**Week-2 PLSQL Exercises**

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:**

INSERTING DATAS:

CREATE TABLE loan\_customers (

customer\_id NUMBER,

interest\_rate NUMBER,

age NUMBER

);

INSERT INTO loan\_customers VALUES (1, 10.5, 65);

INSERT INTO loan\_customers VALUES (2, 9.0, 45);

INSERT INTO loan\_customers VALUES (3, 11.0, 70);

COMMIT;

-- PL/SQL block to update interest rates and show output

BEGIN

FOR cust IN (

SELECT customer\_id, interest\_rate

FROM loan\_customers

WHERE age > 60

)

LOOP

UPDATE loan\_customers

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = cust.customer\_id;

END LOOP;

COMMIT;

END;

/

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:

PL/SQL Block for Promoting VIPs:

BEGIN

FOR cust IN (

SELECT customer\_id

FROM customers

WHERE balance > 10000

)

LOOP

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = cust.customer\_id;

END LOOP;

COMMIT;

END;

/

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;

DECLARE

CURSOR due\_loans IS

SELECT loan\_id, customer\_id, due\_date

FROM loans

WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan IN due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID ' || loan.customer\_id ||

' has a loan (Loan ID: ' || loan.loan\_id || ') due on ' ||

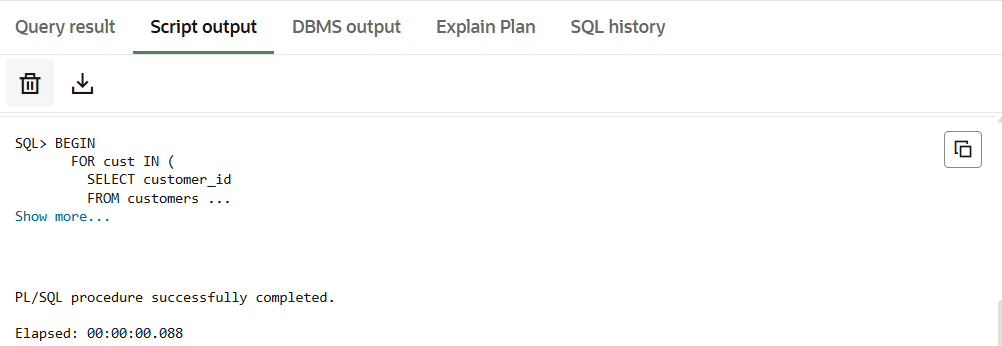
TO\_CHAR(loan.due\_date, 'DD-MON-YYYY'));

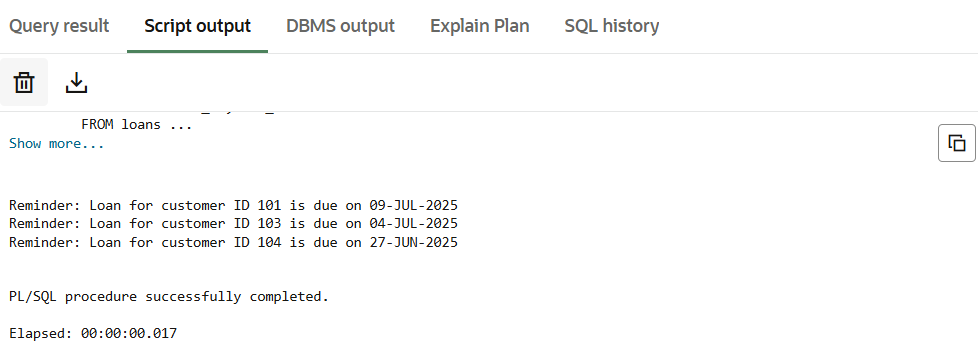
END LOOP;

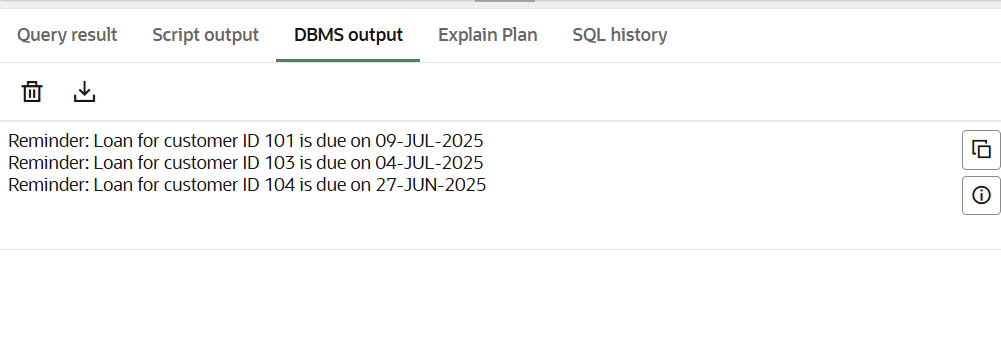
END;

/

Output:







1. **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:**

**INSERT DATA:**

CREATE TABLE savings\_accounts (

  account\_id NUMBER,

  customer\_id NUMBER,

  balance NUMBER

);

INSERT INTO savings\_accounts VALUES (1, 101, 10000);

INSERT INTO savings\_accounts VALUES (2, 102, 20000);

COMMIT;

**Process Monthly Interest (1% on Savings Accounts):**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01);

COMMIT;

END;

/

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:**

**INSERT DATA:**

CREATE TABLE employees (

emp\_id NUMBER,

emp\_name VARCHAR2(100),

department\_id NUMBER,

salary NUMBER

);

-- Sample data

INSERT INTO employees VALUES (1, 'Ravi', 10, 50000);

INSERT INTO employees VALUES (2, 'Anita', 20, 60000);

INSERT INTO employees VALUES (3, 'Vikram', 10, 55000);

COMMIT;

**Update Employee Bonus by Department:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

  dept\_id IN NUMBER,

  bonus\_percent IN NUMBER

) AS

BEGIN

  UPDATE employees

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

  WHERE department\_id = dept\_id;

  COMMIT;

END;

/

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:**

**INSERT DATA:**

CREATE TABLE bank\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

-- Sample data

INSERT INTO bank\_accounts VALUES (101, 1, 15000);

INSERT INTO bank\_accounts VALUES (102, 1, 5000);

COMMIT;

**Transfer Funds Between Accounts:**

CREATE OR REPLACE PROCEDURE TransferFunds(

  from\_account\_id IN NUMBER,

  to\_account\_id IN NUMBER,

  amount IN NUMBER

) AS

  insufficient\_balance EXCEPTION;

BEGIN

  -- Check balance

  DECLARE

    current\_balance NUMBER;

  BEGIN

    SELECT balance INTO current\_balance

    FROM bank\_accounts

    WHERE account\_id = from\_account\_id;

    IF current\_balance < amount THEN

      RAISE insufficient\_balance;

    END IF;

    -- Deduct from source

    UPDATE bank\_accounts

    SET balance = balance - amount

    WHERE account\_id = from\_account\_id;

    -- Add to destination

    UPDATE bank\_accounts

    SET balance = balance + amount

    WHERE account\_id = to\_account\_id;

    COMMIT;

  END;

EXCEPTION

  WHEN insufficient\_balance THEN

    DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance for transfer.');

  WHEN NO\_DATA\_FOUND THEN

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

  WHEN OTHERS THEN

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

END;

/

Output:

