**WEEK-02 HANDS ON SOLUTIONS**

**PL/SQL EXERCISE SOLUTIONS**

**Exercise 1: Control Structures**

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

o 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.

o 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.

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

This PL/SQL program is designed to demonstrate the use of control structures in Oracle by simulating a real-world banking scenario.

Two tables are created: customers and loans.

The customers table holds details about each customer, such as their customer\_id, name, age, balance, and a flag isvip which denotes if they are considered a VIP.

The loans table stores loan information including loan\_id, customer\_id (foreign key), interest\_rate, and due\_date.

These two tables are linked via a foreign key relationship to represent which customer owns which loan.

The script proceeds with three main procedures (PL/SQL blocks), each addressing a specific business rule using control structures.

In the first scenario, a FOR loop and an IF condition are used to iterate through customers and apply a 1% interest rate discount to all customers over the age of 60.

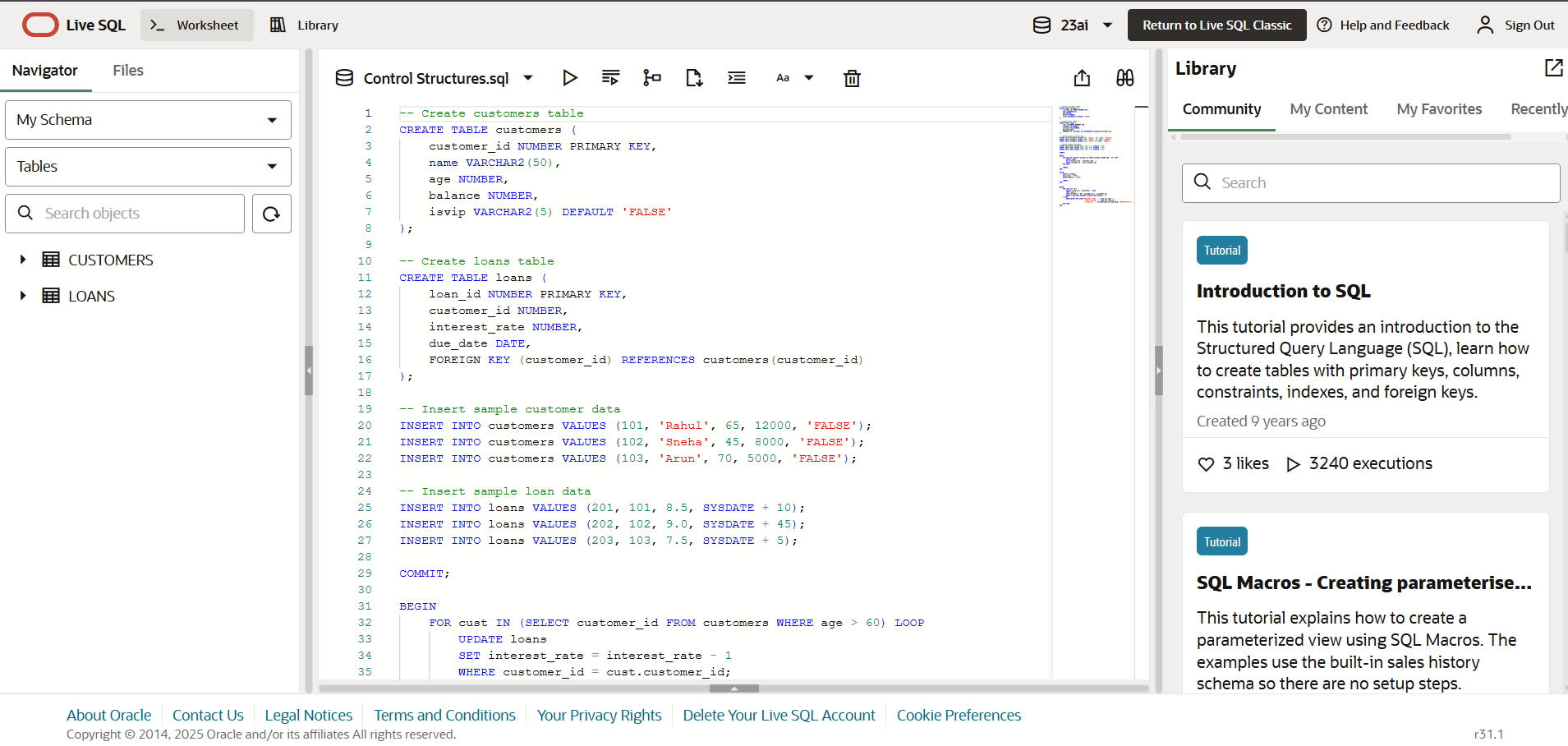
In the second block, the script directly updates the isvip status of all customers with balances exceeding $10,000 by setting the flag to 'TRUE'.

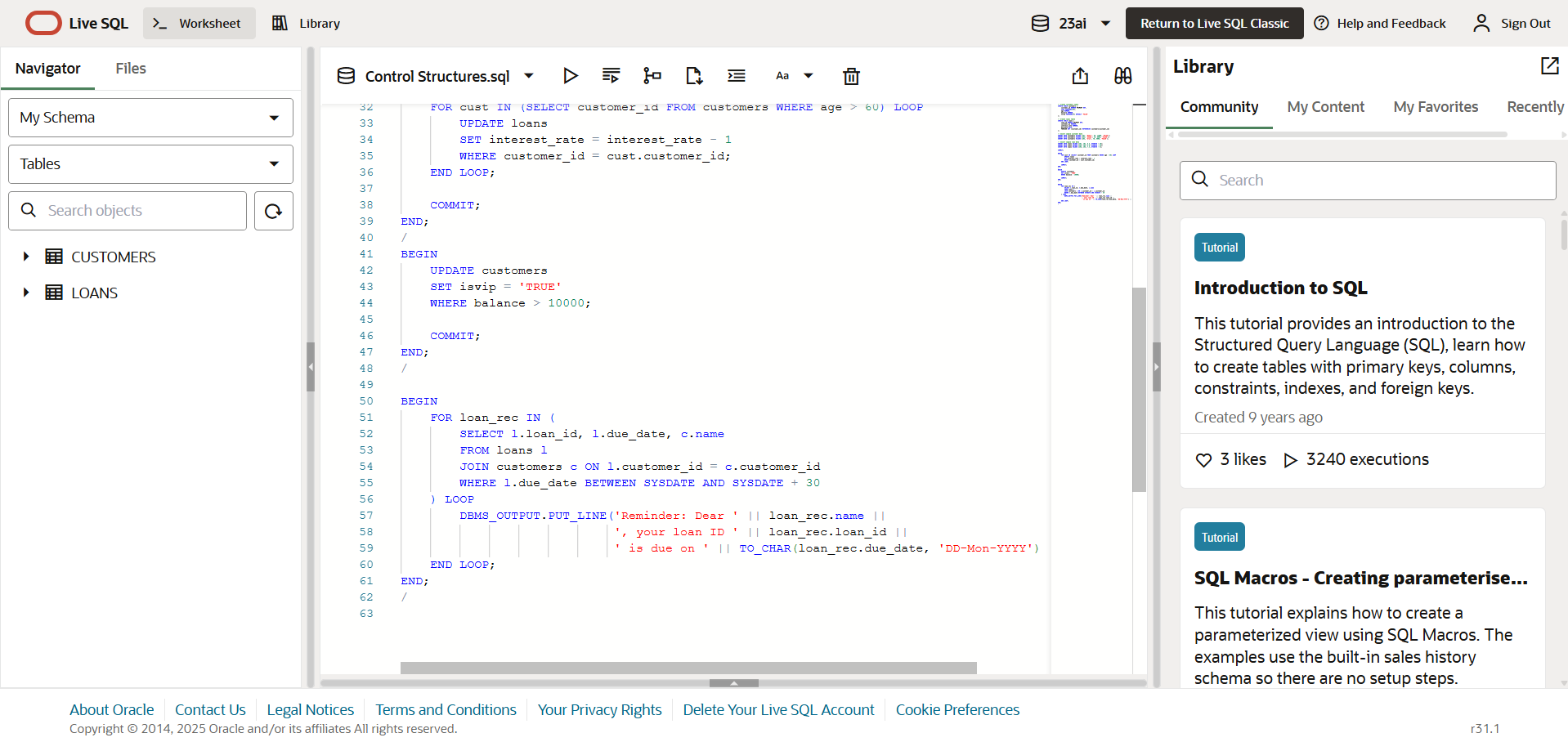
This step is useful for classifying high-value customers. The third block is a reporting operation that finds all loans due within the next 30 days.

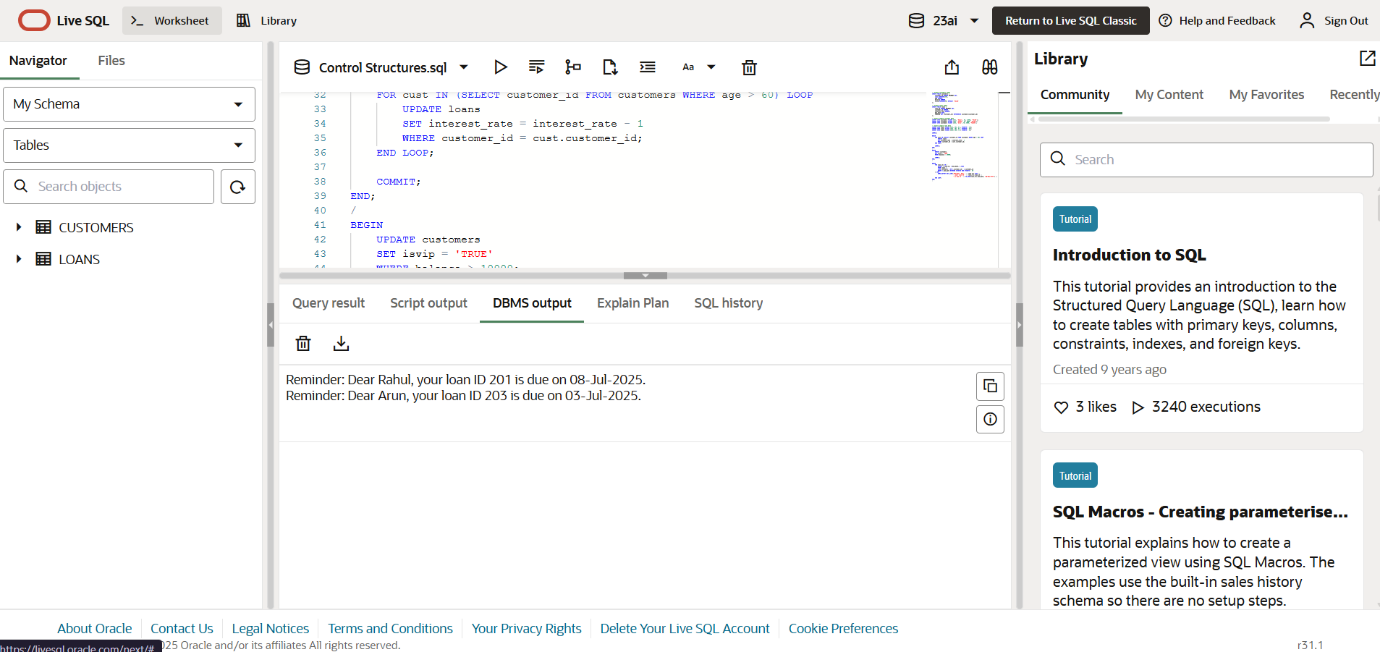
For each such loan, a reminder message is printed using DBMS\_OUTPUT.PUT\_LINE, including the customer’s name, loan ID, and due date.

When this PL/SQL code is executed, it modifies the loan interest rates and customer VIP flags according to the given conditions and displays reminder messages for loans due soon.

The procedures demonstrate key PL/SQL features such as cursor-based loops, conditional logic, data manipulation (UPDATE), and output statements.







**Exercise 3: Stored Procedures**

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

o 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.

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

o 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.

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

o 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.

**EXPLANATION:**

In this exercise, PL/SQL stored procedures are used to simulate core banking operations such as interest processing, bonus calculation, and fund transfers.

To support these operations, I created three tables: savings\_accounts, employees, and accounts.

Each table is designed with attributes that reflect real-world data structures used in banking systems.

The savings\_accounts table contains three fields: account\_id (primary key), customer\_id, and balance.

This table stores information about individual customer savings accounts. The first stored procedure, ProcessMonthlyInterest, operates on this table by applying a fixed 1% interest to each balance using a simple UPDATE query.

This simulates monthly interest being credited to every customer's savings account.

The second table, employees, includes attributes such as emp\_id (primary key), name, department\_id, and salary.

It represents the bank's employee database. The procedure UpdateEmployeeBonus uses this table to provide performance-based salary increments.

It takes a department ID and bonus percentage as parameters and increases the salary of every employee in that department accordingly.

Lastly, the accounts table is used to simulate a general-purpose customer account system, which is useful for fund transfers.

It includes account\_id (primary key), customer\_id, and balance.

The TransferFunds procedure operates on this table to move money from one account to another.

Before performing the transfer, it checks whether the source account has sufficient funds.

If the balance is insufficient, the procedure raises an application error; otherwise, it updates both accounts accordingly.

Each stored procedure performs database updates without requiring user input/output during execution.

The correctness of the procedure is verified by executing SELECT queries after running the procedures, which confirm that interest is applied, bonuses are added, and funds are successfully transferred between accounts.

This exercise effectively demonstrates how PL/SQL procedures can be used to implement modular, secure, and reusable components in a banking database application.

