PL/SQL Bank Scenarios Implementation

# Introduction

This document outlines PL/SQL implementations for common banking scenarios involving control structures and stored procedures. These scenarios simulate real-life banking operations such as interest calculations, account management, customer notifications, and fund transfers.

# Objectives

- Apply conditional logic using PL/SQL control structures.

- Utilize stored procedures for reusable banking operations.

- Enhance customer experience by automating routine tasks.

- Ensure data consistency and transactional integrity.

# Implementation

## EXERCISE 1: CONTROL STRUCTURES

### Scenario 1: Apply Discount to Loan Interest Rates for Senior Citizens

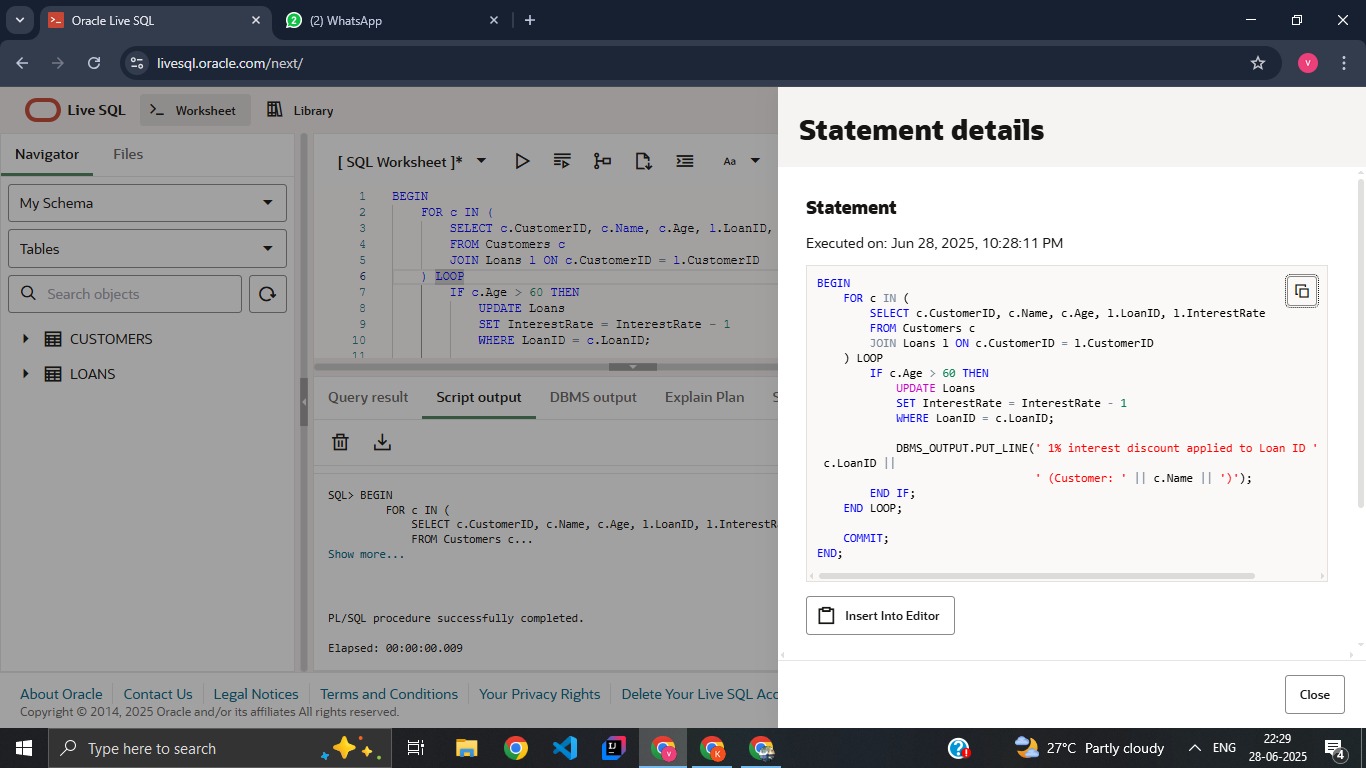
BEGIN  
 FOR cust\_rec IN (SELECT CustomerID, Age FROM Customers) LOOP  
 IF cust\_rec.Age > 60 THEN  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE CustomerID = cust\_rec.CustomerID;  
 END IF;  
 END LOOP;  
 COMMIT;  
END;  
/

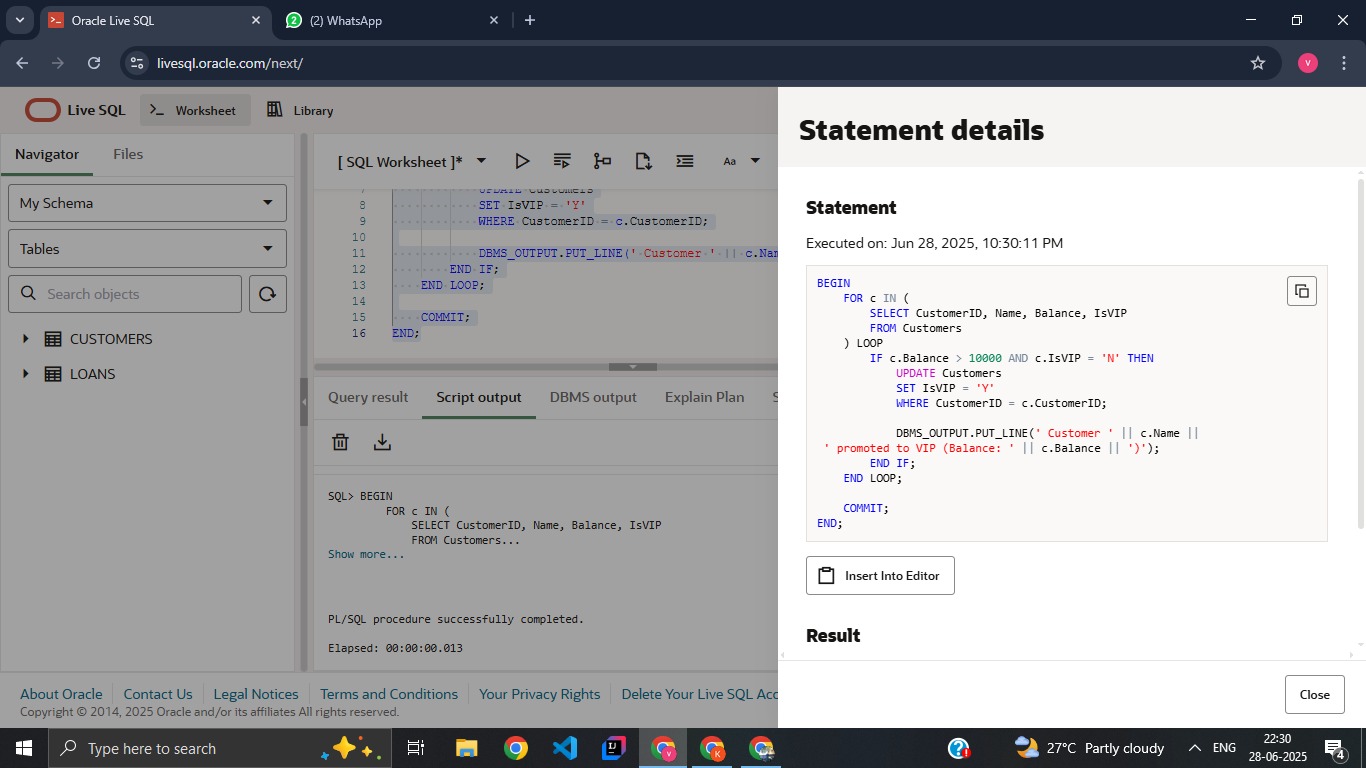
### Scenario 2: Promote Customers to VIP Based on Balance

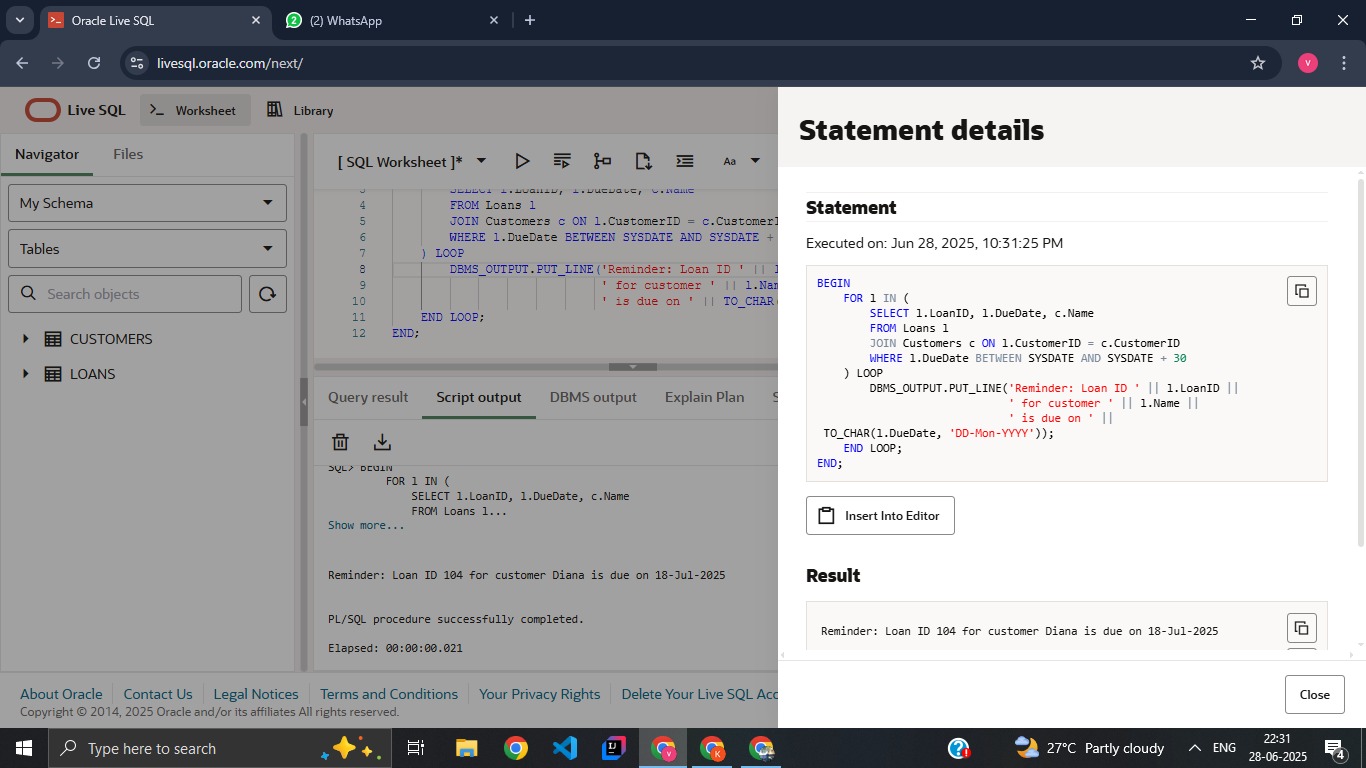
BEGIN  
 FOR cust\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP  
 IF cust\_rec.Balance > 10000 THEN  
 UPDATE Customers  
 SET IsVIP = 'TRUE'  
 WHERE CustomerID = cust\_rec.CustomerID;  
 END IF;  
 END LOOP;  
 COMMIT;  
END;  
/

### Scenario 3: Loan Due Reminders

BEGIN  
 FOR loan\_rec IN (  
 SELECT LoanID, CustomerID, DueDate   
 FROM Loans  
 WHERE DueDate <= SYSDATE + 30  
 ) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan\_rec.LoanID ||  
 ' for customer ' || loan\_rec.CustomerID ||  
 ' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));  
 END LOOP;  
END;  
/

**OUTPUT  
**

****

****

## EXERCISE 3: STORED PROCEDURES

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

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 FOR acc\_rec IN (SELECT AccountID, Balance FROM SavingsAccounts) LOOP  
 UPDATE SavingsAccounts  
 SET Balance = Balance + (Balance \* 0.01)  
 WHERE AccountID = acc\_rec.AccountID;  
 END LOOP;  
 COMMIT;  
END;  
/

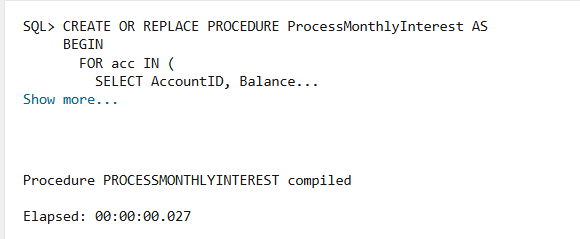
### Scenario 2: Update Employee Bonus

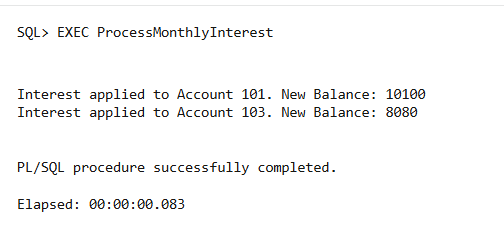
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(  
 dept\_id IN NUMBER,  
 bonus\_pct IN NUMBER  
) IS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* bonus\_pct / 100)  
 WHERE DepartmentID = dept\_id;  
 COMMIT;  
END;  
/

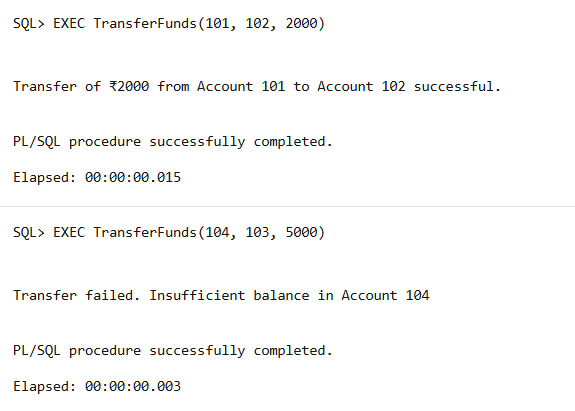
### Scenario 3: Transfer Funds Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds(  
 from\_account IN NUMBER,  
 to\_account IN NUMBER,  
 amount IN NUMBER  
) IS  
 insufficient\_balance EXCEPTION;  
 from\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = from\_account;  
  
 IF from\_balance < 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  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: insufficient balance.');  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed due to an unexpected error.');  
END;  
/

# Output

****

****

****

# Conclusion

The PL/SQL blocks and stored procedures developed in this document demonstrate the power of procedural SQL in automating complex banking operations. Such automation improves efficiency, accuracy, and overall customer satisfaction.