**Exercise 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.

BEGIN

  FOR rec IN(

        SELECT c.CustomerID,l.LoanID

        FROM Customers c

        JOIN LOANS l ON c.CustomerID=l.CustomerID

        where Months\_Between(SYSDATE,c.DOB)/12>60

    )LOOP

    UPDATE Loans

    SET InterestRate=InterestRate-1

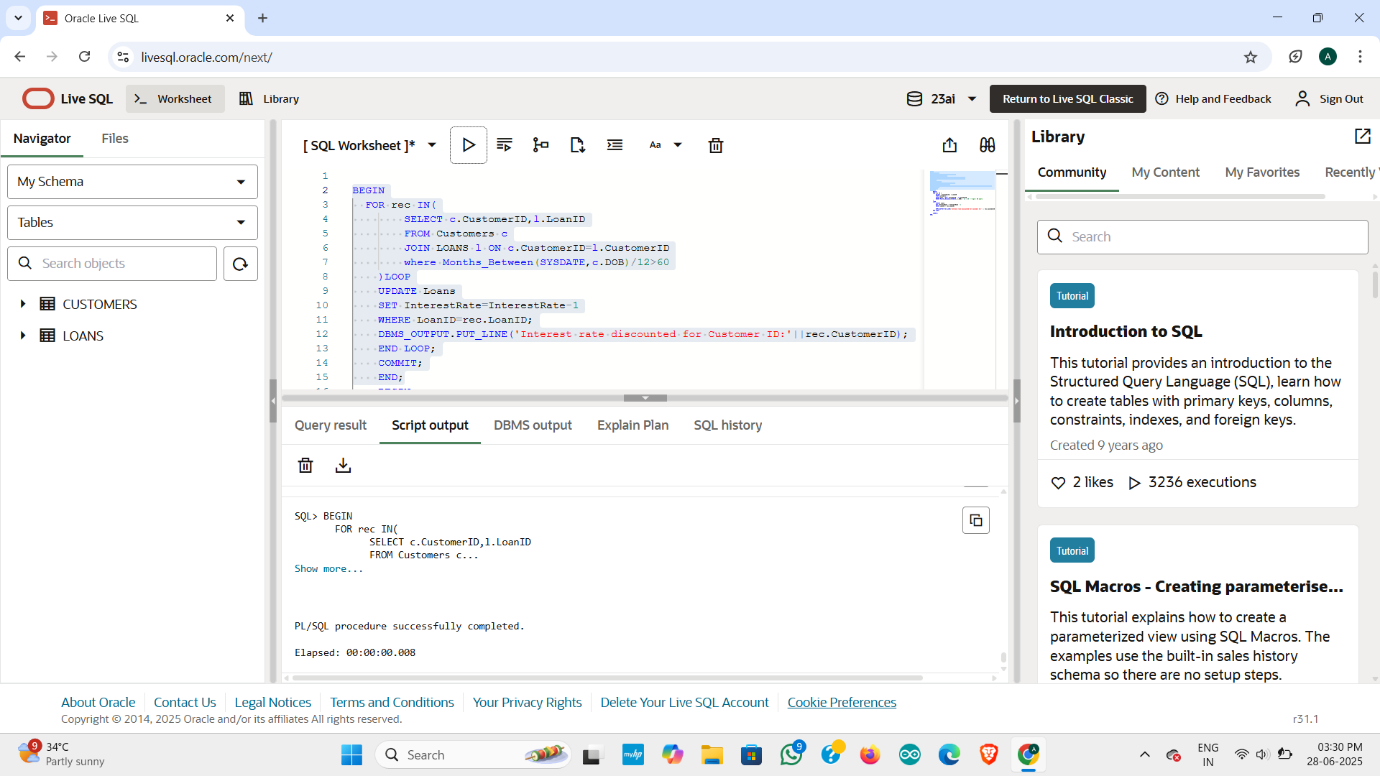
    WHERE LoanID=rec.LoanID;

    DBMS\_OUTPUT.PUT\_LINE('Interest rate discounted for Customer ID:'||rec.CustomerID);

    END LOOP;

    COMMIT;

    END;

**OUTPUT:**

**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.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

BEGIN

FOR rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

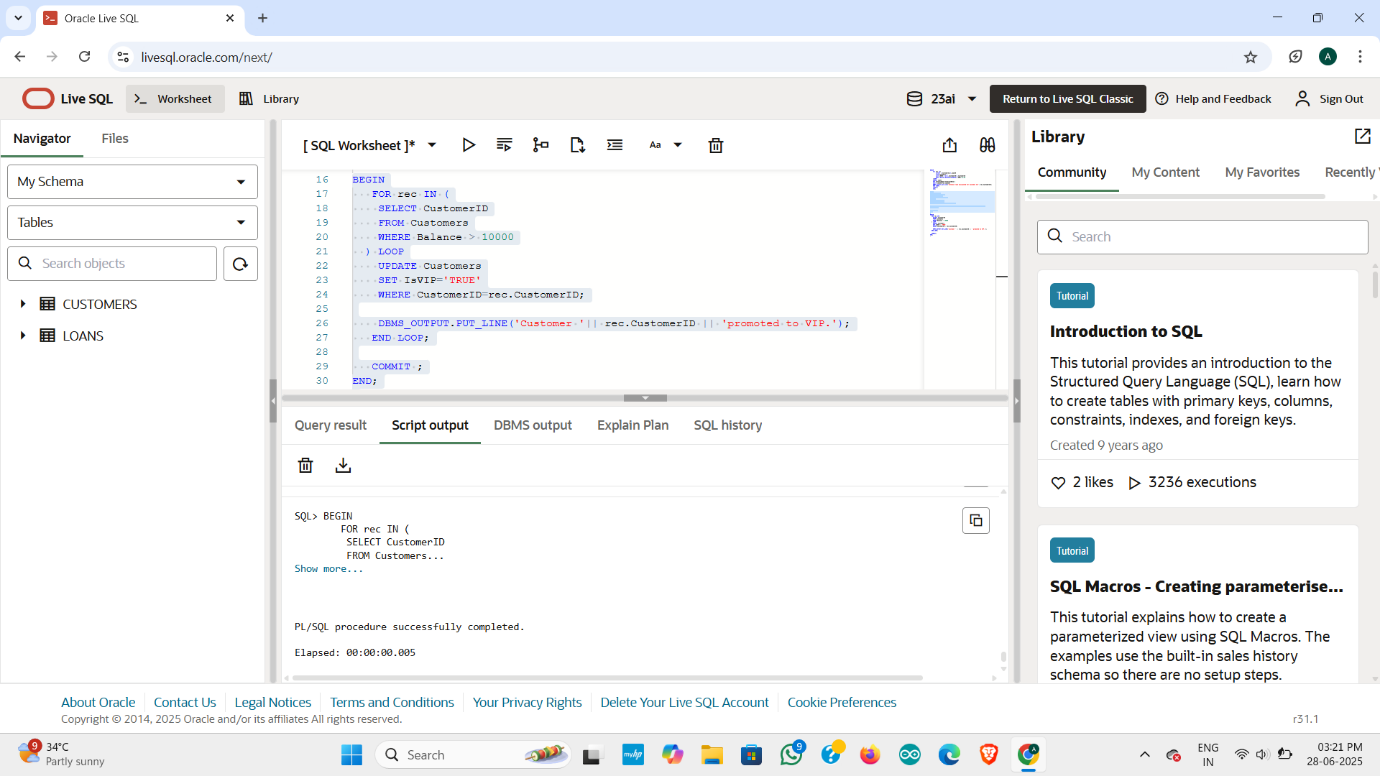
SET IsVIP='TRUE'

WHERE CustomerID=rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer '|| rec.CustomerID || 'promoted to VIP.');

END LOOP;

COMMIT ;

END

**OUTPUT:**

**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.

BEGIN

   FOR rec IN(

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

     FROM loans l

     JOIN customers c ON l.CustomerID=c.CustomerID

     WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

 )  LOOP

    DBMS\_OUTPUT.PUT\_LINE(

        'Remainder:Loan ID'||rec.LoanID||

        'for Customer"'||rec.Name||'"(ID:'||rec.CustomerID||

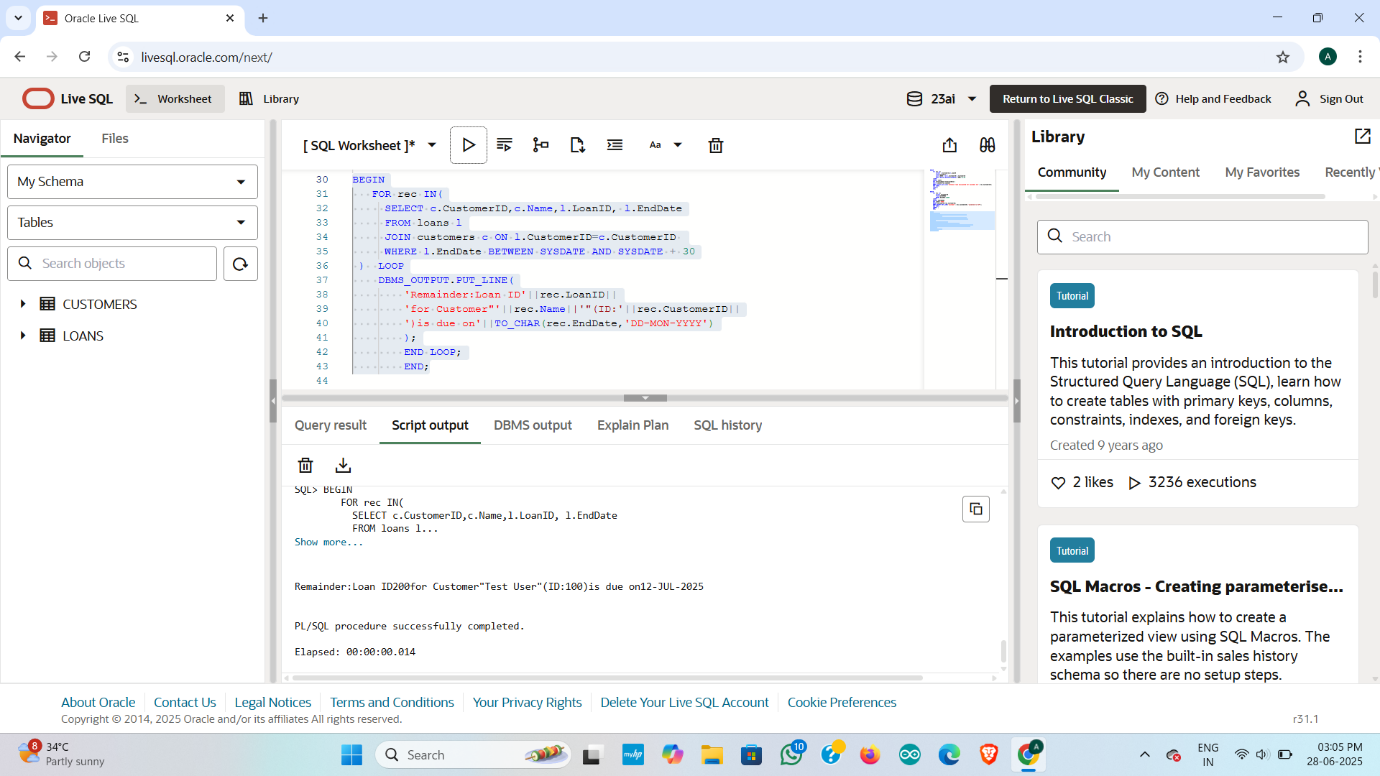
        ')is due on'||TO\_CHAR(rec.EndDate,'DD-MON-YYYY')

        );

        END LOOP;

        END;

**OUTPUT:**



**Exercise 3: 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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN(

SELECT AccountID,Balance

FROM Accounts

WHERE UPPER(AccountType)='SAVINGS'

)LOOP

UPDATE Accounts

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

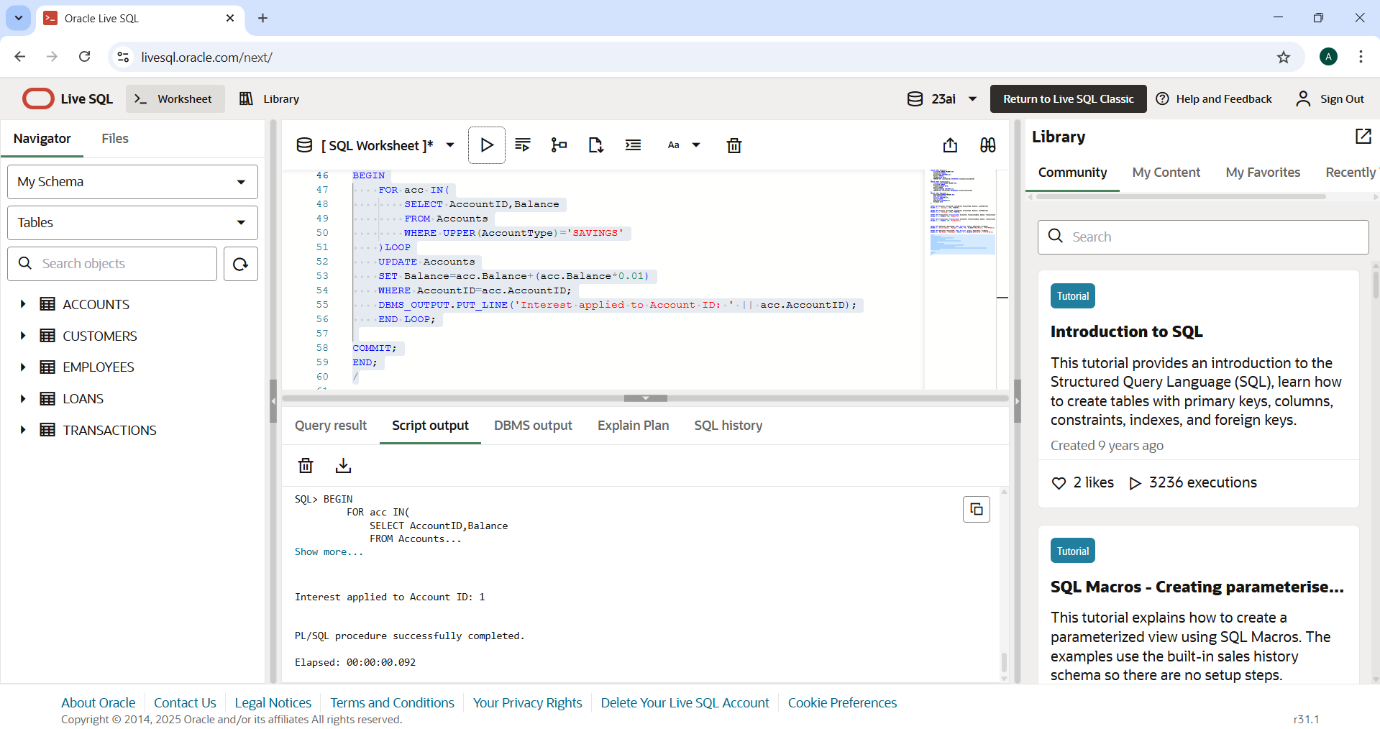
WHERE AccountID=acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

**OUTPUT:**

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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

)IS

BEGIN

    UPDATE Employees

SET Salary=Salary+(Salary\*p\_BonusPercent/100)

WHERE UPPER(Department)=UPPER(p\_Department);

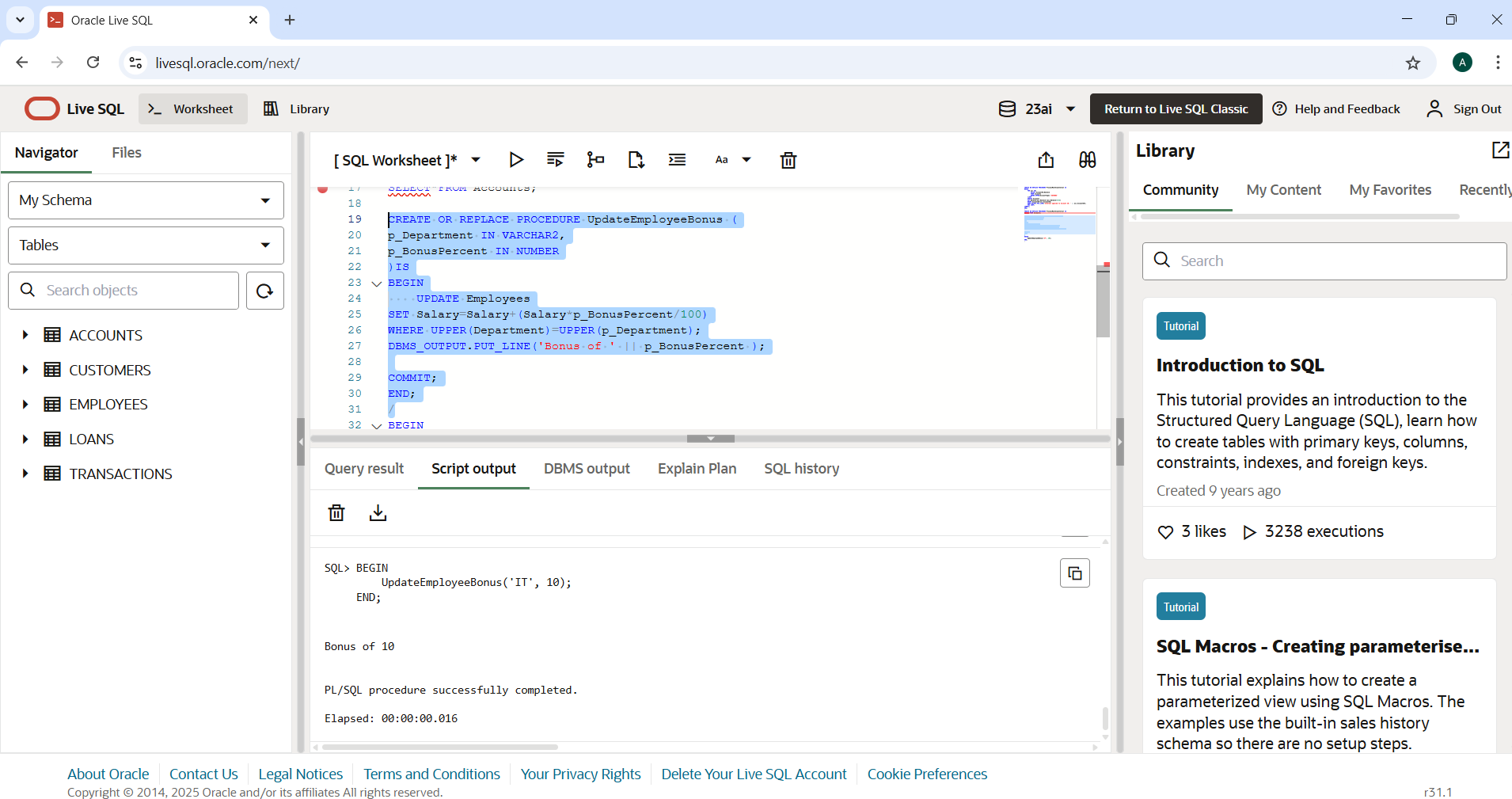
DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_BonusPercent );

COMMIT;

END;

/

OUTPUT:



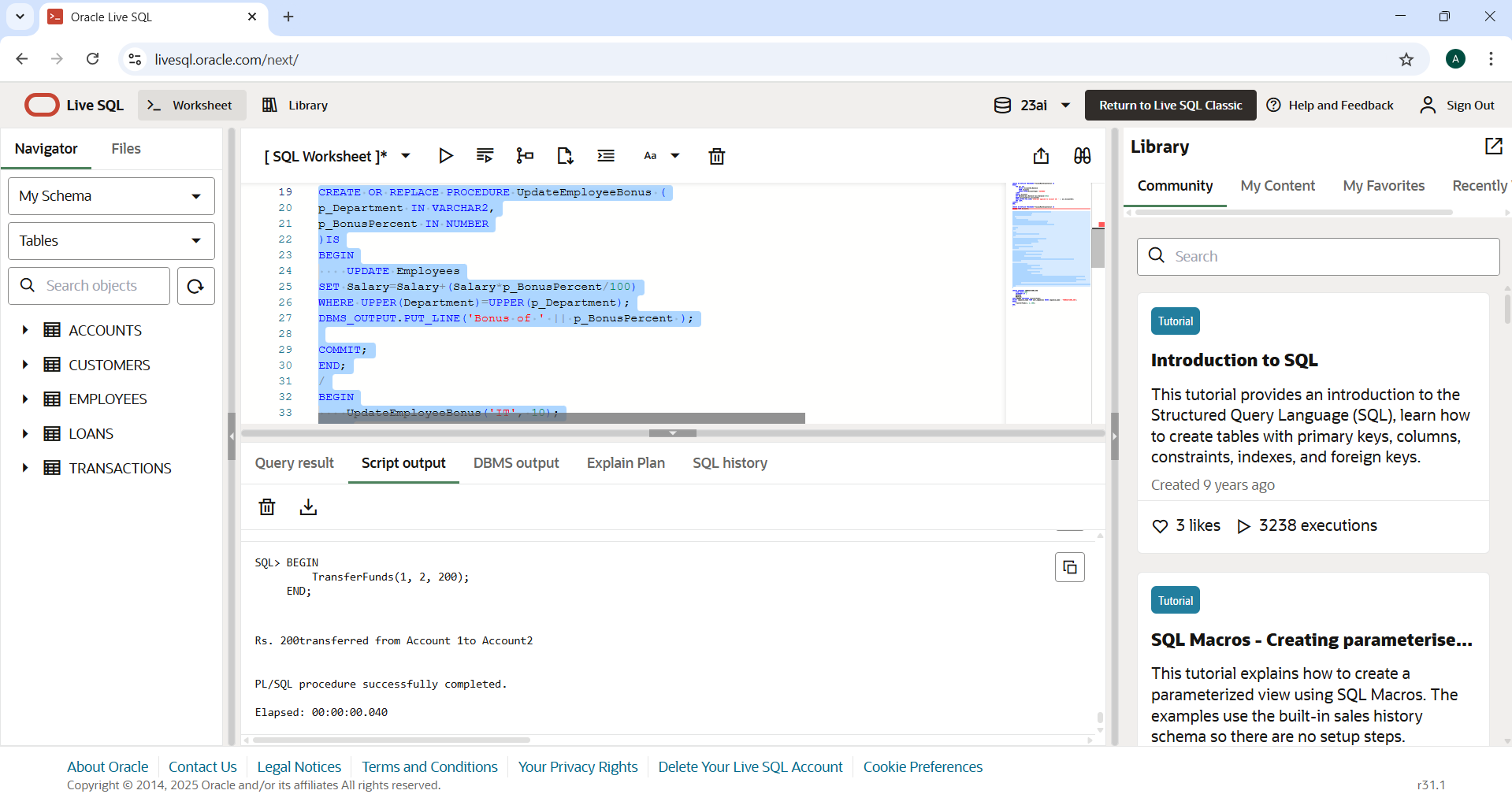
**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **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.

1. CREATE OR REPLACE PROCEDURE TransferFunds (
2. p\_SourceAccountID IN NUMBER,
3. p\_TargetAccountID IN NUMBER,
4. p\_Amount IN NUMBER
5. )IS
6. v\_SourceBalance NUMBER;
7. BEGIN
8. SELECT Balance INTO v\_SourceBalance
9. FROM Accounts
10. WHERE AccountID=p\_SourceAccountID
11. FOR UPDATE;
12. IF v\_SourceBalance<p\_Amount THEN
13. RAISE\_APPLICATION\_ERROR(-20001,'Insufficient balance in source account.');
14. END IF;
15. UPDATE Accounts
16. SET Balance=Balance - p\_Amount,
17. LastModified=SYSDATE
18. WHERE AccountID=p\_SourceAccountID;
19. UPDATE Accounts
20. SET Balance=Balance + p\_Amount,
21. LastModified=SYSDATE
22. WHERE AccountID=p\_TargetAccountID;
23. INSERT INTO Transactions(TransactionID,AccountID,TransactionDate,Amount,TransactionType)
24. VALUES(TRANSACTIONS\_SEQ.NEXTVAL,p\_SourceAccountID,SYSDATE,p\_Amount,'Debit');
25. INSERT INTO Transactions(TransactionID,AccountID,TransactionDate,Amount,TransactionType)
26. VALUES(TRANSACTIONS\_SEQ.NEXTVAL,p\_TargetAccountID,SYSDATE,p\_Amount,'Credit');
27. COMMIT;
28. DBMS\_OUTPUT.PUT\_LINE('Rs. ' || p\_Amount || 'transferred from Account ' || p\_SourceAccountID || 'to Account' || p\_TargetAccountID);
29. END;

/

1. CREATE SEQUENCE TRANSACTIONS\_SEQ
2. START WITH 1001
3. INCREMENT BY 1
4. NOCACHE
5. NOCYCLE;
6. SHOW ERRORS PROCEDURE TransferFunds;
7. SELECT sequence\_name FROM user\_sequences WHERE sequence\_name = 'TRANSACTIONS\_SEQ';
8. BEGIN
9. TransferFunds(1, 2, 200);
10. END;
11. /

**OUTPUT:** ****