**DN 4.0 JAVA FSE SOLUTIONS – WEEK 2**

**SKILL: PL/SQL programming**

**Exercise 1:** Control Structures

**Creating Tables:**

**Customers Table**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

**Loans Table**

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Inserting data in to Customers and Loans tables**

BEGIN

INSERT INTO Customers VALUES (1, 'Ravi', 65, 12000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Anita', 45, 8000, 'FALSE');

INSERT INTO Customers VALUES (3, 'John', 70, 5000, 'FALSE');

INSERT INTO Customers VALUES (4, 'Priya', 30, 15000, 'FALSE');

COMMIT;

END;

/

BEGIN

INSERT INTO Loans VALUES (101, 1, 10, SYSDATE + 20);

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

INSERT INTO Loans VALUES (103, 3, 11, SYSDATE + 10);

INSERT INTO Loans VALUES (104, 4, 9, SYSDATE + 5);

COMMIT;

END;

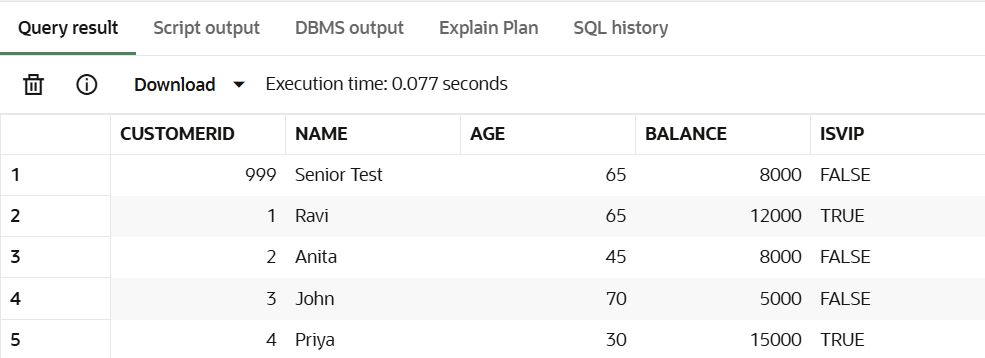
/

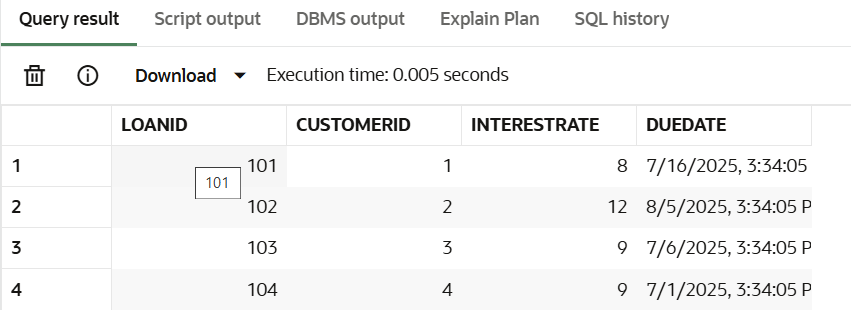
**Print data after commit**

SELECT \* FROM Customers;

SELECT \* FROM Loans;

**Output:**





**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:**

DECLARE

CURSOR cust\_cursor IS

SELECT c.CustomerID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.Age > 60;

BEGIN

FOR rec IN cust\_cursor LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to CustomerID: ' || rec.CustomerID);

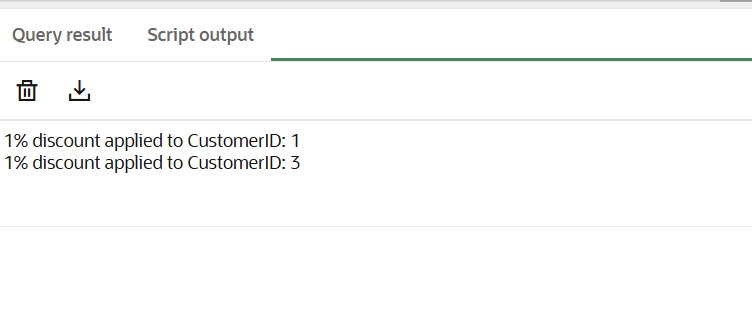
END LOOP;

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:**

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID FROM Customers WHERE Balance > 10000;

BEGIN

FOR rec IN vip\_cursor LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

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

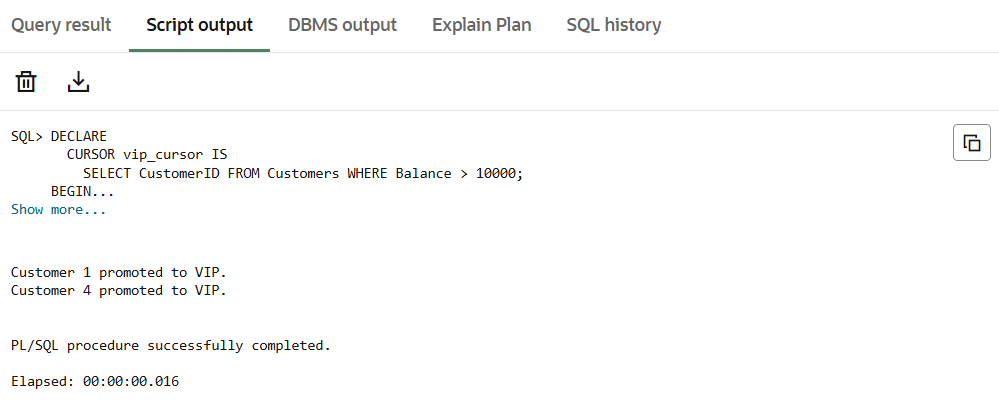
END LOOP;

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:**

DECLARE

CURSOR loan\_cursor IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN loan\_cursor LOOP

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

' for customer ' || rec.Name ||

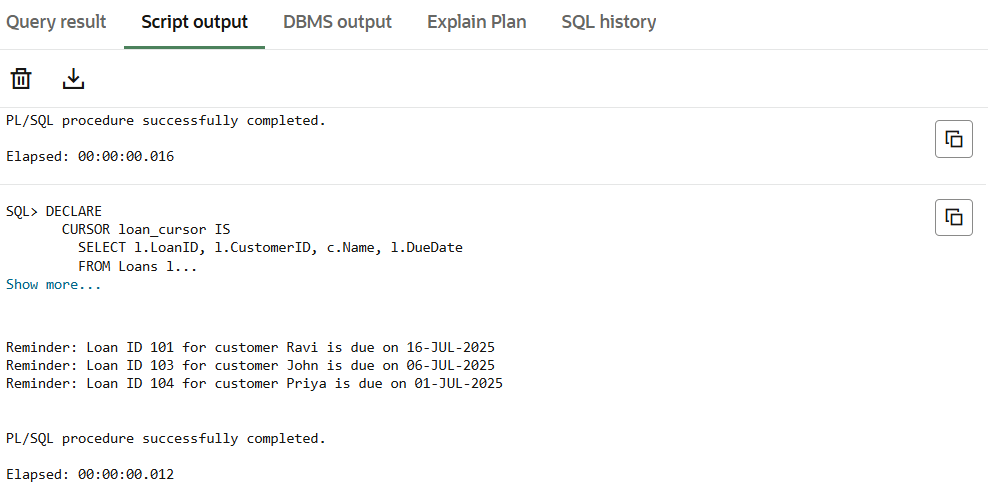
' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output:**



**Exercise 3:** Stored Procedures

**Creating Tables:**

**Accounts Table**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER);

**Employees Table**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Department VARCHAR2(30),

Salary NUMBER);

**Inserting data in to Accounts and Employees tables**

BEGIN

INSERT INTO Accounts VALUES (101, 'Savings', 5000);

INSERT INTO Accounts VALUES (102, 'Savings', 8000);

INSERT INTO Accounts VALUES (103, 'Current', 10000);

INSERT INTO Accounts VALUES (104, 'Savings', 2000);

COMMIT;

END;

/

BEGIN

INSERT INTO Employees VALUES (201, 'Amit', 'HR', 40000);

INSERT INTO Employees VALUES (202, 'Neha', 'IT', 55000);

INSERT INTO Employees VALUES (203, 'Ravi', 'HR', 42000);

INSERT INTO Employees VALUES (204, 'Sara', 'Finance', 60000);

COMMIT;

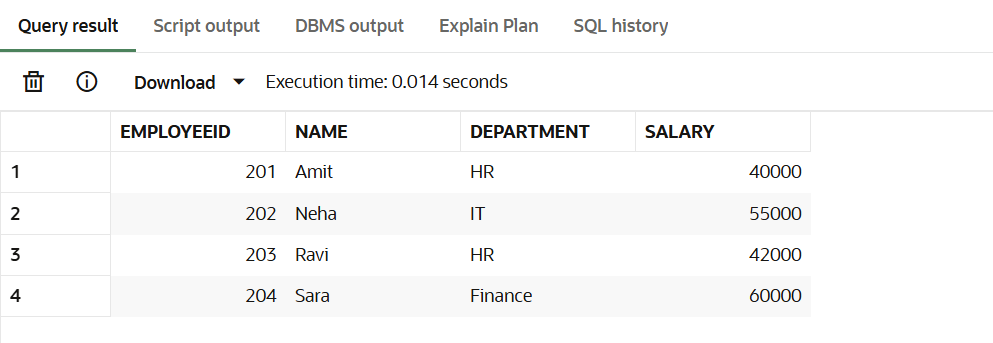
END;

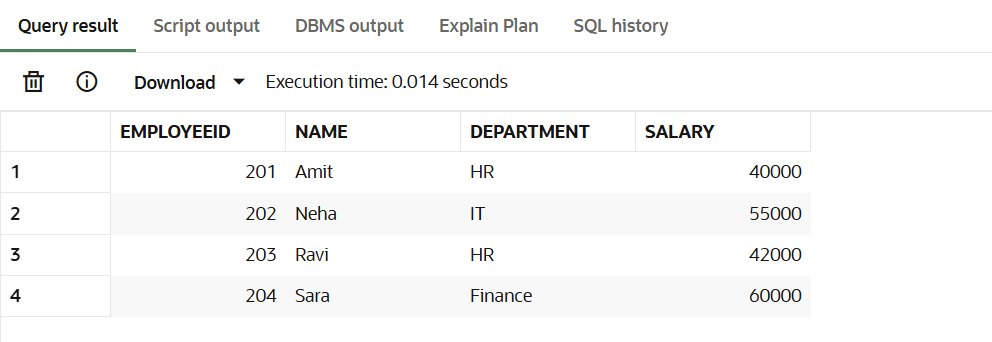
/

SELECT \* FROM ACCOUNTS

SELECT \* FROM EMPLOYEES

**OUTPUT:**





**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 Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

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

COMMIT;

END;

/

-- Call Procedure 1: Process Monthly Interest

BEGIN

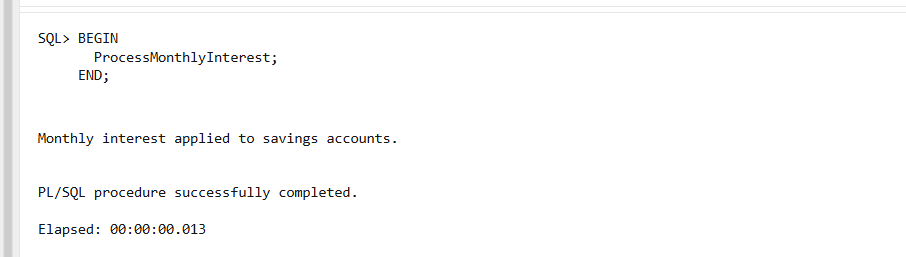
ProcessMonthlyInterest;

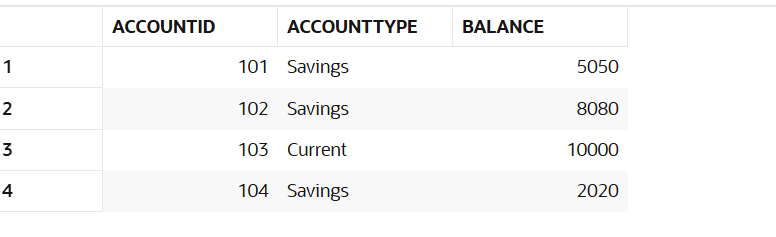
END;

/

SELECT \* FROM ACCOUNTS

**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 applied to employees in department: ' || p\_department);

COMMIT;

END;

/

-- Call Procedure 2: Add 10% bonus to employees in department 10

BEGIN

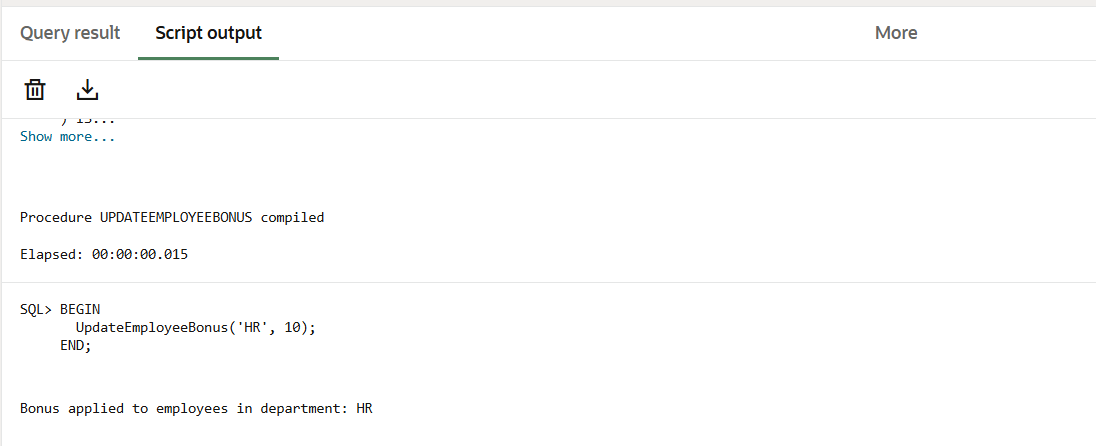
UpdateEmployeeBonus('HR', 10);

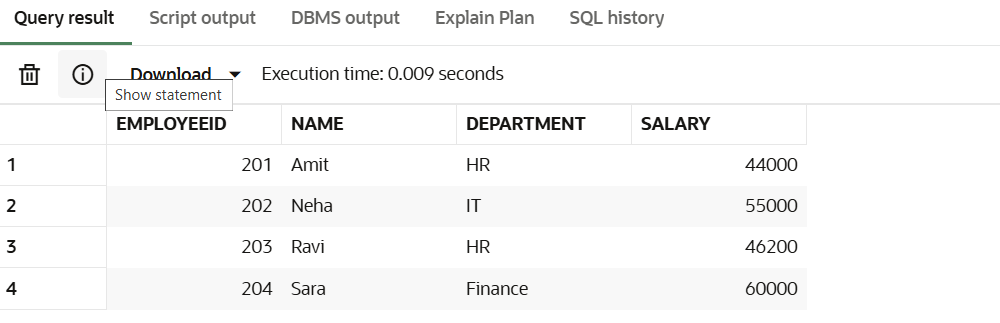
SELECT \* FROM Employees;

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 (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

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;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

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

COMMIT;

END;

/

-- Call Procedure 3: Transfer 3000 from Account 3 to Account 2

BEGIN

TransferFunds(102, 104, 1000);

END;

/

-- Show updated results

SELECT \* FROM Accounts;

**Output**

