**SupersetID-6406181**

**PL/SQL Programming**

**Schema to be Created**

*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'));*

**Exercise 1: Control Structures**

**Scenario 1:**

**Code**:

BEGIN

for cust IN (SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP

IF months\_between(sysdate, cust.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

END IF;

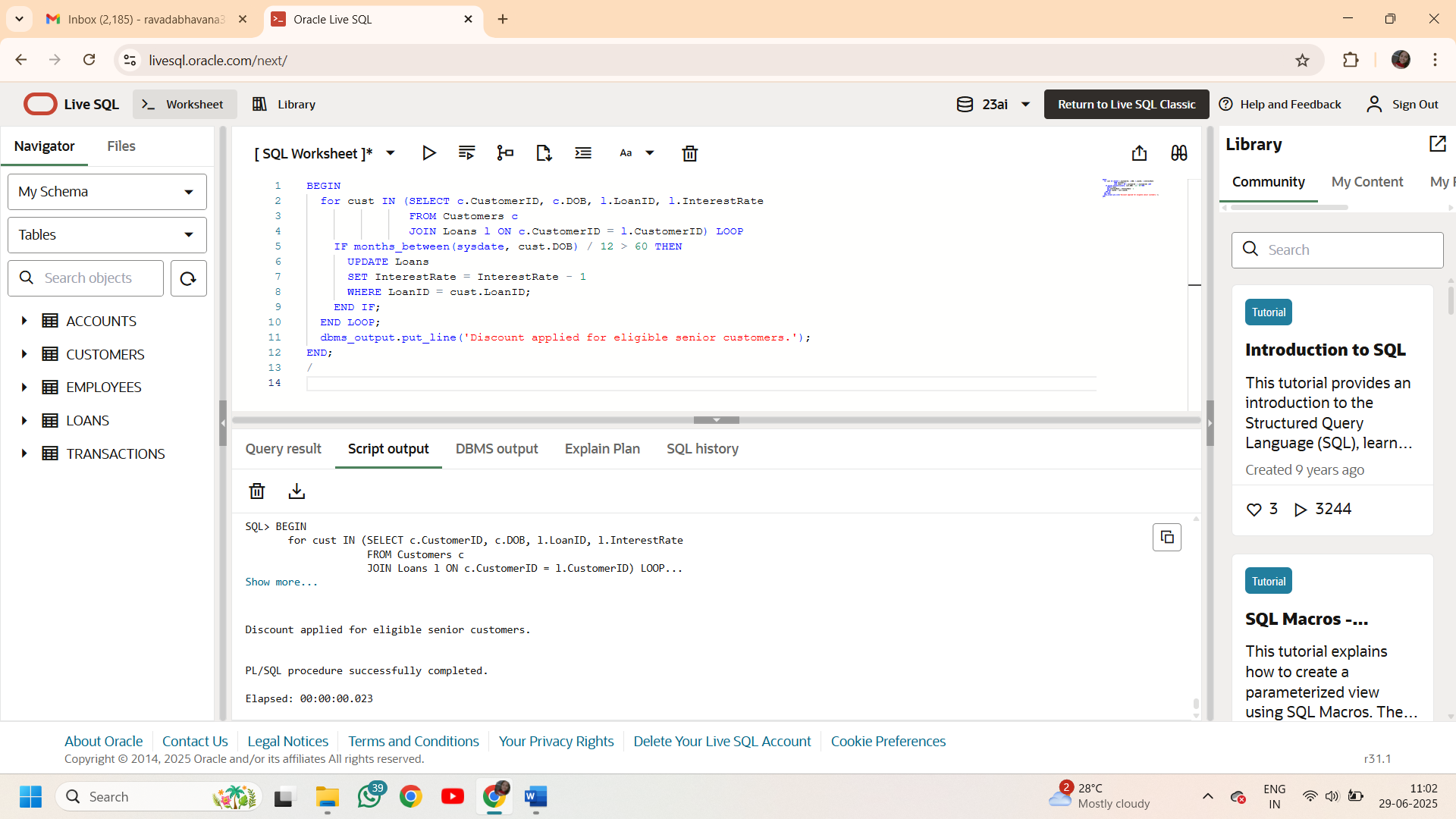
END LOOP;

dbms\_output.put\_line('Discount applied for eligible senior customers.');

END;

/

**Output:**



**Scenario 2:**

**Code:**

DECLARE

IsVIP boolean;

BEGIN

for cust in (select CustomerID,Name,Balance from Customers) loop

if cust.Balance>10000 then

IsVIP:=true;

else

IsVIP:=false;

end if;

if IsVIP then

dbms\_output.put\_line('customer '||cust.Name||' with ID'|| cust.CustomerID||'is a vip');

else

dbms\_output.put\_line('customer '||cust.Name||' with ID'|| cust.CustomerID||' is not a vip');

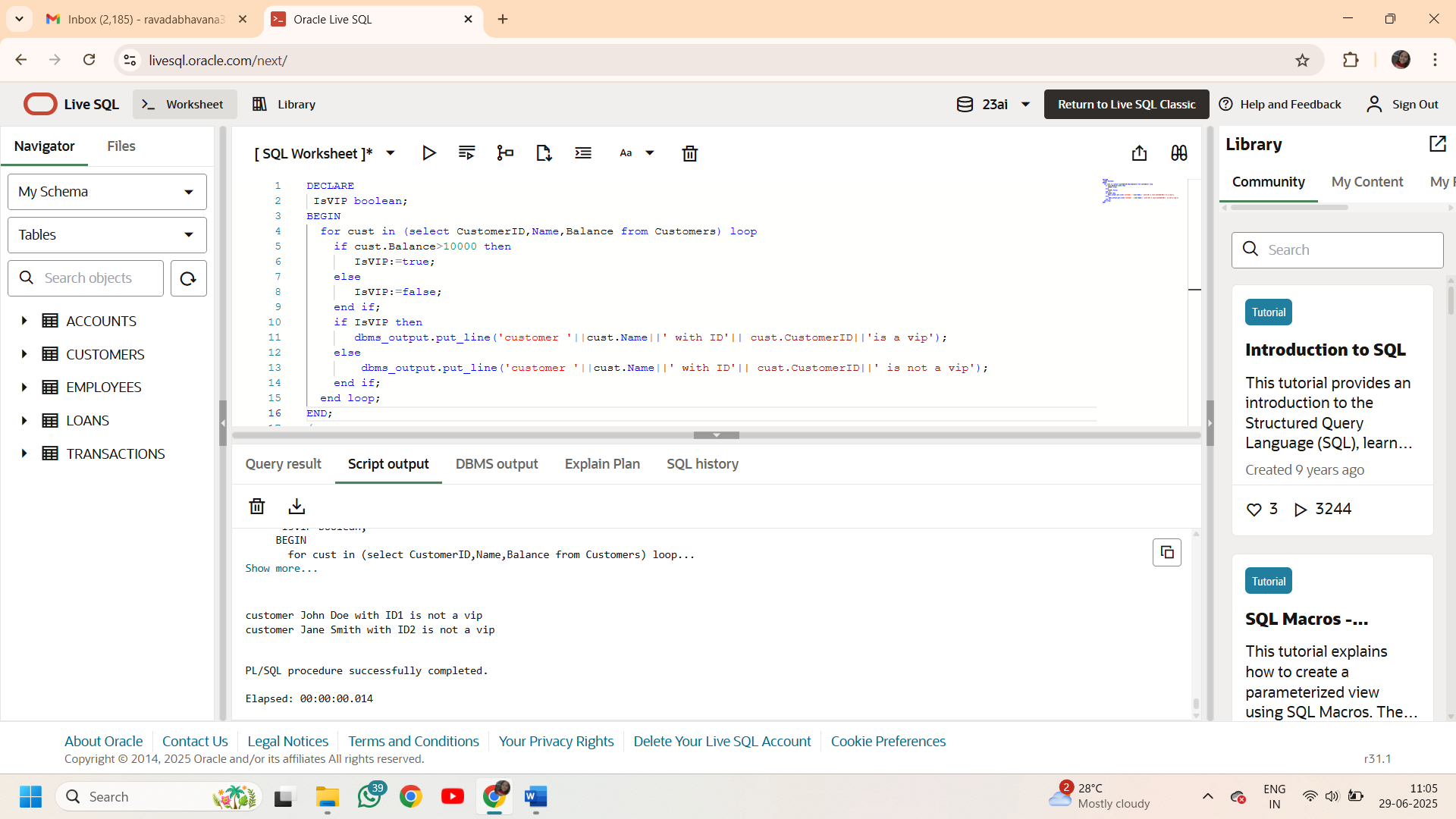
end if;

end loop;

END;

/

**Output:**



**Scenario 3:**

**Code:**

BEGIN

for loan in(select l.LoanID,l.CustomerID as loanCustID,l.EndDate,c.Name,c.CustomerID as custID

from Loans l join Customers c on

l.CustomerID=c.CustomerID

where l.EndDate between ADD\_MONTHS(SYSDATE, 60) and ADD\_MONTHS(SYSDATE,61))

loop

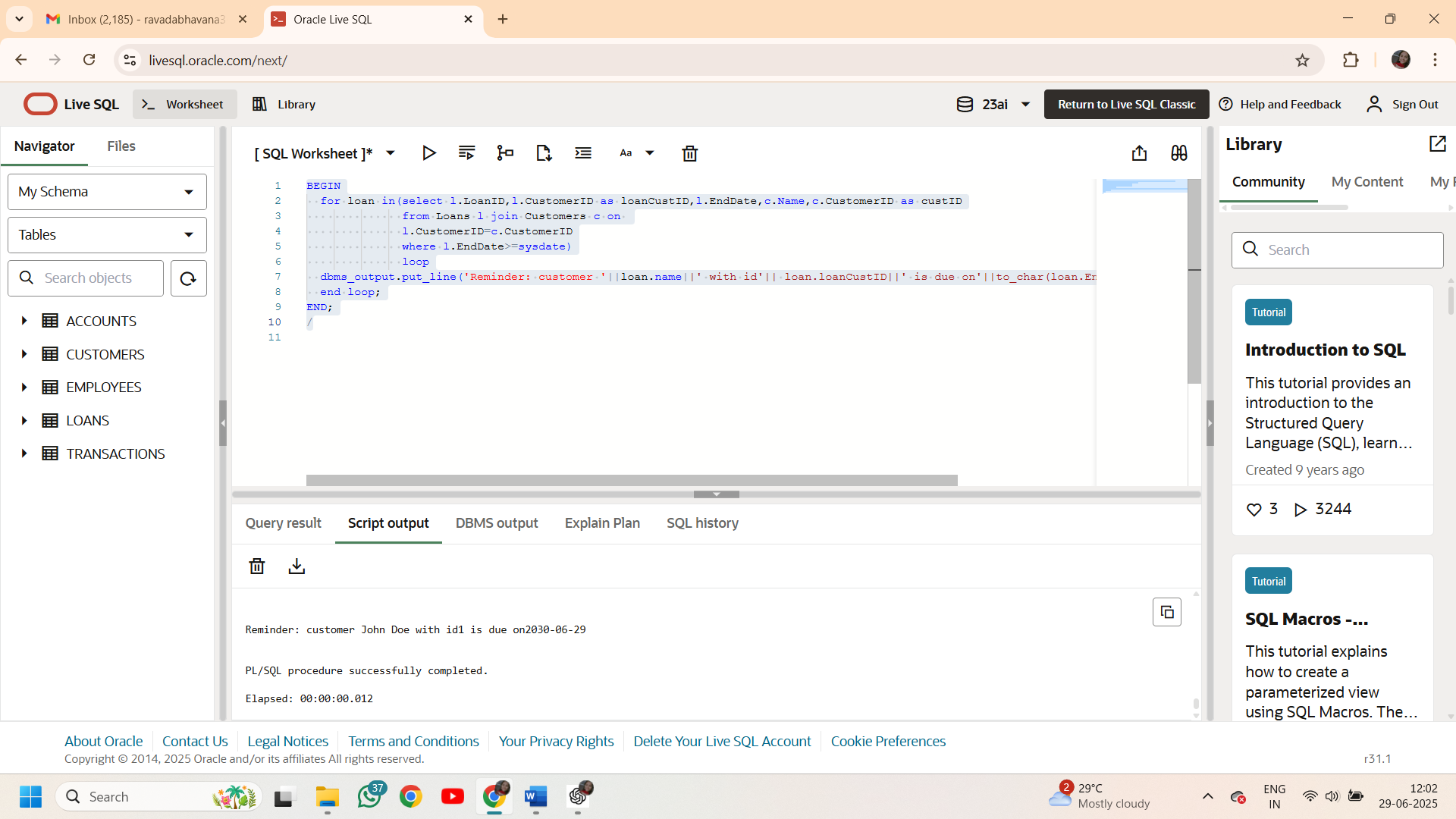
dbms\_output.put\_line('Reminder: customer '||loan.name||' with id'|| loan.loanCustID||' is due on'||to\_char(loan.EndDate,'YYYY-MM-DD'));

end loop;

END;

/

**Output:**



**Exercise 2: Error Handling**

**Scenario 1:**

**Code:**

create or replace procedure SafeTransferFunds(

    p\_fromAccountId number,

    p\_toAccountId number,

    amount number

)

AS

 source\_balance number;

begin

select Balance into source\_balance

from Accounts

where AccountID=p\_fromAccountId;

if source\_balance<amount then

raise\_application\_error(-20001,'insuffiecient funds');

end if;

update Accounts

set Balance=Balance-amount

where AccountID=p\_fromAccountId;

update Accounts

set Balance=Balance+amount

where AccountID=p\_toAccountId;

commit;

exception

when others then

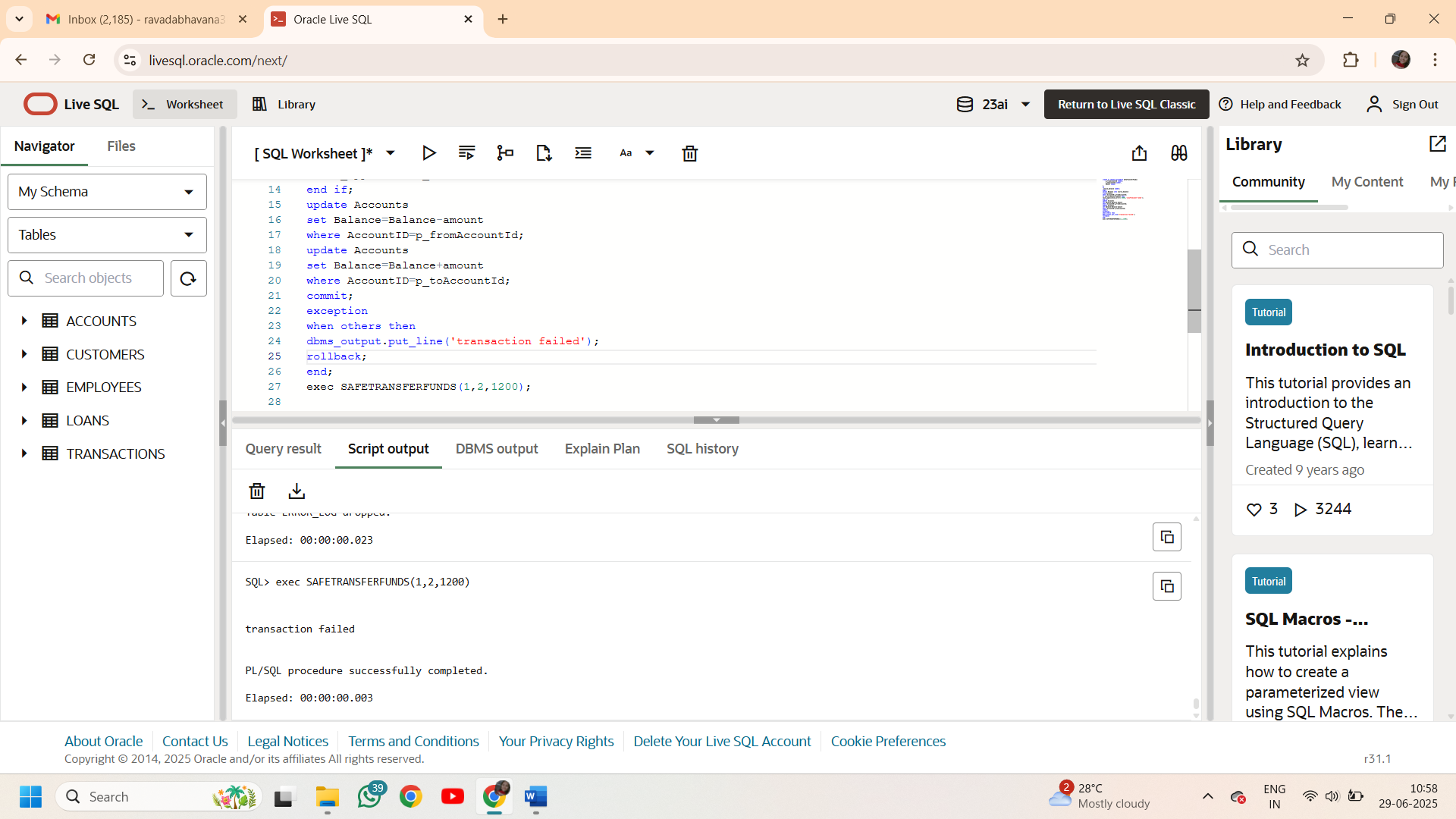
dbms\_output.put\_line('transaction failed');

rollback;

end;

exec SAFETRANSFERFUNDS(1,2,1200);

**Output:**



**Scenario 2:**

**Code:**

create or replace PROCEDURE UpdateSalary(

    p\_employeeID number,

    p\_percent number

)

as

v\_old\_salary number;

BEGIN

select Salary into v\_old\_salary

from Employees

where EmployeeID=p\_employeeID;

update EMPLOYEES

set Salary=Salary+(Salary\*p\_percent/100)

where EmployeeID=p\_employeeID;

dbms\_output.put\_line('salary updated successfully for employeeid '||p\_employeeID);

EXCEPTION

when NO\_DATA\_FOUND THEN

dbms\_output.put\_line('error: employee with '||p\_employeeID||' doesnt exist');

when others THEN

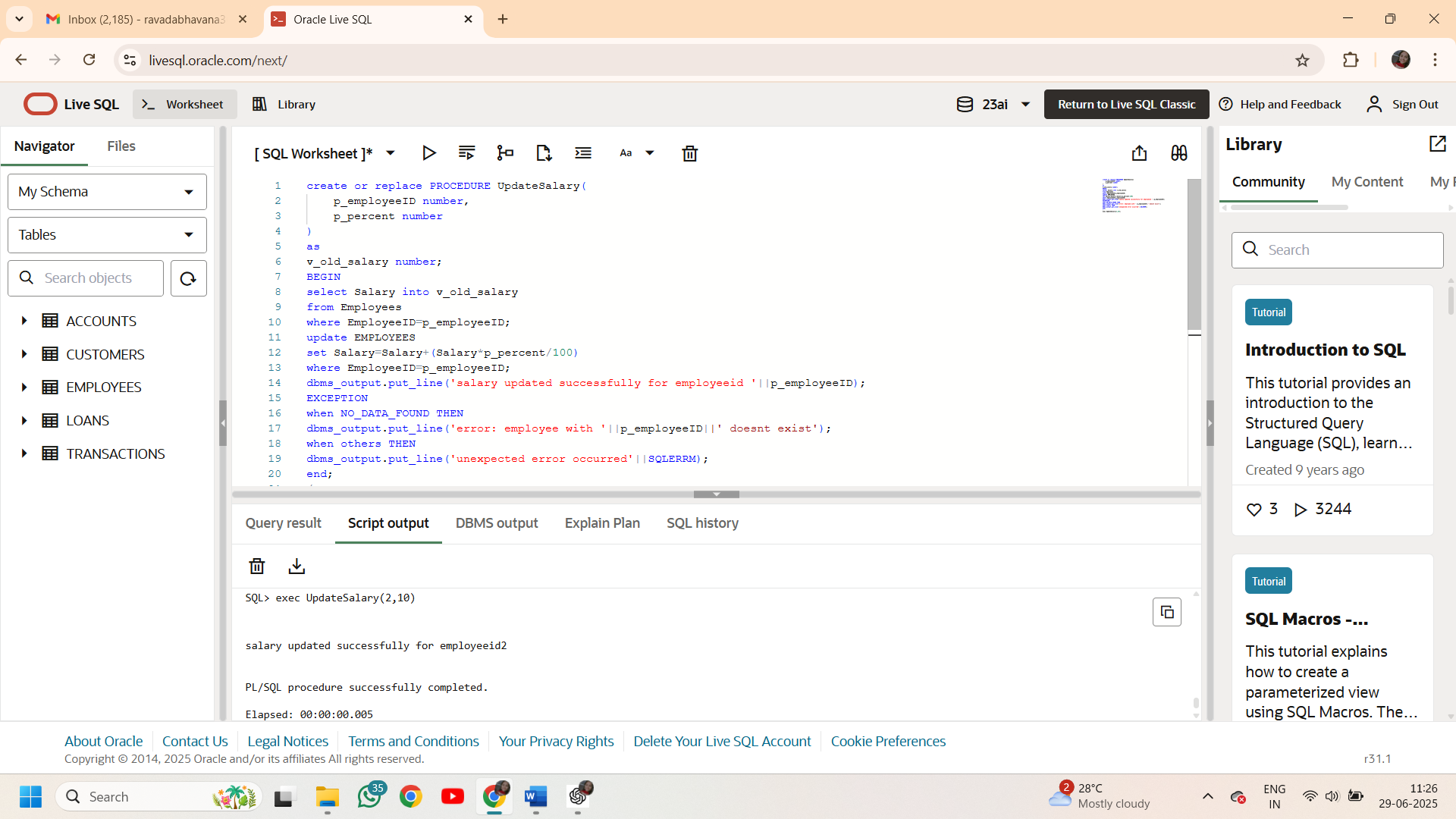
dbms\_output.put\_line('unexpected error occurred'||SQLERRM);

end;

/

exec UpdateSalary(2,10);

**Output:**



**Scenario 3:**

**Code:**

create or replace procedure AddNewCustomer(

    cust\_ID number,

    cust\_name Varchar2,

    cust\_dob date,

    cust\_bal number,

    last\_modified date

)

as

begin

    insert into Customers(CustomerID,Name,DOB,Balance,LastModified)

    values(cust\_ID,cust\_name,cust\_dob,cust\_bal,last\_modified);

    dbms\_output.put\_line('customer inserted successfully with id '||cust\_ID);

exception

when DUP\_VAL\_ON\_INDEX then

dbms\_output.put\_line('error: customer with id'||cust\_ID||'already exists');

when others THEN

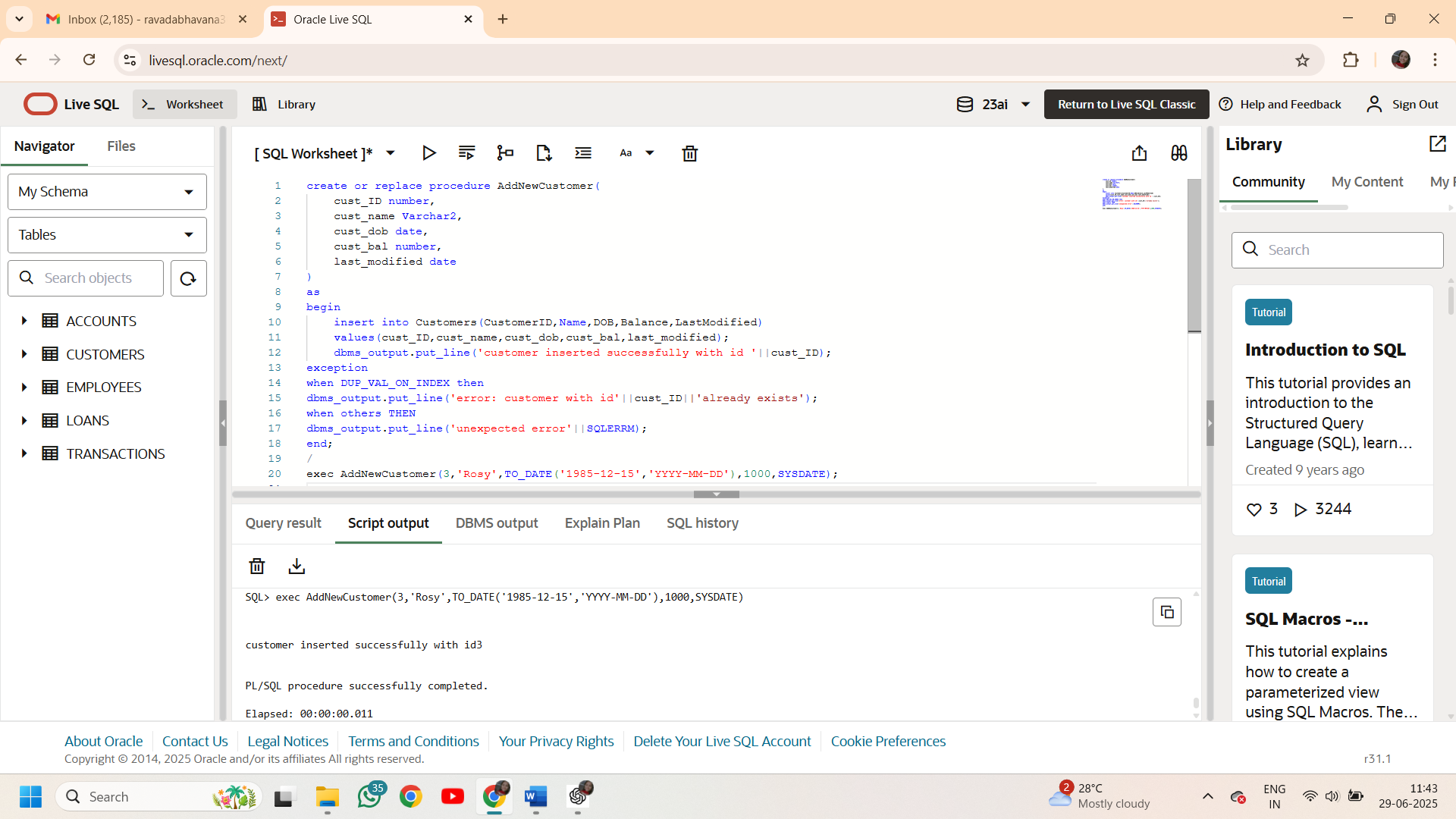
dbms\_output.put\_line('unexpected error'||SQLERRM);

end;

/

exec AddNewCustomer(3,'Rosy',TO\_DATE('1985-12-15','YYYY-MM-DD'),1000,SYSDATE);

**Output:**



**Exercise 3: Stored Procedures**

**Scenario 1:**

Code:

create or replace procedure ProcessMonthlyInterest as

account\_count NUMBER:=0;

BEGIN

update Accounts

set Balance=Balance+(Balance\*0.01),LastModified=SYSDATE

where AccountType='Savings';

account\_count:=SQL%rowcount;

dbms\_output.put\_line(account\_count||' accounts have been updated with 1% interest');

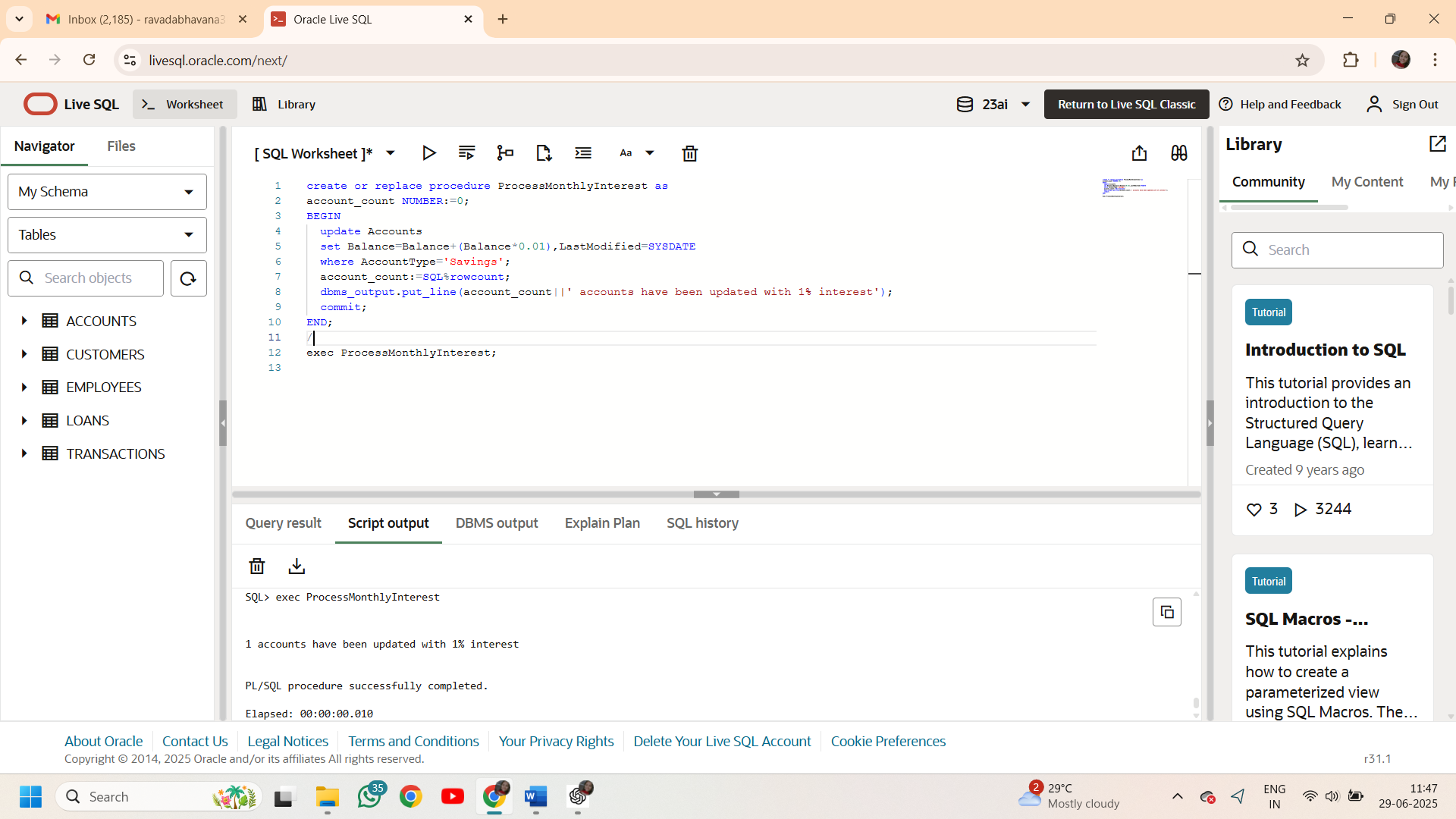
commit;

END;

/

exec ProcessMonthlyInterest;

Output:



**Scenario 2:**

**Code:**

create or replace procedure UpdateEmployeeBonus(

v\_department varchar2,

v\_bonuspercentage number)

as

v\_count number:=0;

BEGIN

update Employees

set salary=salary+(salary\*v\_bonuspercentage/100)

where Department=v\_department;

v\_count:=SQL%rowcount;

dbms\_output.put\_line(v\_count||' employees from '||v\_department||' department got a bonus of '||v\_bonuspercentage||'%');

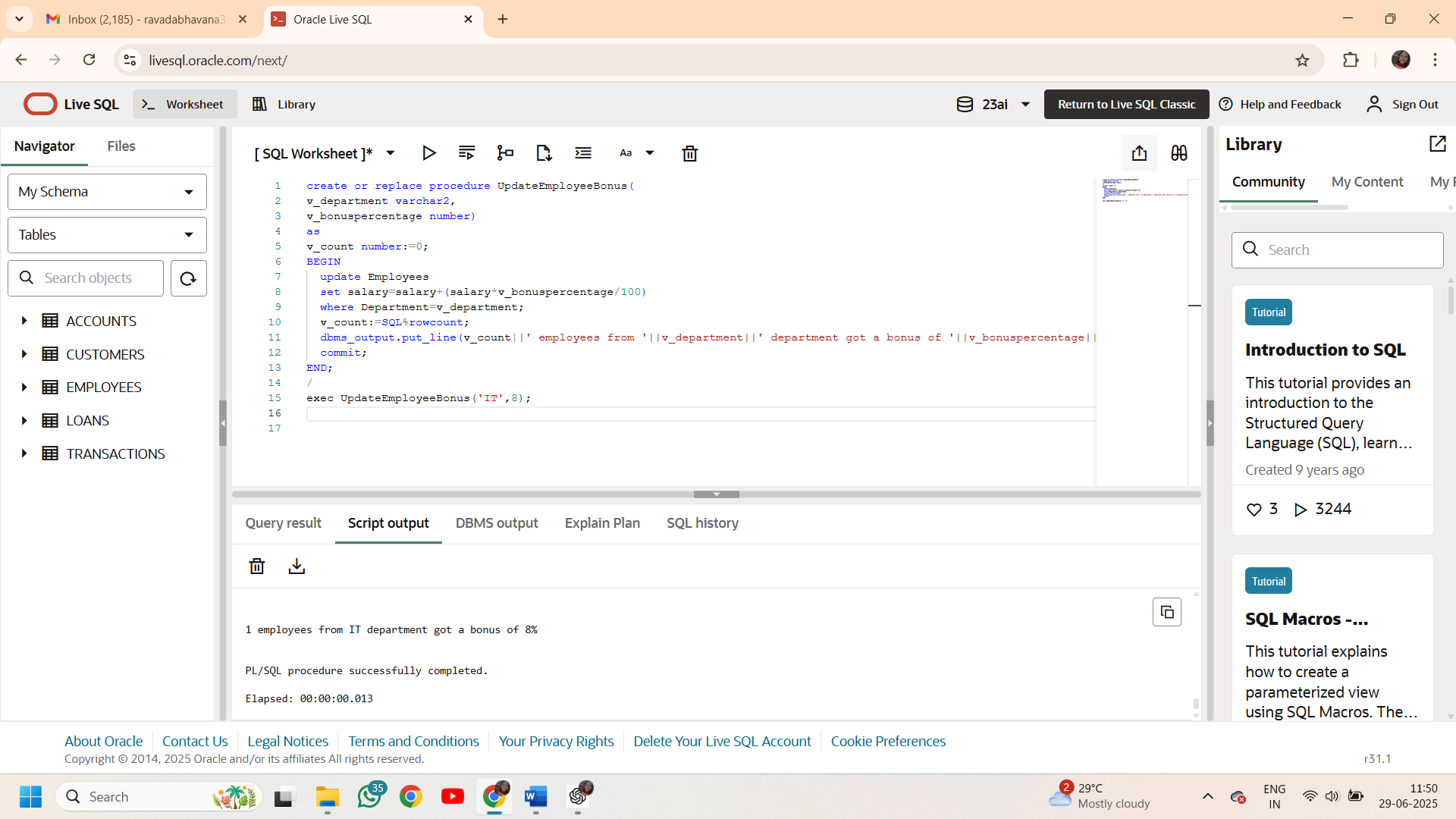
commit;

END;

/

exec UpdateEmployeeBonus('IT',8);

**Output:**



**Scenario 3:**

Code:

create sequence Transaction\_seq start with 1000 increment by 1;

create or replace procedure TransferFunds(

p\_FromAccountId number,

p\_ToAccountId number,

p\_amount number) as

p\_bal number;

BEGIN

if p\_FromAccountId=p\_ToAccountId then

raise\_application\_error(-20002,'self transfer is not possible');

end if;

select Balance into p\_bal from Accounts

where AccountID=p\_FromAccountId;

if p\_bal<p\_amount then

raise\_application\_error(-20001,'sorry! insufficient balance in the source account');

end if;

update Accounts

set Balance=Balance-p\_amount,

LastModified=SYSDATE

where AccountID=p\_FromAccountId;

update Accounts

set Balance=Balance+p\_amount,

LastModified=SYSDATE

where AccountID=p\_ToAccountId;

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

values (Transaction\_seq.NEXTVAL,p\_FromAccountId,SYSDATE,p\_amount,'Withdrawal');

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

values(Transaction\_seq.NEXTVAL,p\_ToAccountId,SYSDATE,p\_amount,'Deposit');

dbms\_output.put\_line(p\_amount||' has been tranferred to account with id '|| p\_ToAccountId);

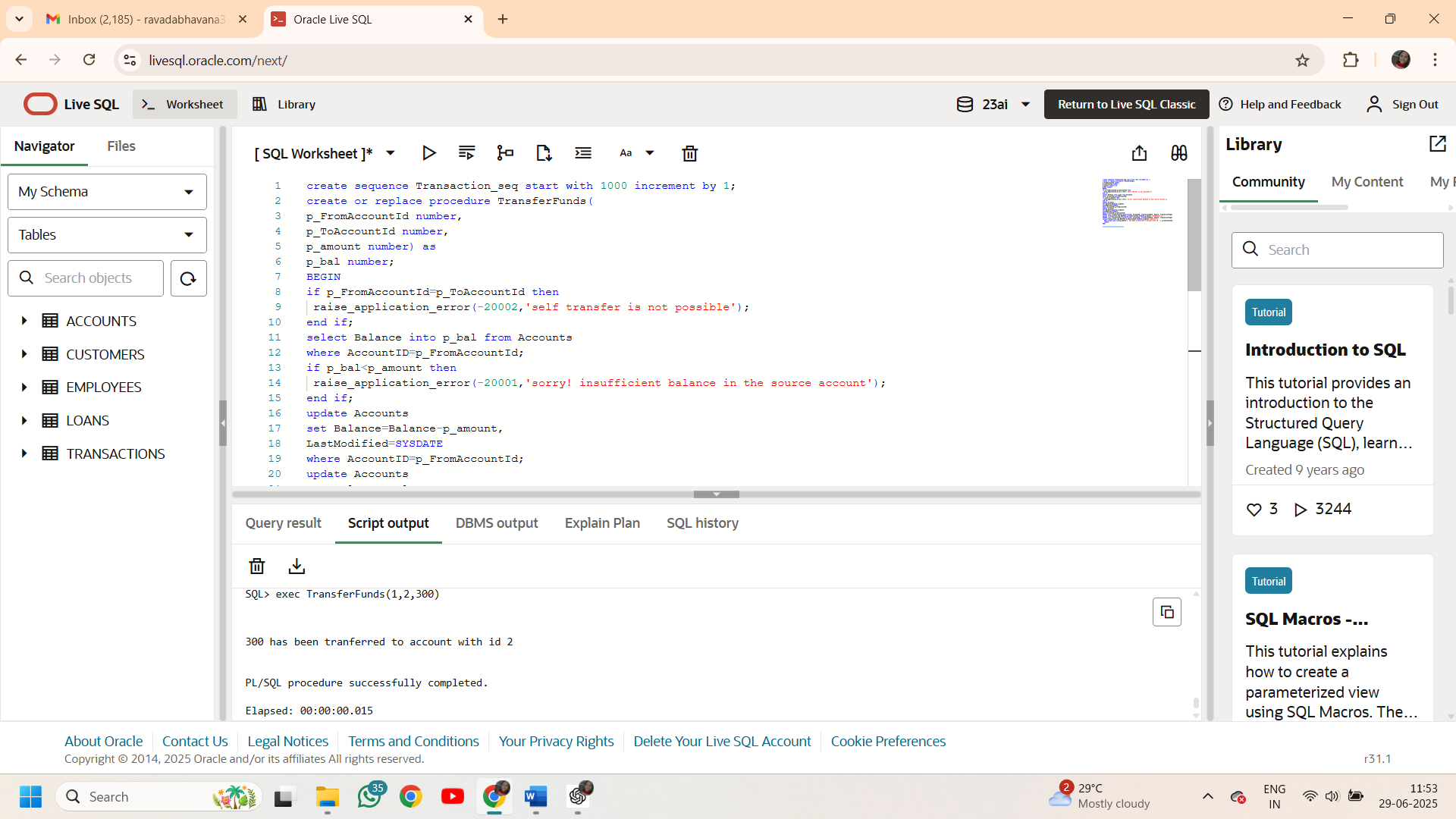
commit;

END;

/

exec TransferFunds(1,2,300);

Output:



**Exercise 4: Functions**

**Scenario 1:**

**Code:**

create or replace function CalculateAge(

    f\_dob date

)return number

is

v\_age number;

begin

    v\_age:=floor(months\_between(sysdate,f\_dob)/12);

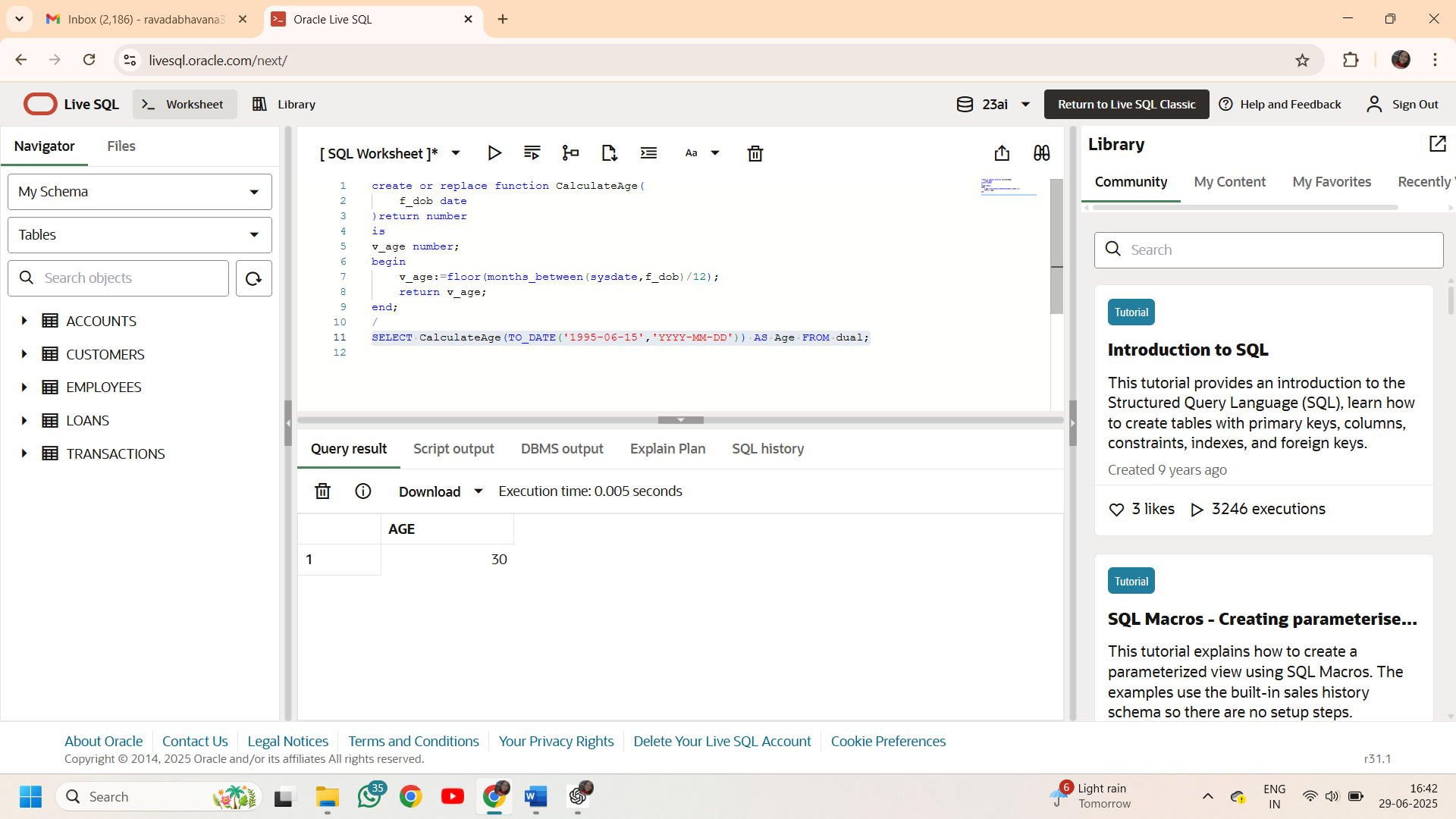
    return v\_age;

end;

/

SELECT CalculateAge(TO\_DATE('1995-06-15','YYYY-MM-DD')) AS Age FROM dual;

**Output:**



**Scenario 2:**

**Code:**

create or replace function CalculateMonthlyInstallment(

    loan\_amount number,

    interest\_rate number,

    loan\_duration number

)return NUMBER

is

v\_emi number;

v\_monthly\_rate number:=interest\_rate/(12\*100);

v\_total\_months number:=loan\_duration\*12;

BEGIN

    if v\_monthly\_rate=0 then

    v\_emi:=loan\_amount/v\_total\_months;

    else

    v\_emi:=(loan\_amount\*v\_monthly\_rate\*power(1+v\_monthly\_rate,v\_total\_months))/(power(1+v\_monthly\_rate,v\_total\_months)-1);

    end if;

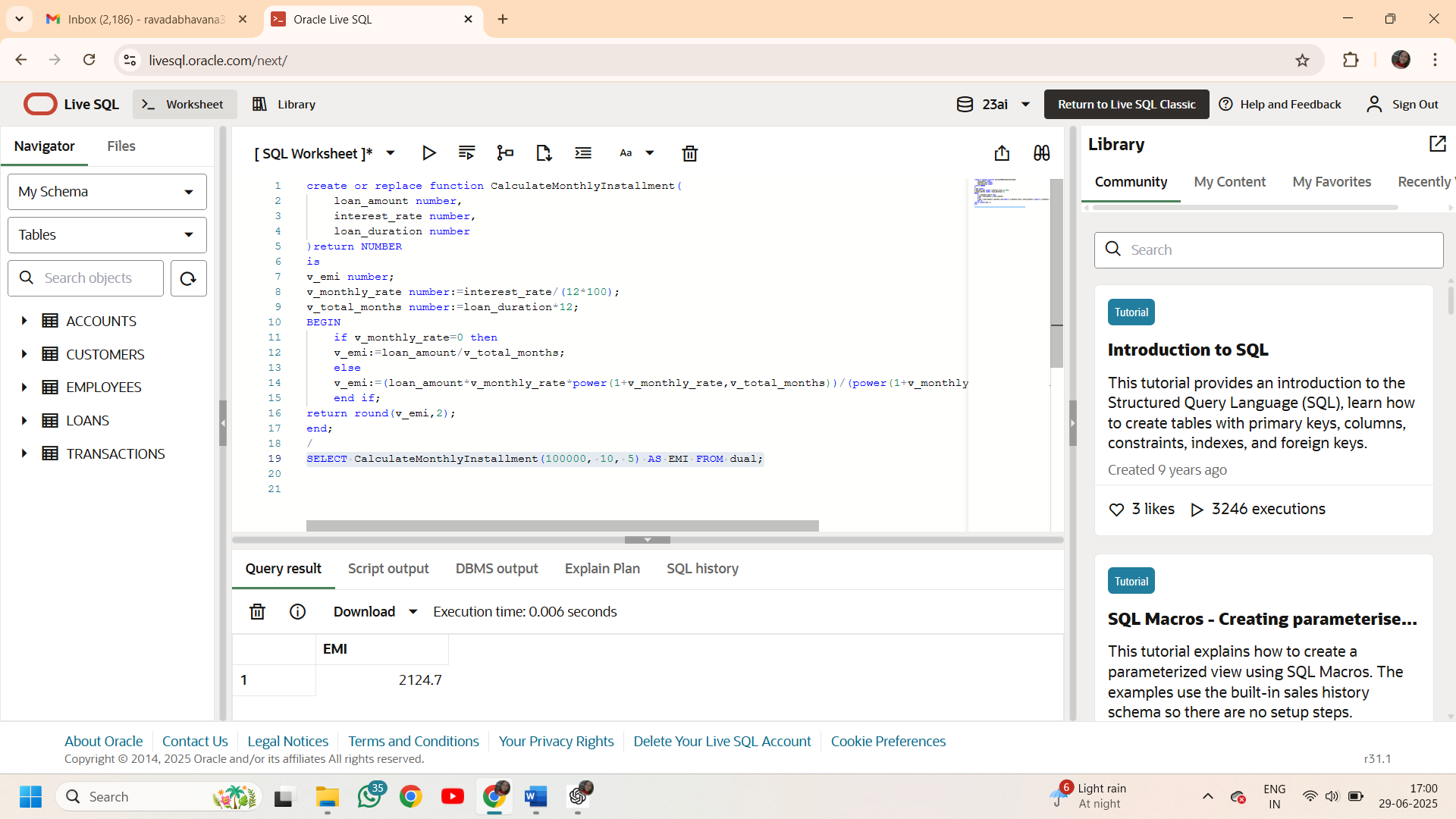
return round(v\_emi,2);

end;

/

SELECT CalculateMonthlyInstallment(100000, 10, 5) AS EMI FROM dual;

**Output:**



**Scenario 3:**

**Code:**

create or replace function HasSufficientBalance(

    f\_account\_id number,

    f\_amount number

)return boolean

is

v\_bal number;

begin

select Balance into v\_bal from Accounts

where AccountID=f\_account\_id;

return v\_bal>=f\_amount;

end;

/

declare

result boolean;

begin

    result:=HasSufficientBalance(1,200);

    if result THEN

    dbms\_output.put\_line('sufficient balance');

    end if;

EXCEPTION

when NO\_DATA\_FOUND then

 dbms\_output.put\_line('Insufficient balance');

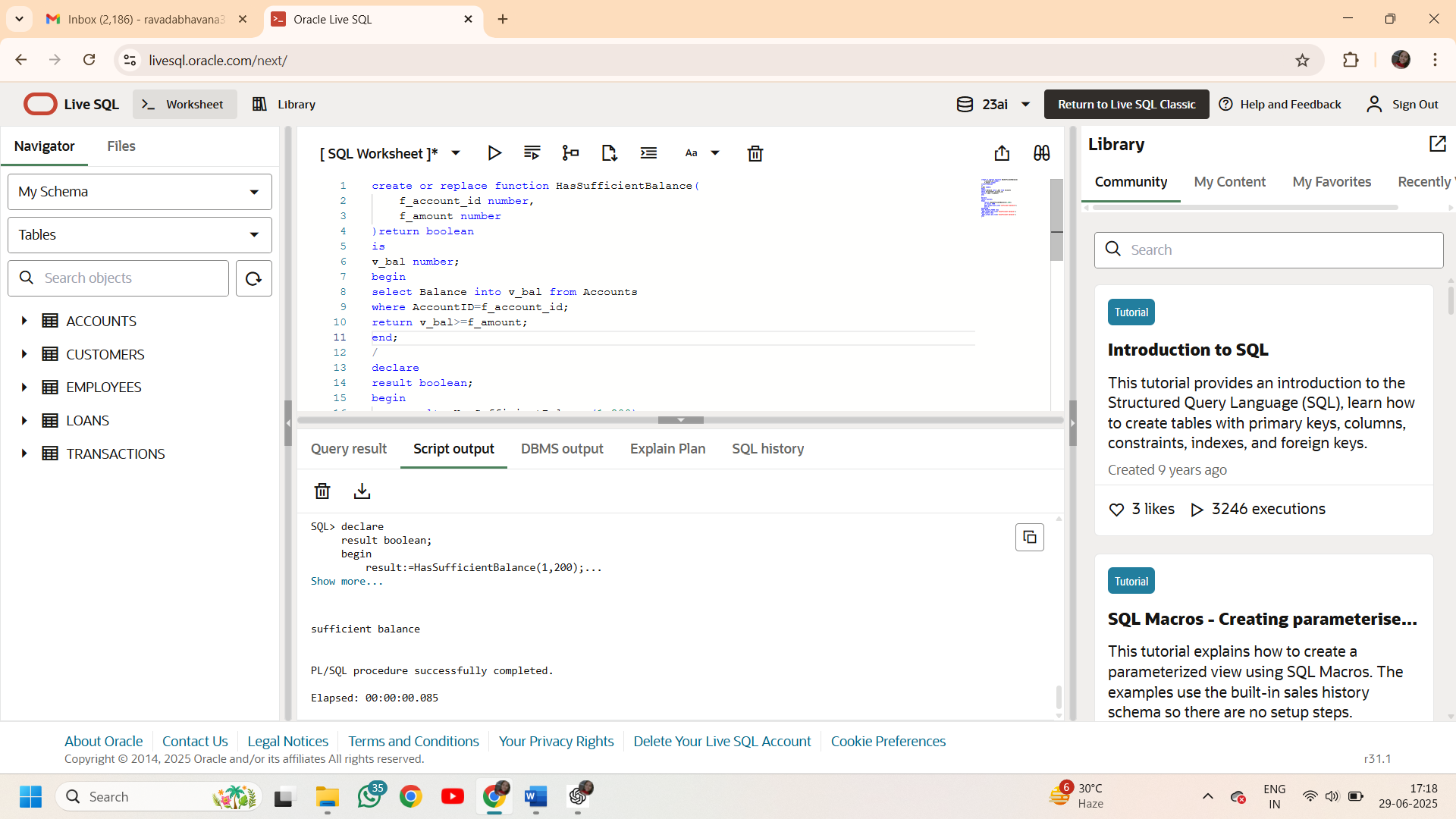
when others then

 dbms\_output.put\_line('Insufficient balance');

end;

/

**Output:**



**Exercise 5: Triggers**

**Scenario 1:**

**Code:**

create or replace trigger UpdateCustomerLastModified

before UPDATE on Customers

FOR EACH ROW

begin

    :NEW.LastModified := sysdate;

end;

/

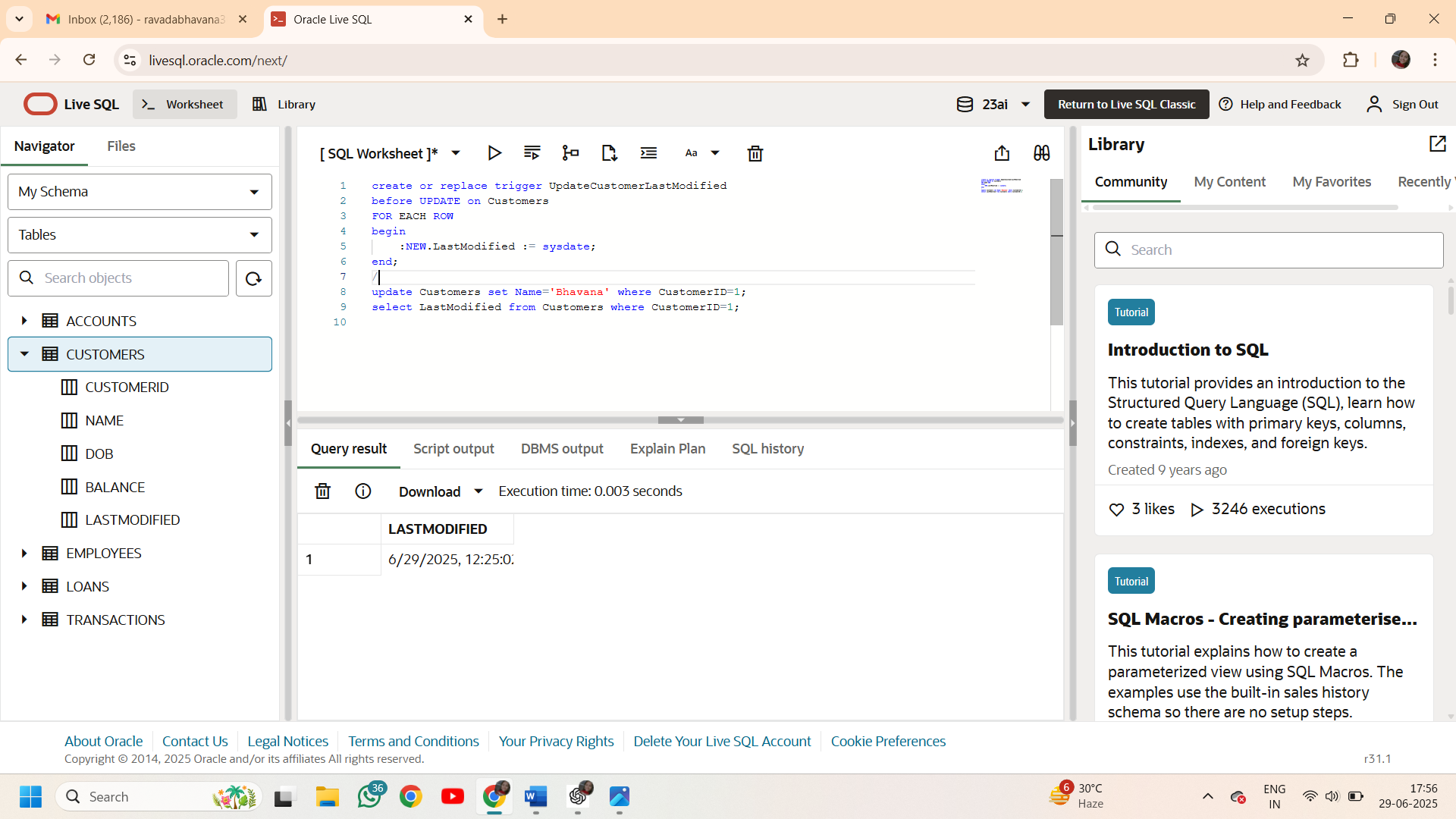
--to update

update Customers set Name='Bhavana' where CustomerID=1;

--to check

select LastModified from Customers where CustomerID=1;

**Output:**



**Scenario 2:**

**Code:**

create table AuditLog (

    AuditID number GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

    TransactionID NUMBER,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    LoggedAt DATE DEFAULT SYSDATE

);

create or replace trigger LogTransaction

after insert on Transactions

for each ROW

BEGIN

    insert into AuditLog(TransactionID,AccountID,TransactionDate,Amount,TransactionType)

    values(:NEW.TransactionID,:NEW.AccountID,:NEW.TransactionDate,:NEW.Amount,:NEW.TransactionTypE);

end;

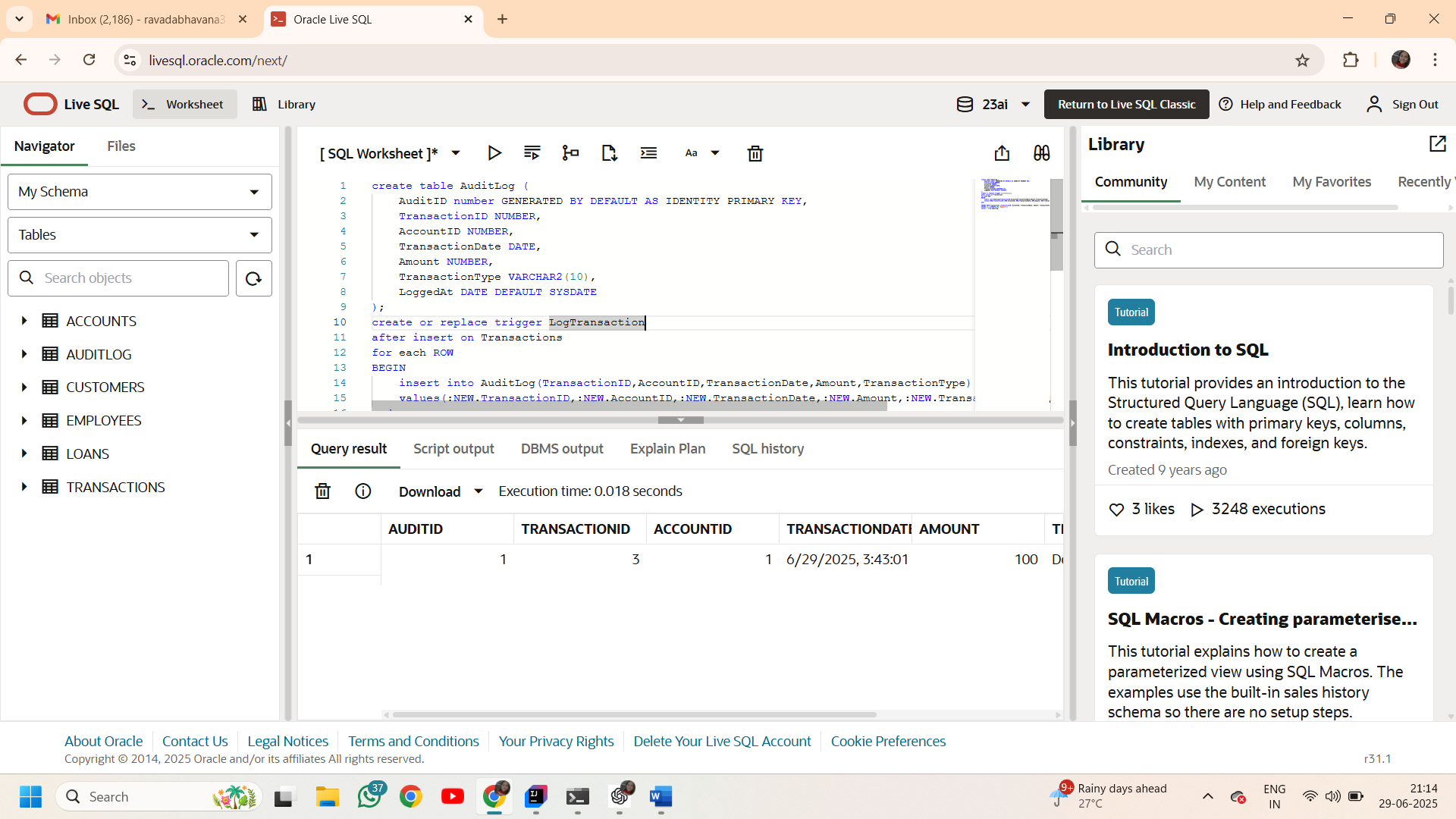
/

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

VALUES (3,1,SYSDATE,100,'Deposit');

select \* from Auditlog;

**Output:**



**Scenario 3:**

**Code:**

create or replace trigger CheckTransactionRules

before insert ON Transactions

FOR EACH ROW

declare

    v\_balance NUMBER;

begin

    select Balance into v\_balance

    from Accounts

    where AccountID = :NEW.AccountID;

    if :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

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

    END IF;

    IF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

    END IF;

end;

/

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

VALUES (4,1,SYSDATE,999999,'Withdrawal');

**Output:**

