**[PL SQL programming](https://github.com/seshadrimr/Digital-Nurture-4.0-JavaFSE/tree/main/Java%20FSE/Deepskilling/PL%20SQL%20programming)**

**Setting Up Schema :**

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

);

**Inserting Values in the SCHEMA**

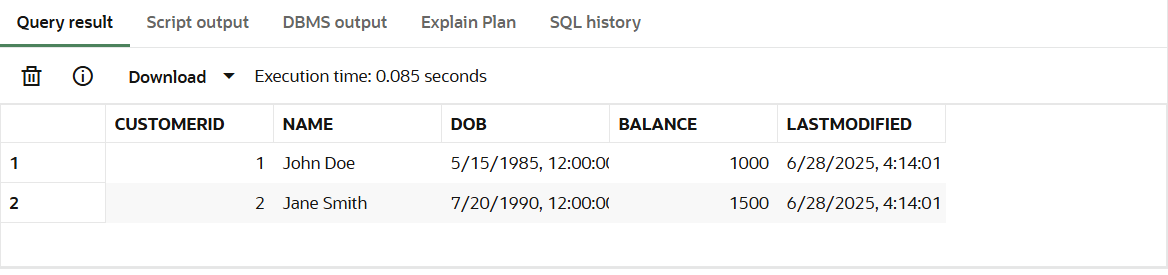
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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);

**Output**



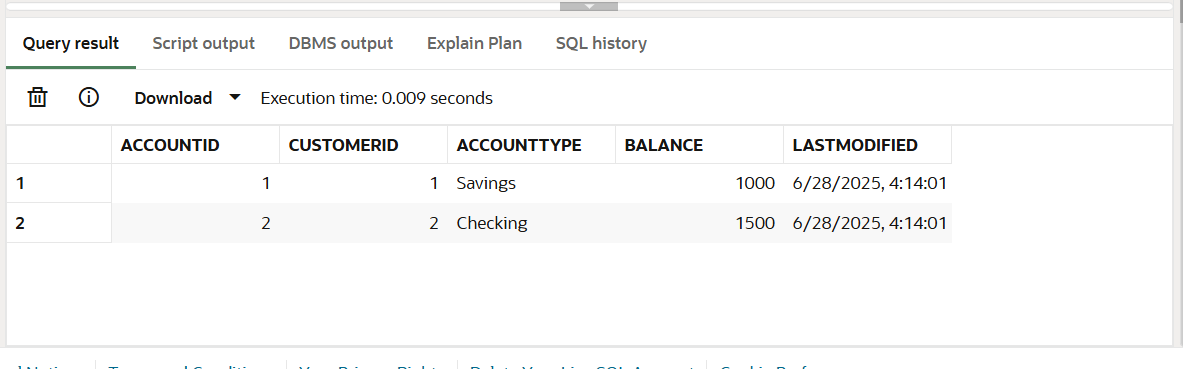
INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

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

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

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

**Output**



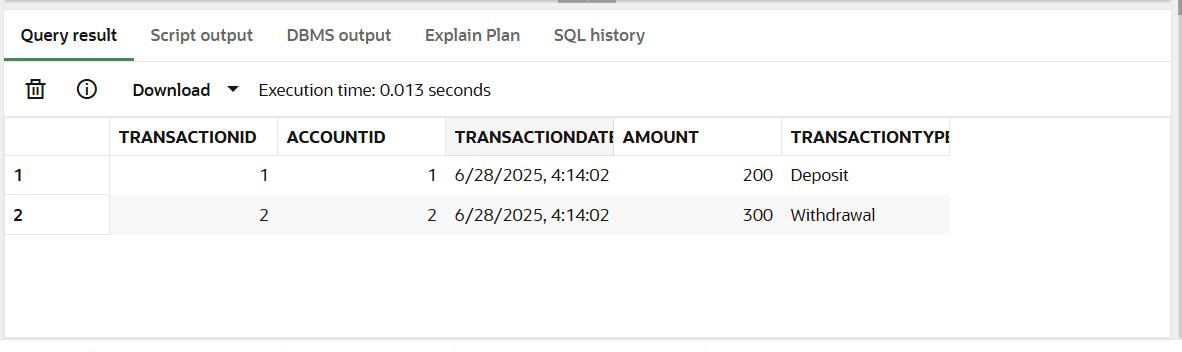
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

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

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

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

**Output**



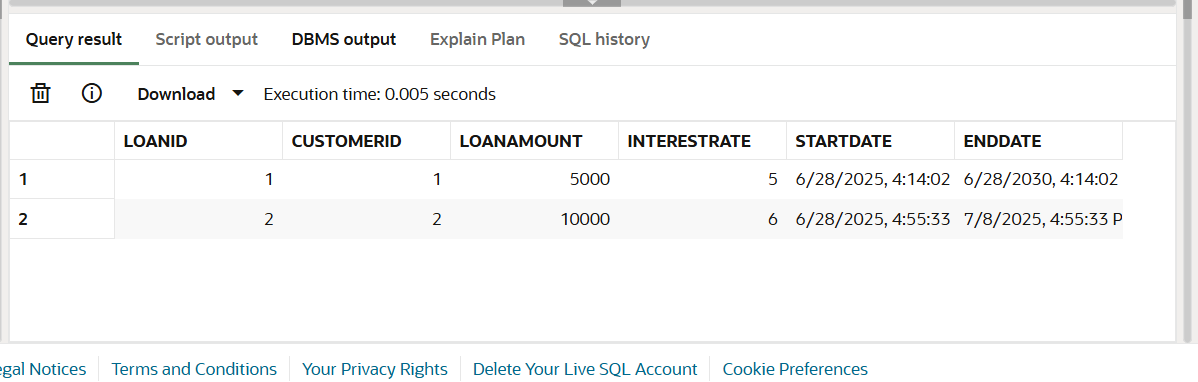
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

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

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

VALUES (2, 2, 10000, 6, SYSDATE, SYSDATE + 10);

**Output**

****

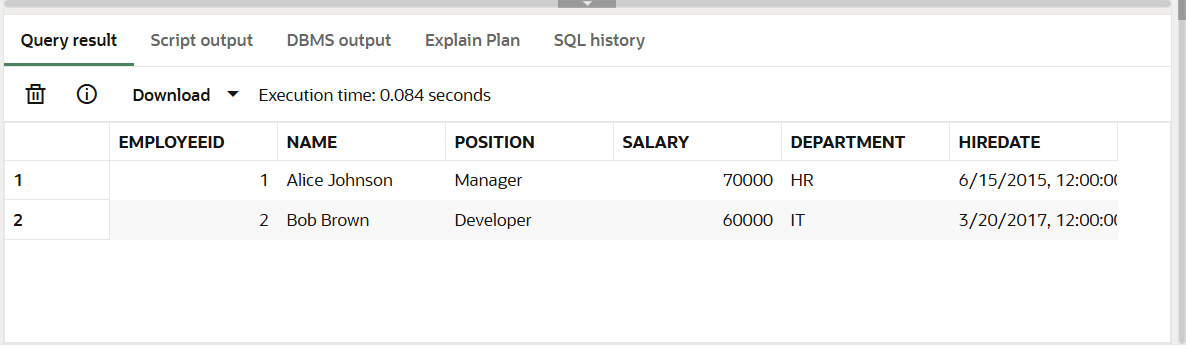
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

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**

****

**Question 1:**

**Scenario 1:**

BEGIN

FOR cust\_rec IN (SELECT CustomerID, DOB FROM Customers) LOOP

IF MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

END IF;

END LOOP;

END;

/

**Scenario 2:**

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));

BEGIN

FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = cust.CustomerID;

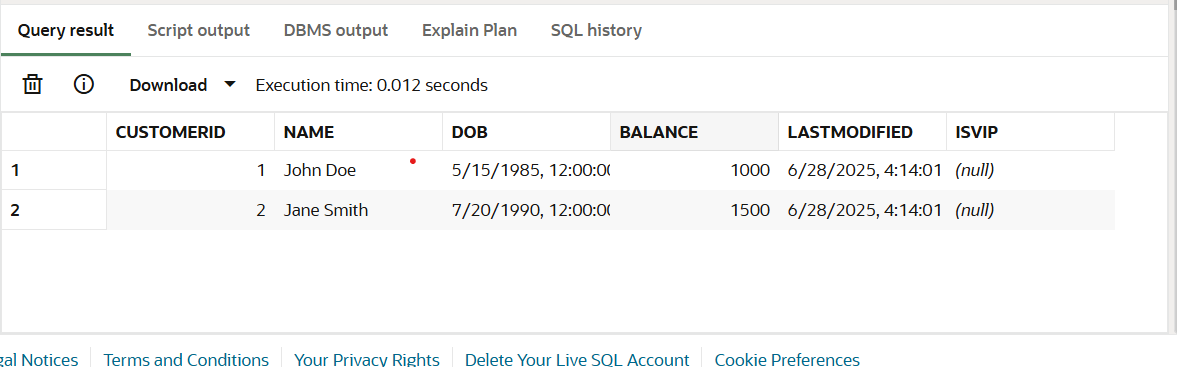
END IF;

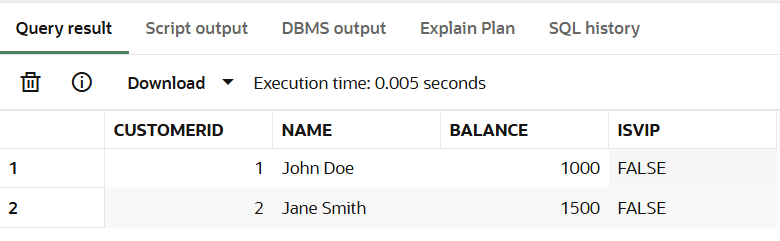
END LOOP;

END;

/

**Outputs:**



****

**Scenario 3:**

BEGIN

FOR loan\_rec IN (

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

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

' for customer ' || loan\_rec.Name ||

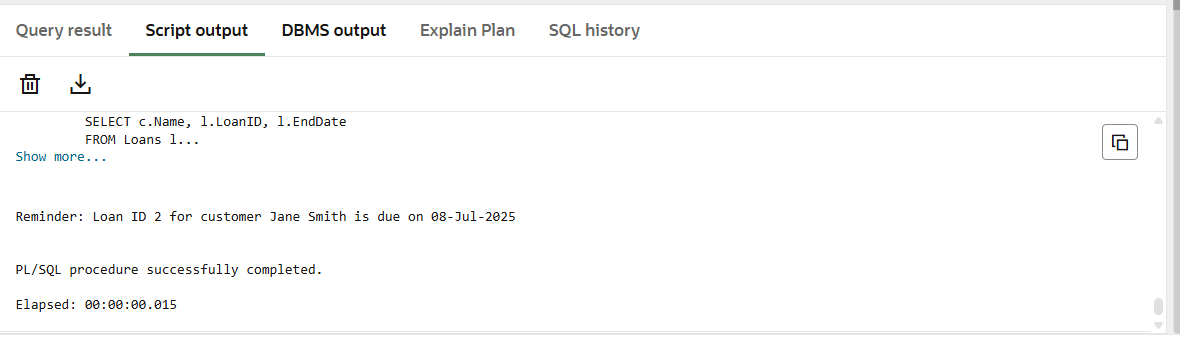
' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-Mon-YYYY'));

END LOOP;

END;

/

**Output:**



**Question 2:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

from\_account\_id IN NUMBER,

to\_account\_id IN NUMBER,

amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

from\_balance NUMBER;

BEGIN

-- Get sender balance

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

-- Check balance

IF from\_balance < amount THEN

RAISE insufficient\_funds;

END IF;

-- Subtract from sender

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account\_id;

-- Add to receiver

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('✅ Transfer of $' || amount || ' successful.');

EXCEPTION

WHEN insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Transfer failed: Insufficient funds.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Transfer failed: ' || SQLERRM);

ROLLBACK;

END;

/

**Test Query:**

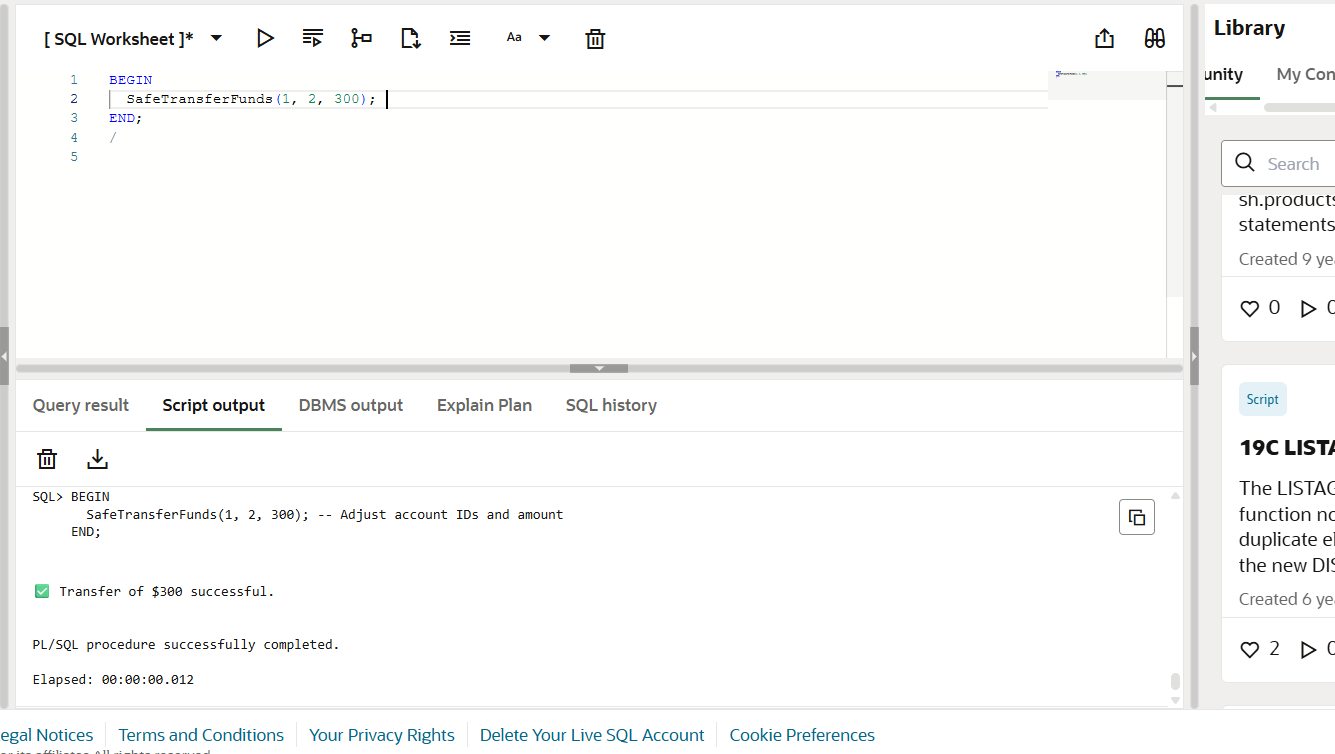
BEGIN

SafeTransferFunds(1, 2, 300);

END;

/

**Output**

****

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

emp\_id IN NUMBER,

percentage IN NUMBER

) IS

emp\_not\_found EXCEPTION;

emp\_salary Employees.Salary%TYPE;

BEGIN

-- Check if employee exists

SELECT Salary INTO emp\_salary FROM Employees WHERE EmployeeID = emp\_id;

-- Update salary

UPDATE Employees

SET Salary = Salary + (Salary \* percentage / 100)

WHERE EmployeeID = emp\_id;

DBMS\_OUTPUT.PUT\_LINE('✅ Salary updated by ' || percentage || '% for Employee ID ' || emp\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: Employee ID ' || emp\_id || ' does not exist.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: ' || SQLERRM);

END;

/

**Test Query:**

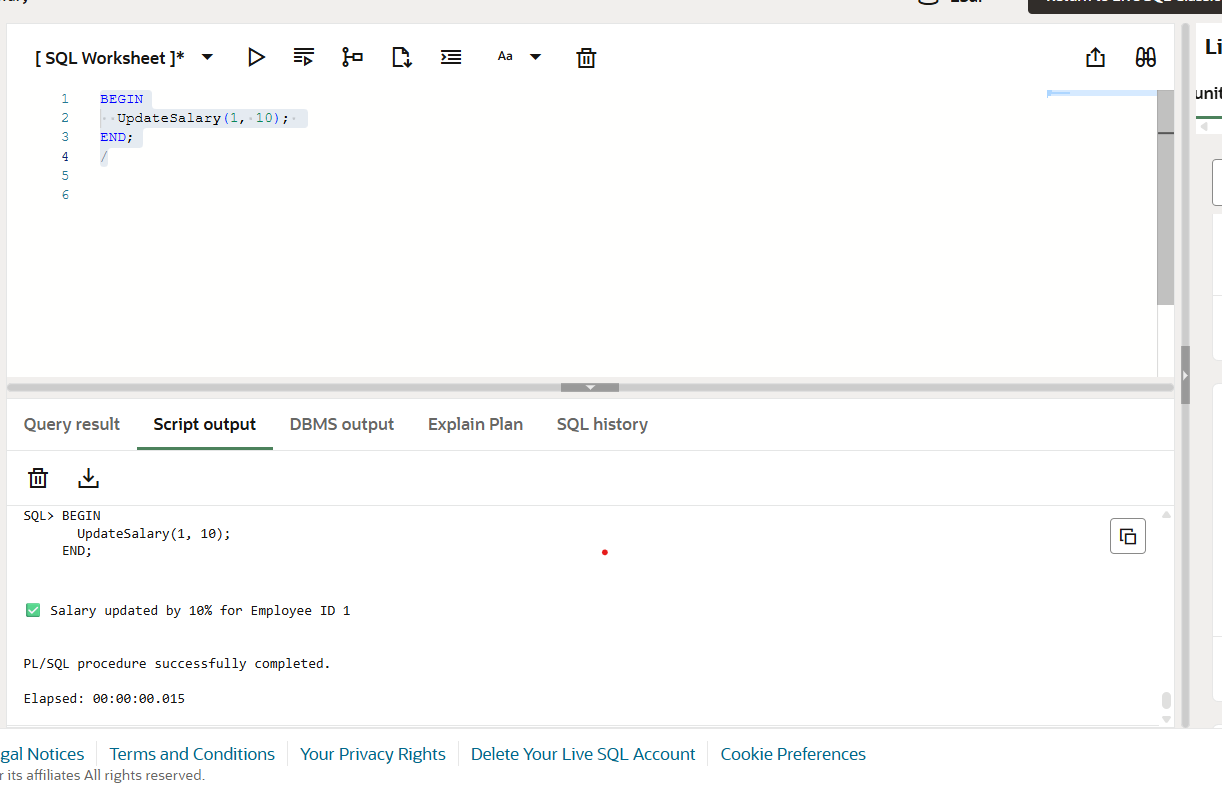
BEGIN

UpdateSalary(1, 10);

END;

/

**Outputs**

****

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

BEGIN

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

VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('✅ Customer ' || p\_name || ' added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: Customer ID ' || p\_id || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: ' || SQLERRM);

END;

/  
  
**Test Query:**

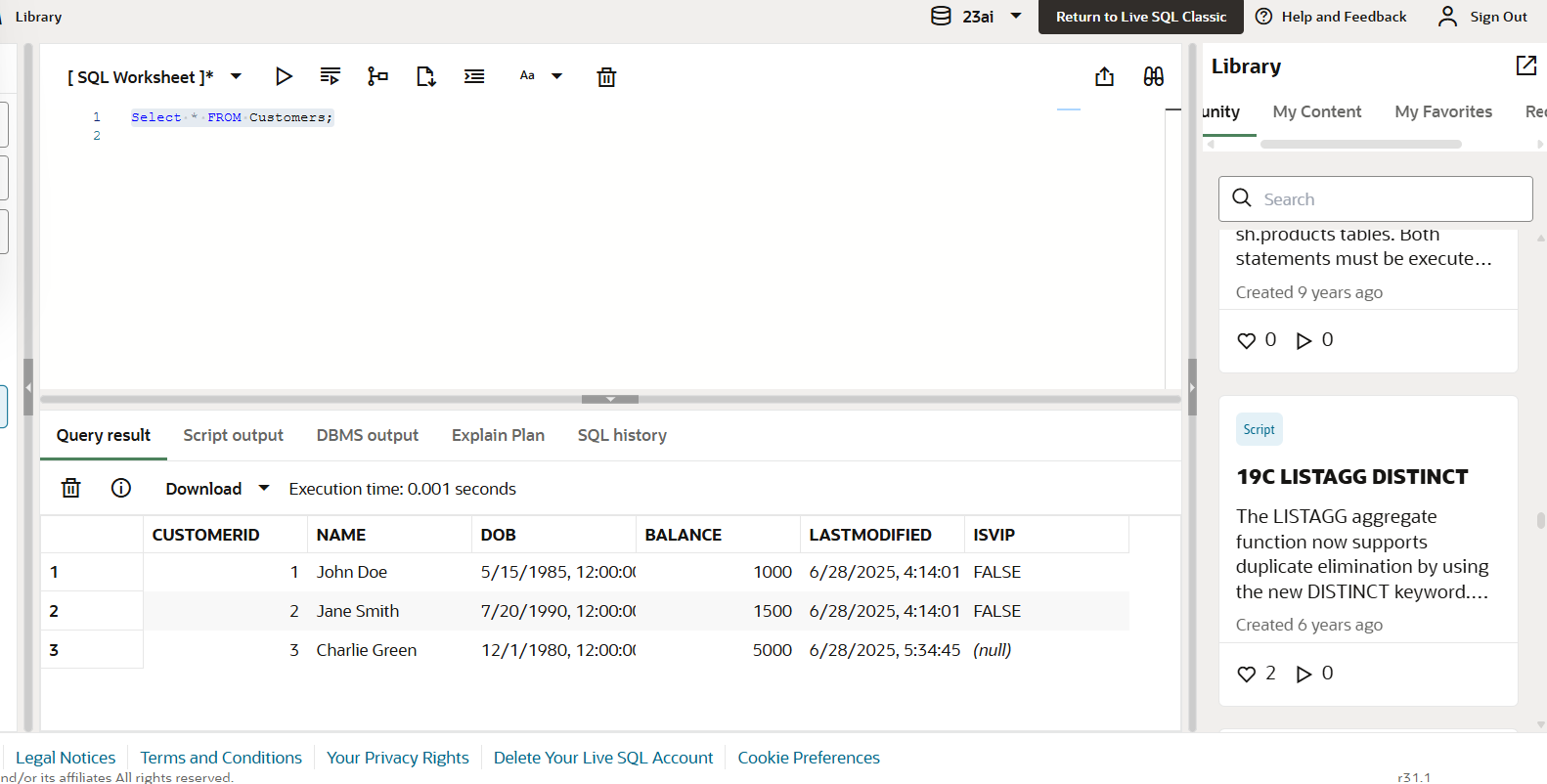
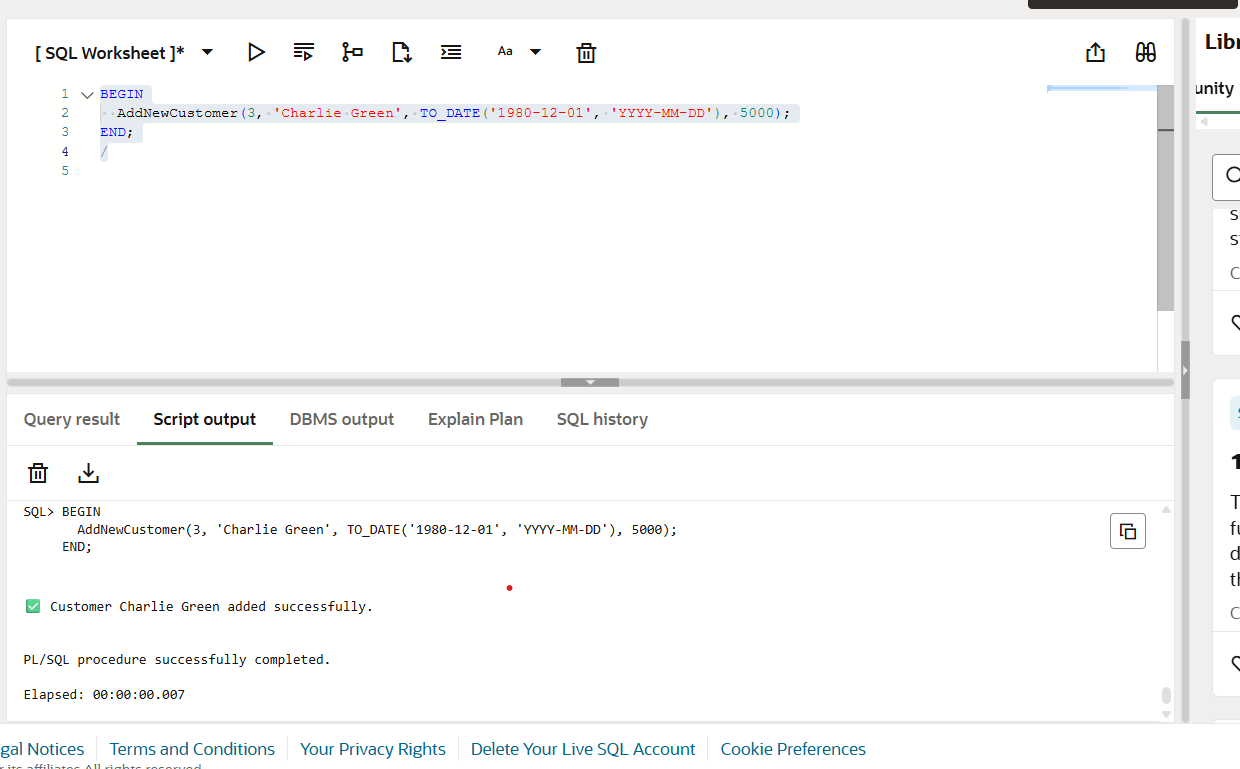
BEGIN

AddNewCustomer(3, 'Charlie Green', TO\_DATE('1980-12-01', 'YYYY-MM-DD'), 5000);

END;

/

**Outputs:**

****

**Question 3:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

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

END LOOP;

END;

/

**Test Query:**

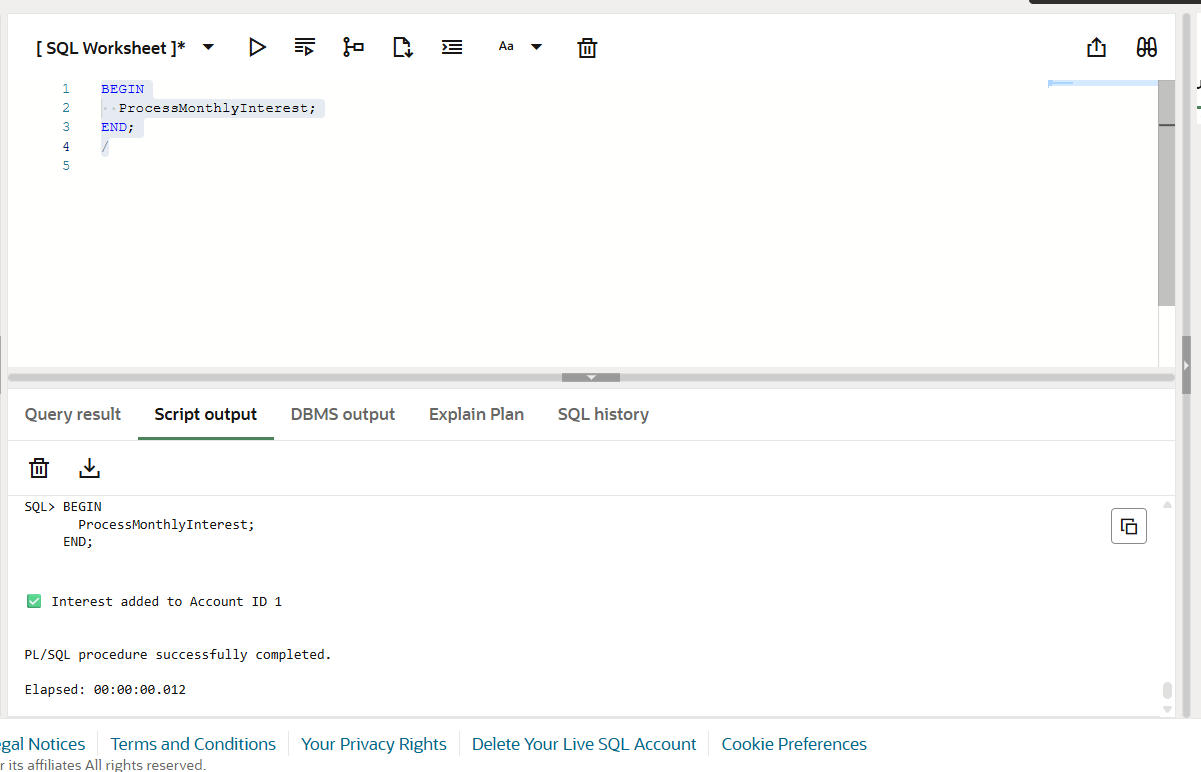
BEGIN

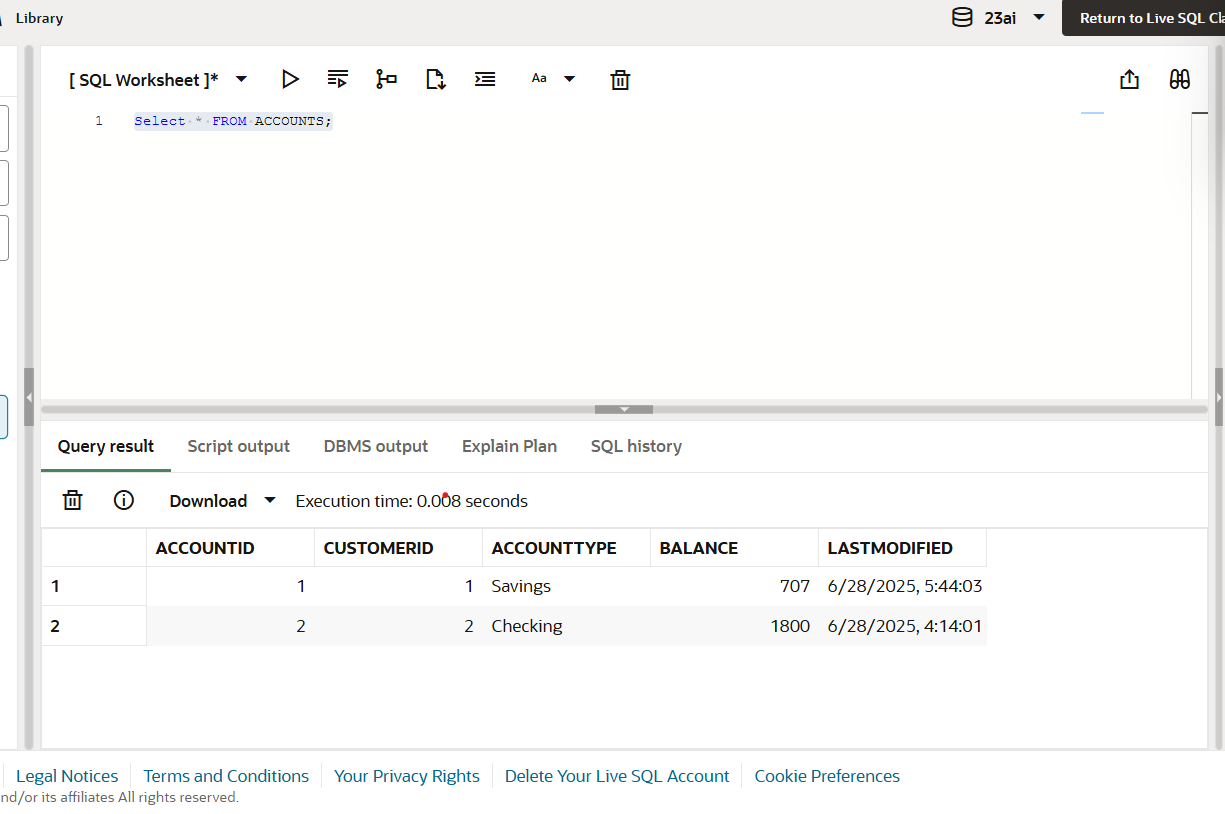
ProcessMonthlyInterest;

END;

/

SELECT \* FROM Customers;

**Outputs:**

****

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = dept\_name;

DBMS\_OUTPUT.PUT\_LINE('✅ Bonus of ' || bonus\_percent || '% applied to department ' || dept\_name);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: ' || SQLERRM);

END;

/

**Test Query:**

BEGIN

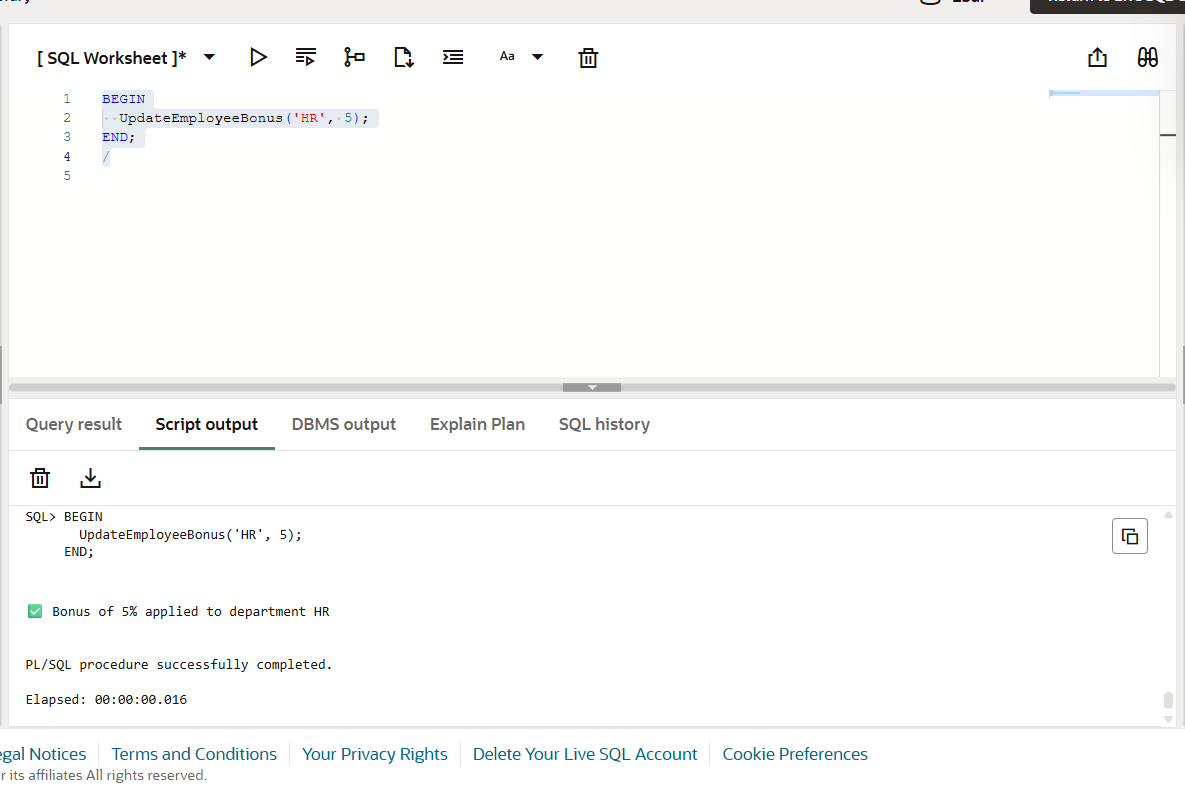
UpdateEmployeeBonus('HR', 5);

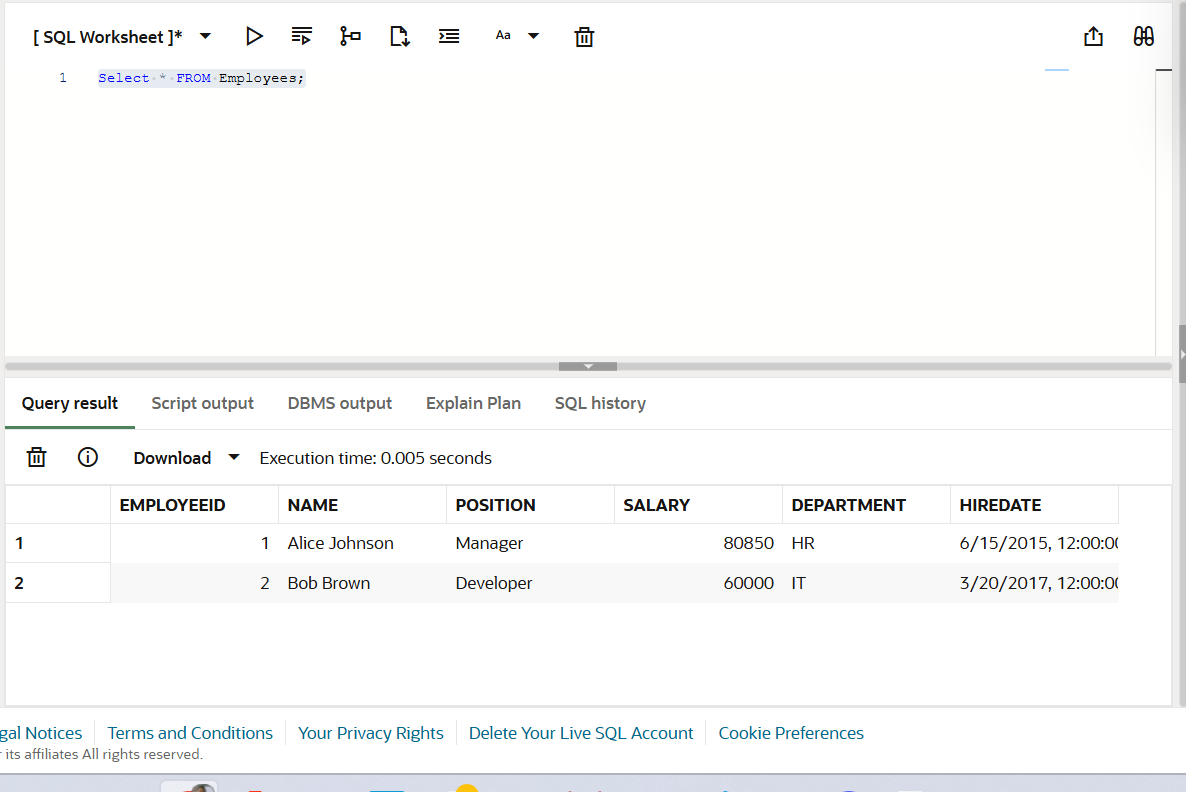
END;

/

SELECT \* FROM Employees;

**Outputs:**

****

****

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

-- Check available balance

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

IF from\_balance < amount THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Transfer failed: Insufficient funds in Account ID ' || from\_account);

RETURN;

END IF;

-- Deduct from sender

UPDATE Accounts

SET Balance = Balance - amount,

LastModified = SYSDATE

WHERE AccountID = from\_account;

-- Credit to receiver

UPDATE Accounts

SET Balance = Balance + amount,

LastModified = SYSDATE

WHERE AccountID = to\_account;

DBMS\_OUTPUT.PUT\_LINE('✅ Transferred $' || amount || ' from Account ID ' || from\_account || ' to Account ID ' || to\_account);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: One or both account IDs not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Error: ' || SQLERRM);

END;

/

**Test Query:**

BEGIN

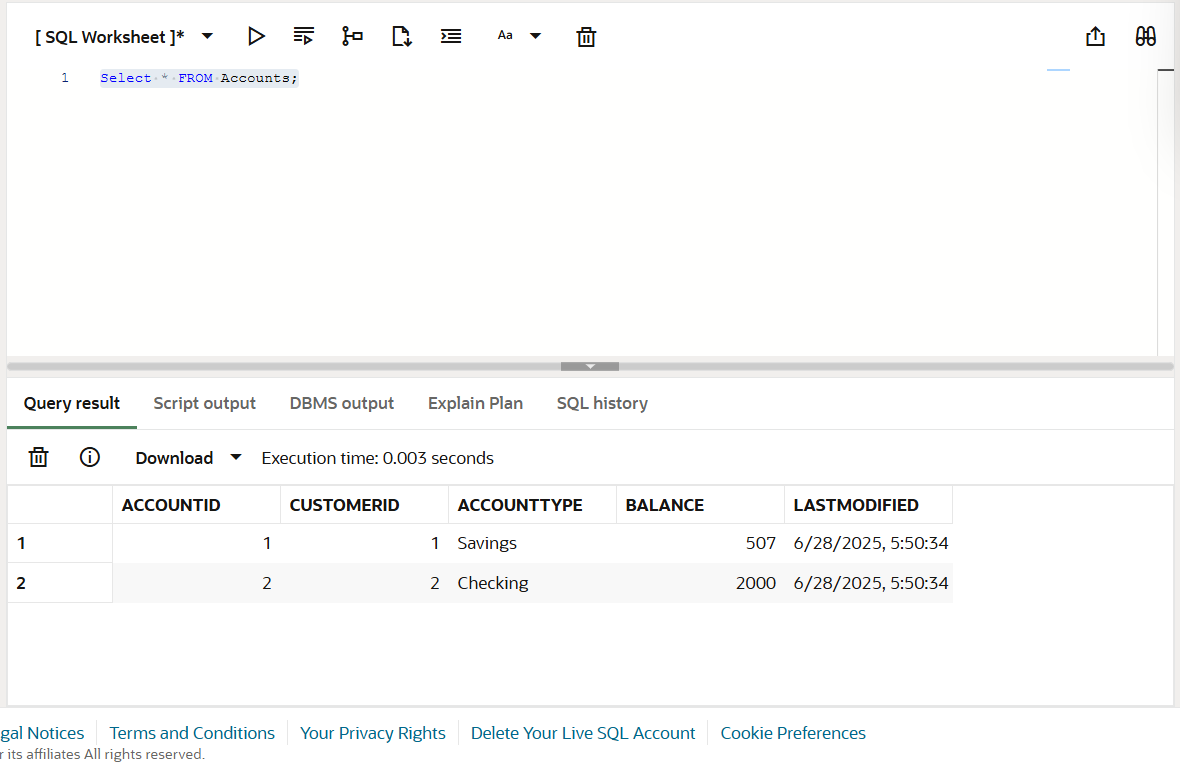
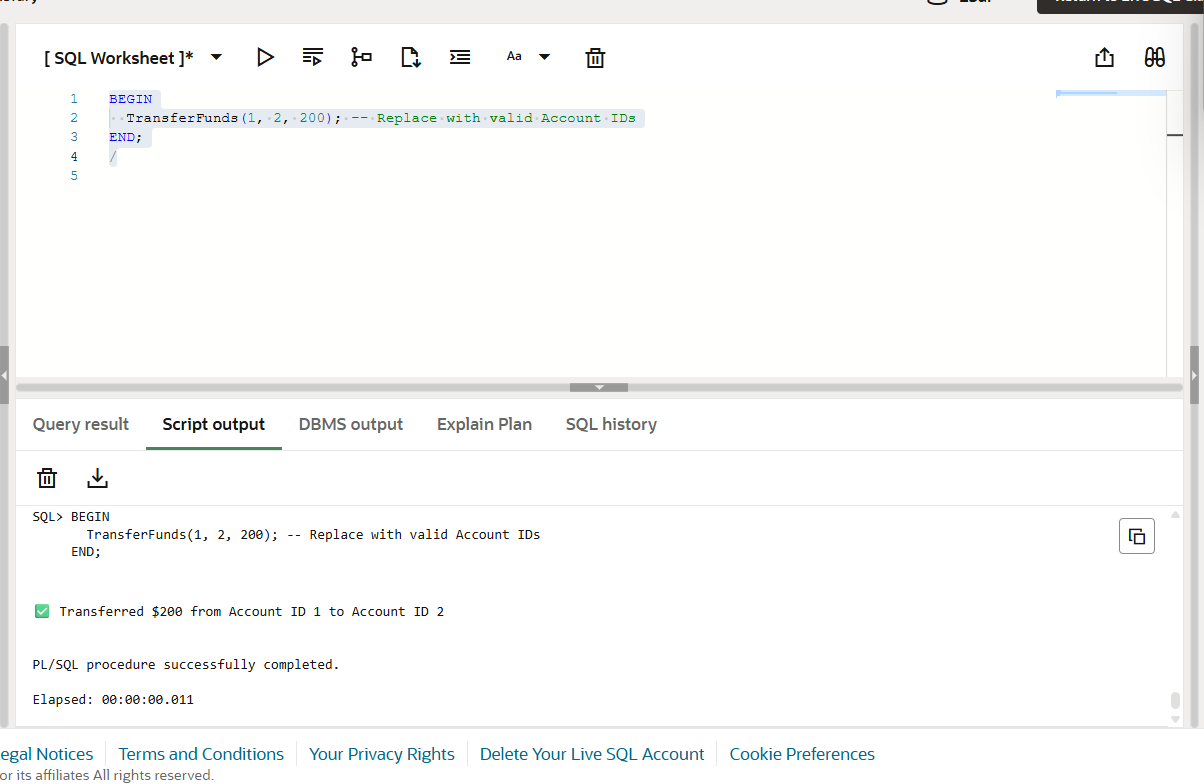
TransferFunds(1, 2, 200);

END;

/

Select \* FROM Accounts;

**Outputs:**



**Question 4:**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge (

  dob IN DATE

) RETURN NUMBER IS

  age NUMBER;

BEGIN

  age := FLOOR(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

  RETURN age;

END;

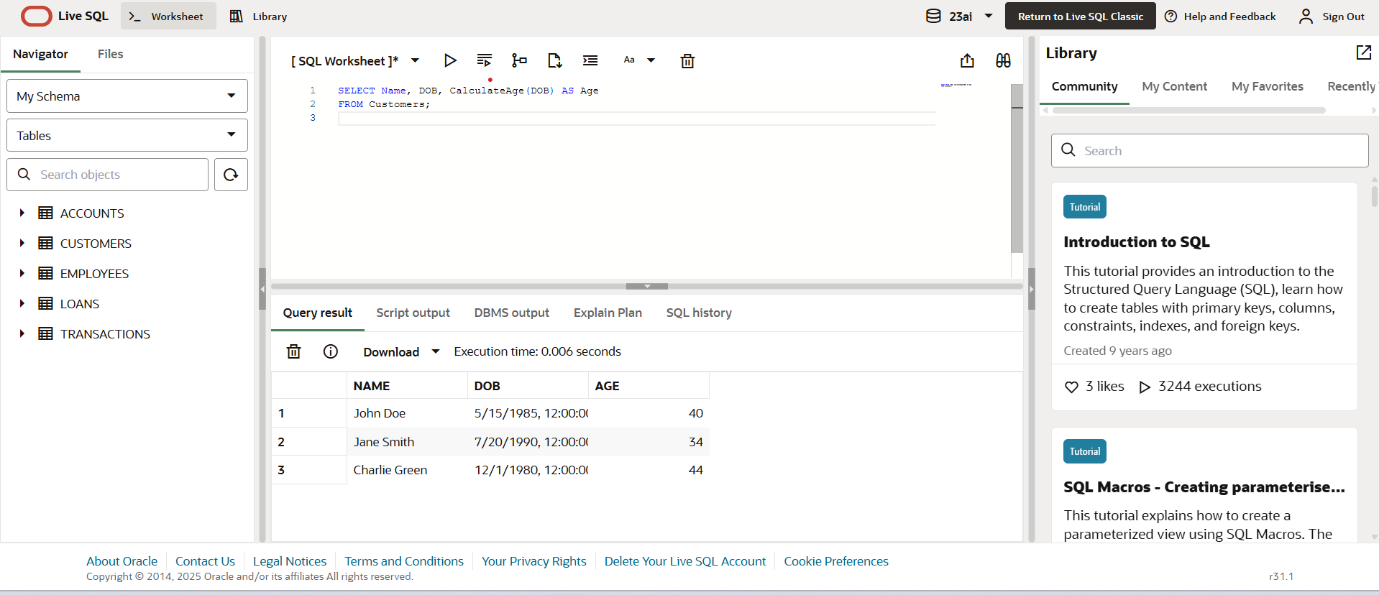
/

**Test Query:**

SELECT Name, DOB, CalculateAge(DOB) AS Age

FROM Customers;

**Output**



**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

  loan\_amount IN NUMBER,

  annual\_interest\_rate IN NUMBER,

  years IN NUMBER

) RETURN NUMBER IS

  r NUMBER; -- monthly interest rate

  n NUMBER; -- total months

  emi NUMBER;

BEGIN

  r := annual\_interest\_rate / 12 / 100;

  n := years \* 12;

  IF r = 0 THEN

    emi := loan\_amount / n;

  ELSE

    emi := loan\_amount \* r \* POWER(1 + r, n) / (POWER(1 + r, n) - 1);

  END IF;

  RETURN ROUND(emi, 2);

END;

/

**Test Query:**

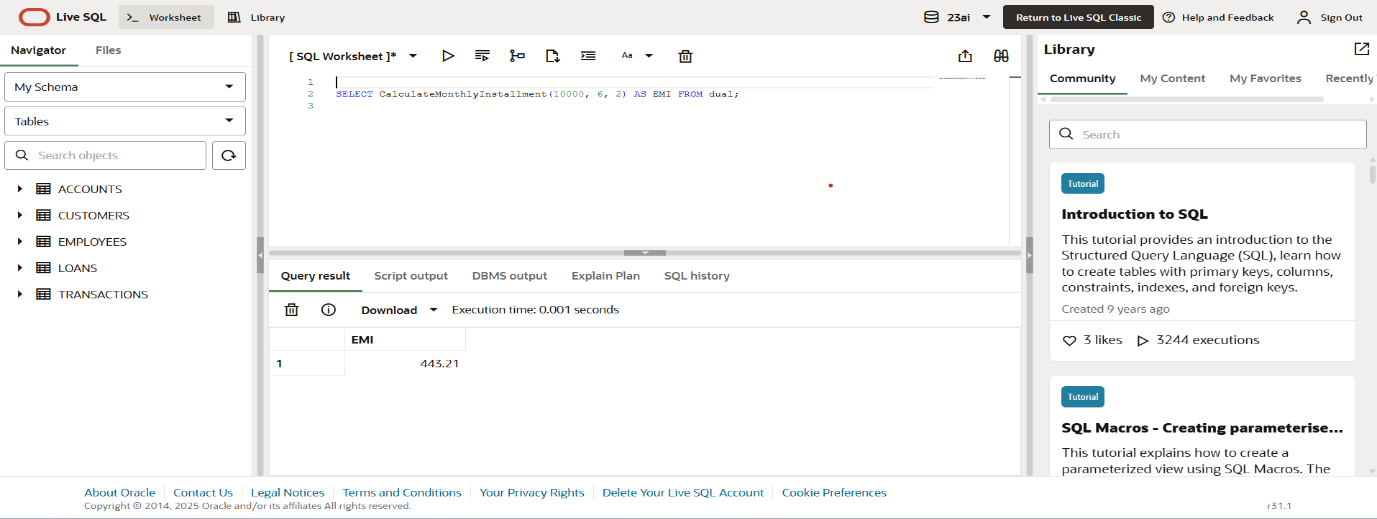
Formula Used : P \* R \* (1 + R)^N / ((1 + R)^N - 1) where :

* P = Loan Amount
* R = Monthly interest rate (annual rate / 12 / 100)
* N = Total number of monthly payments (years × 12)

Code:

SELECT CalculateMonthlyInstallment(10000, 6, 2) AS EMI FROM dual;

**Output:**



**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

account\_id IN NUMBER,

amount IN NUMBER

) RETURN NUMBER IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = account\_id;

IF bal >= amount THEN

RETURN 1; -- TRUE

ELSE

RETURN 0; -- FALSE

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

RETURN 0;

END;

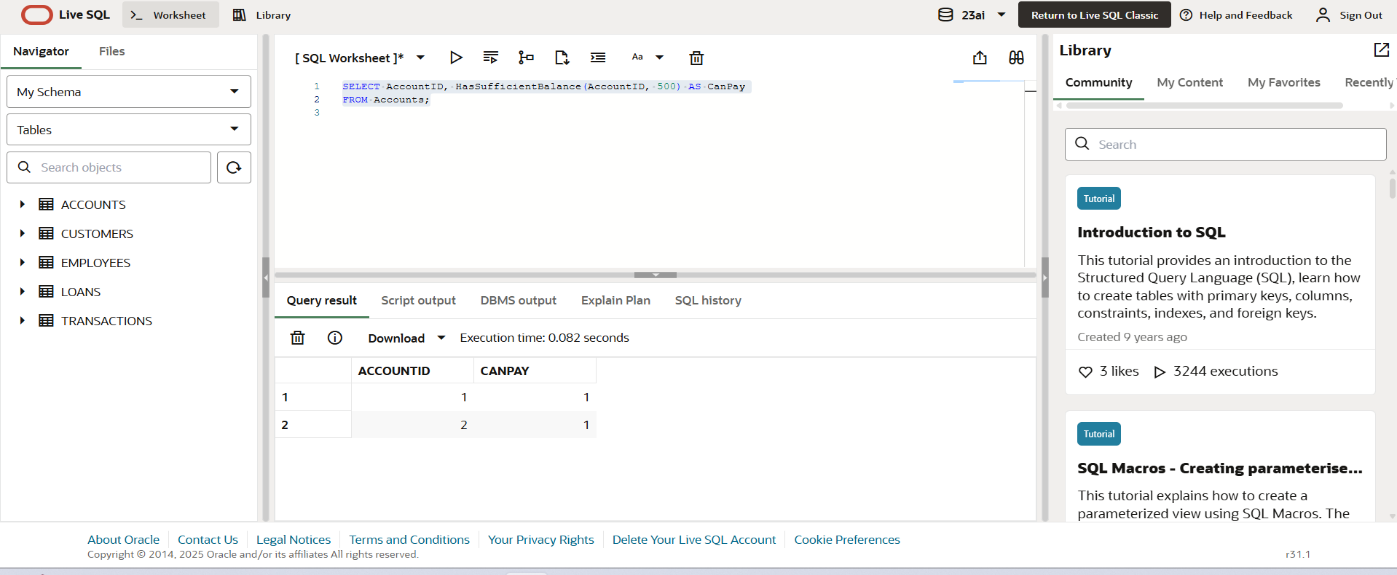
/

**Test Query:**

SELECT AccountID, HasSufficientBalance(AccountID, 500) AS CanPay

FROM Accounts;

**Outputs:**

****

**Question 5:**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Test Query:**

UPDATE Customers

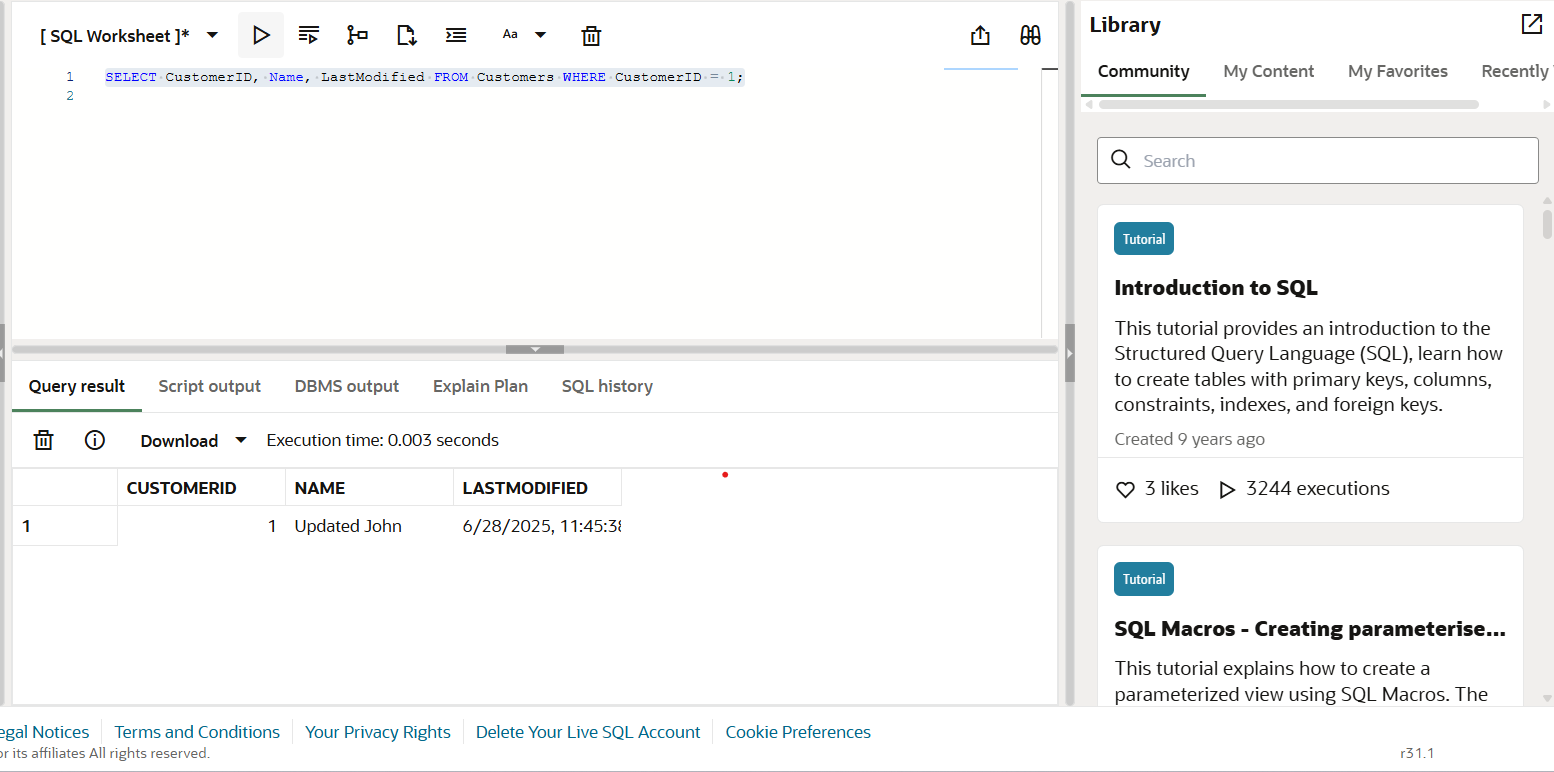
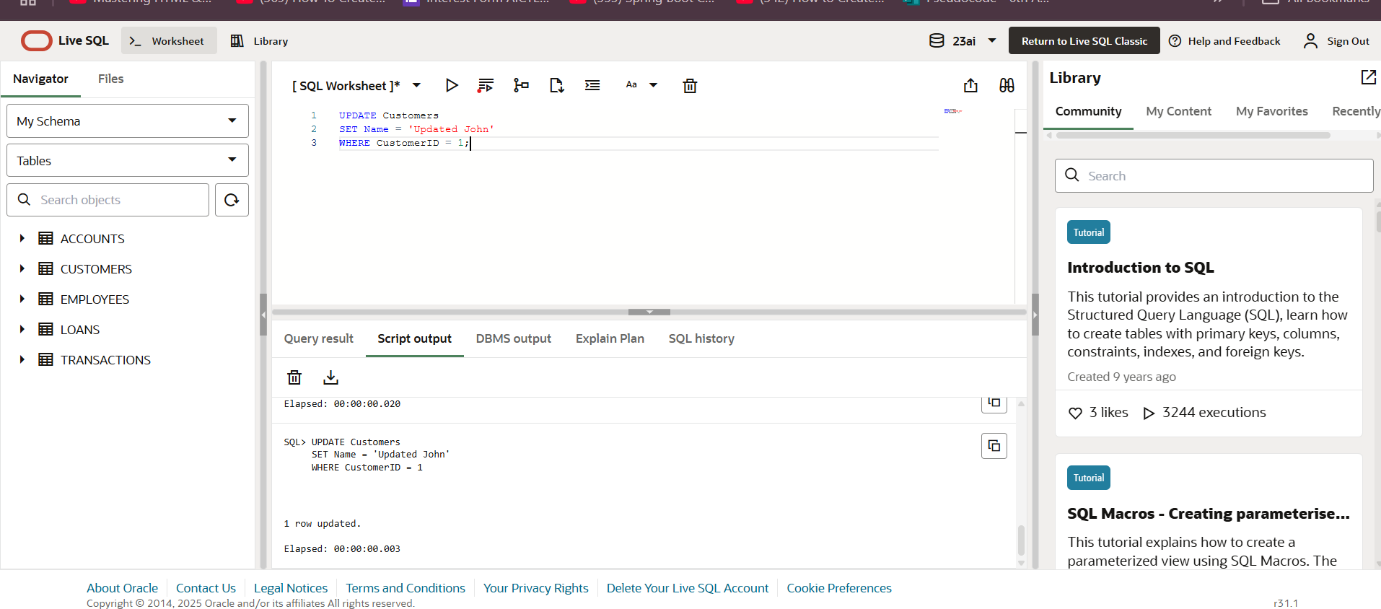
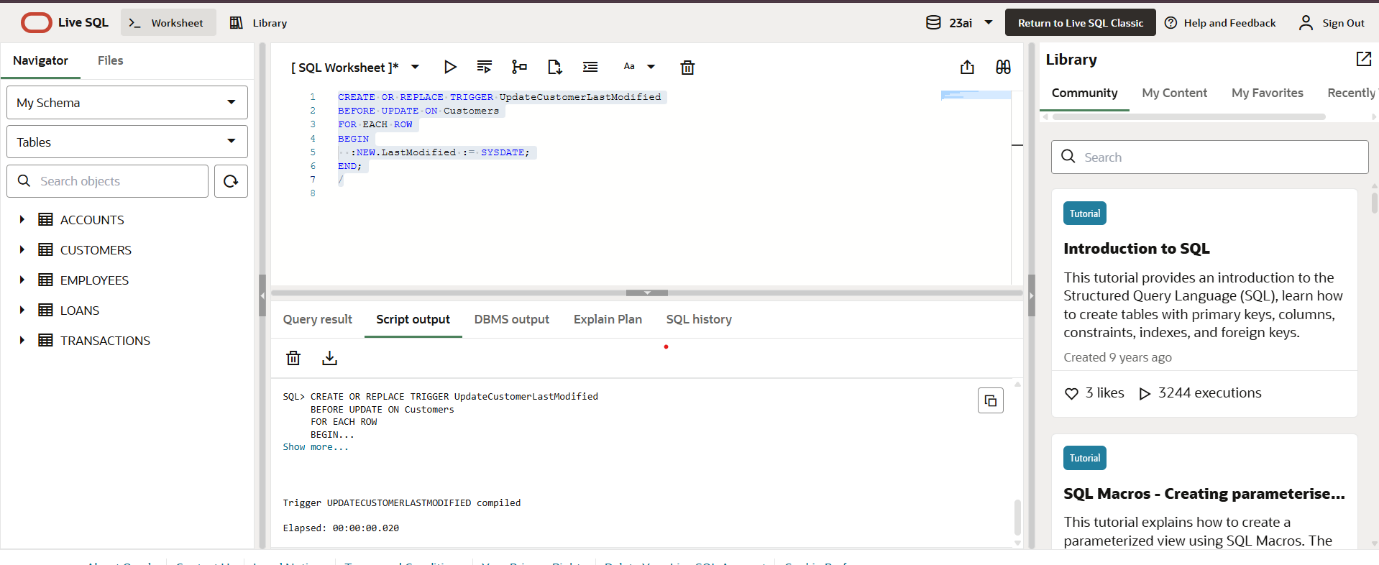
SET Name = 'Updated John'

WHERE CustomerID = 1;

//for checking if LastModified is Updated or not

SELECT CustomerID, Name, LastModified FROM Customers WHERE CustomerID = 1;

**Output**



**Scenario 2:**

*Creating Audit Table*

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

LoggedAt DATE

);

*Trigger LogTransaction*

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Amount, TransactionType, LoggedAt)

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END;

/

**Test Query:**

*Inserting a new transaction*

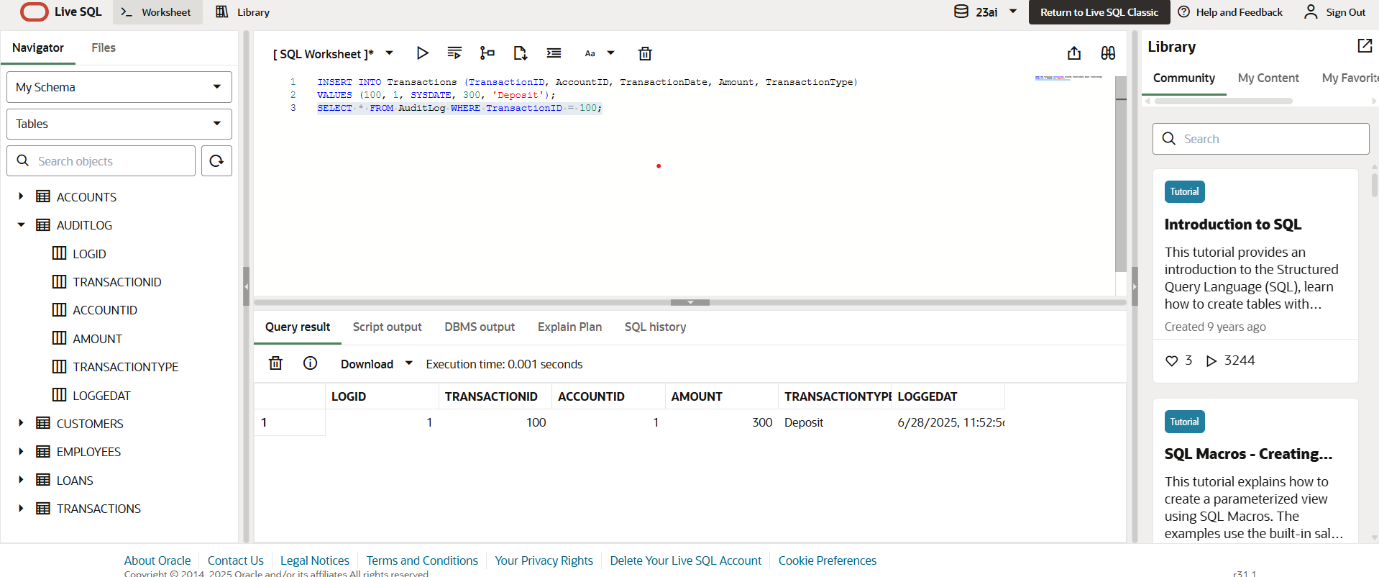
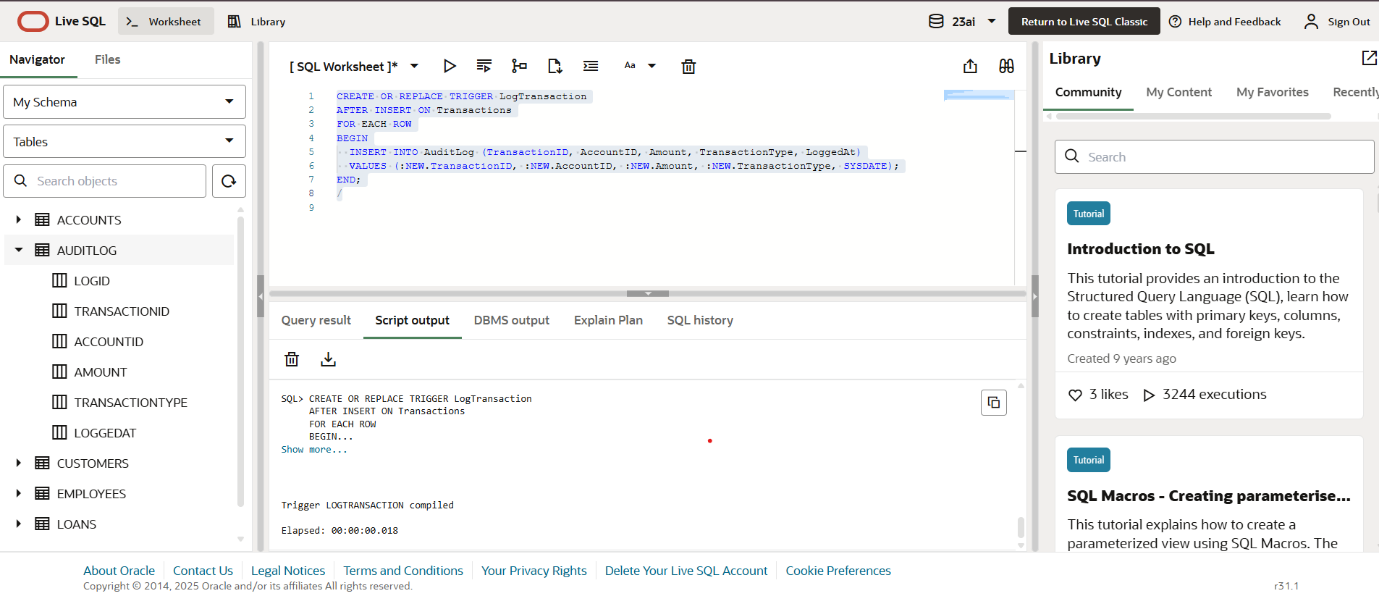
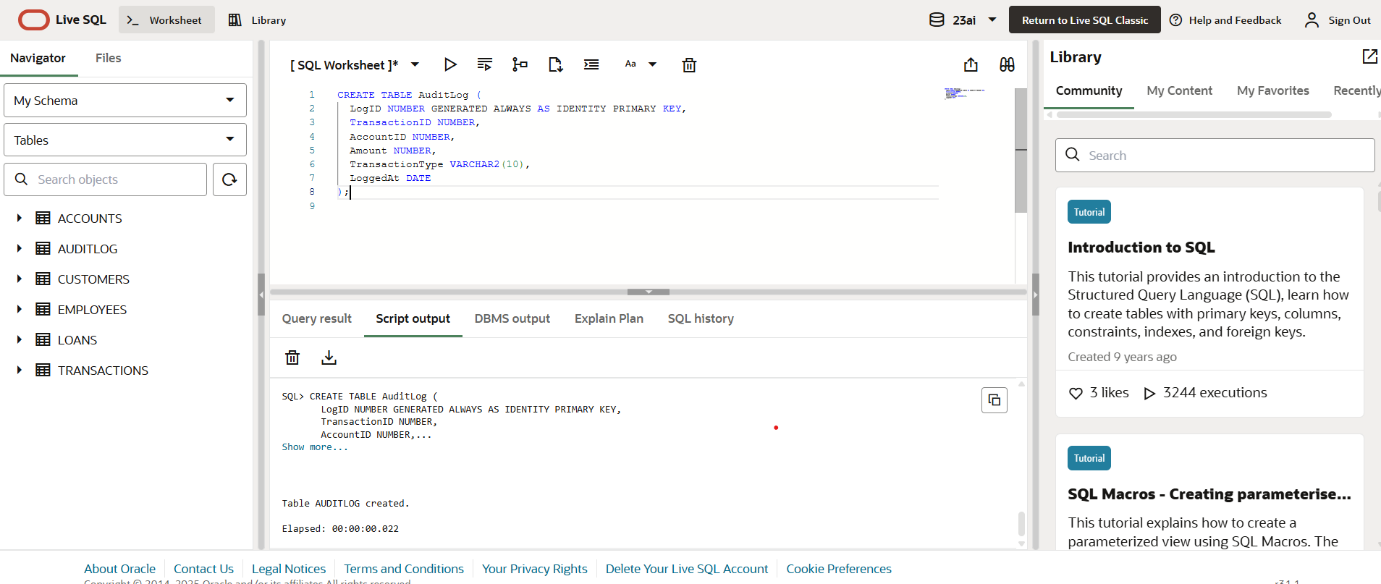
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (100, 1, SYSDATE, 300, 'Deposit');

*Checking the audit log*

SELECT \* FROM AuditLog WHERE TransactionID = 100;

**Outputs:**

****

**Scenario 3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

acc\_balance NUMBER;

BEGIN

-- Get account balance

SELECT Balance INTO acc\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

-- Rule: Deposit must be positive

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

RAISE\_APPLICATION\_ERROR(-20001, '❌ Deposit amount must be positive.');

END IF;

-- Rule: Withdrawal must not exceed balance

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > acc\_balance THEN

RAISE\_APPLICATION\_ERROR(-20002, '❌ Withdrawal exceeds account balance.');

END IF;

END;

/

**Test Queries:**

*Valid Insert:*

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

VALUES (101, 1, SYSDATE, 100, 'Withdrawal');

*Invalid Insert – Deposit <= 0*

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

VALUES (102, 1, SYSDATE, 0, 'Deposit');

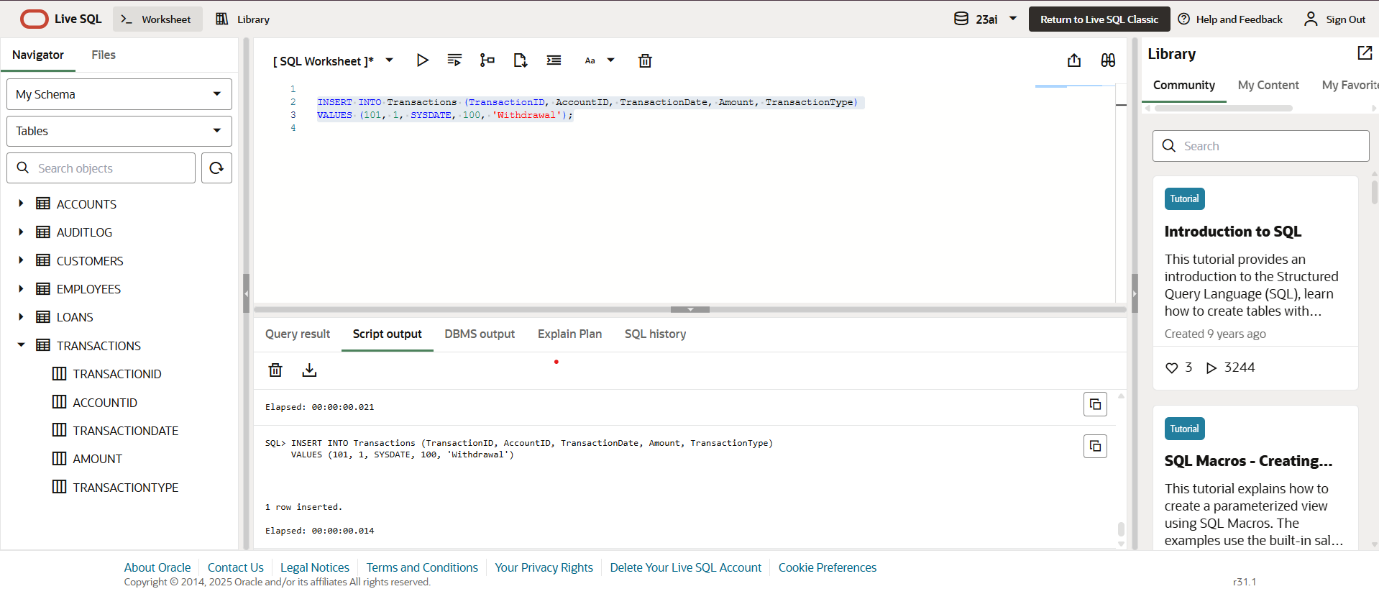
*Invalid Insert – Withdrawal > Balance*

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

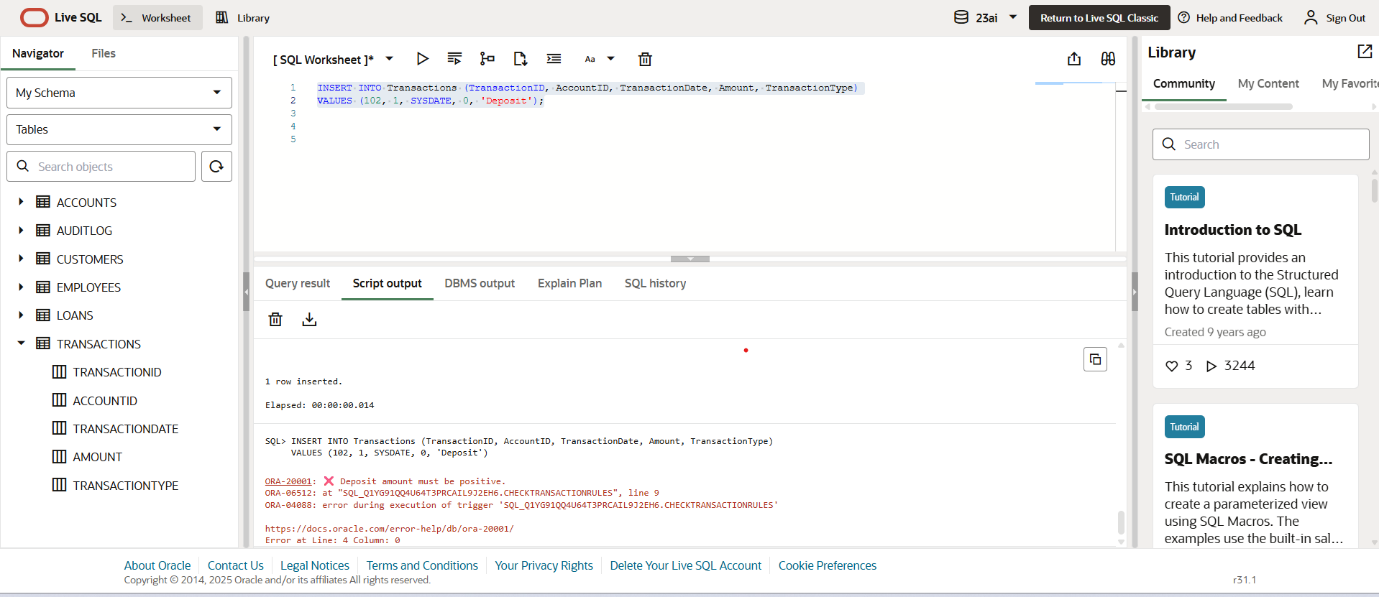
VALUES (103, 1, SYSDATE, 999999, 'Withdrawal');

**Outputs:**

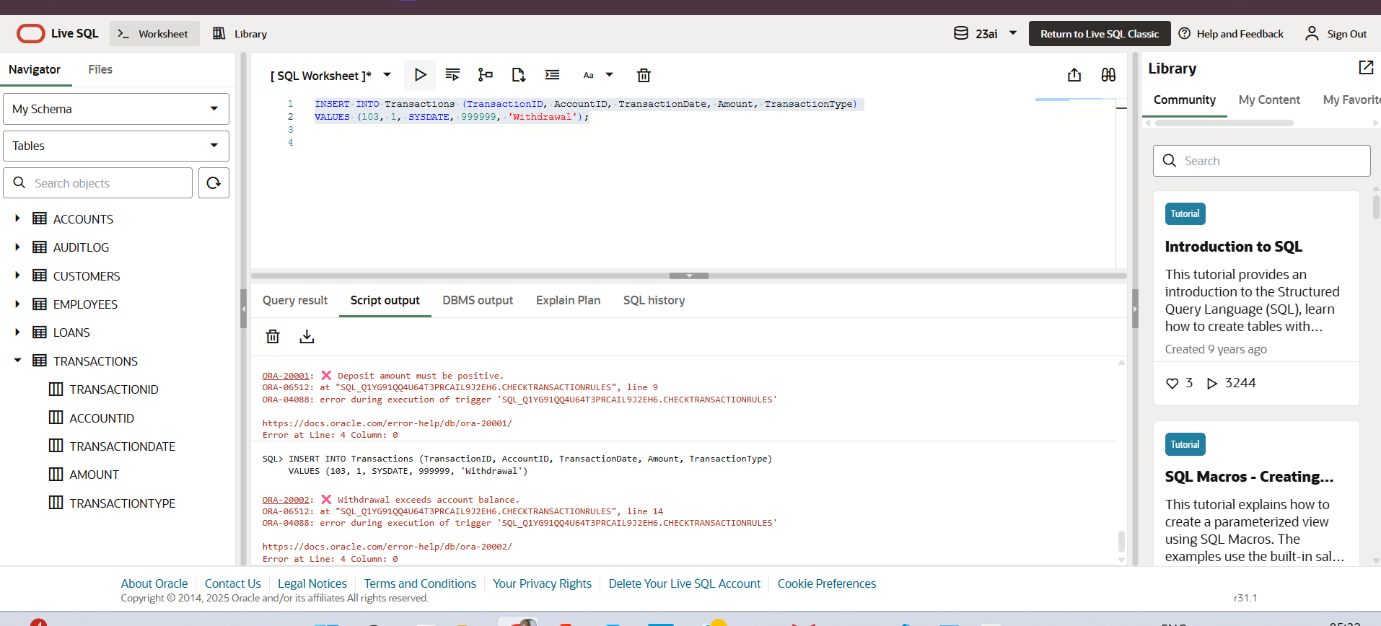
**Valid**

****

**Invalid**

****

**Invalid**

****

**Question 6:**

**Scenario 1:**

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, Name FROM Customers;

CURSOR txn\_cursor (p\_cust\_id NUMBER) IS

SELECT t.TransactionDate, t.Amount, t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

WHERE a.CustomerID = p\_cust\_id

AND TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_cust\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_type Transactions.TransactionType%TYPE;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('📄 Statement for: ' || cust\_rec.Name);

OPEN txn\_cursor(cust\_rec.CustomerID);

LOOP

FETCH txn\_cursor INTO v\_date, v\_amount, v\_type;

EXIT WHEN txn\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(' → ' || TO\_CHAR(v\_date, 'DD-Mon') || ' - ' || v\_type || ': $' || v\_amount);

END LOOP;

CLOSE txn\_cursor;

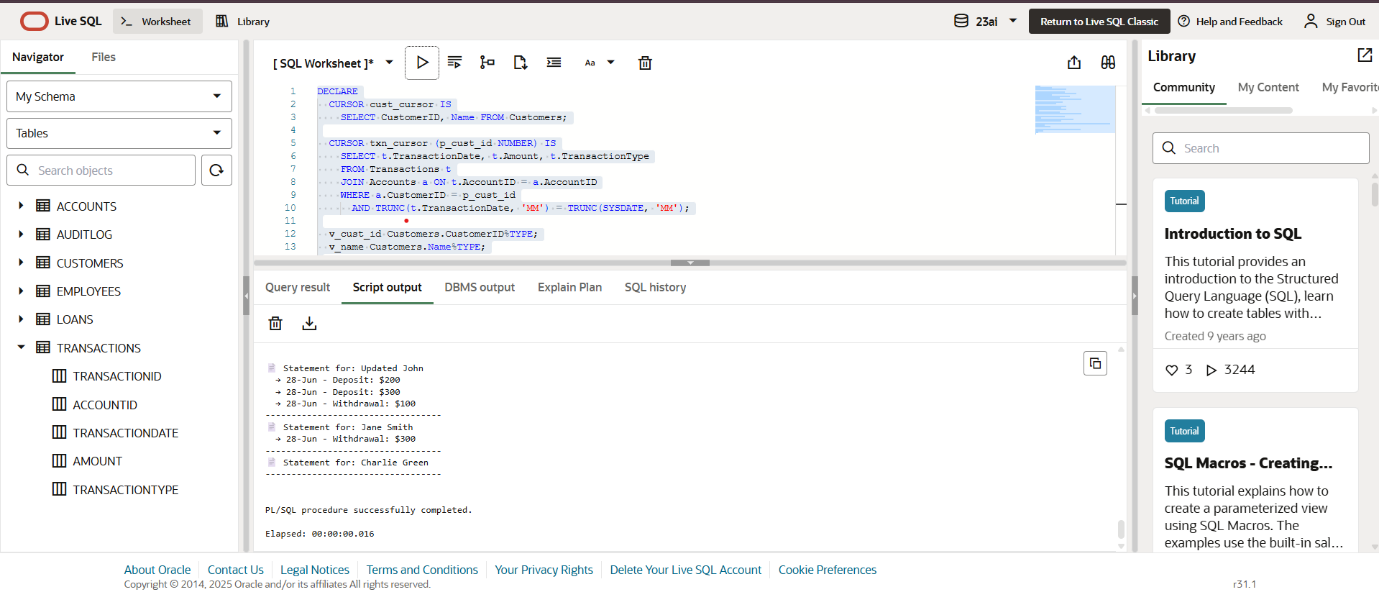
DBMS\_OUTPUT.PUT\_LINE('----------------------------------');

END LOOP;

END;

/

**Outputs**

****

**Scenario 2:**

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_id Accounts.AccountID%TYPE;

v\_bal Accounts.Balance%TYPE;

annual\_fee CONSTANT NUMBER := 50;

BEGIN

FOR acc\_rec IN account\_cursor LOOP

IF acc\_rec.Balance >= annual\_fee THEN

UPDATE Accounts

SET Balance = Balance - annual\_fee,

LastModified = SYSDATE

WHERE AccountID = acc\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE(' Annual fee deducted from Account ID: ' || acc\_rec.AccountID);

ELSE

DBMS\_OUTPUT.PUT\_LINE(' Skipped Account ID ' || acc\_rec.AccountID || ': Insufficient funds.');

END IF;

END LOOP;

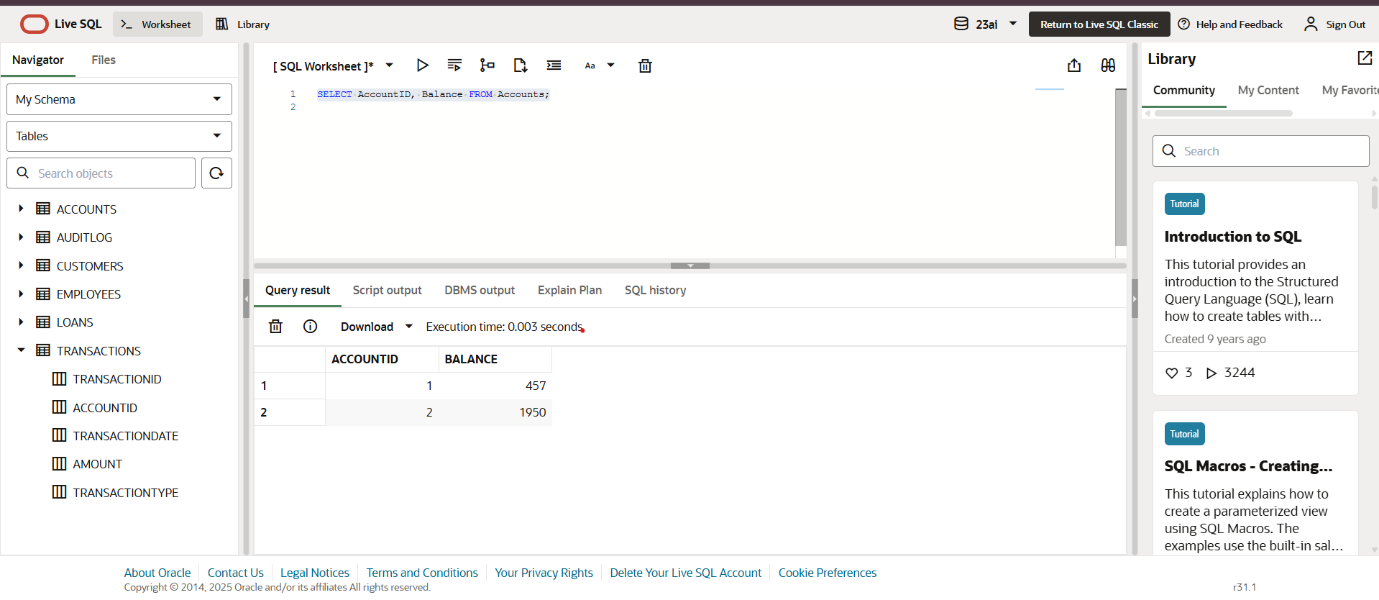
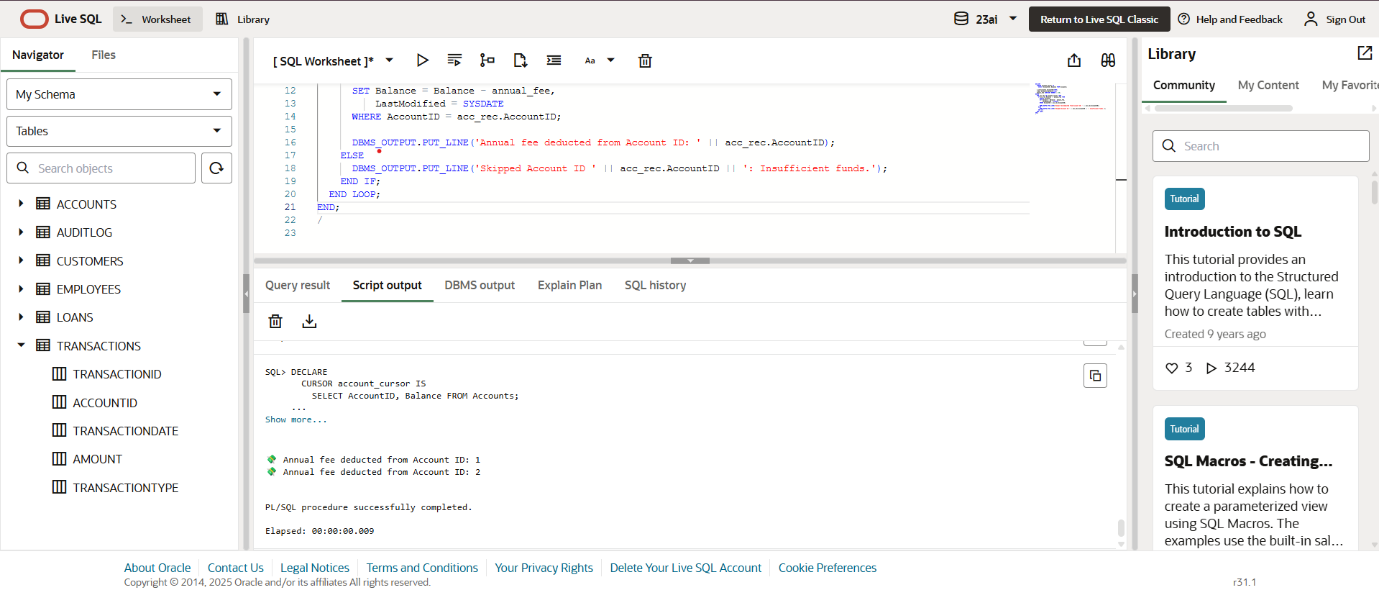
END;

/

**Test Query:**

SELECT AccountID, Balance FROM Accounts;

**Outputs**

****

**Scenario 3:**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

v\_id Loans.LoanID%TYPE;

v\_amt Loans.LoanAmount%TYPE;

v\_rate Loans.InterestRate%TYPE;

new\_rate NUMBER;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

IF loan\_rec.LoanAmount >= 10000 THEN

new\_rate := loan\_rec.InterestRate + 1.5;

ELSE

new\_rate := loan\_rec.InterestRate + 1.0;

END IF;

UPDATE Loans

SET InterestRate = new\_rate

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('📈 Loan ID ' || loan\_rec.LoanID || ' interest updated to ' || new\_rate || '%');

END LOOP;

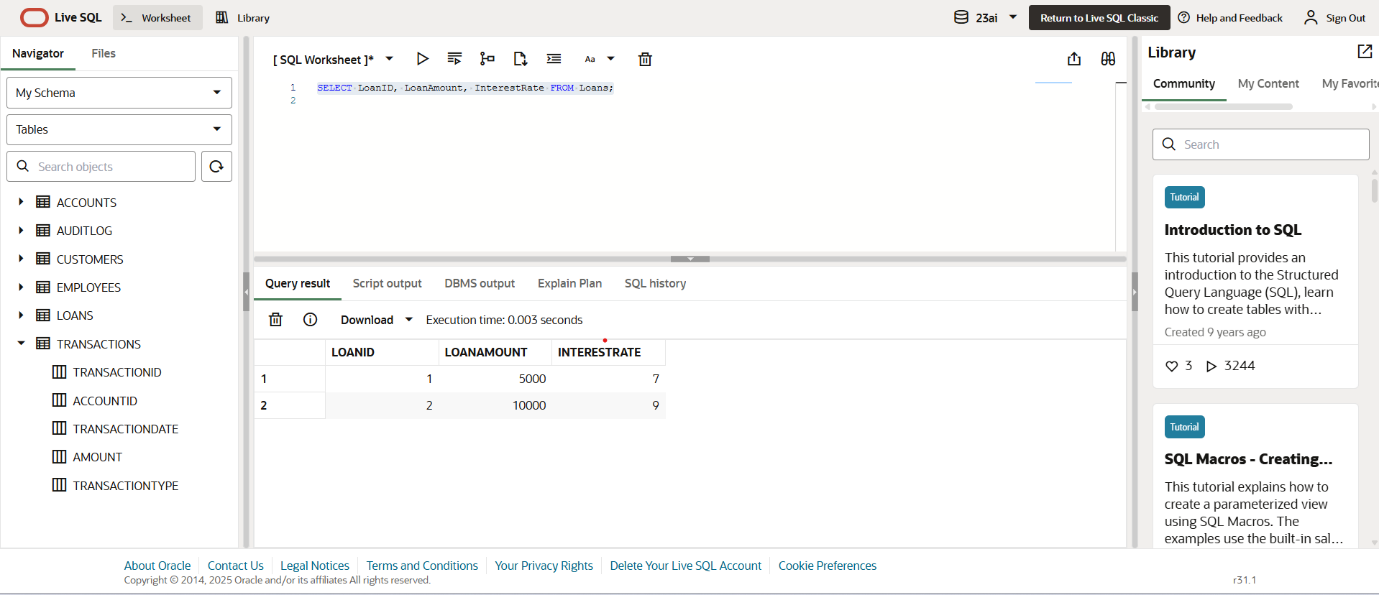
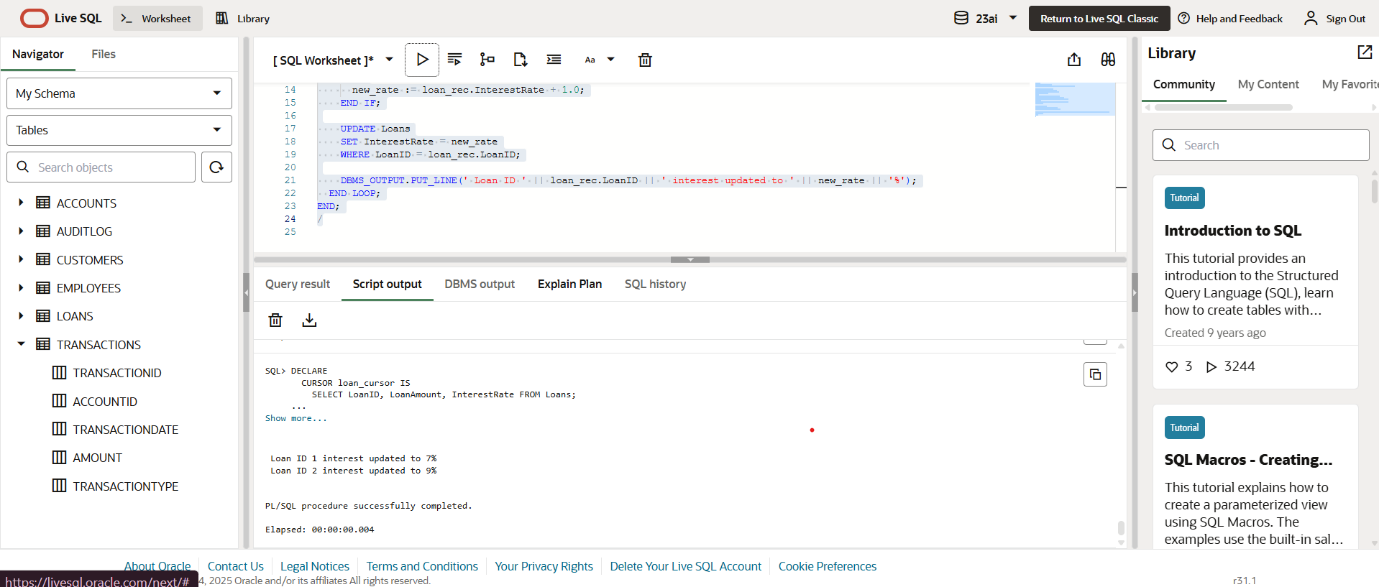
END;

/

**Test Query:**

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

**Outputs**



**Question 7:**

**Scenario 1:**

*Package Specification*

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomerDetails(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

*Package Body*

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

END;

PROCEDURE UpdateCustomerDetails(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE) IS

BEGIN

UPDATE Customers

SET Name = p\_name,

DOB = p\_dob,

LastModified = SYSDATE

WHERE CustomerID = p\_id;

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Customers WHERE CustomerID = p\_id;

RETURN bal;

END;

END CustomerManagement;

/

**Test Query:**

BEGIN

CustomerManagement.AddCustomer(10, 'Charlie Blue', TO\_DATE('1980-08-20', 'YYYY-MM-DD'), 5000);

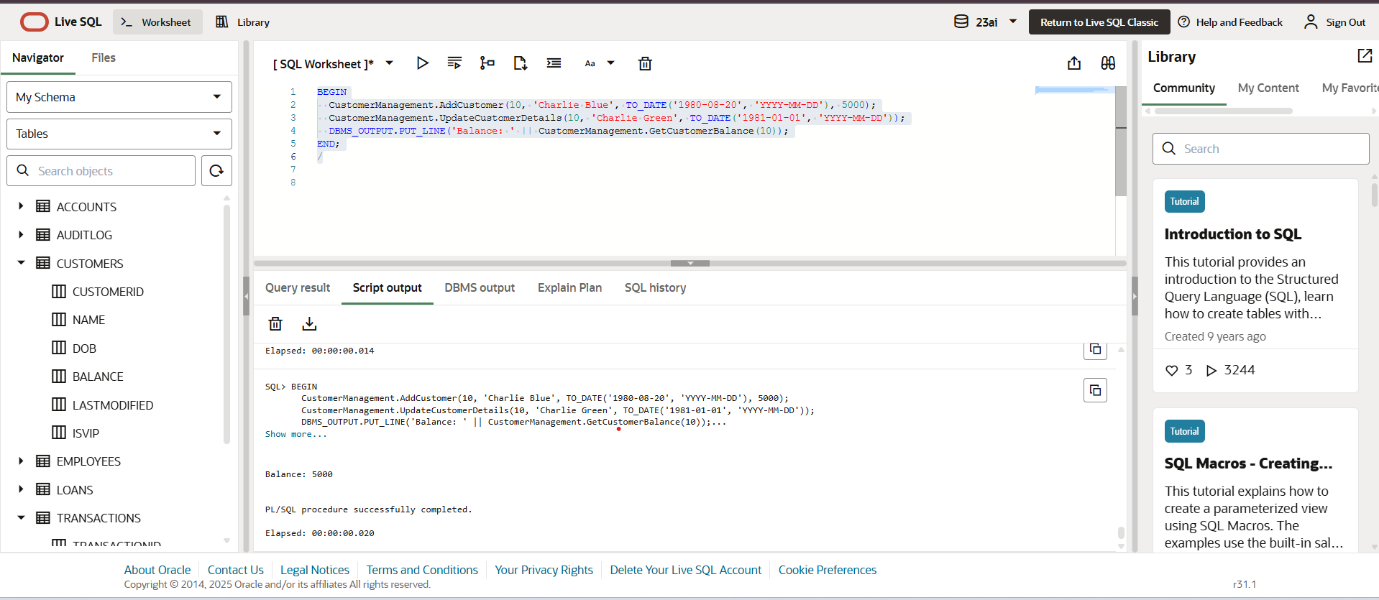
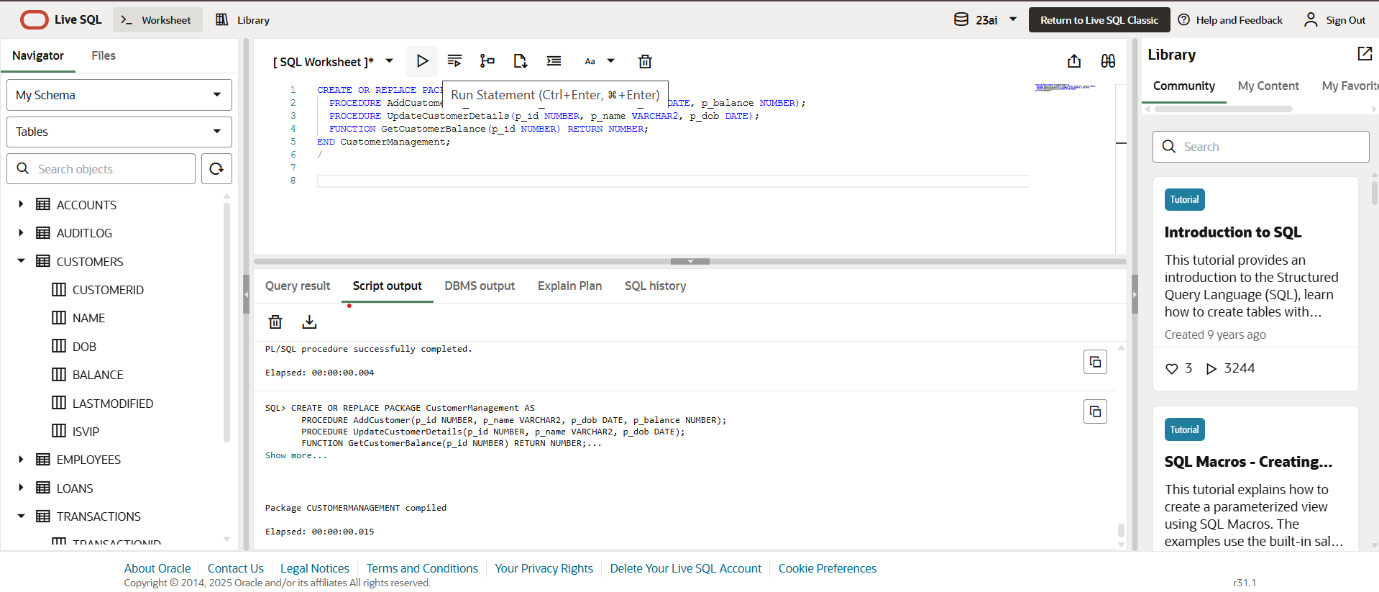
CustomerManagement.UpdateCustomerDetails(10, 'Charlie Green', TO\_DATE('1981-01-01', 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(10));

END;

/

**Outputs:**



**Scenario 2:**

*Hiring*

DECLARE

v\_name HR.EMPLOYEES.FIRST\_NAME%TYPE;

v\_job HR.EMPLOYEES.JOB\_ID%TYPE;

v\_salary HR.EMPLOYEES.SALARY%TYPE;

v\_dept HR.EMPLOYEES.DEPARTMENT\_ID%TYPE;

v\_hire HR.EMPLOYEES.HIRE\_DATE%TYPE;

BEGIN

SELECT FIRST\_NAME, JOB\_ID, SALARY, DEPARTMENT\_ID, HIRE\_DATE

INTO v\_name, v\_job, v\_salary, v\_dept, v\_hire

FROM HR.EMPLOYEES

WHERE EMPLOYEE\_ID = 101;

DBMS\_OUTPUT.PUT\_LINE('Simulated Hire: ' || v\_name || ' | Job: ' || v\_job || ' | Dept: ' || v\_dept || ' | Salary: ' || v\_salary);

END;

/

*Update*

DECLARE

v\_old\_salary HR.EMPLOYEES.SALARY%TYPE;

v\_new\_salary NUMBER;

v\_old\_dept HR.EMPLOYEES.DEPARTMENT\_ID%TYPE;

v\_new\_dept NUMBER := 90;

BEGIN

SELECT SALARY, DEPARTMENT\_ID

INTO v\_old\_salary, v\_old\_dept

FROM HR.EMPLOYEES

WHERE EMPLOYEE\_ID = 101;

v\_new\_salary := v\_old\_salary + 1000;

DBMS\_OUTPUT.PUT\_LINE('Simulated Update: New Salary = ' || v\_new\_salary || ', New Department = ' || v\_new\_dept);

END;

/

*Actual Annual Salary Calculation*

DECLARE

v\_salary HR.EMPLOYEES.SALARY%TYPE; -- declares variable with correct type

v\_annual NUMBER; -- declares another variable

BEGIN

SELECT SALARY INTO v\_salary

FROM HR.EMPLOYEES

WHERE EMPLOYEE\_ID = 101;

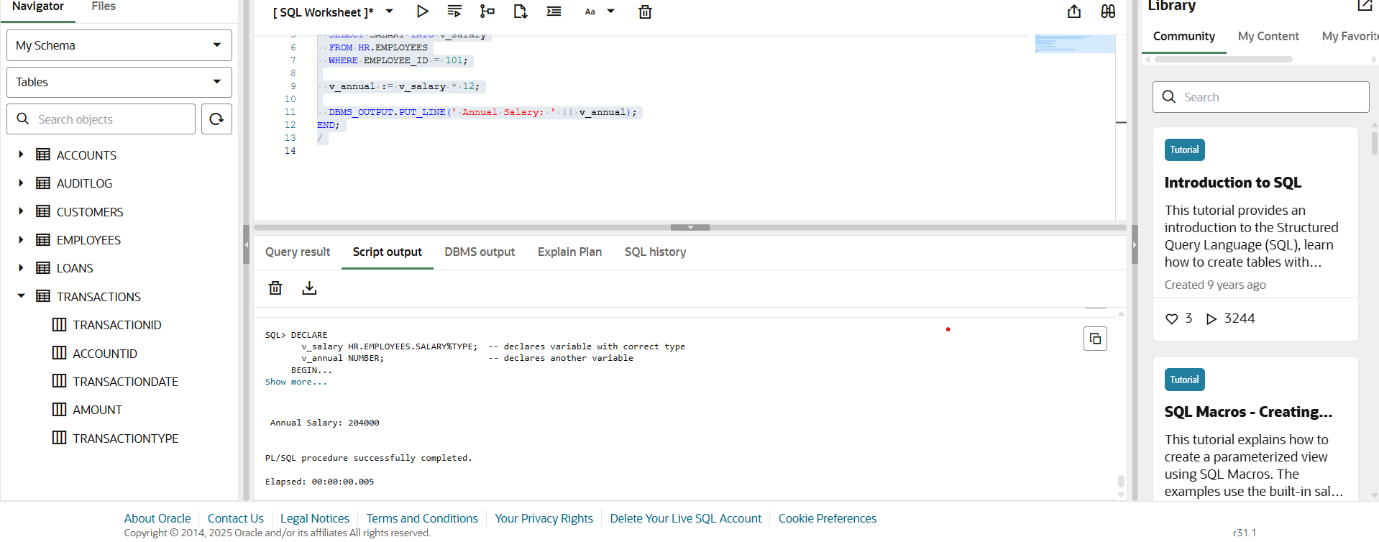
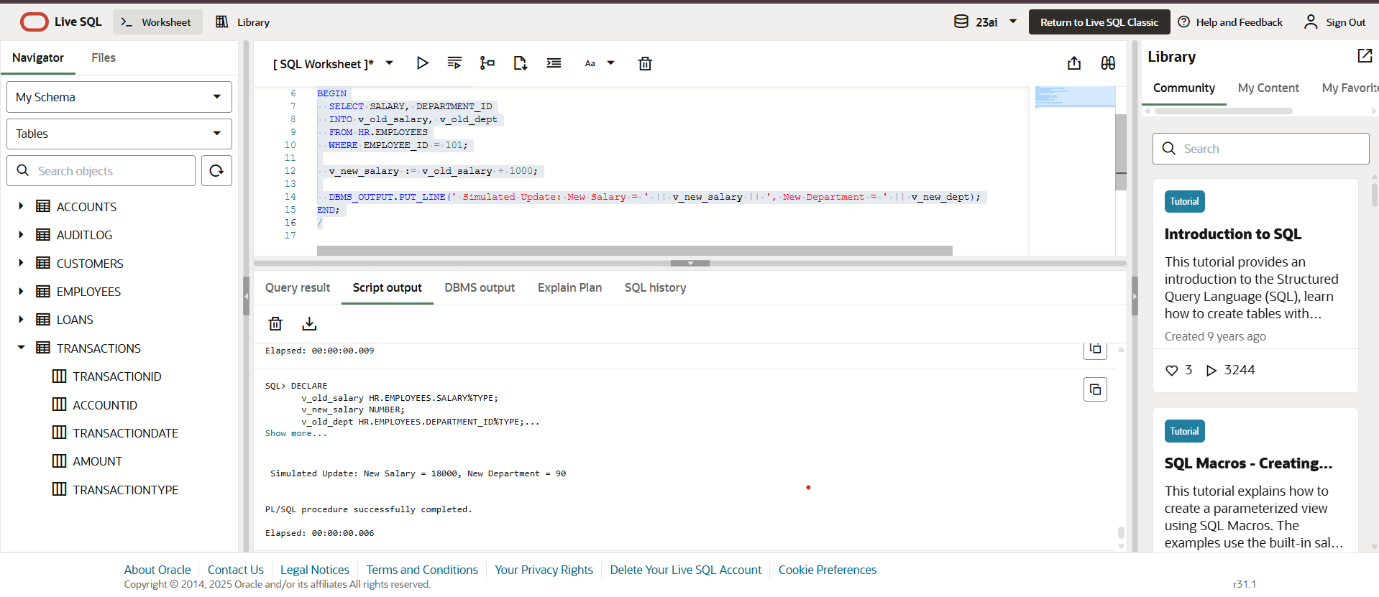
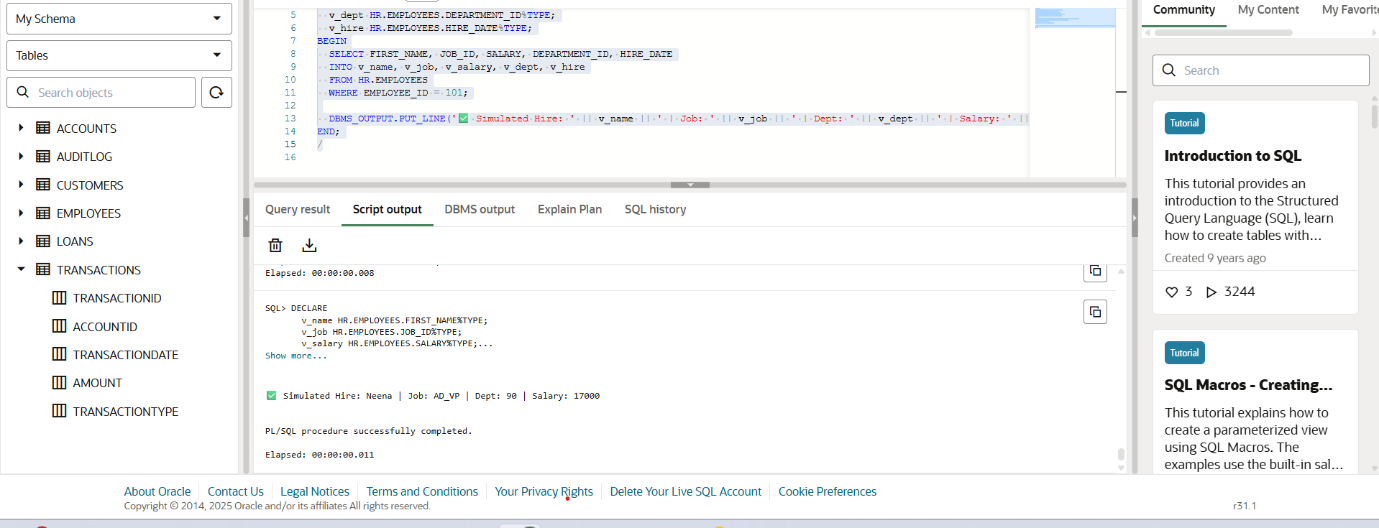
v\_annual := v\_salary \* 12;

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || v\_annual);

END;

/

**Outputs**



**Scenario 3:**

*Package Specification*

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

*Package Body*

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

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

VALUES (p\_acc\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

END;

FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

total NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO total FROM Accounts WHERE CustomerID = p\_cust\_id;

RETURN total;

END;

END AccountOperations;

/

**Test Query :**

BEGIN

AccountOperations.OpenAccount(20, 10, 'Savings', 800);

AccountOperations.OpenAccount(21, 10, 'Checking', 1200);

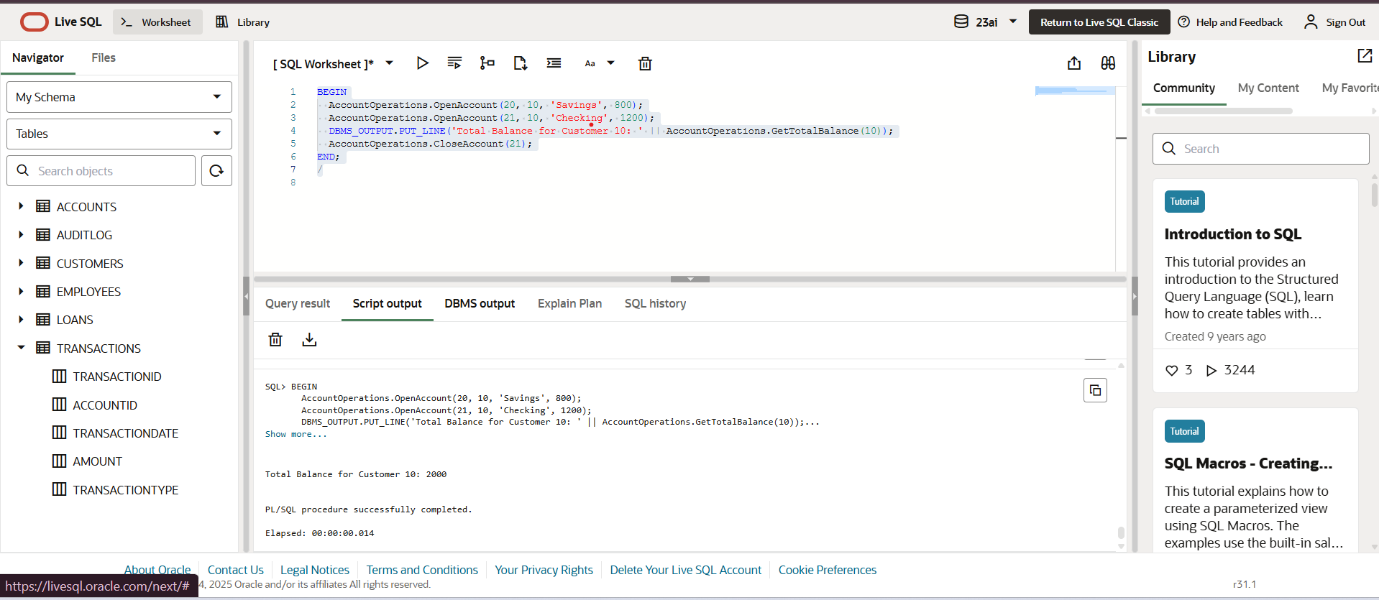
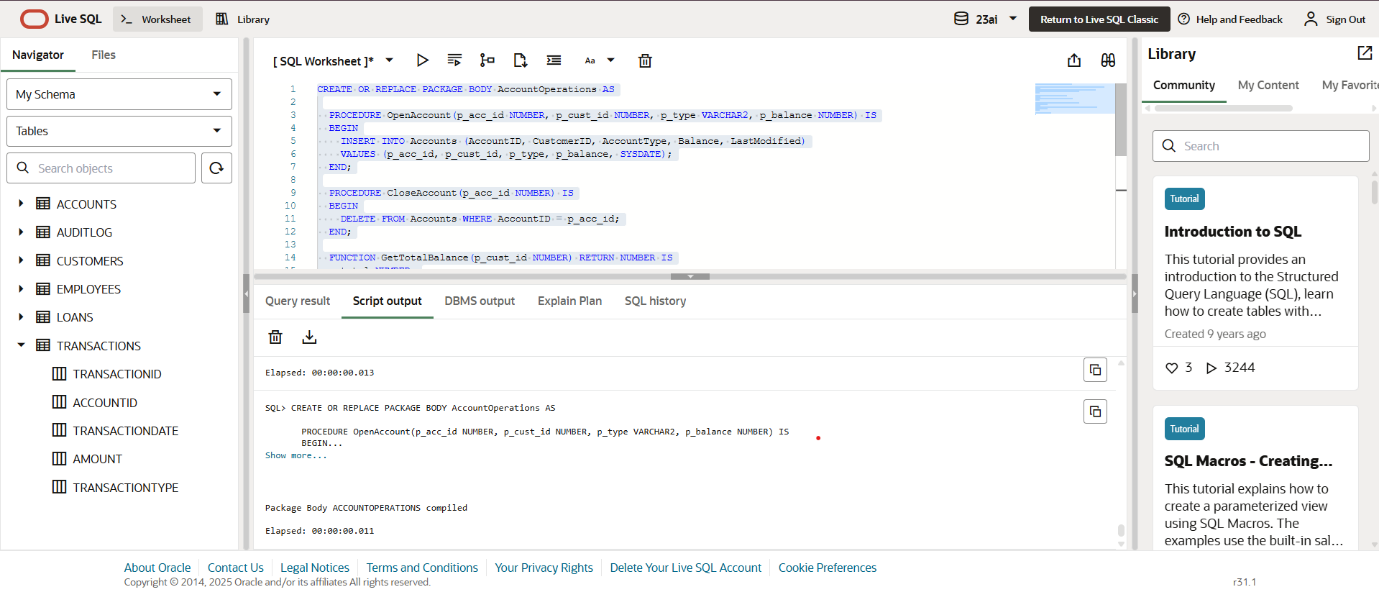
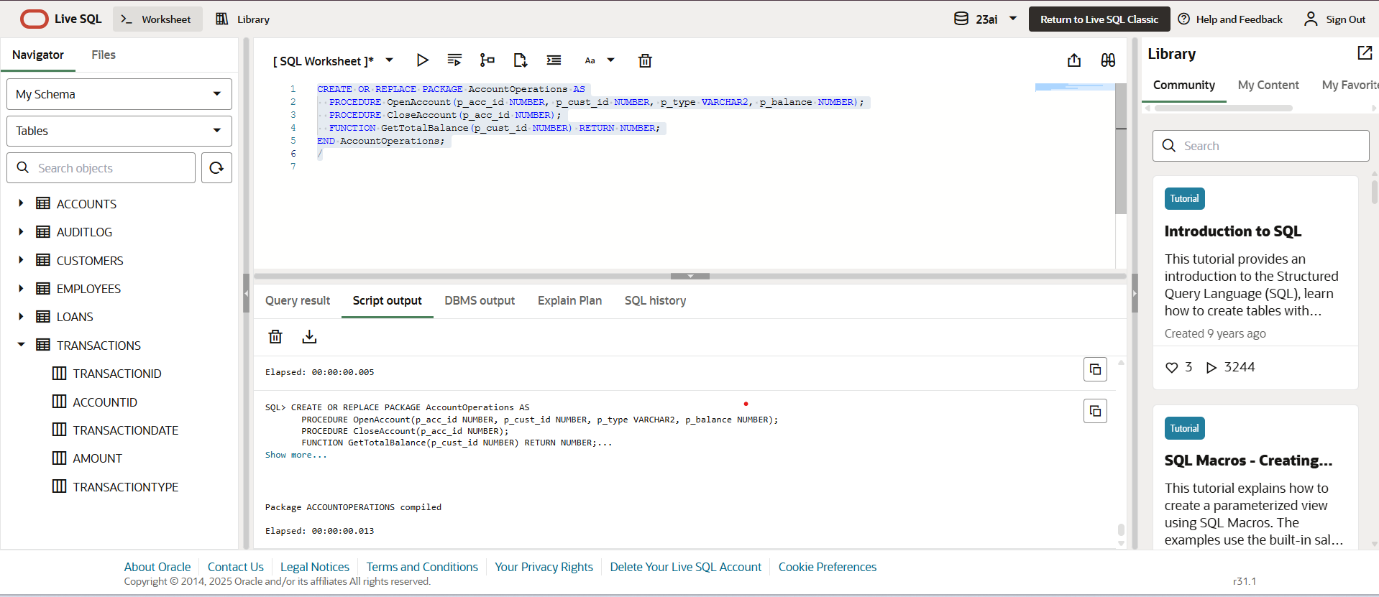
DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 10: ' || AccountOperations.GetTotalBalance(10));

AccountOperations.CloseAccount(21);

END;

/

**Outputs:**

****