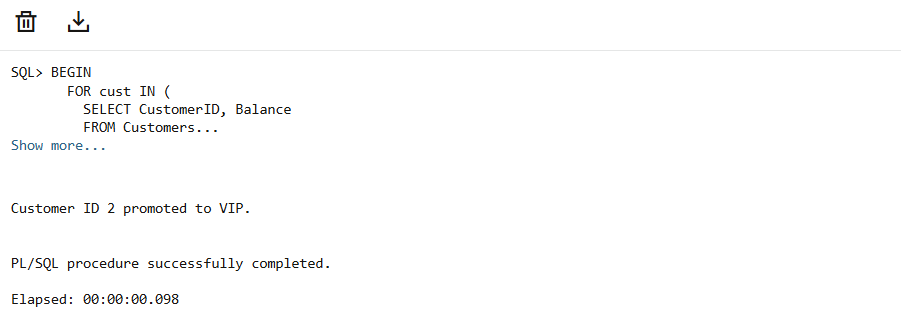
    DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust.CustomerID || ' promoted to VIP.');

  END LOOP;

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, l.CustomerID, l.EndDate, c.Name

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

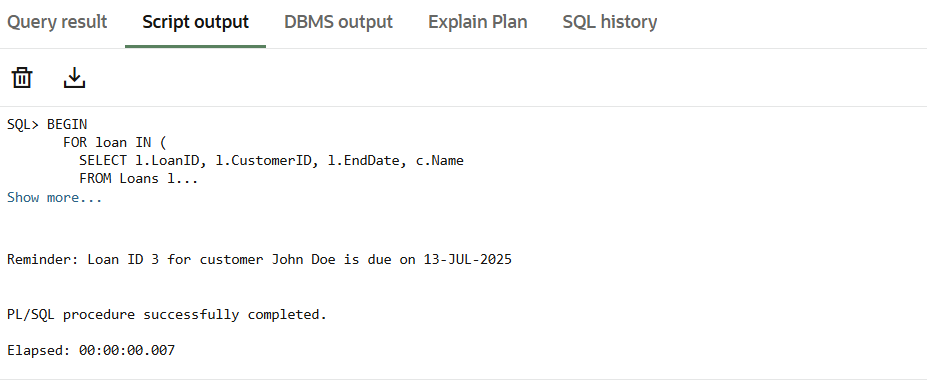
    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan.LoanID ||' for customer ' || loan.Name ||

                         ' is due on ' || TO\_CHAR(loan.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:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR acc IN (

    SELECT AccountID, Balance

    FROM Accounts

    WHERE AccountType = 'Savings'

  ) LOOP

    UPDATE Accounts

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

        LastModified = SYSDATE

    WHERE AccountID = acc.AccountID;

    DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID);

  END LOOP;

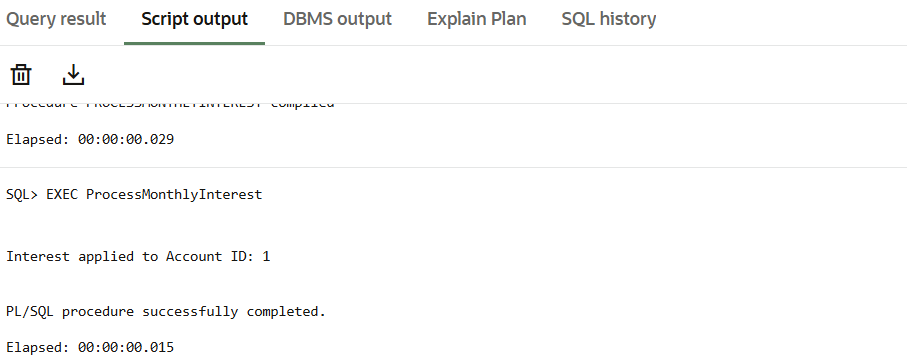
END;

/

SET SERVEROUTPUT ON;

EXEC ProcessMonthlyInterest;

**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 (

  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;

  DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to department: ' || p\_department);

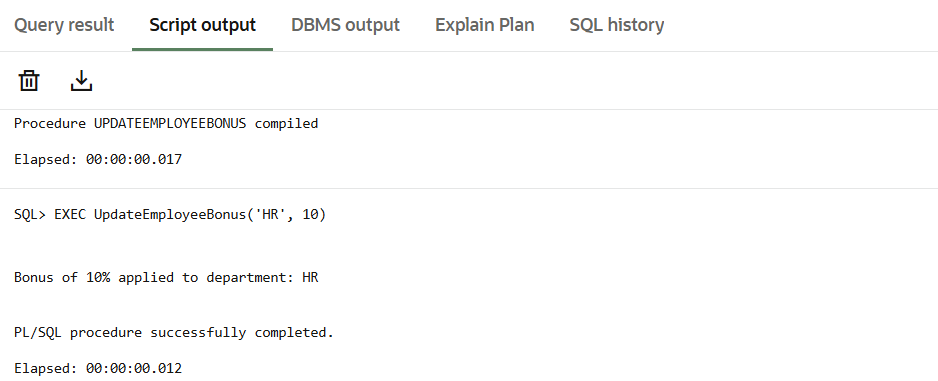
END;

/

SET SERVEROUTPUT ON;

EXEC UpdateEmployeeBonus('HR', 10);

**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 (

  p\_from\_account IN NUMBER,

  p\_to\_account IN NUMBER,

  p\_amount IN NUMBER

) IS

  v\_balance NUMBER;

BEGIN

  -- Check balance of source account

  SELECT Balance INTO v\_balance

  FROM Accounts

  WHERE AccountID = p\_from\_account;

  IF v\_balance < p\_amount THEN

    RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

  END IF;

  -- Deduct from source

  UPDATE Accounts

  SET Balance = Balance - p\_amount,

      LastModified = SYSDATE

  WHERE AccountID = p\_from\_account;

  -- Add to destination

  UPDATE Accounts

  SET Balance = Balance + p\_amount,

      LastModified = SYSDATE

  WHERE AccountID = p\_to\_account;

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

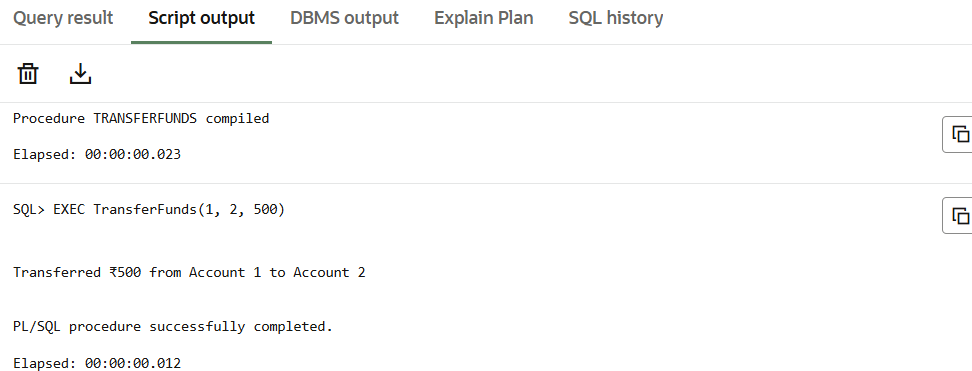
END;

/

SET SERVEROUTPUT ON;

EXEC TransferFunds(1, 2, 500);

**OUTPUT:**

****