**Exercise 1: Control Structures**

**SCENARIO 1: 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.**

BEGIN

  FOR cust\_rec IN (

    SELECT customer\_id, loan\_interest\_rate, age

    FROM customers

    WHERE age > 60

  ) LOOP

    UPDATE customers

    SET loan\_interest\_rate = loan\_interest\_rate - 1

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

    DBMS\_OUTPUT.PUT\_LINE('Discount applied for Customer ID: ' || cust\_rec.customer\_id);

  END LOOP;

  COMMIT;

END;

**SCENARIO 2: 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.**

BEGIN

  FOR cust\_rec IN (

    SELECT customer\_id, balance

    FROM customers

    WHERE balance > 10000

  ) LOOP

    UPDATE customers

    SET isvip = 'TRUE'

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

    DBMS\_OUTPUT.PUT\_LINE('VIP status granted to Customer ID: ' || cust\_rec.customer\_id);

  END LOOP;

  COMMIT;

END;

**SCENARIO 3: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.**

BEGIN

  FOR loan\_rec IN (

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

    FROM loans l

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

    WHERE l.due\_date <= SYSDATE + 30

  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE(

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

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

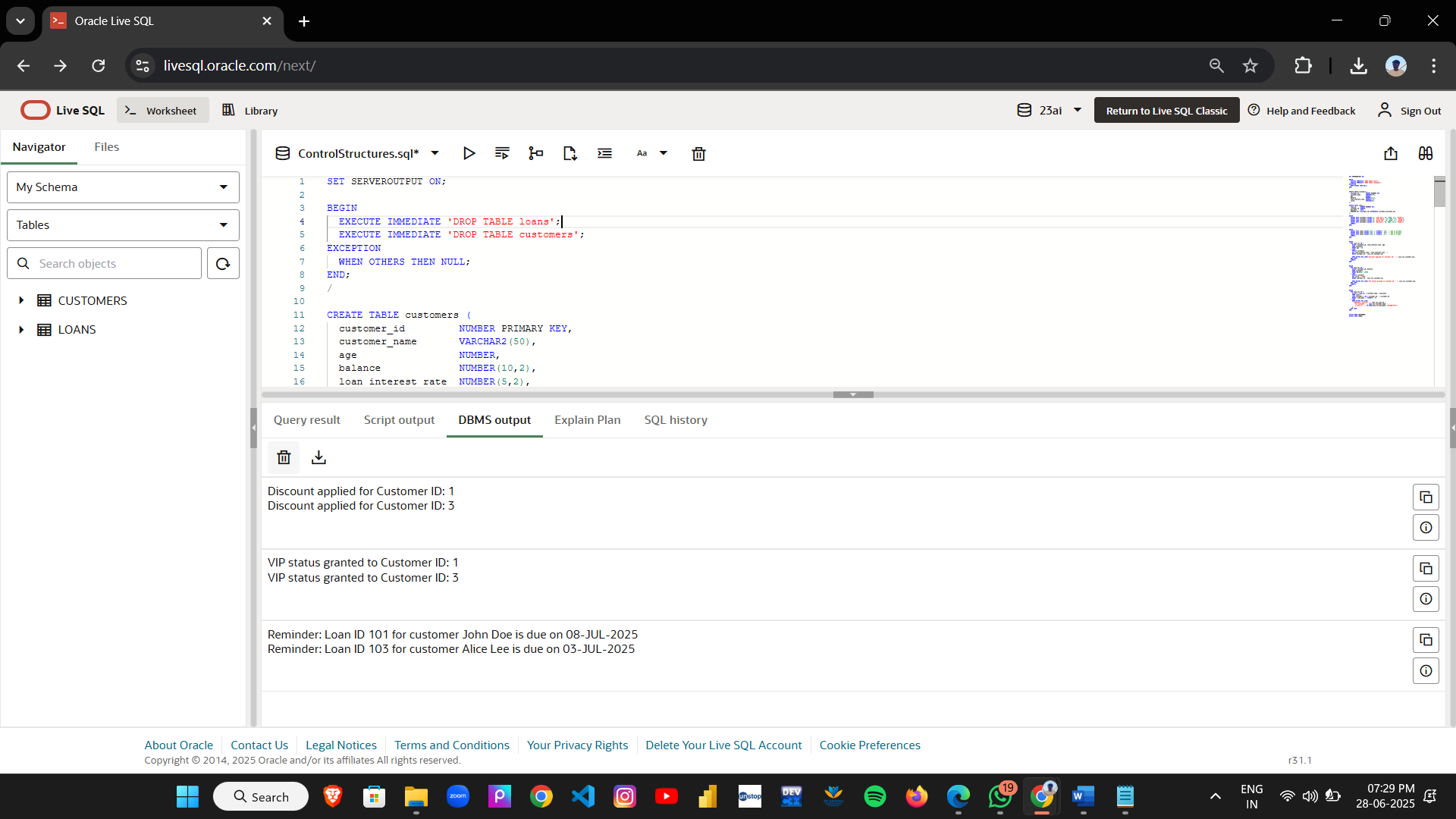
      ' is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY')

    );

  END LOOP;

END;

**OUTPUT**:



**Exercise 3: Stored Procedures**

**Scenario 1: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.**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE UPPER(account\_type) = 'SAVINGS';

DBMS\_OUTPUT.PUT\_LINE('Interest applied to all savings accounts.');

COMMIT;

END;

**Scenario 2:Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_bonus\_pct / 100)

WHERE LOWER(department) = LOWER(p\_dept);

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_pct || '% applied to ' || p\_dept || ' department.');

COMMIT;

END;

**Scenario 3: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**

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 account\_id = p\_from\_account;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Not enough balance in source account.');

END IF;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

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

COMMIT;

END;

**Execution of all procedures:**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('ProcessMonthlyInterest:');

ProcessMonthlyInterest;

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('UpdateEmployeeBonus (Sales, 10%):');

UpdateEmployeeBonus('Sales', 10);

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('TransferFunds (103 -> 102, amount 2000):');

TransferFunds(103, 102, 2000);

DBMS\_OUTPUT.PUT\_LINE('');

END;

**Output:**

