**SKILL : PL/SQL PROGRAMMING**

**Exercise : 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.

**Program:**

CREATE TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    date\_of\_birth DATE);

CREATE TABLE loans (

    loan\_id NUMBER PRIMARY KEY,

    customer\_id NUMBER REFERENCES customers(customer\_id),

    interest\_rate NUMBER,

    status VARCHAR2(20)

);

INSERT INTO customers VALUES (1, '01-JAN-1950'); -- Over 60

INSERT INTO customers VALUES (2, '01-JAN-1990'); -- Under 60

INSERT INTO loans VALUES (101, 1, 5.0, 'ACTIVE'); -- Eligible

INSERT INTO loans VALUES (102, 1, 4.5, 'CLOSED'); -- Ineligible status

INSERT INTO loans VALUES (103, 2, 6.0, 'ACTIVE'); -- Ineligible customer age

SELECT \*from CUSTOMERS;

-- Adding necessary columns for Scenarios 2 and 3

ALTER TABLE customers ADD (balance NUMBER, IsVIP VARCHAR2(5) DEFAULT 'FALSE');

ALTER TABLE loans ADD (due\_date DATE);

UPDATE customers SET balance = 15000, IsVIP = 'FALSE' WHERE customer\_id = 1; -- VIP candidate

UPDATE customers SET balance = 5000, IsVIP = 'FALSE' WHERE customer\_id = 2;  -- Not VIP

UPDATE loans SET due\_date = SYSDATE + 20 WHERE loan\_id = 101; -- Due in 20 days

UPDATE loans SET due\_date = SYSDATE + 40 WHERE loan\_id = 103; -- Due in 40 days

-- Scenario 1: Apply discount to loan interest rates for customers above 60

DECLARE

    CURSOR c\_customers IS

        SELECT c.customer\_id, l.loan\_id, l.interest\_rate

        FROM customers c

        JOIN loans l ON c.customer\_id = l.customer\_id

        WHERE MONTHS\_BETWEEN(SYSDATE, c.date\_of\_birth)/12 > 60

        AND l.status = 'ACTIVE'

        FOR UPDATE OF l.interest\_rate;

    v\_discount\_rate NUMBER := 0.01;

    v\_count NUMBER := 0;

    v\_vip\_count NUMBER := 0;

    v\_reminder\_count NUMBER := 0;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== STARTING BANK OPERATIONS ===');

    DBMS\_OUTPUT.PUT\_LINE('');

    DBMS\_OUTPUT.PUT\_LINE('SCENARIO 1: Applying senior citizen discounts...');

    FOR cust\_rec IN c\_customers LOOP

        v\_count := v\_count + 1;

        DBMS\_OUTPUT.PUT\_LINE('Processing customer ' || cust\_rec.customer\_id ||

                            ', loan ' || cust\_rec.loan\_id ||

                            ', current rate: ' || cust\_rec.interest\_rate);

        UPDATE loans

        SET interest\_rate = GREATEST(interest\_rate - v\_discount\_rate, 0)

        WHERE loan\_id = cust\_rec.loan\_id;

        DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount. New rate: ' ||

                            (cust\_rec.interest\_rate - v\_discount\_rate));

    END LOOP;

    IF v\_count = 0 THEN

        DBMS\_OUTPUT.PUT\_LINE('No eligible customers found for discount');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Successfully applied discounts to ' || v\_count || ' loans');

    END IF;

    DBMS\_OUTPUT.PUT\_LINE('');

    -- Scenario 2: VIP status promotion

    DBMS\_OUTPUT.PUT\_LINE('SCENARIO 2: Promoting customers to VIP status...');

    FOR vip\_rec IN (SELECT customer\_id FROM customers WHERE balance > 10000 FOR UPDATE) LOOP

        v\_vip\_count := v\_vip\_count + 1;

        UPDATE customers

        SET IsVIP = 'TRUE'

        WHERE customer\_id = vip\_rec.customer\_id;

        DBMS\_OUTPUT.PUT\_LINE('Promoted customer ' || vip\_rec.customer\_id || ' to VIP status');

    END LOOP;

    IF v\_vip\_count = 0 THEN

        DBMS\_OUTPUT.PUT\_LINE('No customers qualified for VIP status');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Promoted ' || v\_vip\_count || ' customers to VIP status');

    END IF;

    DBMS\_OUTPUT.PUT\_LINE('');

    -- Scenario 3: Loan due reminders

    DBMS\_OUTPUT.PUT\_LINE('SCENARIO 3: Sending loan due reminders...');

    FOR due\_rec IN (

        SELECT c.customer\_id, l.loan\_id, l.due\_date,

               TRUNC(l.due\_date - SYSDATE) AS days\_remaining

        FROM customers c

        JOIN loans l ON c.customer\_id = l.customer\_id

        WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

        AND l.status = 'ACTIVE'

    ) LOOP

        v\_reminder\_count := v\_reminder\_count + 1;

        DBMS\_OUTPUT.PUT\_LINE('REMINDER: Customer ' || due\_rec.customer\_id ||

                            ', your loan ' || due\_rec.loan\_id ||

                            ' is due in ' || due\_rec.days\_remaining ||

                            ' days on ' || TO\_CHAR(due\_rec.due\_date, 'DD-MON-YYYY'));

    END LOOP;

    IF v\_reminder\_count = 0 THEN

        DBMS\_OUTPUT.PUT\_LINE('No loans due in the next 30 days');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Sent ' || v\_reminder\_count || ' loan due reminders');

    END IF;

    DBMS\_OUTPUT.PUT\_LINE('');

    DBMS\_OUTPUT.PUT\_LINE('=== ALL OPERATIONS COMPLETED SUCCESSFULLY ===');

    COMMIT;

EXCEPTION

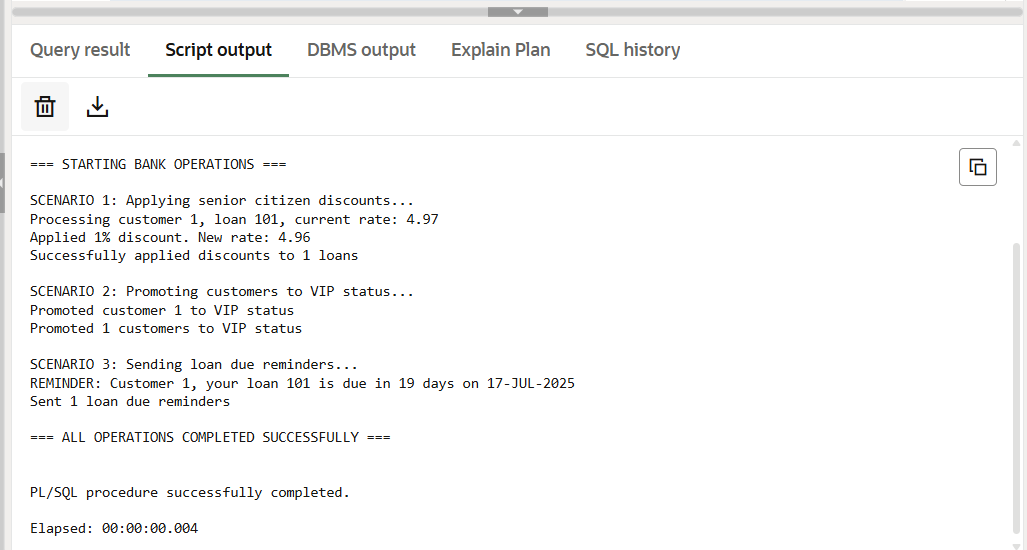
    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error in bank operations: ' || SQLERRM);

END;

Output:



**Exercise : Stored Procedures**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

**Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Scenario 2:** Manage errors when updating employee salaries.

**Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Scenario 3:** Ensure data integrity when adding a new customer.

**Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**Program:**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountHolder VARCHAR2(100),

Balance NUMBER(10,2),

AccountType VARCHAR2(20)

);

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

EmpName VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER(10,2)

);

INSERT INTO Accounts VALUES (1, 'Alice', 10000, 'Savings');

INSERT INTO Accounts VALUES (2, 'Bob', 15000, 'Savings');

INSERT INTO Accounts VALUES (3, 'Charlie', 8000, 'Current');

INSERT INTO Accounts VALUES (4, 'David', 5000, 'Savings');

INSERT INTO Employees VALUES (101, 'Ravi', 'HR', 50000);

INSERT INTO Employees VALUES (102, 'Priya', 'HR', 52000);

INSERT INTO Employees VALUES (103, 'Amit', 'Sales', 48000);

INSERT INTO Employees VALUES (104, 'Neha', 'Sales', 55000);

COMMIT;

----------------------------------------------------------

-- SCENARIO 1: Process Monthly Interest

----------------------------------------------------------

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

END;

BEGIN

ProcessMonthlyInterest;

END;

SELECT \* FROM Accounts;

----------------------------------------------------------

-- SCENARIO 2: Update Employee Bonus

----------------------------------------------------------

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE Department = dept\_name;

END;

BEGIN

UpdateEmployeeBonus('HR', 10);

END;

SELECT \* FROM Employees;

----------------------------------------------------------

-- SCENARIO 3: Transfer Funds

----------------------------------------------------------

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) AS

insufficient\_balance EXCEPTION;

balance\_from NUMBER;

BEGIN

SELECT Balance INTO balance\_from FROM Accounts WHERE AccountID = from\_account;

IF balance\_from < amount THEN

RAISE insufficient\_balance;

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

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

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'One of the accounts does not exist.');

END;

BEGIN

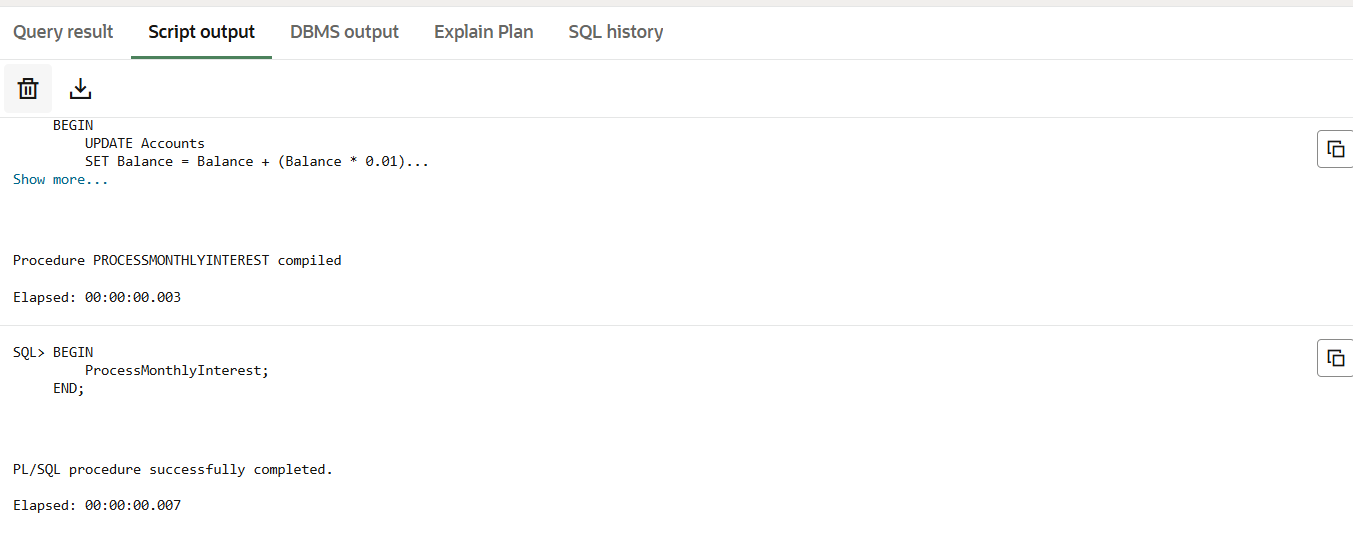
TransferFunds(2, 4, 1000);

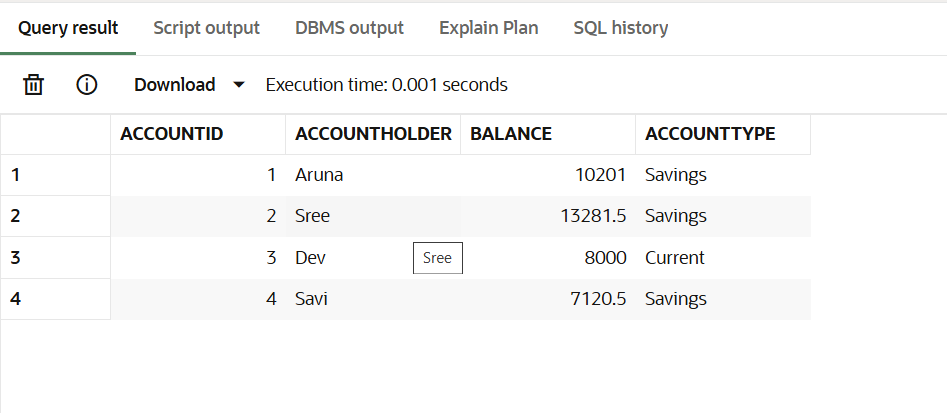
END;

SELECT \* FROM Accounts;

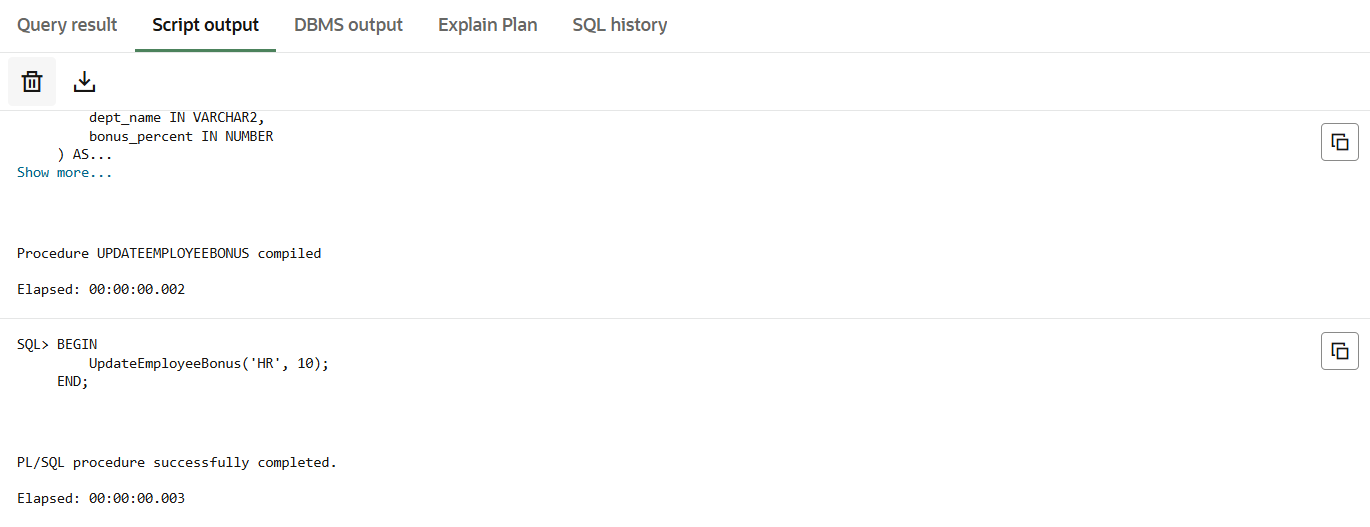
**Output:**

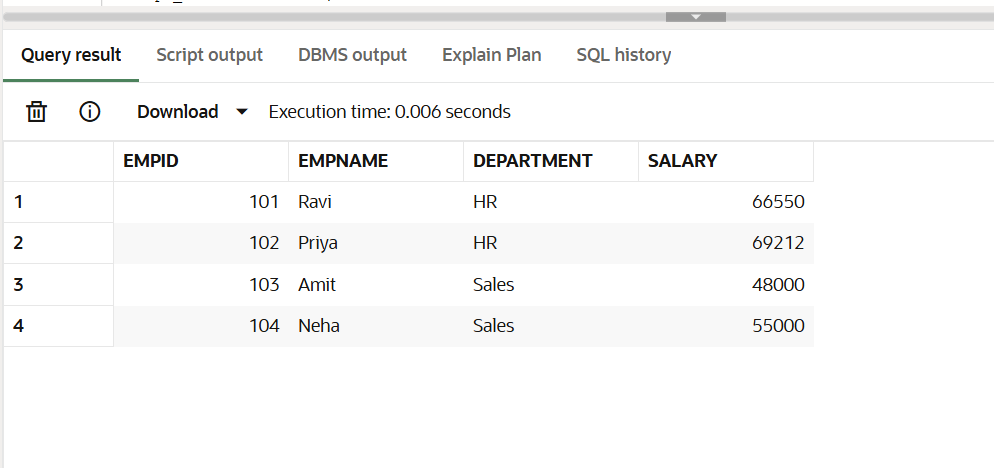
**After Running ProcessMonthlyInterest**





**After Running UpdateEmployeeBonus('HR', 10)**





**After Running TransferFunds(2, 4, 1000)**

