**Banking System Simulation**

**Objective**

The primary objective of this assignment is to demonstrate the understanding and application of **PL/SQL control structures** and **stored procedures** in solving real-time banking system scenarios. These exercises simulate how financial institutions manage operations such as interest calculations, customer status upgrades, employee compensation, and fund transfers using Oracle PL/SQL. This report presents a structured approach to handling such tasks using SQL logic, procedural constructs, and data manipulation statements.

**Exercise 1: Control Structures (PL/SQL Blocks)**

This section demonstrates the use of PL/SQL loops and conditionals to perform operations on customer and loan data.

**Code:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

**Example Scripts for Sample Data Insertion**

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

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

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

VALUES (1, 1, 'Savings', 1000, SYSDATE);

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

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

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

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

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

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

**Scenario 1: Apply Loan Interest Discount for Senior Citizens**

**Requirement:**  
The bank wants to apply a 1% discount to the current loan interest rates for all customers aged above 60.

**Approach:**

1. Fetch all customers and calculate their age based on date of birth.
2. Check if the customer is above 60 years.
3. If yes, apply a 1% discount to their current loan interest rate.
4. Update the corresponding record in the Loans table.

**PL/SQL Code:**

BEGIN

FOR c IN (

SELECT CustomerID, TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB)/12) AS Age

FROM Customers

) LOOP

IF c.Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

END IF;

END LOOP;

END;

**Scenario 2: Promote Customers to VIP Based on Balance**

**Requirement:**  
Customers with a balance greater than $10,000 should be promoted to VIP status by setting a flag.

**Approach:**

1. Loop through all customers.
2. Check if the balance is greater than 10,000.
3. If true, update the IsVIP flag to 'TRUE'.

**PL/SQL Code:**

BEGIN

FOR c IN (

SELECT CustomerID, Balance

FROM Customers

) LOOP

IF c.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = c.CustomerID;

END IF;

END LOOP;

END;

**Scenario 3: Send Loan Due Reminders**

**Requirement:**  
The bank wants to send reminder messages to customers whose loans are due within the next 30 days.

**Approach:**

1. Select loans where the due date (EndDate) is within 30 days from the current date.
2. Join with the Customers table to get customer names.
3. Print a reminder message using DBMS\_OUTPUT.PUT\_LINE.

**PL/SQL Code:**

BEGIN

FOR l IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || l.LoanID ||

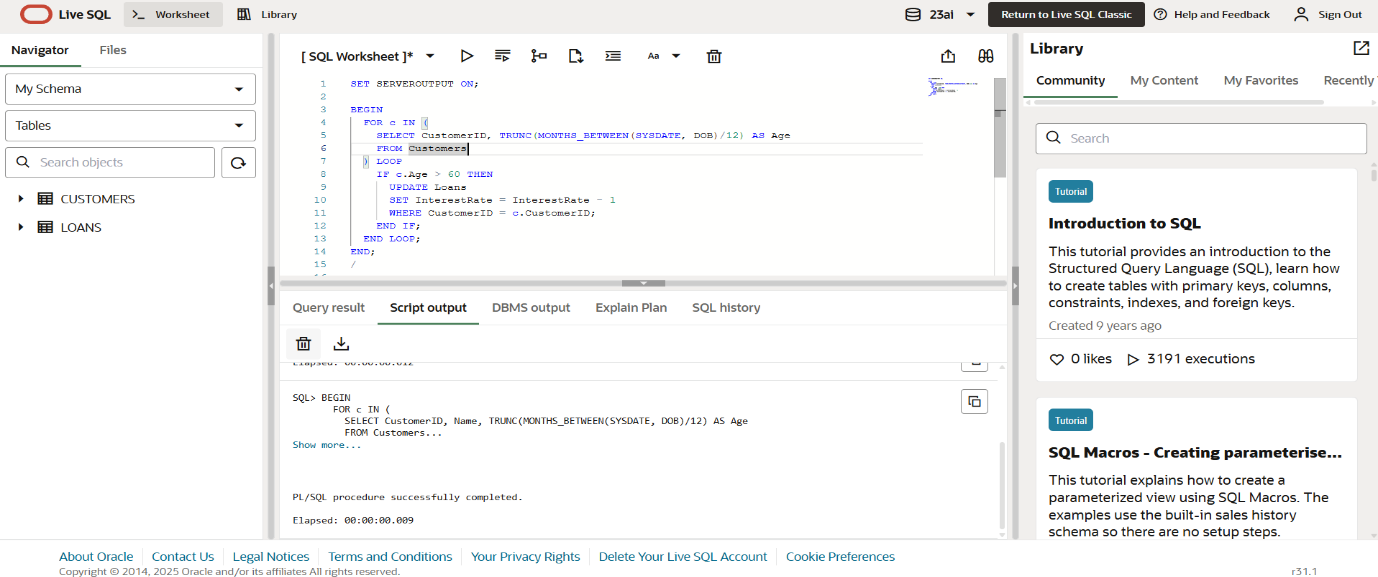
' for customer ' || l.Name ||

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

END LOOP;

END;

**Output:**

****

**Exercise 2: Stored Procedures**

This section includes reusable PL/SQL stored procedures that encapsulate logic for processing interest, updating bonuses, and handling fund transfers.

**Scenario 1: Process Monthly Interest for Savings Accounts**

**Procedure Name:** ProcessMonthlyInterest

**Requirement:**  
Apply 1% monthly interest to the balance of all savings accounts.

**Approach:**

1. Identify all accounts where AccountType = 'Savings'.
2. For each savings account, increase the balance by 1%.
3. Update the balance accordingly.

**Stored Procedure:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = acc.Balance + (acc.Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

END;

**Scenario 2: Employee Bonus Allocation Based on Department**

**Procedure Name:** UpdateEmployeeBonus

**Requirement:**  
The bank wants to provide a performance-based bonus to employees of a specific department.

**Approach:**

1. Accept department name and bonus percentage as input parameters.
2. Calculate bonus based on salary.
3. Update the salary field.

**Stored Procedure:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = dept;

END;

**Scenario 3: Transfer Funds Between Accounts**

**Procedure Name:** TransferFunds

**Requirement:**  
Allow customers to transfer funds from one account to another, ensuring the source account has sufficient balance.

**Approach:**

1. Accept source account ID, target account ID, and amount as parameters.
2. Check if the source account has a balance greater than or equal to the amount.
3. If sufficient, debit from source and credit to target.
4. If not, raise an error.

**Stored Procedure:**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amt IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = from\_acc;

IF from\_balance < amt THEN

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

ELSE

UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = from\_acc;

UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = to\_acc;

END IF;

END;

**Sample Data Used for Testing**

Below are the insert statements used to populate the database for simulation and testing.

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

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

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

VALUES (1, 1, 'Savings', 1000, SYSDATE);

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

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

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

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

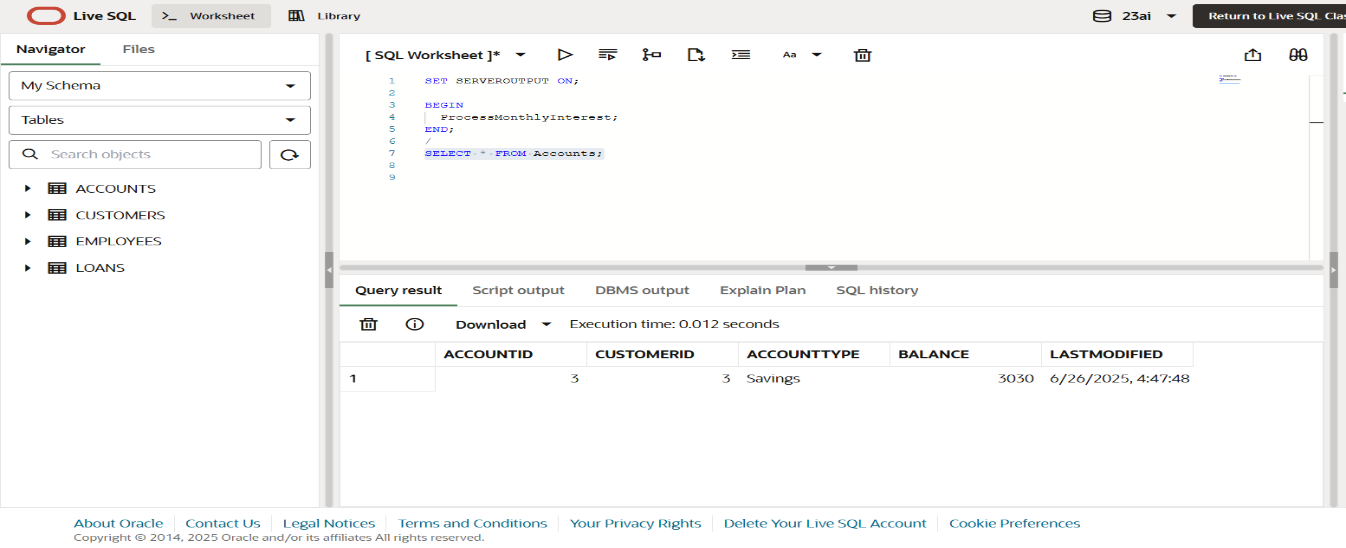
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

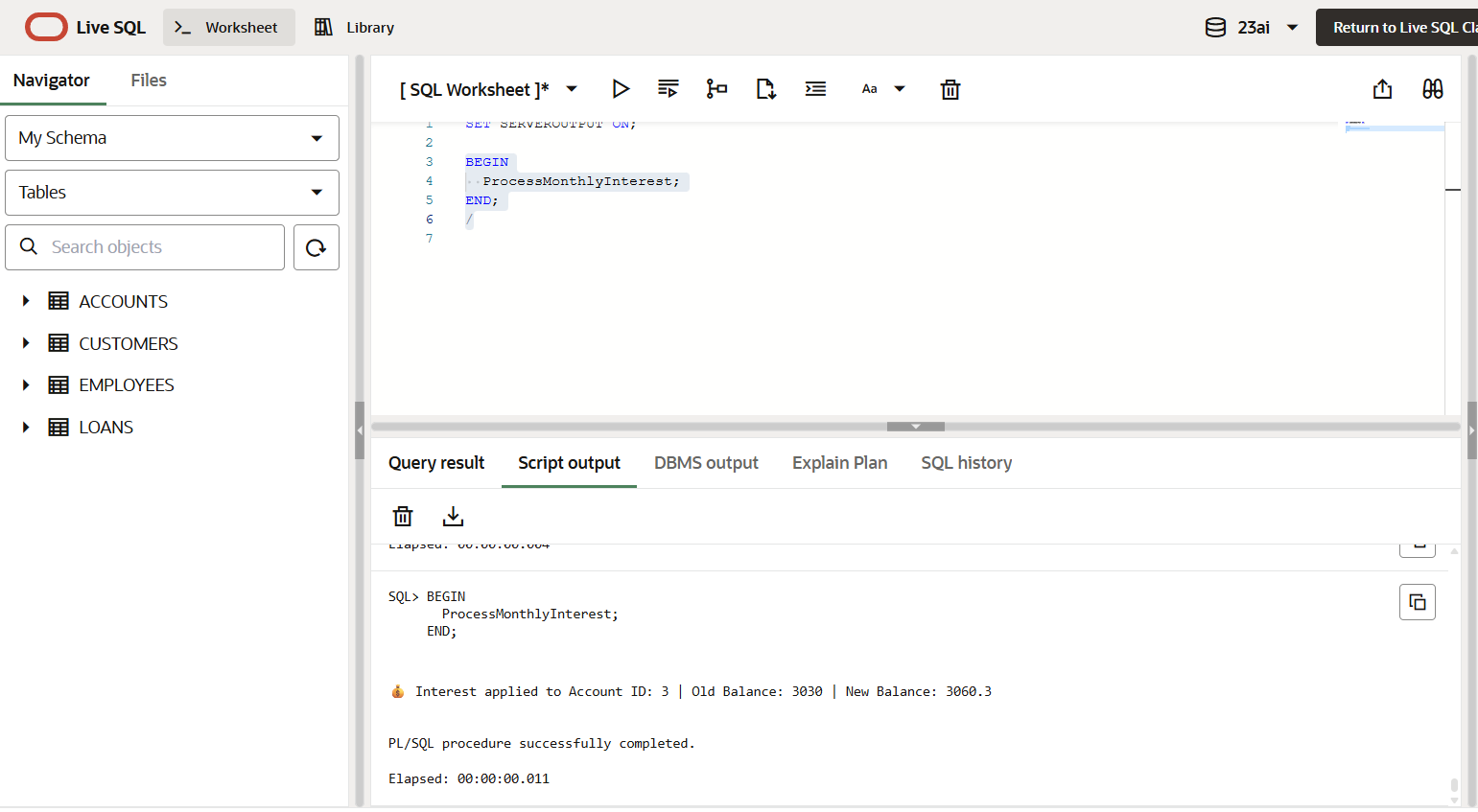
VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

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

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

**Output:**

****

****

**Conclusion**

This assignment provided hands-on exposure to writing PL/SQL blocks and procedures to manage various banking operations. By simulating customer management, account operations, employee bonus processing, and inter-account transfers, it becomes evident how PL/SQL enables robust and reliable transaction control in enterprise applications. The structured use of control structures and procedures enhances maintainability, performance, and modularity in real-world database systems.