**WEEK 2 : PL/SQL PROGRAMMING**

***Exercise 1 Control Structures***

**Scenario 1: Apply a 1% discount to loan interest rates for customers above 60**

CODE

BEGIN

FOR rec IN (

SELECT l.LoanID, l.InterestRate, c.DOB

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE FLOOR(MONTHS\_BETWEEN(SYSDATE, c.DOB)/12) > 60

) LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

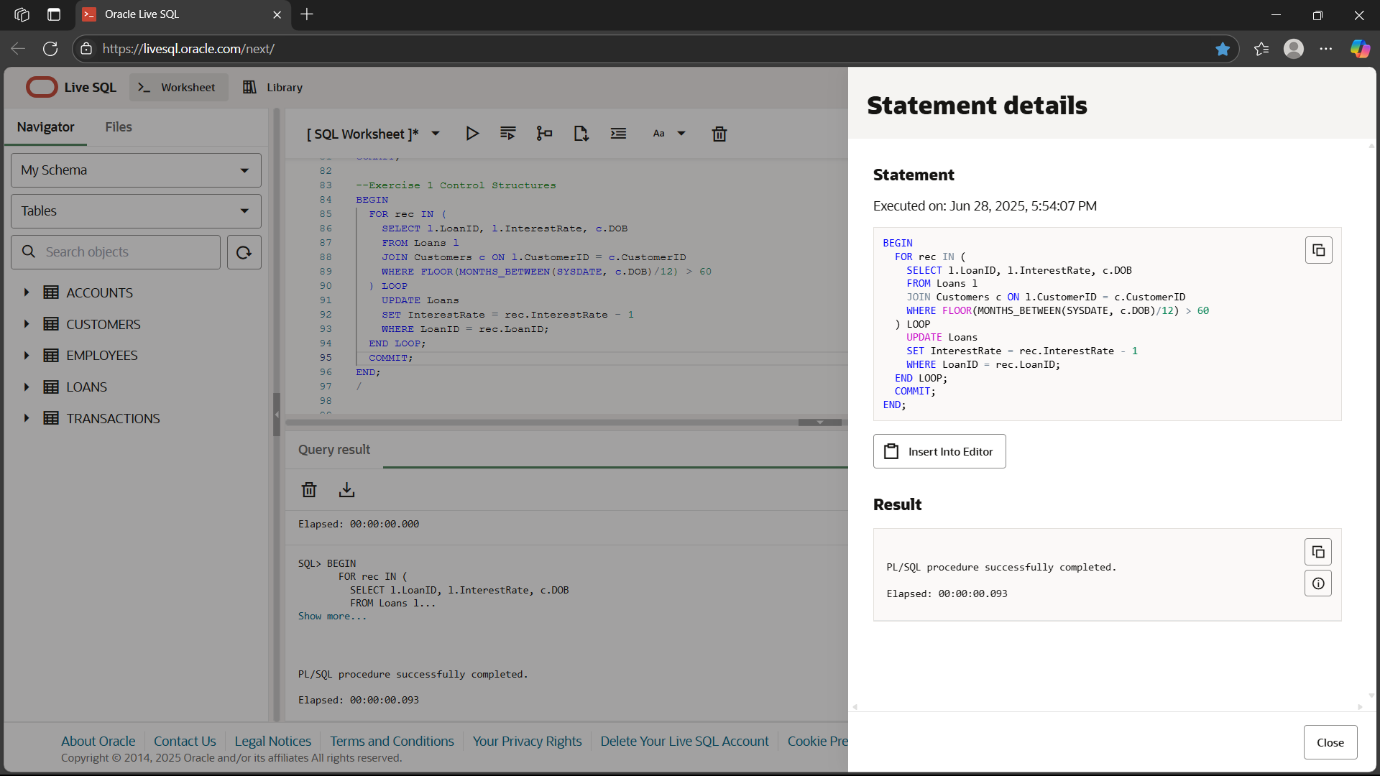
END LOOP;

COMMIT;

END;

/

**OUTPUT:**



**Scenario 2: Promote customers with balance over $10,000 to VIP**

ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N';

BEGIN

FOR rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

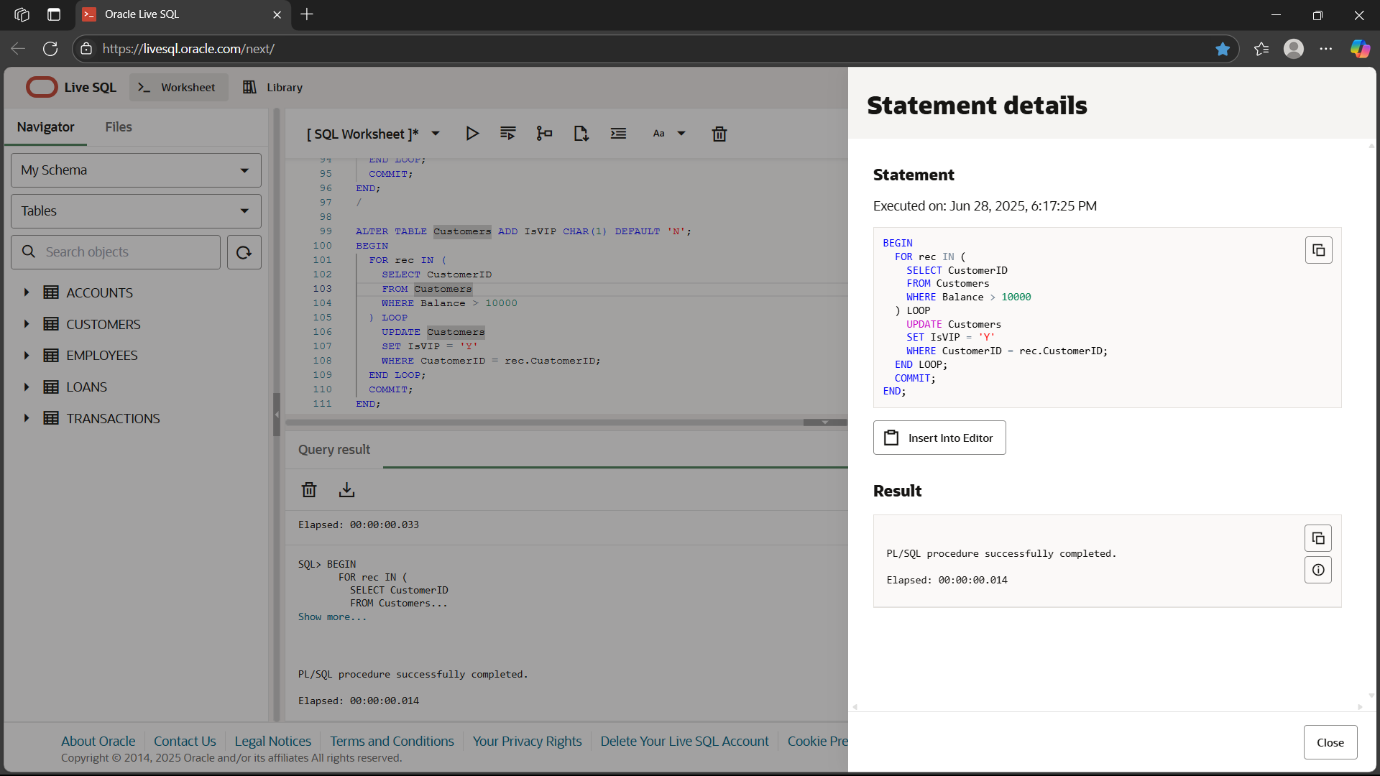
END LOOP;

COMMIT;

END;

/

**OUTPUT:**

****

**Scenario 3: Print loan reminders for loans due in next 30 days**

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || rec.Name ||

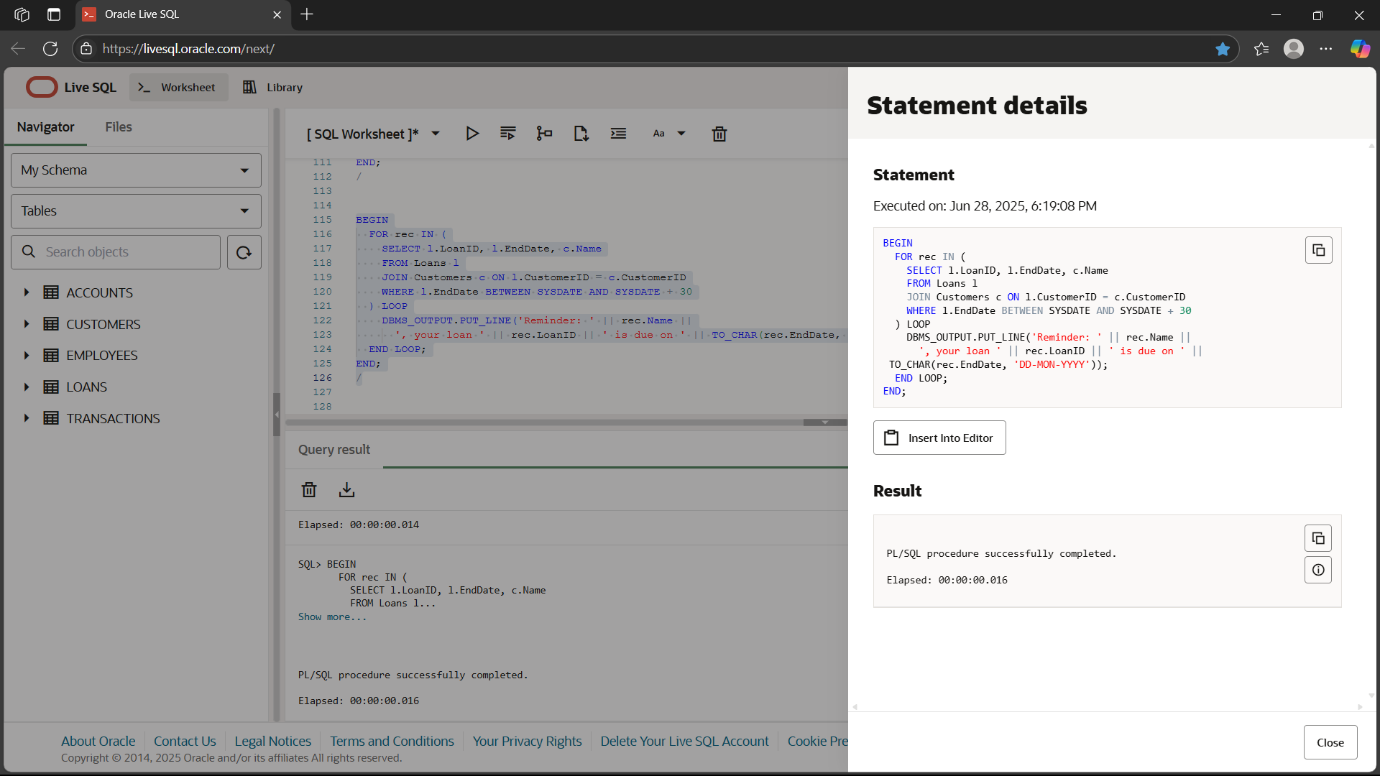
', your loan ' || rec.LoanID || ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**OUTPUT:**

****

***EXERCISE 2 : ERROR HANDLING***

**Scenario 1: Safe fund transfer between accounts**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

)

IS

insufficient\_funds EXCEPTION;

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE insufficient\_funds;

ELSE

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful');

END IF;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: insufficient funds');

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

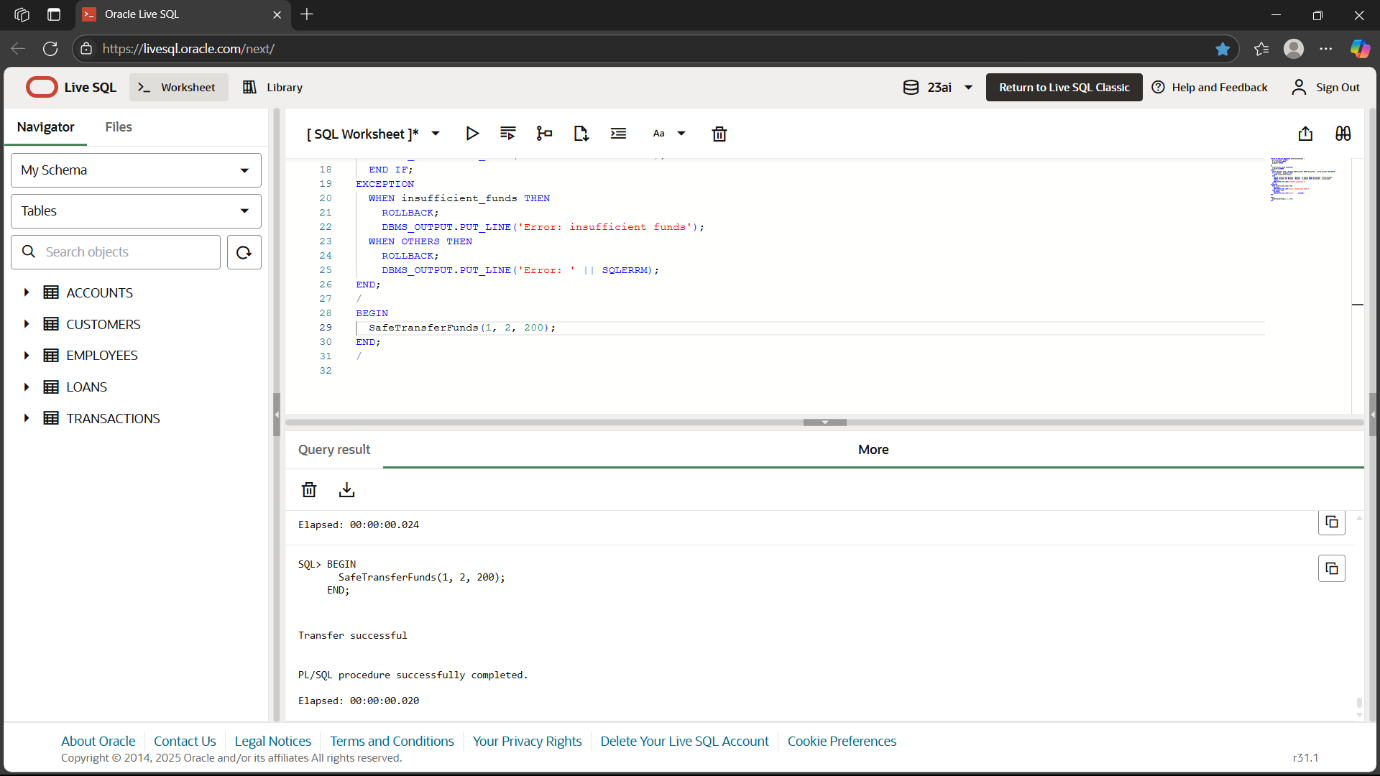
BEGIN

SafeTransferFunds(1, 2, 200);

END;

/

**OUTPUT:**

****

**Scenario 2: Update salary with error handling**

CREATE OR REPLACE PROCEDURE UpdateSalary (

  p\_empid NUMBER,

  p\_pct NUMBER

)

IS

  no\_employee EXCEPTION;

  v\_salary NUMBER;

BEGIN

  SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_empid;

  UPDATE Employees

  SET Salary = v\_salary + (v\_salary \* p\_pct/100)

  WHERE EmployeeID = p\_empid;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Salary updated');

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found');

  WHEN OTHERS THEN

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

END;

/

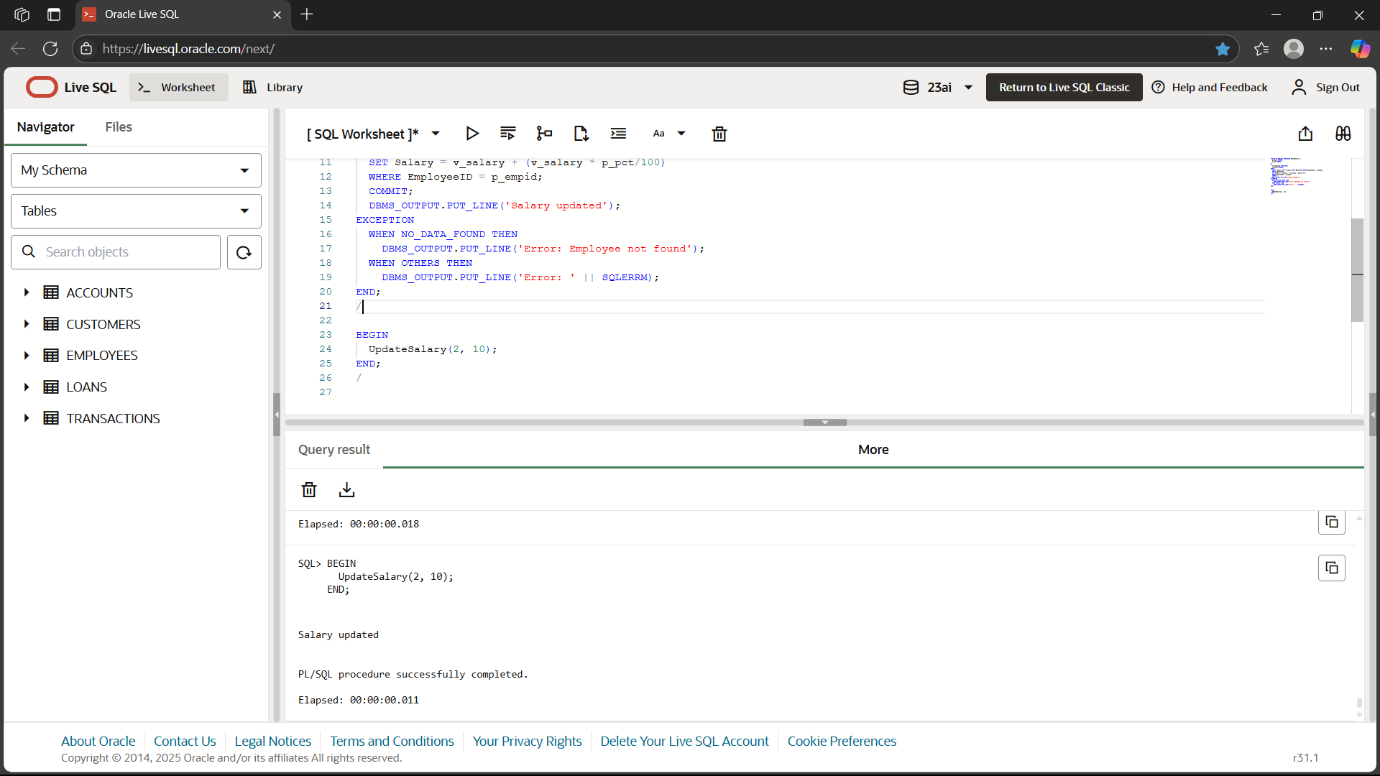
BEGIN

  UpdateSalary(2, 10);

END;

/

**OUTPUT:**

****

**Scenario 3: Add new customer with duplicate error handling**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

  p\_custid NUMBER,

  p\_name VARCHAR2,

  p\_dob DATE,

  p\_balance NUMBER

)

IS

  duplicate\_id EXCEPTION;

BEGIN

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

  VALUES (p\_custid, p\_name, p\_dob, p\_balance, SYSDATE, 'N');

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Customer added');

EXCEPTION

  WHEN DUP\_VAL\_ON\_INDEX THEN

    DBMS\_OUTPUT.PUT\_LINE('Error: Customer with same ID already exists');

  WHEN OTHERS THEN

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

END;

/

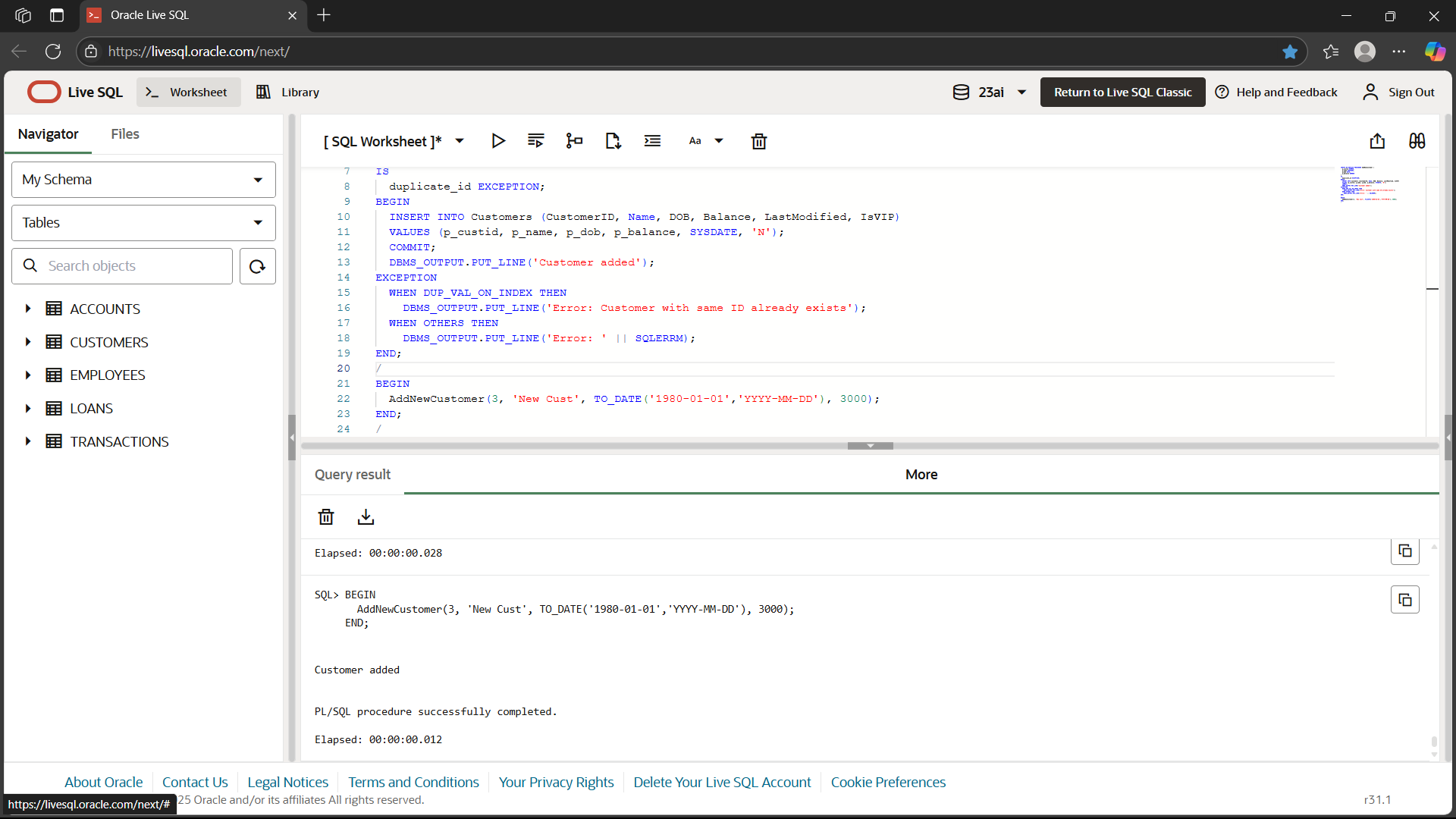
BEGIN

  AddNewCustomer(3, 'New Cust', TO\_DATE('1980-01-01','YYYY-MM-DD'), 3000);

END;

/

**OUTPUT:**

****

***EXERCISE 3 : STORED PROCEDURES***

**Scenario 1: ProcessMonthlyInterest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest processed');

END;

/

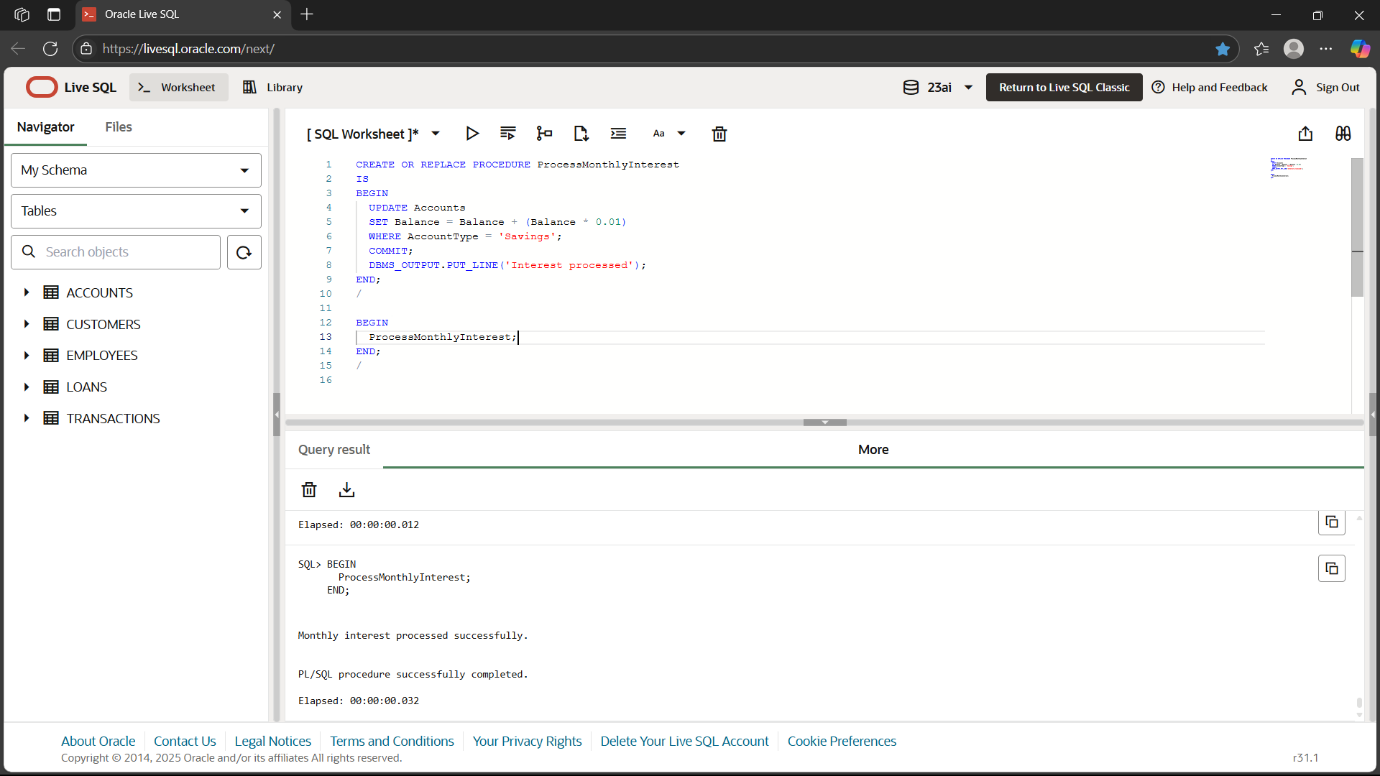
BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT:**

****

**Scenario 2: UpdateEmployeeBonus**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept VARCHAR2,

p\_bonus\_pct NUMBER

)

IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_pct/100)

WHERE Department = p\_dept;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || p\_dept);

END;

/

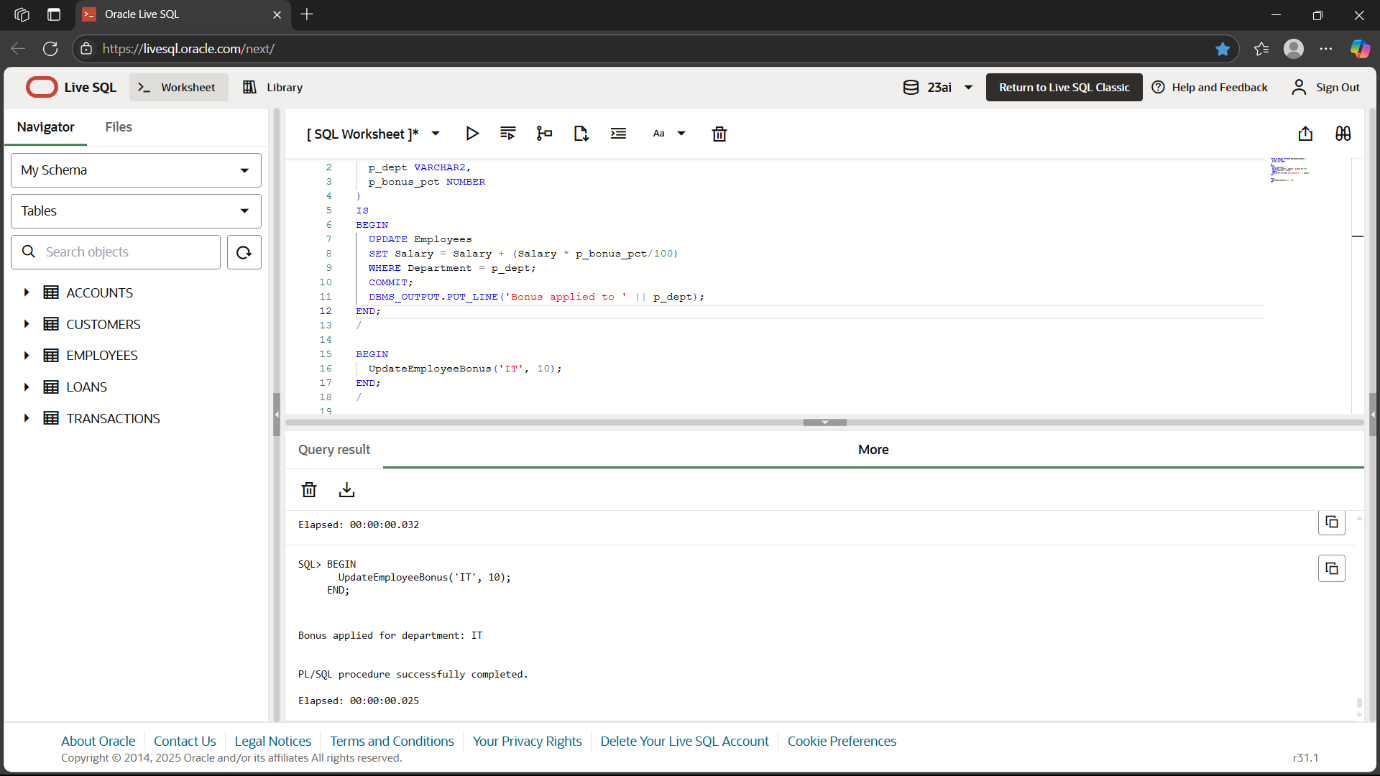
BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

**OUTPUT:**



**Scenario 3: TransferFunds**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from NUMBER,

p\_to NUMBER,

p\_amt NUMBER

)

IS

v\_bal NUMBER;

BEGIN

SELECT Balance INTO v\_bal FROM Accounts WHERE AccountID = p\_from FOR UPDATE;

IF v\_bal >= p\_amt THEN

UPDATE Accounts SET Balance = Balance - p\_amt WHERE AccountID = p\_from;

UPDATE Accounts SET Balance = Balance + p\_amt WHERE AccountID = p\_to;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Error: insufficient funds');

END IF;

END;

/

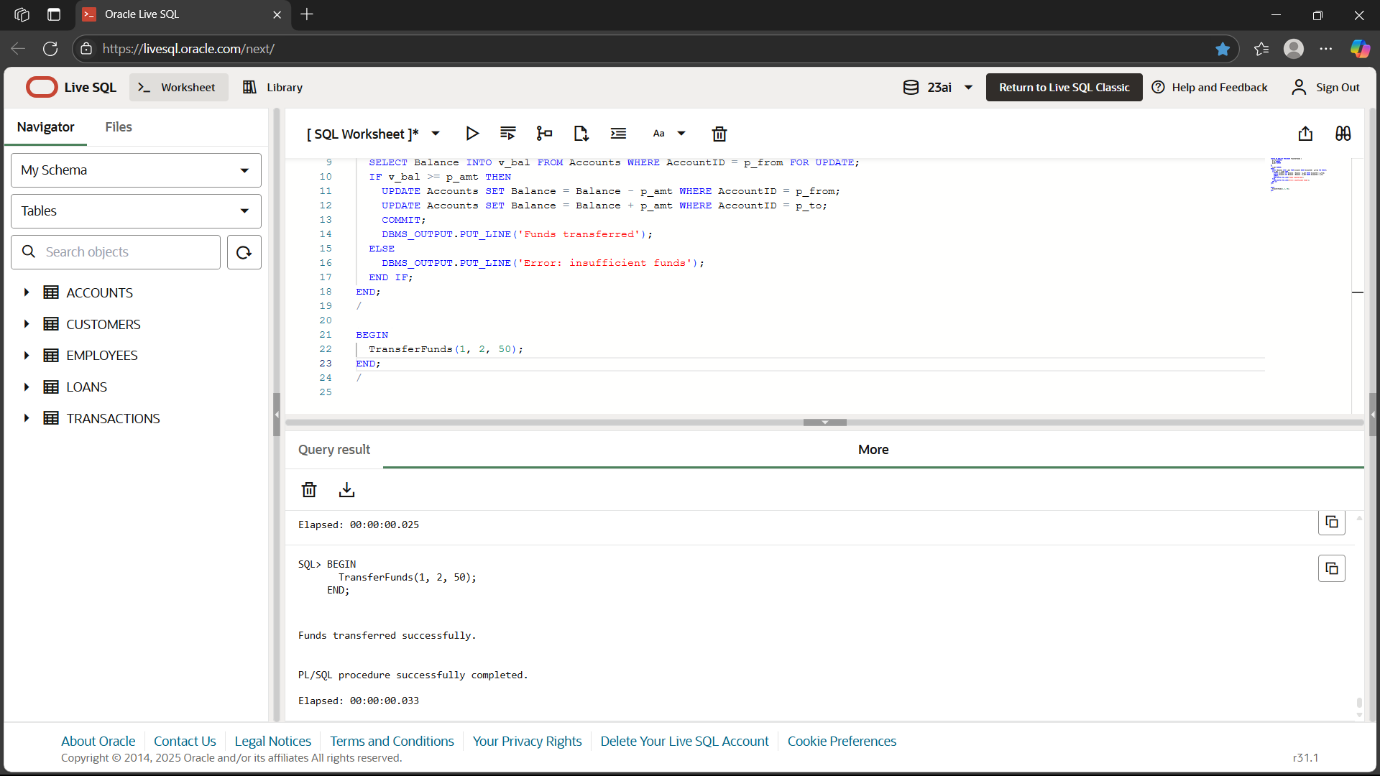
BEGIN

TransferFunds(1, 2, 50);

END;

/

**OUTPUT:**



***EXERCISE 4 : FUNCTIONS***

**Scenario 1: CalculateAge**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER

IS

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);

END;

/

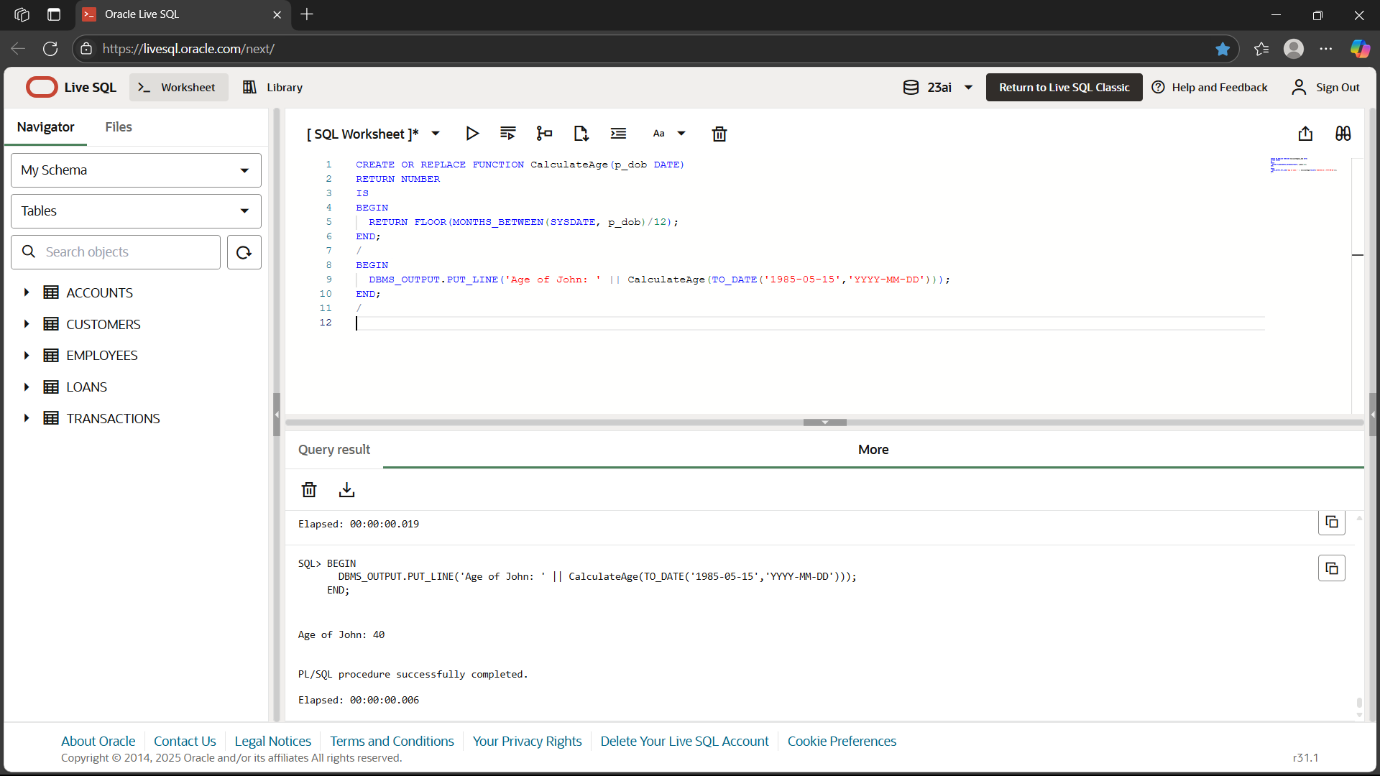
BEGIN

DBMS\_OUTPUT.PUT\_LINE('Age of John: ' || CalculateAge(TO\_DATE('1985-05-15','YYYY-MM-DD')));

END;

/

**OUTPUT:**



**Scenario 2: CalculateMonthlyInstallment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

  p\_loanamt NUMBER,

  p\_rate NUMBER,

  p\_years NUMBER

)

RETURN NUMBER

IS

  r NUMBER;

  n NUMBER;

BEGIN

  r := p\_rate/100/12;

  n := p\_years\*12;

  RETURN (p\_loanamt \* r)/(1 - POWER(1+r,-n));

END;

/

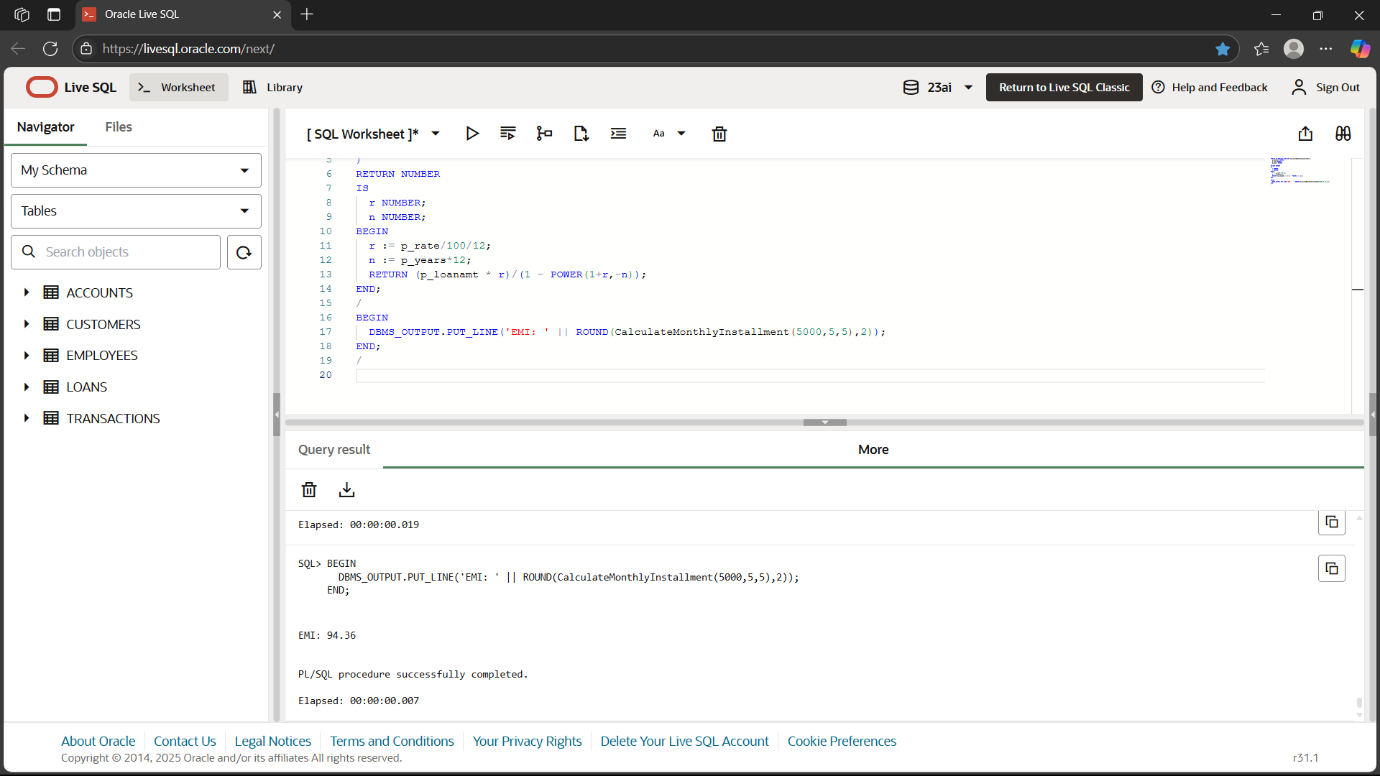
BEGIN

  DBMS\_OUTPUT.PUT\_LINE('EMI: ' || ROUND(CalculateMonthlyInstallment(5000,5,5),2));

END;

/

**OUTPUT:**



**Scenario 3: HasSufficientBalance**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

  p\_accid NUMBER,

  p\_amt NUMBER

)

RETURN BOOLEAN

IS

  v NUMBER;

BEGIN

  SELECT Balance INTO v FROM Accounts WHERE AccountID = p\_accid;

  RETURN v >= p\_amt;

END;

/

DECLARE

  result BOOLEAN;

BEGIN

  result := HasSufficientBalance(1, 200);

  IF result THEN

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

  ELSE

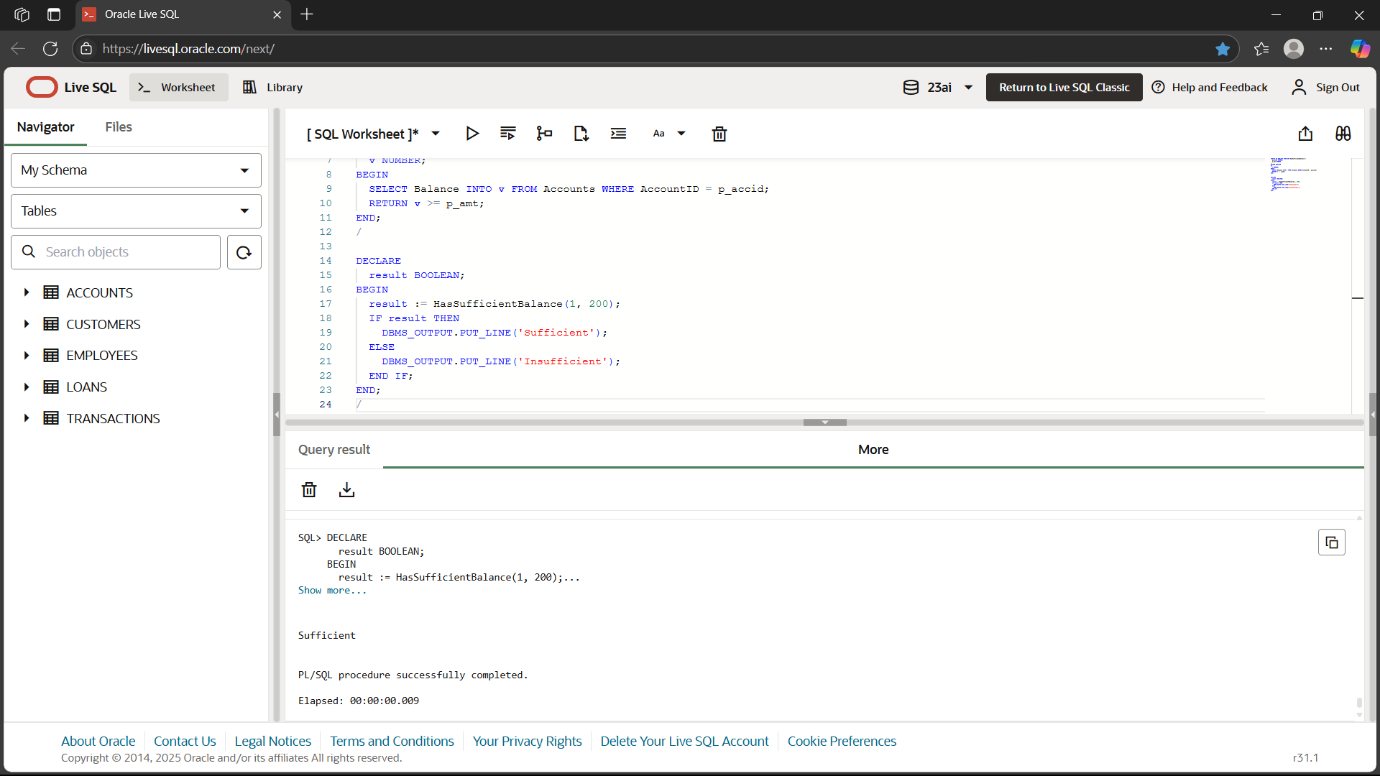
    DBMS\_OUTPUT.PUT\_LINE('Insufficient');

  END IF;

END;

/

**OUTPUT:**



***EXERCISE 5 : TRIGGERS***

**Scenario 1: UpdateCustomerLastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

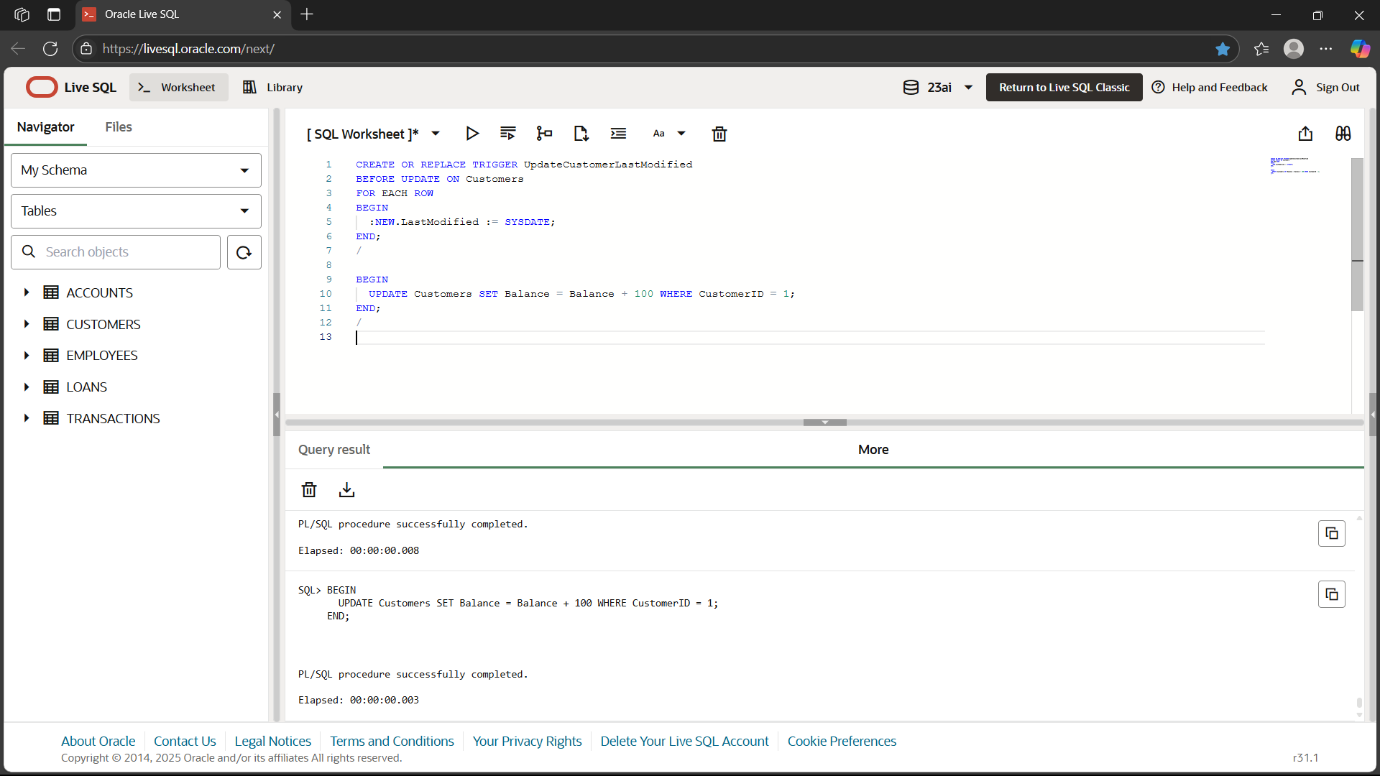
BEGIN

UPDATE Customers SET Balance = Balance + 100 WHERE CustomerID = 1;

END;

/

**OUTPUT:**



**Scenario 2: LogTransaction**

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Message VARCHAR2(200)

);

CREATE SEQUENCE AuditLog\_seq START WITH 1;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog

VALUES (AuditLog\_seq.NEXTVAL, :NEW.TransactionID, SYSDATE, 'Transaction inserted');

END;

/

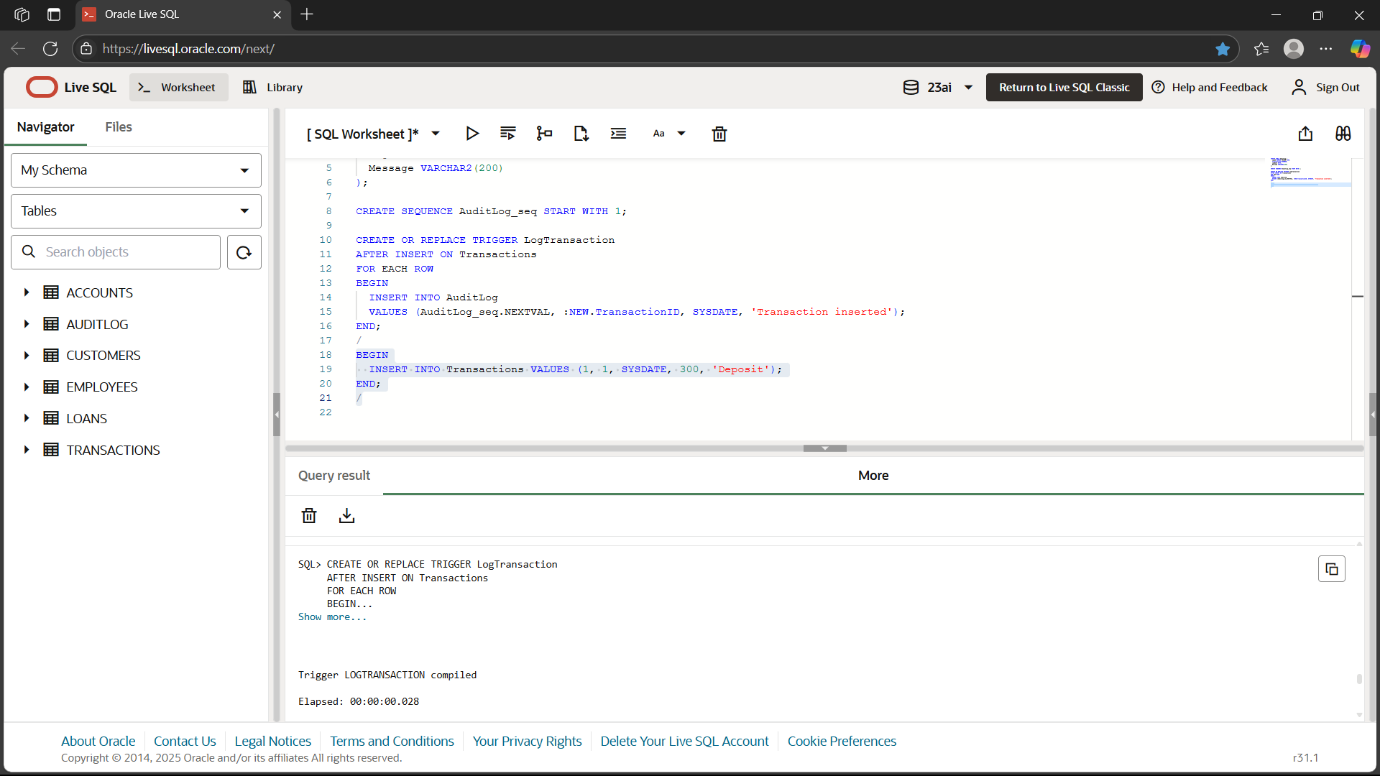
BEGIN

INSERT INTO Transactions VALUES (3, 1, SYSDATE, 300, 'Deposit');

END;

/

**OUTPUT:**



**Scenario 3: CheckTransactionRules**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.Amount > v THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Withdrawal exceeds available balance');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit must be positive');

END IF;

END IF;

END;

/

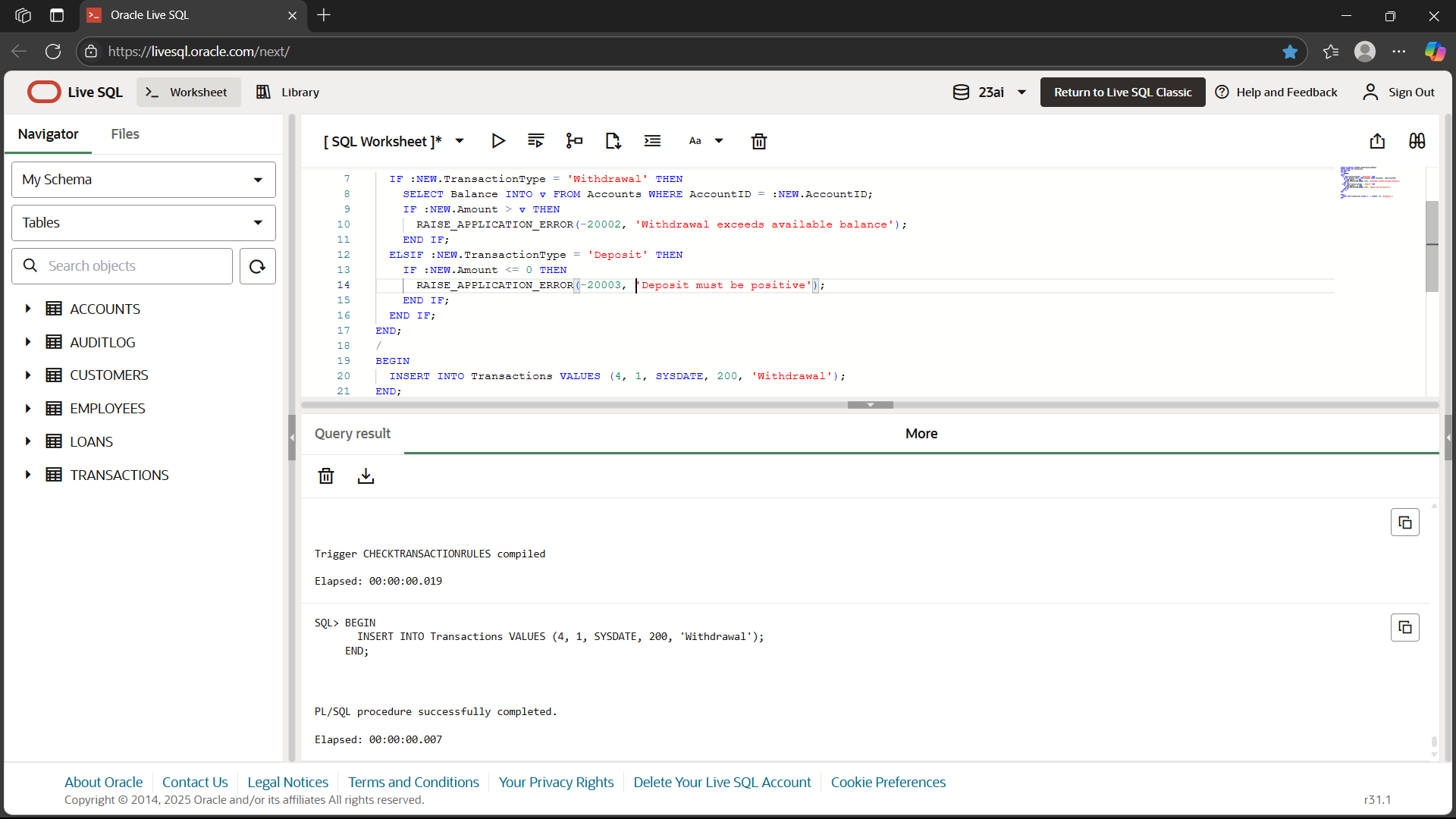
BEGIN

INSERT INTO Transactions VALUES (4, 1, SYSDATE, 50000, 'Withdrawal');

END;

/

**OUTPUT:**



***EXERCISE 6 : CURSORS***

**Scenario 1: GenerateMonthlyStatements**

DECLARE

CURSOR c IS

SELECT c.Name, t.Amount, t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

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

BEGIN

FOR rec IN c LOOP

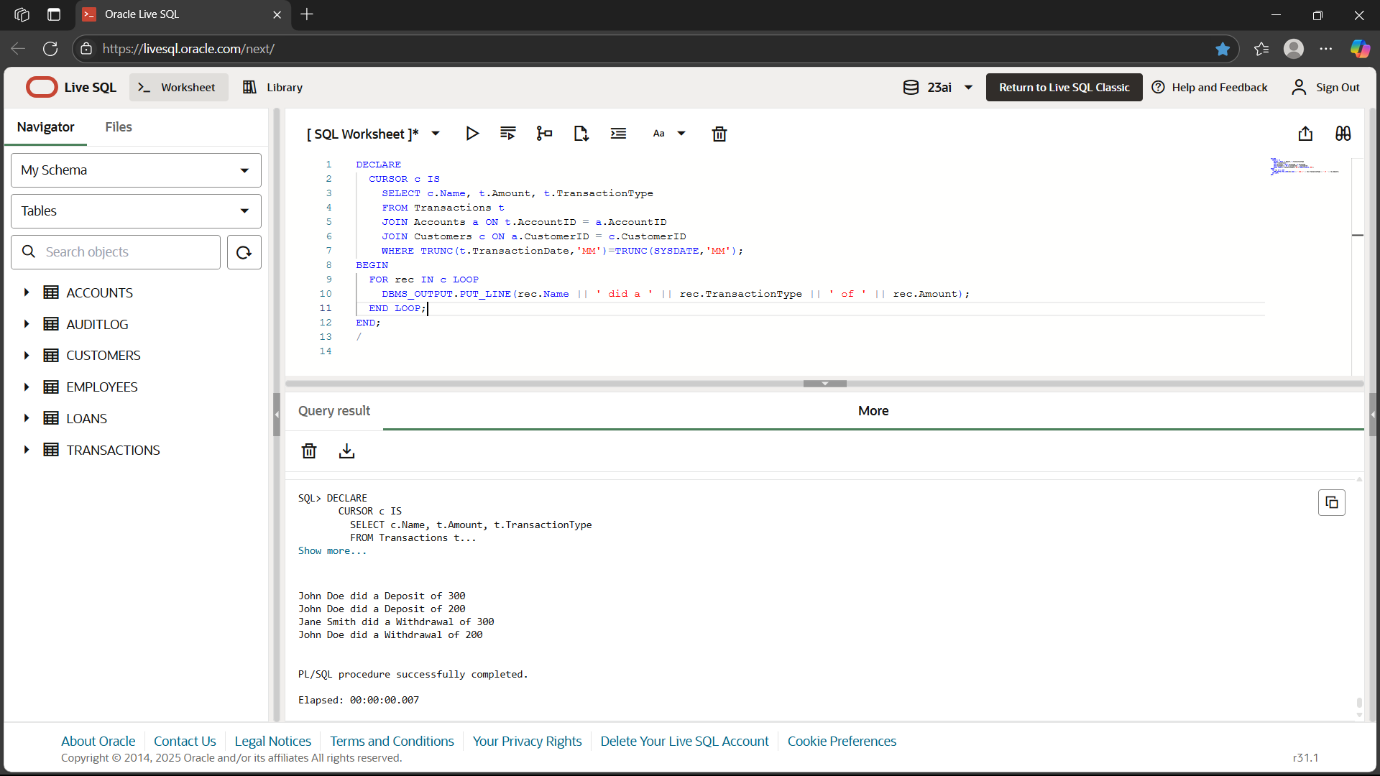
DBMS\_OUTPUT.PUT\_LINE(rec.Name || ' did a ' || rec.TransactionType || ' of ' || rec.Amount);

END LOOP;

END;

/

**OUTPUT:**



**Scenario 2: ApplyAnnualFee**

DECLARE

CURSOR c IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR rec IN c LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Annual fee applied to account ' || rec.AccountID);

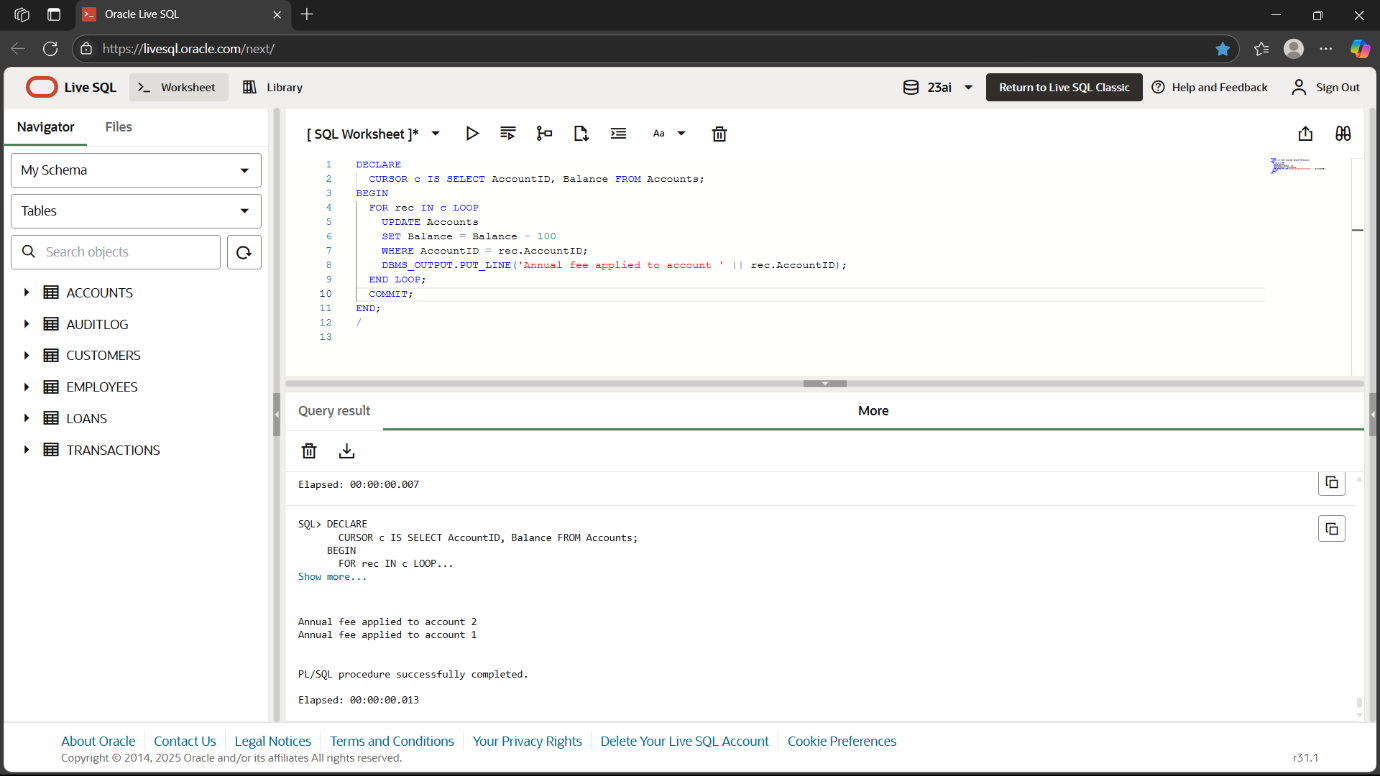
END LOOP;

COMMIT;

END;

/

**OUTPUT:**



**Scenario 3: UpdateLoanInterestRates**

DECLARE

CURSOR c IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR rec IN c LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate + 0.5

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Interest updated for Loan ' || rec.LoanID);

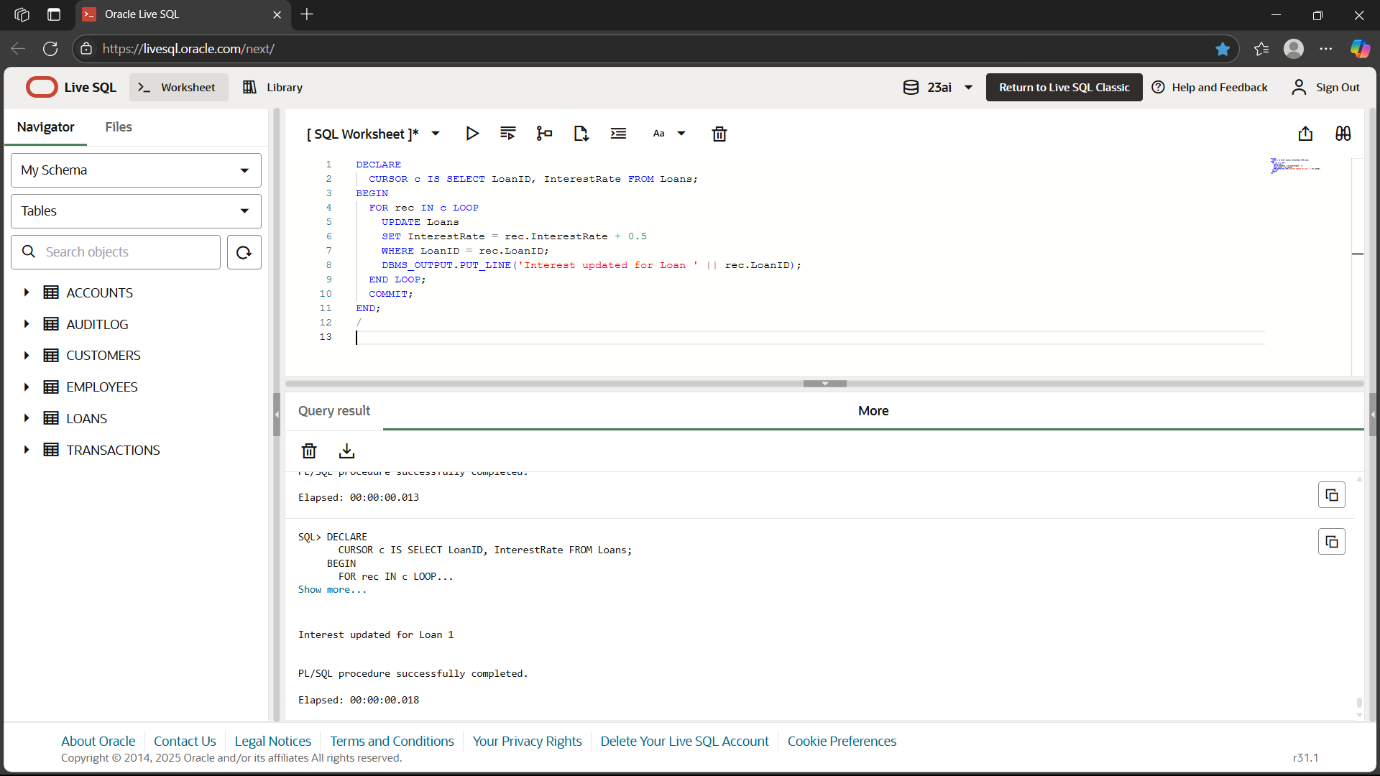
END LOOP;

COMMIT;

END;

/

**OUTPUT:**



***EXERCISE 7 : PACKAGES***

**Scenario 1: CustomerManagement**

-- PACKAGE

DROP PACKAGE CustomerManagement-- PACKAGE

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

  PROCEDURE UpdateCustomer(p\_custid NUMBER, p\_name VARCHAR2);

  FUNCTION GetCustomerBalance(p\_custid NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

  IS

  BEGIN

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

    VALUES (p\_custid, p\_name, p\_dob, p\_balance, SYSDATE, 'N');

  END;

  PROCEDURE UpdateCustomer(p\_custid NUMBER, p\_name VARCHAR2)

  IS

  BEGIN

    UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_custid;

  END;

  FUNCTION GetCustomerBalance(p\_custid NUMBER)

  RETURN NUMBER

  IS

    v\_balance NUMBER;

  BEGIN

    SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_custid;

    RETURN v\_balance;

  END;

END CustomerManagement;

/

 BEGIN

  CustomerManagement.AddCustomer(3, 'Tom Parker', TO\_DATE('1970-04-10','YYYY-MM-DD'), 5000);

  DBMS\_OUTPUT.PUT\_LINE('Added Tom Parker')

  CustomerManagement.UpdateCustomer(3, 'Thomas Parker');

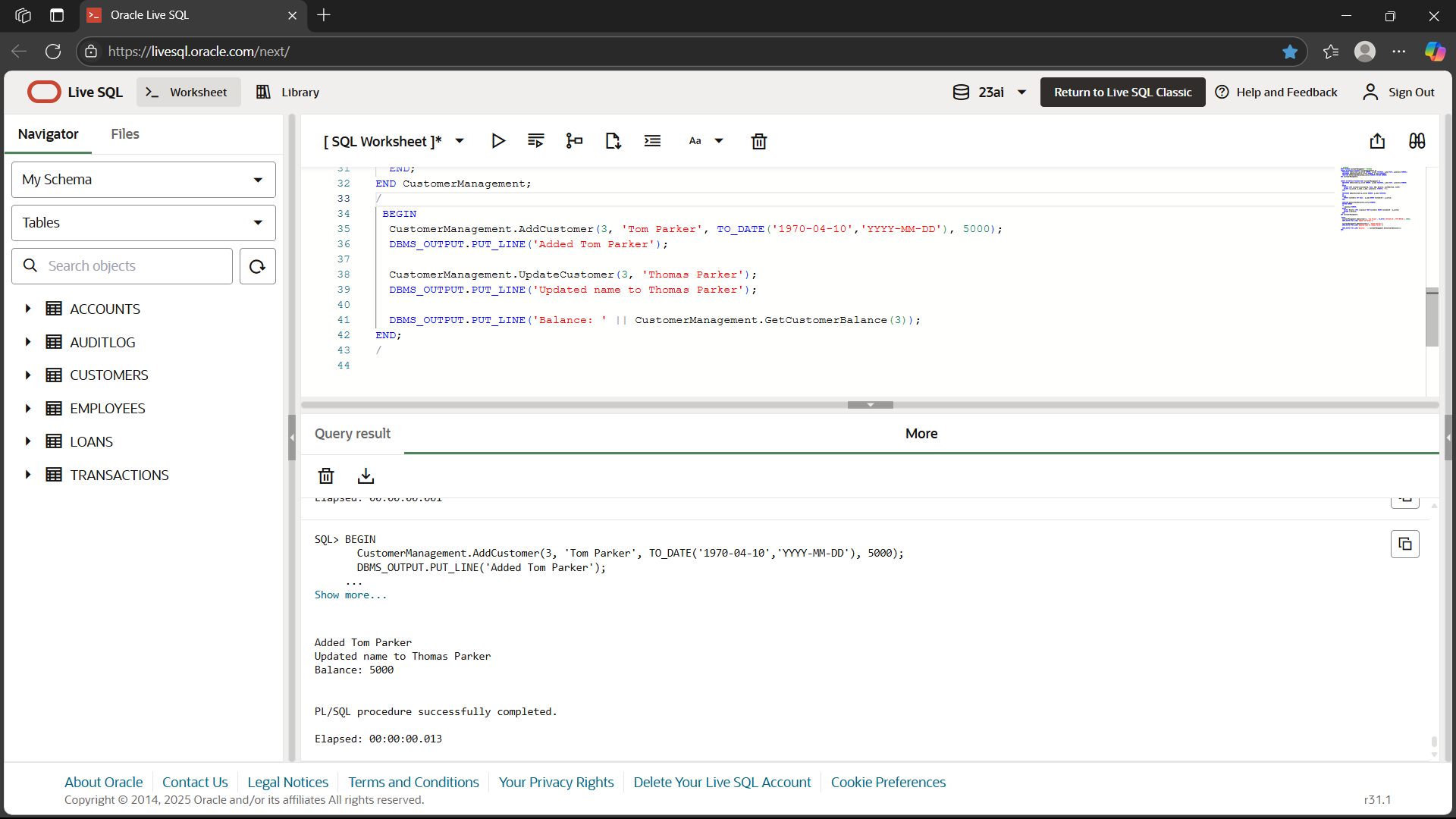
  DBMS\_OUTPUT.PUT\_LINE('Updated name to Thomas Parker');

  DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(3));

END;

/

**OUTPUT:**



**Scenario 2: EmployeeManagement**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

  PROCEDURE HireEmployee(p\_empid NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_hire DATE);

  PROCEDURE UpdateEmployee(p\_empid NUMBER, p\_salary NUMBER);

  FUNCTION GetAnnualSalary(p\_empid NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(p\_empid NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_hire DATE)

  IS

  BEGIN

    INSERT INTO Employees VALUES (p\_empid, p\_name, p\_pos, p\_sal, p\_dept, p\_hire);

  END;

  PROCEDURE UpdateEmployee(p\_empid NUMBER, p\_salary NUMBER)

  IS

  BEGIN

    UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_empid;

  END;

  FUNCTION GetAnnualSalary(p\_empid NUMBER)

  RETURN NUMBER

  IS

    v\_salary NUMBER;

  BEGIN

    SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_empid;

    RETURN v\_salary \* 12;

  END;

END EmployeeManagement;

/

BEGIN

  EmployeeManagement.HireEmployee(3, 'George Wilson', 'Analyst', 50000, 'Finance', SYSDATE);

  DBMS\_OUTPUT.PUT\_LINE('George Wilson hired');

  EmployeeManagement.UpdateEmployee(3, 55000);

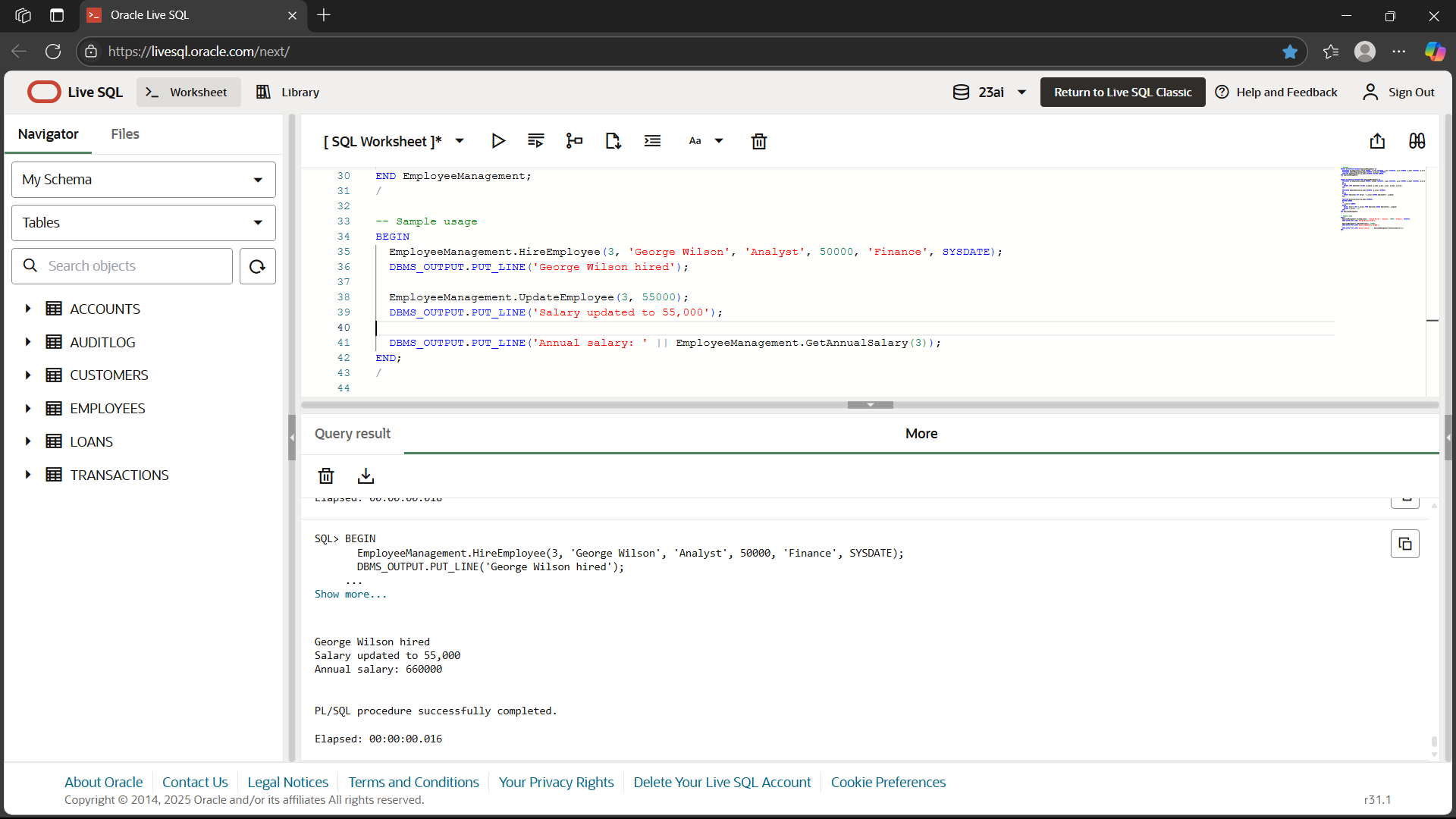
  DBMS\_OUTPUT.PUT\_LINE('Salary updated to 55,000');

  DBMS\_OUTPUT.PUT\_LINE('Annual salary: ' || EmployeeManagement.GetAnnualSalary(3));

END;

/

**OUTPUT:**



**Scenario 3: AccountOperations**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_accid NUMBER);

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER)

IS

BEGIN

INSERT INTO Accounts VALUES (p\_accid, p\_custid, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_accid NUMBER)

IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_accid;

END;

FUNCTION GetTotalBalance(p\_custid NUMBER)

RETURN NUMBER

IS

v\_total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;

RETURN v\_total;

END;

END AccountOperations;

/

BEGIN

AccountOperations.OpenAccount(3, 1, 'Savings', 3000);

DBMS\_OUTPUT.PUT\_LINE('Account 3 opened for John Doe');

DBMS\_OUTPUT.PUT\_LINE('Total balance for John Doe: ' || AccountOperations.GetTotalBalance(1));

AccountOperations.CloseAccount(3);

DBMS\_OUTPUT.PUT\_LINE('Account 3 closed');

END;

/

**OUTPUT:**

