Digital Nurture 4.0 – Deep Skilling Program

Java Full Stack Engineer (FSE)

Week 2 – Mandatory Exercises Submission

Name: Khushi

GitHub Profile: https://github.com/KhushiSGowda

Email:khushisnagraj12@gmail.com

Skills Covered:

PL/SQL Programming

PL/SQL Programming

**Exercise 1: Control Structures**

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

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

VALUES (101, 'Senior Person', TO\_DATE('1950-01-01', 'YYYY-MM-DD'), 15000);

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

VALUES (201, 101, 50000, 10, TO\_DATE('2020-01-01', 'YYYY-MM-DD'), TO\_DATE('2025-12-31', 'YYYY-MM-DD'));

BEGIN

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

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

END;

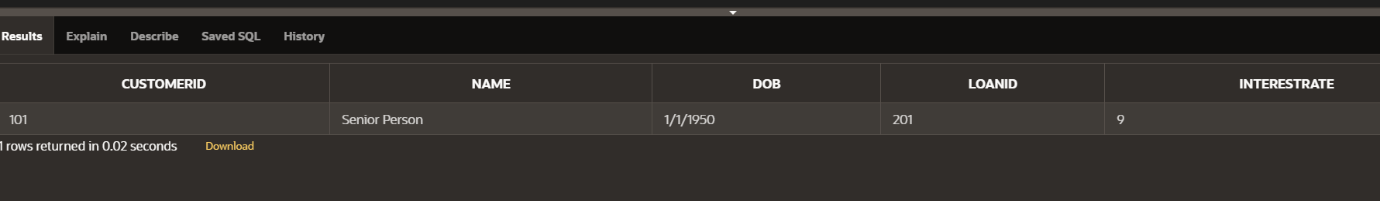
SELECT c.CustomerID, c.Name, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60;

**OUTPUT:**



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

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

VALUES (111, 'Ravi Kumar', TO\_DATE('1980-05-10', 'YYYY-MM-DD'), 20000, SYSDATE);

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

VALUES (112, 'Sita Sharma', TO\_DATE('1992-12-01', 'YYYY-MM-DD'), 8000, SYSDATE);

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

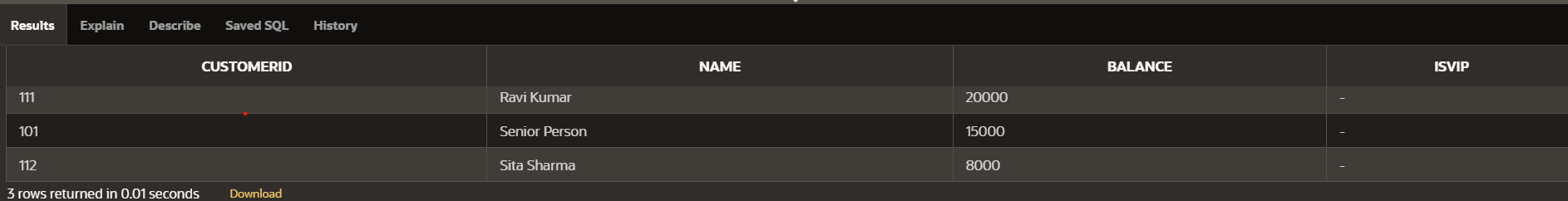
END;

SELECT CustomerID, Name, Balance, IsVIP

FROM Customers

ORDER BY Balance DESC;

**OUTPUT:**

****

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

INSERT INTO Customers (CustomerID, Name, DOB)

VALUES (1, 'Test User', TO\_DATE('1960-01-01', 'YYYY-MM-DD'));

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

VALUES (101, 1, 10000, 5.5, SYSDATE, SYSDATE + 15);

BEGIN

FOR loan\_rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-Mon-YYYY')

);

END LOOP;

END;

SELECT

'Loan ID ' || l.LoanID || ' for customer ' || c.Name ||

' is due on ' || TO\_CHAR(l.EndDate, 'DD-Mon-YYYY') AS Reminder

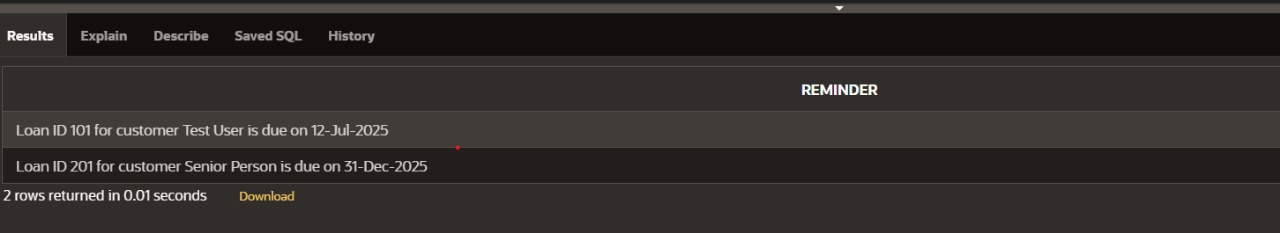
FROM

Loans l

JOIN

Customers c ON l.CustomerID = c.CustomerID;

**OUTPUT:**



**Exercise 3: Stored Procedures**

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

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

VALUES (1, 101, 'Savings', 10000);

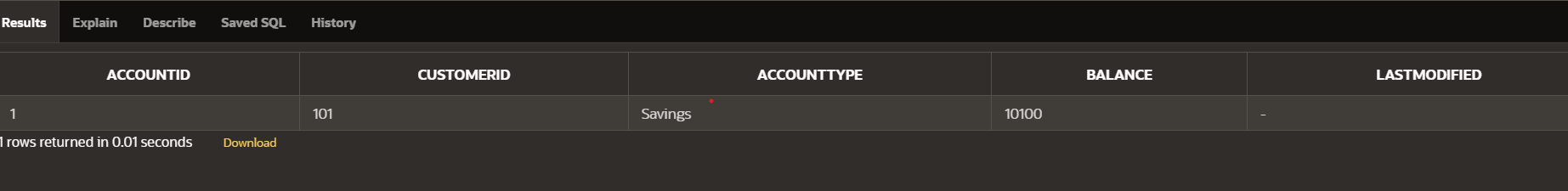
BEGIN

ProcessMonthlyInterest;

END;

SELECT \* FROM Accounts WHERE AccountType = 'Savings';

**OUTPUT:**

****

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

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

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

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

)

IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE Department = p\_Department;

COMMIT;

END;

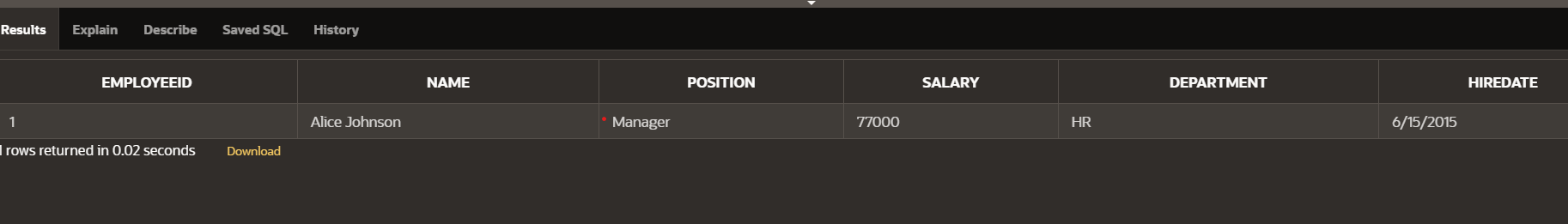
BEGIN

UpdateEmployeeBonus('HR', 10); -- Or use 'IT', based on inserted data

END;

SELECT \* FROM Employees WHERE Department = 'HR';

**OUTPUT:**

****

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

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

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

VALUES (1, 101, 'Savings', 10000);

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccID IN NUMBER,

p\_ToAccID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_balance NUMBER;

insufficient\_funds EXCEPTION;

BEGIN

-- Check balance in source account

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_FromAccID;

IF v\_balance < p\_Amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccID;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_Amount || ' from Account ' || p\_FromAccID || ' to Account ' || p\_ToAccID || ' successful.');

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

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

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Account not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

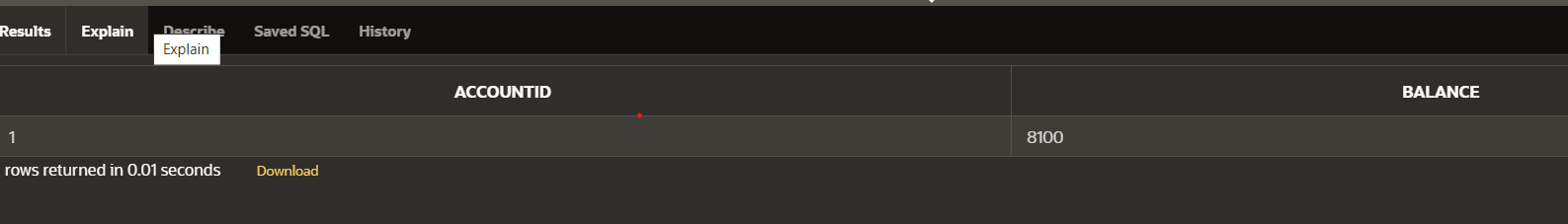
BEGIN

TransferFunds(1, 2, 2000);

END;

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1, 2);

**OUTPUT:**

****