**Exercise 1: Control Structures**

-- STEP 1: Create Tables

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

balance NUMBER,

interest\_rate NUMBER(5,2),

IsVIP VARCHAR2(5)

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

due\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

/

-- STEP 2: Insert Sample Data

INSERT INTO customers VALUES (1, 'John Doe', 65, 12000, 8.5, 'FALSE');

INSERT INTO customers VALUES (2, 'Jane Smith', 45, 9000, 7.5, 'FALSE');

INSERT INTO customers VALUES (3, 'Mary Brown', 70, 15000, 9.0, 'FALSE');

INSERT INTO loans VALUES (101, 1, SYSDATE + 10); -- Due in 10 days

INSERT INTO loans VALUES (102, 2, SYSDATE + 40); -- Due in 40 days

INSERT INTO loans VALUES (103, 3, SYSDATE + 5); -- Due in 5 days

COMMIT;

/

-- STEP 3: Scenario 1 - Apply 1% Discount for Customers Over 60

BEGIN

FOR cust\_rec IN (SELECT customer\_id, interest\_rate, age FROM customers) LOOP

IF cust\_rec.age > 60 THEN

UPDATE customers

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = cust\_rec.customer\_id;

END IF;

END LOOP;

COMMIT;

END;

/

-- STEP 4: Scenario 2 - Promote Customers to VIP Based on Balance

BEGIN

FOR bal\_rec IN (SELECT customer\_id, balance FROM customers) LOOP

IF bal\_rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = bal\_rec.customer\_id;

END IF;

END LOOP;

COMMIT;

END;

/

-- STEP 5: Scenario 3 - Send Loan Reminders (Due in Next 30 Days)

BEGIN

FOR loan\_rec IN (

SELECT loan\_id, customer\_id, due\_date

FROM loans

WHERE due\_date <= SYSDATE + 30

) LOOP

DECLARE

customer\_name customers.name%TYPE;

BEGIN

SELECT name INTO customer\_name

FROM customers

WHERE customer\_id = loan\_rec.customer\_id;

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

' for customer ' || customer\_name ||

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

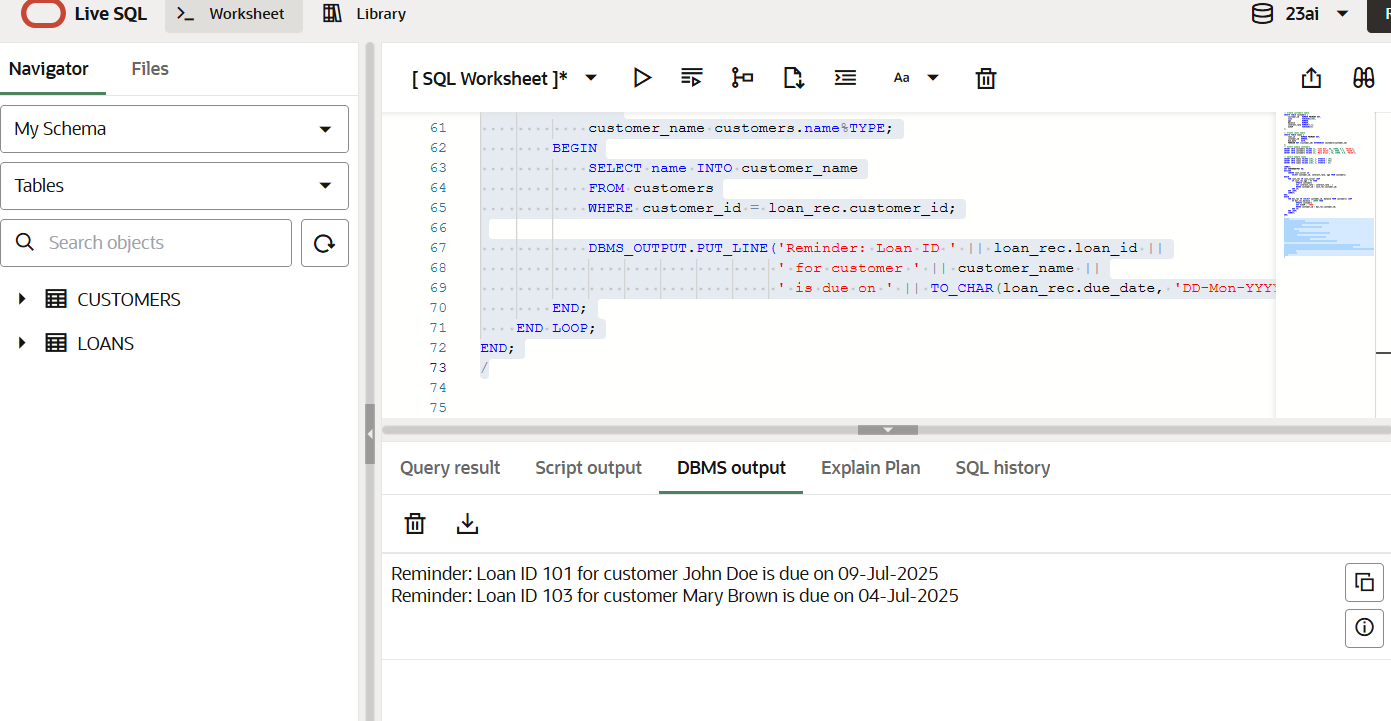
END;

END LOOP;

END;

/

**OUTPUT**

****

**Exercise 2: Error Handling**

-- ===========================

-- STEP 1: CREATE TABLES

-- ===========================

CREATE TABLE accounts (

acc\_id NUMBER PRIMARY KEY,

acc\_name VARCHAR2(100),

balance NUMBER

);

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

emp\_name VARCHAR2(100),

salary NUMBER

);

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

cust\_name VARCHAR2(100),

age NUMBER

);

CREATE TABLE log\_table (

log\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

log\_time TIMESTAMP DEFAULT SYSTIMESTAMP,

message VARCHAR2(4000)

);

/

-- ===========================

-- STEP 2: INSERT SAMPLE DATA

-- ===========================

INSERT INTO accounts VALUES (1, 'Alice', 5000);

INSERT INTO accounts VALUES (2, 'Bob', 2000);

INSERT INTO employees VALUES (101, 'John', 30000);

COMMIT;

/

-- ===========================

-- STEP 3: PROCEDURE 1 - SafeTransferFunds

-- ===========================

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

v\_message VARCHAR2(4000);

BEGIN

SELECT balance INTO v\_balance FROM accounts WHERE acc\_id = p\_from\_acc;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE accounts SET balance = balance - p\_amount WHERE acc\_id = p\_from\_acc;

UPDATE accounts SET balance = balance + p\_amount WHERE acc\_id = p\_to\_acc;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_message := 'Transfer failed from account ' || TO\_CHAR(p\_from\_acc) ||

' to account ' || TO\_CHAR(p\_to\_acc) || ': ' || SQLERRM;

INSERT INTO log\_table(message) VALUES(v\_message);

END;

/

-- ===========================

-- STEP 4: PROCEDURE 2 - UpdateSalary

-- ===========================

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

)

IS

v\_message VARCHAR2(4000);

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_percent / 100)

WHERE emp\_id = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_message := 'Salary update failed for emp\_id ' || TO\_CHAR(p\_emp\_id) || ': ' || SQLERRM;

INSERT INTO log\_table(message) VALUES (v\_message);

END;

/

-- ===========================

-- STEP 5: PROCEDURE 3 - AddNewCustomer

-- ===========================

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_cust\_id IN NUMBER,

p\_cust\_name IN VARCHAR2,

p\_age IN NUMBER

)

IS

v\_message VARCHAR2(4000);

BEGIN

INSERT INTO customers(cust\_id, cust\_name, age)

VALUES(p\_cust\_id, p\_cust\_name, p\_age);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

v\_message := 'Customer ID ' || TO\_CHAR(p\_cust\_id) || ' already exists.';

INSERT INTO log\_table(message) VALUES(v\_message);

WHEN OTHERS THEN

v\_message := 'Add customer failed for cust\_id ' || TO\_CHAR(p\_cust\_id) || ': ' || SQLERRM;

INSERT INTO log\_table(message) VALUES(v\_message);

END;

/

-- ===========================

-- STEP 6: TEST PROCEDURES

-- ===========================

-- Transfer funds from Alice to Bob

EXEC SafeTransferFunds(1, 2, 1000);

-- Update John's salary by 10%

EXEC UpdateSalary(101, 10);

-- Add a new customer (should succeed)

EXEC AddNewCustomer(201, 'Alice Smith', 28);

-- Add same customer again (should log error)

EXEC AddNewCustomer(201, 'Duplicate Entry', 40);

-- ===========================

-- STEP 7: VIEW LOGS AND DATA

-- ===========================

-- Check accounts

SELECT \* FROM accounts;

-- Check employees

SELECT \* FROM employees;

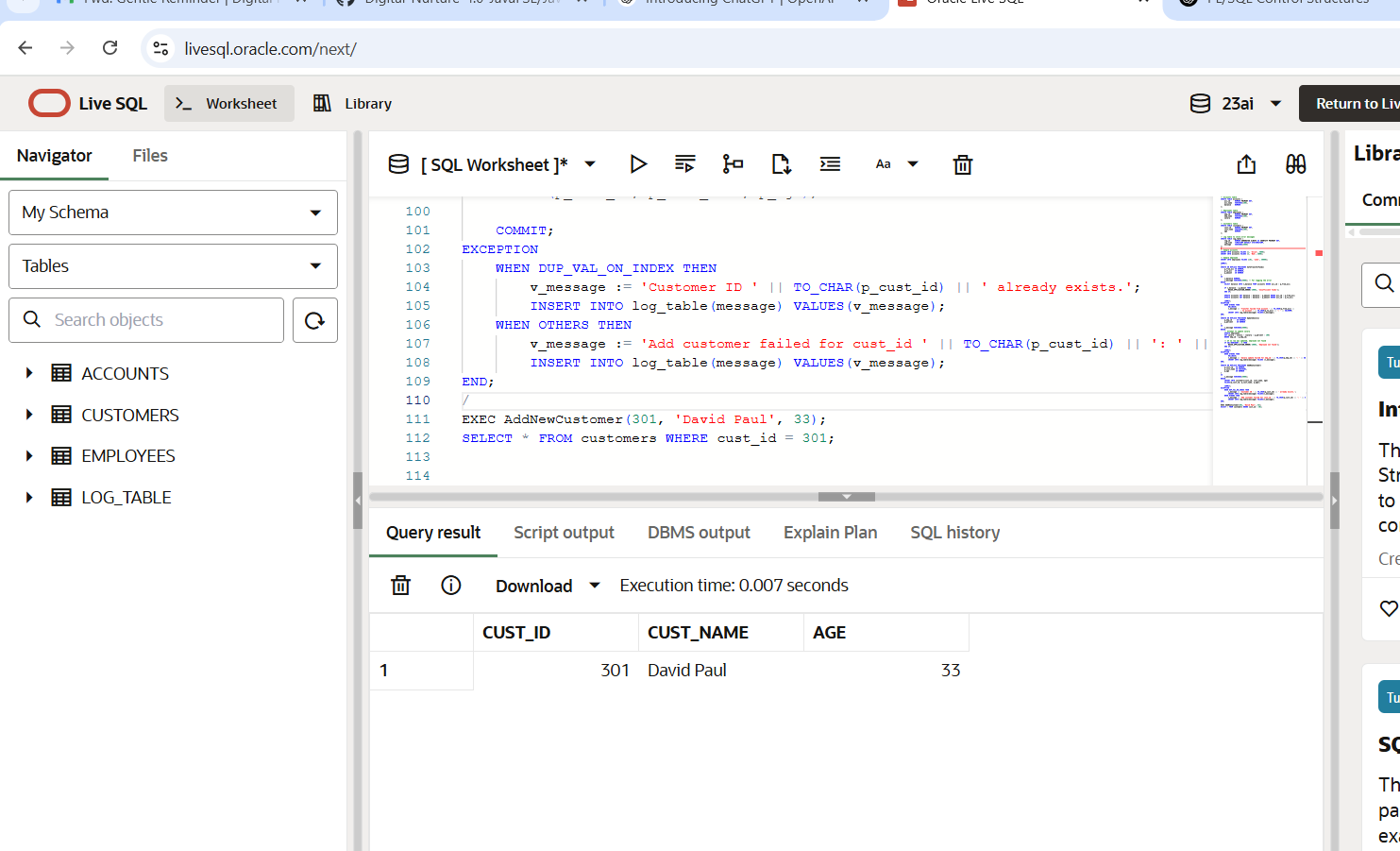
-- Check customers

SELECT \* FROM customers;

-- Check log table

SELECT \* FROM log\_table ORDER BY log\_time DESC;

**OUTPUT**

****

**Exercise 3: Stored Procedures**

-- =======================================

-- STEP 1: Create Tables (if not existing)

-- =======================================

-- Accounts table

CREATE TABLE accounts (

acc\_id NUMBER PRIMARY KEY,

acc\_type VARCHAR2(20), -- 'savings' or 'current'

acc\_name VARCHAR2(100),

balance NUMBER

);

-- Employees table

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

emp\_name VARCHAR2(100),

department VARCHAR2(50),

salary NUMBER

);

-- Log table

CREATE TABLE log\_table (

log\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

log\_time TIMESTAMP DEFAULT SYSTIMESTAMP,

message VARCHAR2(4000)

);

/

-- =======================================

-- STEP 2: Insert Sample Data

-- =======================================

-- Insert savings and current accounts

INSERT INTO accounts VALUES (1, 'savings', 'Alice', 10000);

INSERT INTO accounts VALUES (2, 'savings', 'Bob', 5000);

INSERT INTO accounts VALUES (3, 'current', 'Charlie', 8000);

-- Insert employees

INSERT INTO employees VALUES (101, 'John', 'HR', 30000);

INSERT INTO employees VALUES (102, 'Priya', 'IT', 40000);

INSERT INTO employees VALUES (103, 'David', 'IT', 45000);

COMMIT;

/

-- =======================================

-- STEP 3: PROCEDURE - ProcessMonthlyInterest

-- =======================================

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE acc\_type = 'savings';

COMMIT;

END;

/

-- =======================================

-- STEP 4: PROCEDURE - UpdateEmployeeBonus

-- =======================================

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

)

IS

v\_message VARCHAR2(4000);

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_bonus\_percent / 100)

WHERE department = p\_department;

IF SQL%ROWCOUNT = 0 THEN

v\_message := 'No employees found in department: ' || p\_department;

INSERT INTO log\_table(message) VALUES(v\_message);

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_message := 'Bonus update failed for department ' || p\_department || ': ' || SQLERRM;

INSERT INTO log\_table(message) VALUES(v\_message);

END;

/

-- =======================================

-- STEP 5: PROCEDURE - TransferFunds

-- =======================================

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

v\_message VARCHAR2(4000);

BEGIN

SELECT balance INTO v\_balance FROM accounts WHERE acc\_id = p\_from\_acc;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE accounts SET balance = balance - p\_amount WHERE acc\_id = p\_from\_acc;

UPDATE accounts SET balance = balance + p\_amount WHERE acc\_id = p\_to\_acc;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_message := 'Transfer failed from account ' || TO\_CHAR(p\_from\_acc) ||

' to account ' || TO\_CHAR(p\_to\_acc) || ': ' || SQLERRM;

INSERT INTO log\_table(message) VALUES(v\_message);

END;

/

-- =======================================

-- STEP 6: TEST THE PROCEDURES

-- =======================================

-- Apply interest to all savings accounts

EXEC ProcessMonthlyInterest;

-- Give 5% bonus to IT department

EXEC UpdateEmployeeBonus('IT', 5);

-- Transfer 2000 from Alice to Bob

EXEC TransferFunds(1, 2, 2000);

-- Test error: Transfer too much

EXEC TransferFunds(3, 2, 9000);

-- =======================================

-- STEP 7: VIEW TABLES

-- =======================================

-- View account balances

SELECT \* FROM accounts;

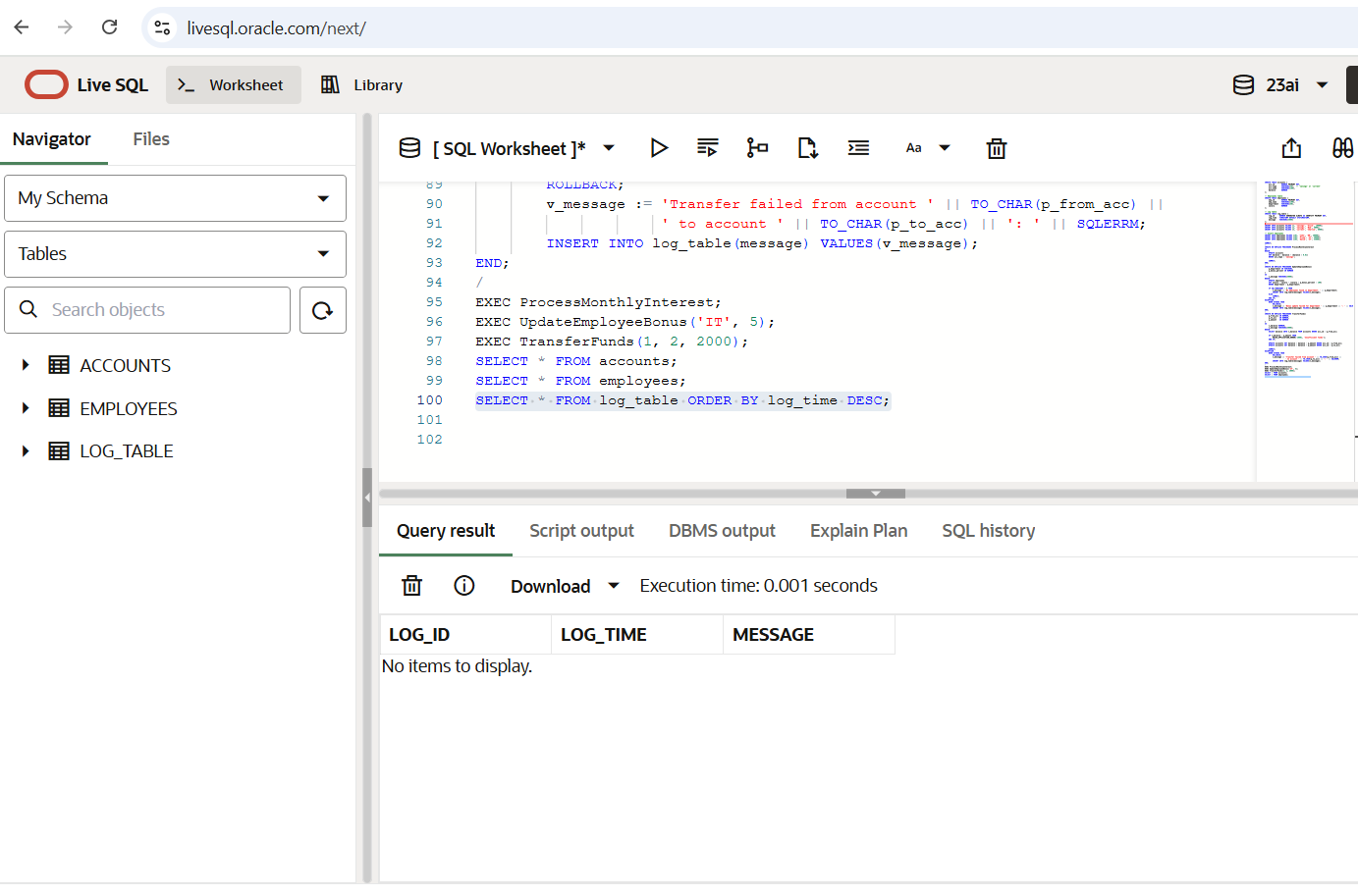
-- View employee salaries

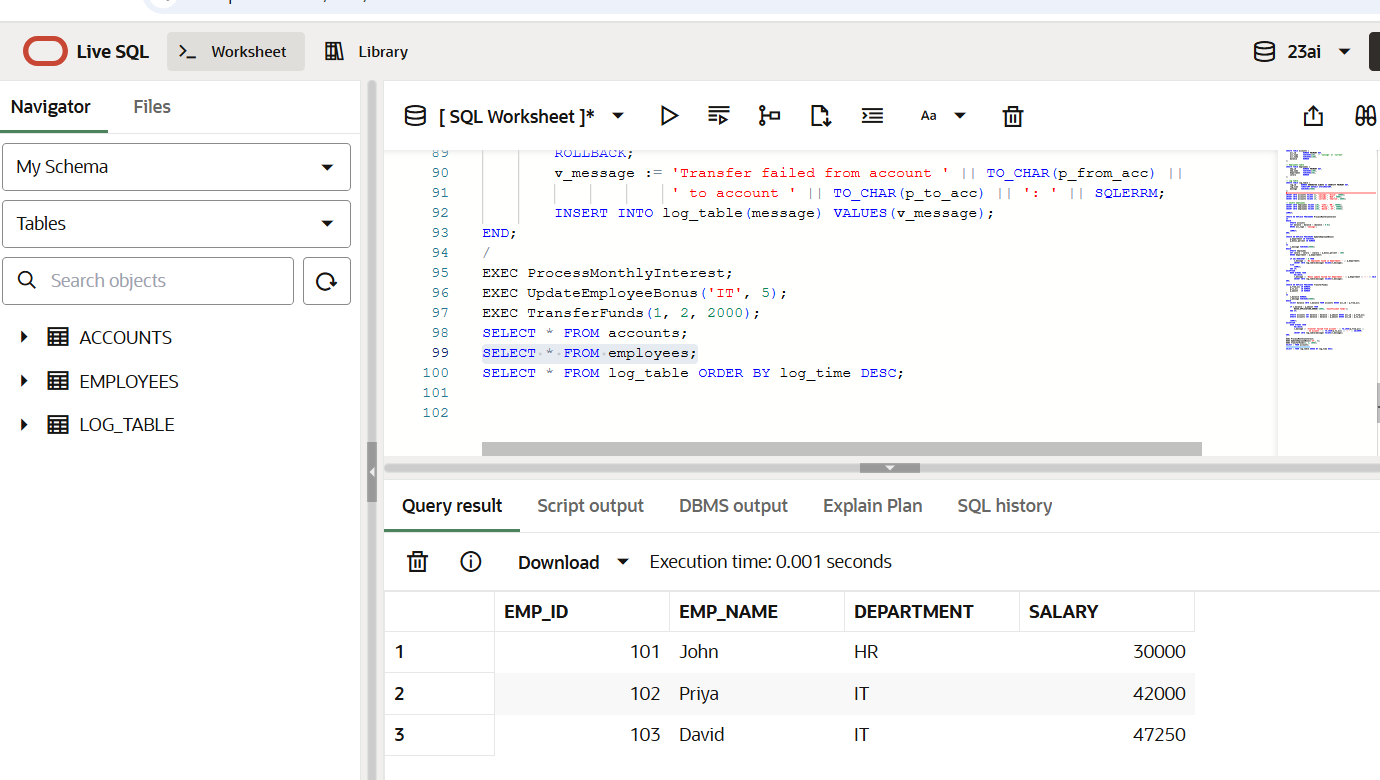
SELECT \* FROM employees;

-- View logged errors

SELECT \* FROM log\_table ORDER BY log\_time DESC;

**OUTPUT**

****

****

**Exercise 4: Functions**

-- ======================================

-- STEP 1: Table Setup (if not already created)

-- ======================================

-- Customers table with date of birth

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

cust\_name VARCHAR2(100),

dob DATE

);

-- Accounts table with balances

CREATE TABLE accounts (

acc\_id NUMBER PRIMARY KEY,

acc\_name VARCHAR2(100),

balance NUMBER

);

/

-- ======================================

-- STEP 2: Insert Sample Data

-- ======================================

-- Insert sample customers

INSERT INTO customers VALUES (101, 'Alice', TO\_DATE('1985-05-10', 'YYYY-MM-DD'));

INSERT INTO customers VALUES (102, 'Bob', TO\_DATE('2000-08-20', 'YYYY-MM-DD'));

-- Insert sample accounts

INSERT INTO accounts VALUES (1, 'Alice', 8000);

INSERT INTO accounts VALUES (2, 'Bob', 3000);

COMMIT;

/

-- ======================================

-- STEP 3: FUNCTION - CalculateAge

-- ======================================

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

/

-- Example usage:

-- SELECT CalculateAge(TO\_DATE('1990-01-01','YYYY-MM-DD')) FROM dual;

-- ======================================

-- STEP 4: FUNCTION - CalculateMonthlyInstallment

-- Formula: EMI = [P \* R \* (1+R)^N] / [(1+R)^N - 1]

-- Where:

-- P = loan amount

-- R = monthly interest rate (annual rate / 12 / 100)

-- N = number of monthly payments (years \* 12)

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_annual\_interest NUMBER,

p\_years NUMBER

)

RETURN NUMBER

IS

r NUMBER := p\_annual\_interest / 12 / 100;

n NUMBER := p\_years \* 12;

emi NUMBER;

BEGIN

IF r = 0 THEN

emi := p\_loan\_amount / n;

ELSE

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

END IF;

RETURN ROUND(emi, 2);

END;

/

-- Example usage:

-- SELECT CalculateMonthlyInstallment(100000, 10, 5) FROM dual;

-- ======================================

-- STEP 5: FUNCTION - HasSufficientBalance

-- ======================================

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_acc\_id IN NUMBER,

p\_amount IN NUMBER

)

RETURN BOOLEAN

IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM accounts WHERE acc\_id = p\_acc\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

-- Example usage (in PL/SQL block):

-- DECLARE

-- result BOOLEAN;

-- BEGIN

-- result := HasSufficientBalance(1, 5000);

-- IF result THEN

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

-- ELSE

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

-- END IF;

-- END;

-- /

-- ======================================

-- STEP 6: TEST FUNCTION CALLS

-- ======================================

-- Age of Alice

SELECT cust\_name, CalculateAge(dob) AS age FROM customers;

-- Calculate EMI

SELECT CalculateMonthlyInstallment(500000, 7.5, 10) AS Monthly\_EMI FROM dual;

-- Check balance

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(1, 6000);

IF result THEN

DBMS\_OUTPUT.PUT\_LINE('Account 1: Sufficient balance');

ELSE

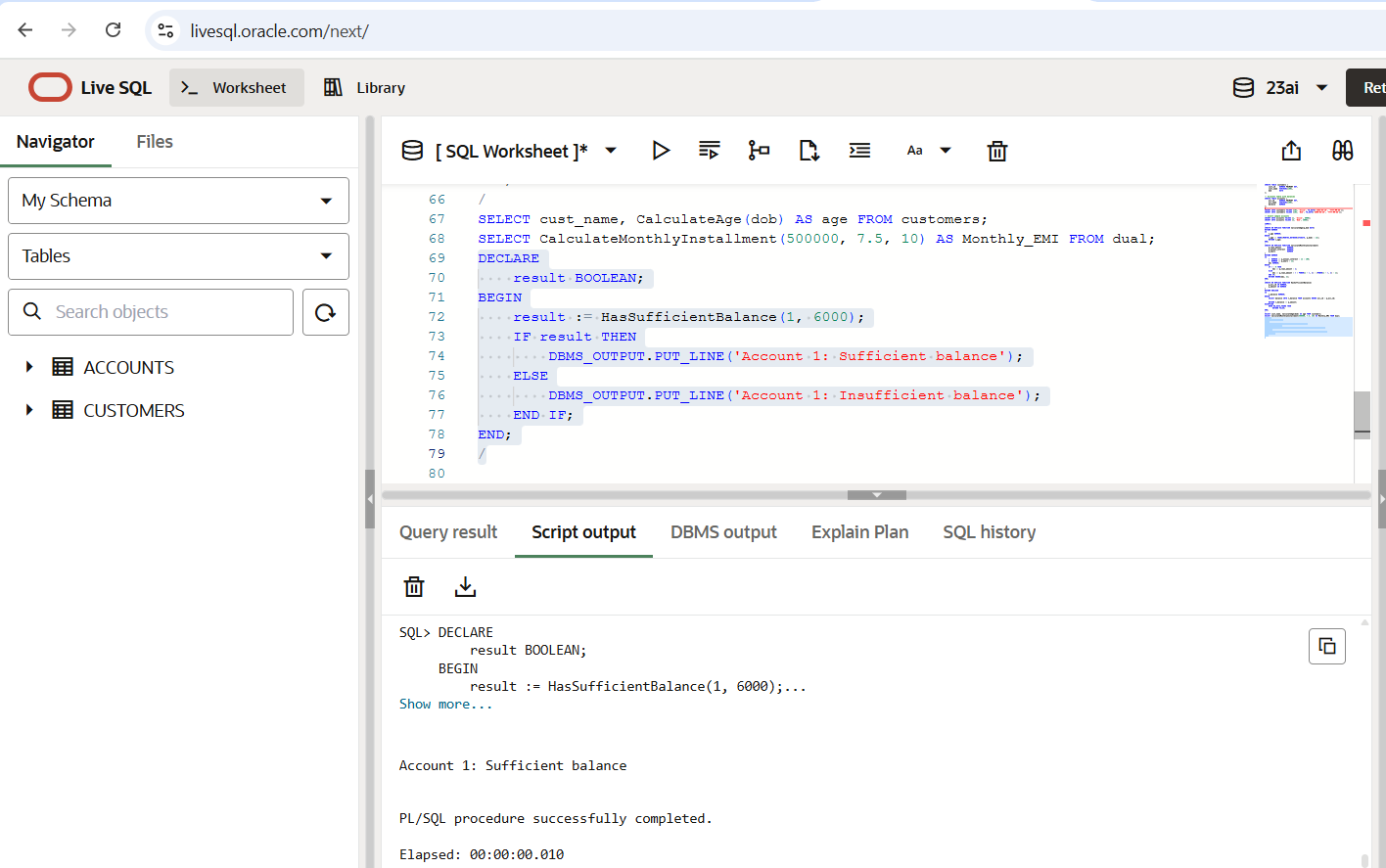
DBMS\_OUTPUT.PUT\_LINE('Account 1: Insufficient balance');

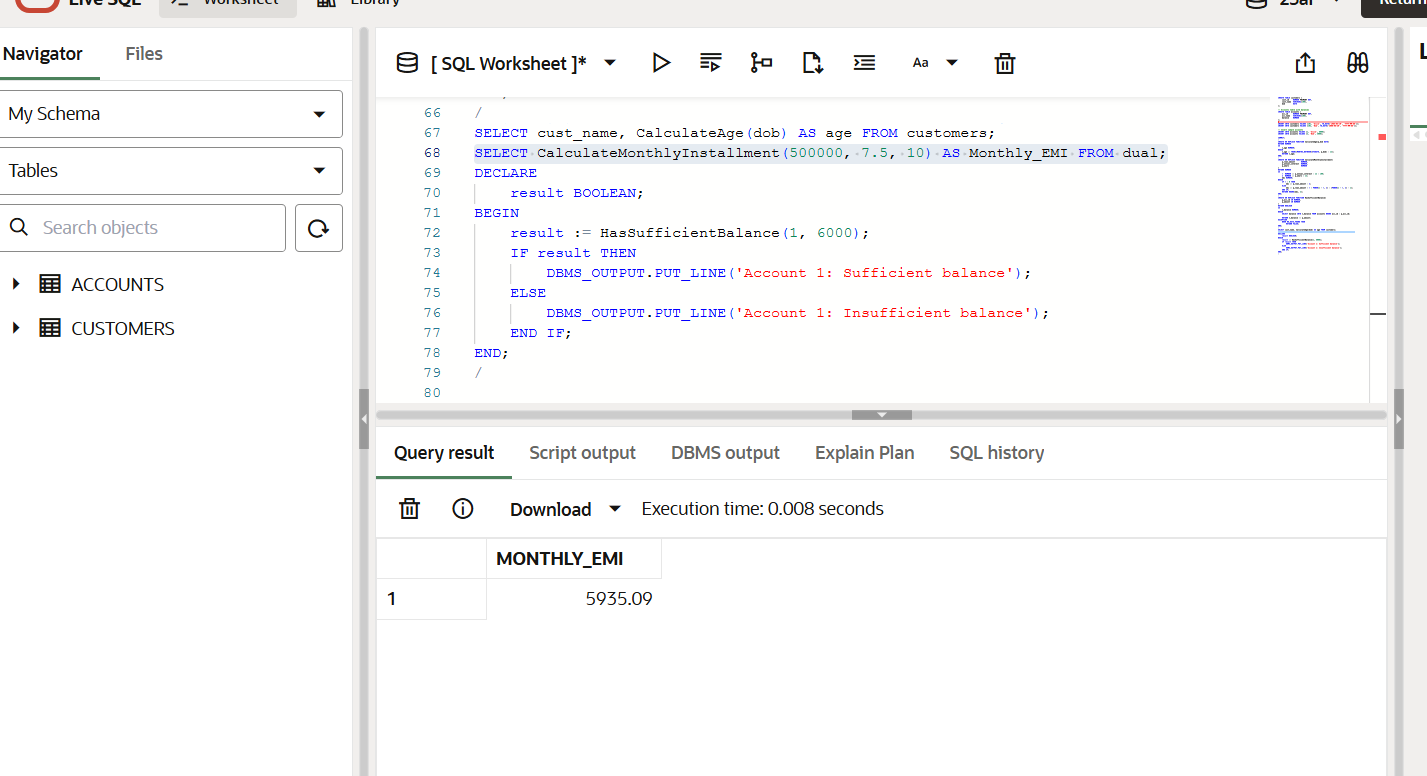
END IF;

END;

/

**OUTPUT**

****

****

**Exercise 5: Triggers**

-- ======================================

-- STEP 1: Create Tables

-- ======================================

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

cust\_name VARCHAR2(100),

dob DATE,

last\_modified DATE

);

CREATE TABLE transactions (

trans\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

acc\_id NUMBER,

trans\_type VARCHAR2(20), -- 'deposit' or 'withdrawal'

amount NUMBER,

trans\_date DATE DEFAULT SYSDATE

);

CREATE TABLE accounts (

acc\_id NUMBER PRIMARY KEY,

acc\_name VARCHAR2(100),

balance NUMBER

);

CREATE TABLE audit\_log (

audit\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

action\_time TIMESTAMP DEFAULT SYSTIMESTAMP,

trans\_id NUMBER,

acc\_id NUMBER,

trans\_type VARCHAR2(20),

amount NUMBER

);

/

-- ======================================

-- STEP 2: Insert Sample Data

-- ======================================

INSERT INTO customers VALUES (1, 'Alice', TO\_DATE('1985-05-10','YYYY-MM-DD'), NULL);

INSERT INTO customers VALUES (2, 'Bob', TO\_DATE('2000-08-20','YYYY-MM-DD'), NULL);

INSERT INTO accounts VALUES (1, 'Alice', 10000);

INSERT INTO accounts VALUES (2, 'Bob', 5000);

COMMIT;

/

-- ======================================

-- STEP 3: Trigger - UpdateCustomerLastModified

-- ======================================

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

:NEW.last\_modified := SYSDATE;

END;

/

-- ======================================

-- STEP 4: Trigger - LogTransaction

-- ======================================

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON transactions

FOR EACH ROW

BEGIN

INSERT INTO audit\_log(trans\_id, acc\_id, trans\_type, amount)

VALUES(:NEW.trans\_id, :NEW.acc\_id, :NEW.trans\_type, :NEW.amount);

END;

/

-- ======================================

-- STEP 5: Trigger - CheckTransactionRules

-- ======================================

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Check if amount is positive

IF :NEW.amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Amount must be greater than zero');

END IF;

-- Check withdrawal rules

IF :NEW.trans\_type = 'withdrawal' THEN

SELECT balance INTO v\_balance FROM accounts WHERE acc\_id = :NEW.acc\_id;

IF :NEW.amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal');

END IF;

END IF;

END;

/

-- ======================================

-- STEP 6: Test the Triggers

-- ======================================

-- Update customer to see last\_modified change

UPDATE customers SET cust\_name = 'Alice Updated' WHERE cust\_id = 1;

-- Valid deposit (should be logged)

INSERT INTO transactions(acc\_id, trans\_type, amount) VALUES (1, 'deposit', 500);

-- Invalid withdrawal (should raise error)

-- INSERT INTO transactions(acc\_id, trans\_type, amount) VALUES (1, 'withdrawal', 999999); -- Uncomment to test

-- ======================================

-- STEP 7: View Results

-- ======================================

-- Check customer last modified

SELECT cust\_id, cust\_name, last\_modified FROM customers;

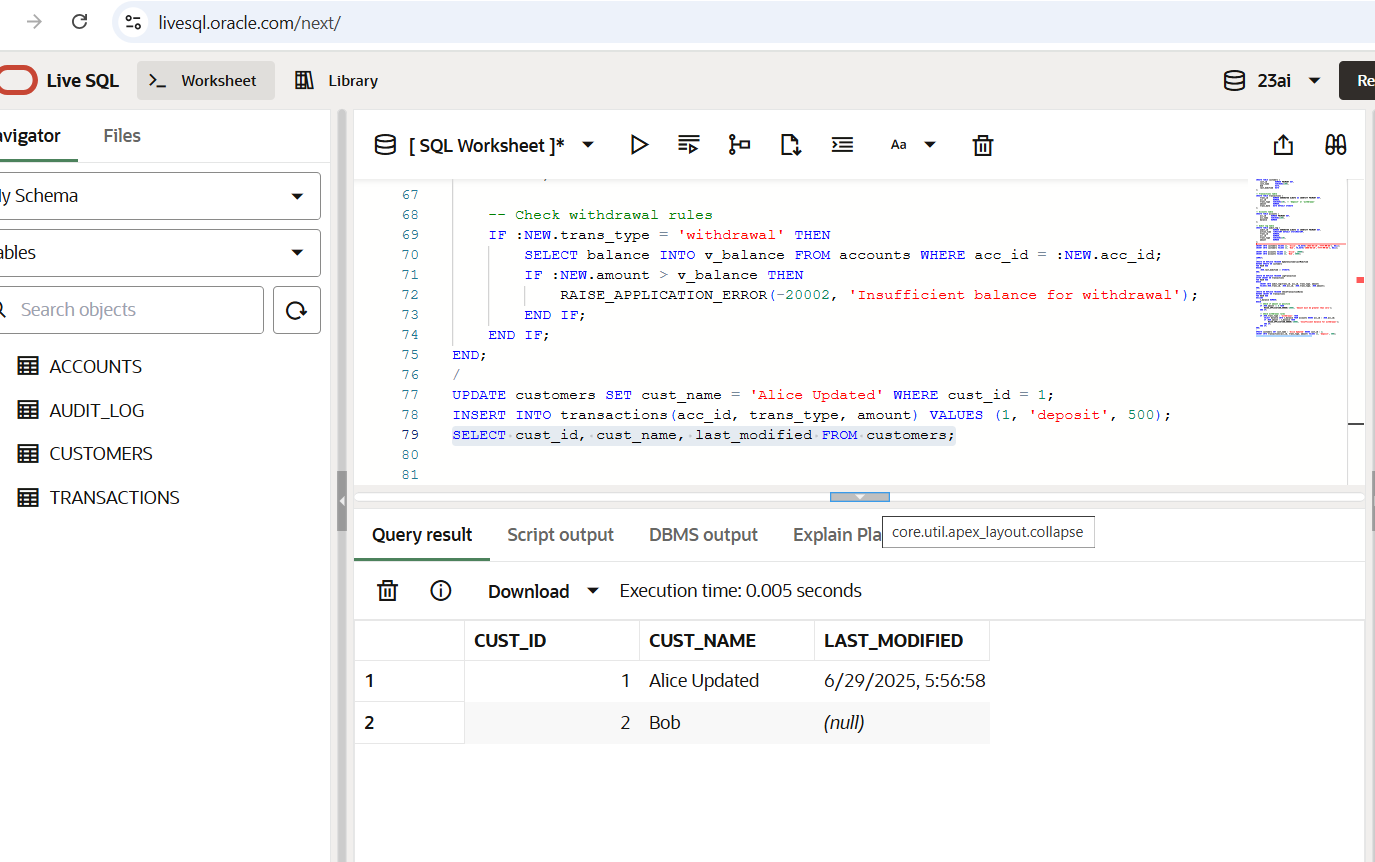
-- Check all transactions

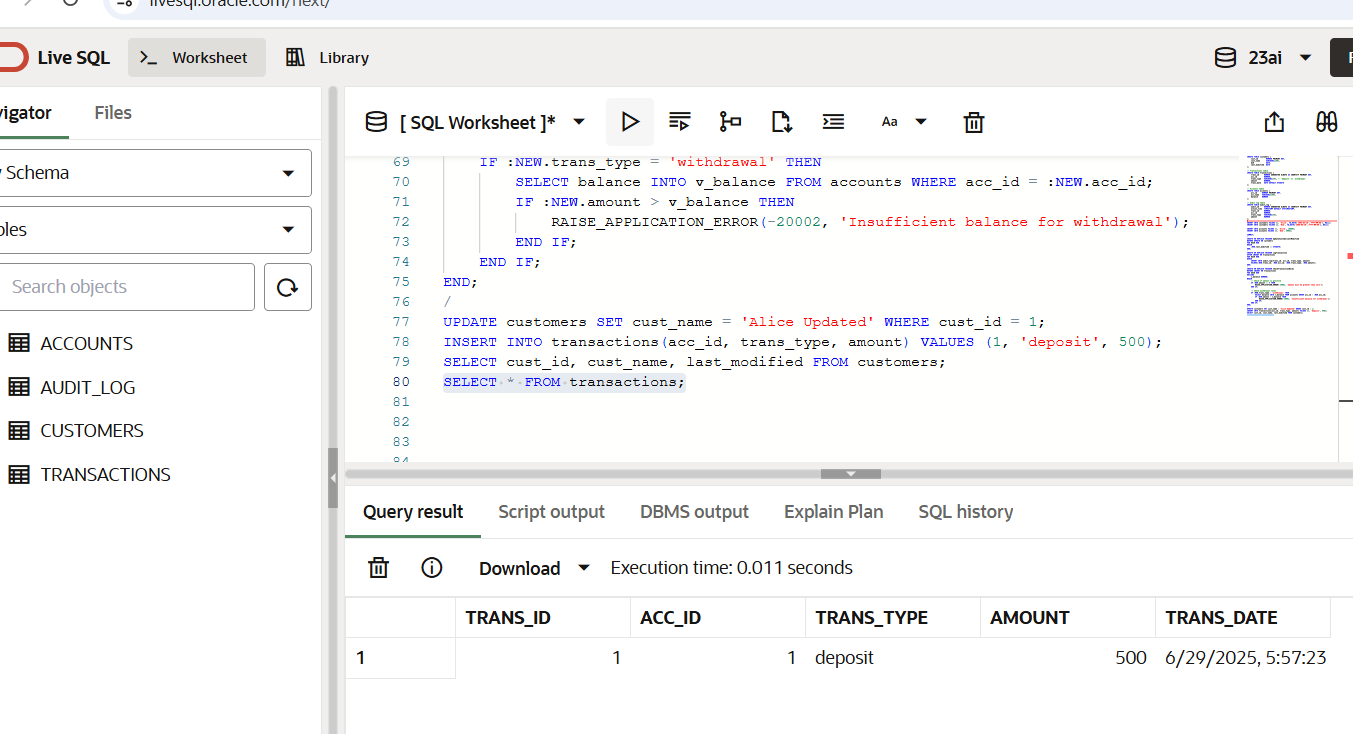
SELECT \* FROM transactions;

-- Check audit log

SELECT \* FROM audit\_log;

**OUTPUT**

****

****

**Exercise 6: Cursors**

-- ============================================

-- STEP 1: TABLE SETUP

-- ============================================

CREATE TABLE transactions (

trans\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

cust\_id NUMBER,

acc\_id NUMBER,

trans\_type VARCHAR2(20), -- deposit / withdrawal

amount NUMBER,

trans\_date DATE

);

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

cust\_name VARCHAR2(100)

);

CREATE TABLE accounts (

acc\_id NUMBER PRIMARY KEY,

cust\_id NUMBER,

balance NUMBER

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

cust\_id NUMBER,

principal NUMBER,

interest\_rate NUMBER -- in percent

);

/

-- ============================================

-- STEP 2: SAMPLE DATA

-- ============================================

-- Insert customers

INSERT INTO customers VALUES (1, 'Alice');

INSERT INTO customers VALUES (2, 'Bob');

-- Insert accounts

INSERT INTO accounts VALUES (101, 1, 10000);

INSERT INTO accounts VALUES (102, 2, 8000);

-- Insert transactions

INSERT INTO transactions (cust\_id, acc\_id, trans\_type, amount, trans\_date) VALUES (1, 101, 'deposit', 2000, SYSDATE);

INSERT INTO transactions (cust\_id, acc\_id, trans\_type, amount, trans\_date) VALUES (1, 101, 'withdrawal', 1000, SYSDATE - 20);

INSERT INTO transactions (cust\_id, acc\_id, trans\_type, amount, trans\_date) VALUES (2, 102, 'deposit', 3000, SYSDATE);

-- Insert loans

INSERT INTO loans VALUES (1, 1, 50000, 8.5);

INSERT INTO loans VALUES (2, 2, 75000, 9.0);

COMMIT;

/

-- ============================================

-- SCENARIO 1: GenerateMonthlyStatements

-- ============================================

DECLARE

CURSOR c\_statements IS

SELECT c.cust\_name, t.acc\_id, t.trans\_type, t.amount, t.trans\_date

FROM transactions t

JOIN customers c ON t.cust\_id = c.cust\_id

WHERE TO\_CHAR(t.trans\_date, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY')

ORDER BY c.cust\_name, t.trans\_date;

v\_name customers.cust\_name%TYPE;

v\_acc\_id transactions.acc\_id%TYPE;

v\_type transactions.trans\_type%TYPE;

v\_amt transactions.amount%TYPE;

v\_date transactions.trans\_date%TYPE;

BEGIN

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

FOR rec IN c\_statements LOOP

v\_name := rec.cust\_name;

v\_acc\_id := rec.acc\_id;

v\_type := rec.trans\_type;

v\_amt := rec.amount;

v\_date := rec.trans\_date;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ', Account: ' || v\_acc\_id ||

', ' || v\_type || ': ' || v\_amt || ', Date: ' || TO\_CHAR(v\_date, 'DD-MON-YYYY'));

END LOOP;

END;

/

-- ============================================

-- SCENARIO 2: ApplyAnnualFee

-- ============================================

DECLARE

CURSOR c\_accounts IS

SELECT acc\_id, balance FROM accounts;

v\_acc\_id accounts.acc\_id%TYPE;

v\_balance accounts.balance%TYPE;

v\_fee CONSTANT NUMBER := 500;

BEGIN

FOR rec IN c\_accounts LOOP

v\_acc\_id := rec.acc\_id;

v\_balance := rec.balance;

IF v\_balance >= v\_fee THEN

UPDATE accounts SET balance = balance - v\_fee WHERE acc\_id = v\_acc\_id;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_fee || ' applied to Account ' || v\_acc\_id);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ' || v\_acc\_id || ' has insufficient balance for annual fee.');

END IF;

END LOOP;

COMMIT;

END;

/

-- ============================================

-- SCENARIO 3: UpdateLoanInterestRates

-- ============================================

DECLARE

CURSOR c\_loans IS

SELECT loan\_id, interest\_rate FROM loans;

v\_loan\_id loans.loan\_id%TYPE;

v\_interest loans.interest\_rate%TYPE;

BEGIN

FOR rec IN c\_loans LOOP

v\_loan\_id := rec.loan\_id;

v\_interest := rec.interest\_rate;

-- Example rule: reduce 0.5% if current interest > 8.5%

IF v\_interest > 8.5 THEN

UPDATE loans

SET interest\_rate = interest\_rate - 0.5

WHERE loan\_id = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_loan\_id || ': Interest reduced to ' || (v\_interest - 0.5));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_loan\_id || ': No change.');

END IF;

END LOOP;

COMMIT;

END;

/

-- ============================================

-- VIEW RESULTS

-- ============================================

-- Check account balances after fee

SELECT \* FROM accounts;

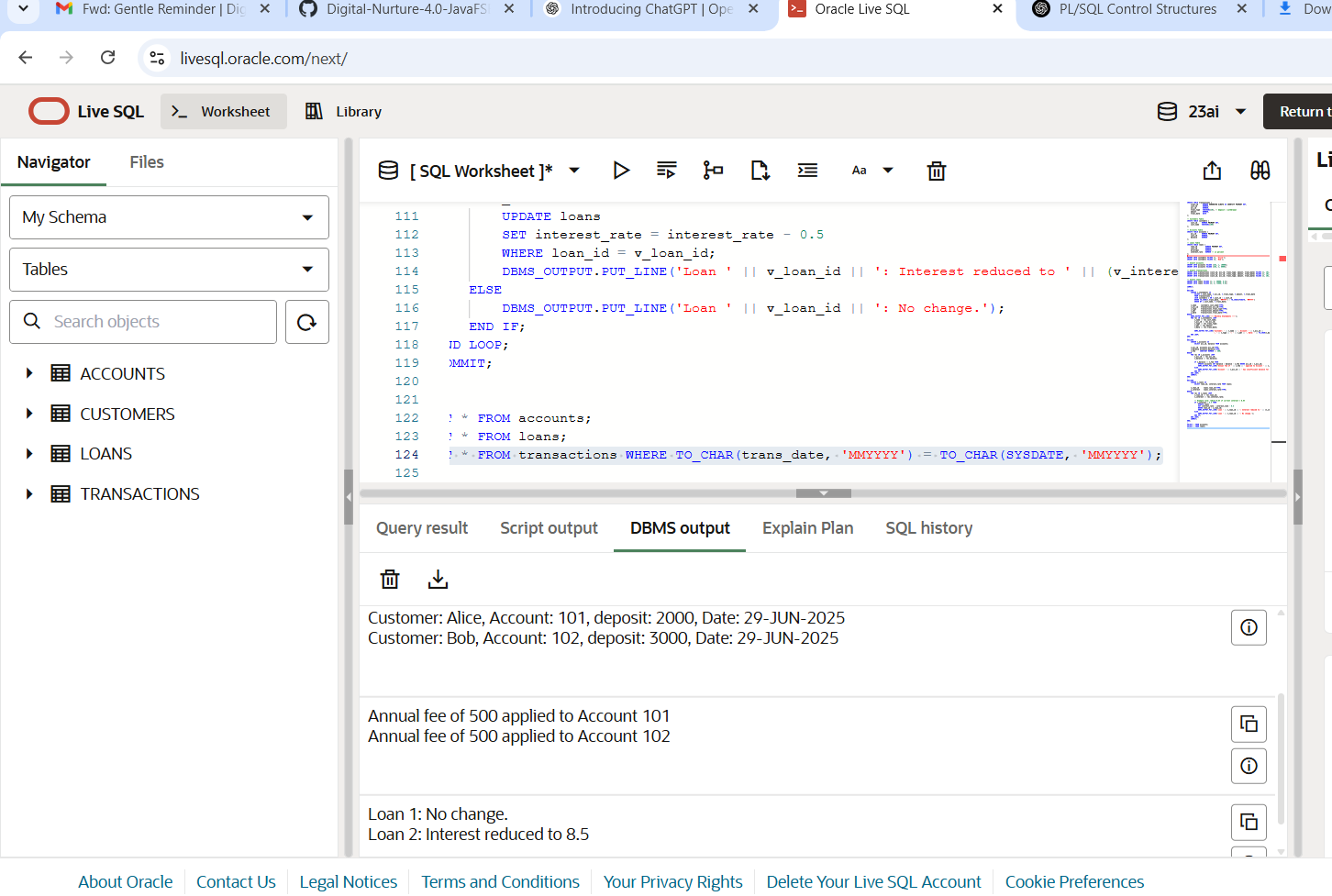
-- Check loan interest updates

SELECT \* FROM loans;

-- Optional: Check current month transactions

SELECT \* FROM transactions WHERE TO\_CHAR(trans\_date, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

**OUTPUT**

****

**Exercise 7: Packages**

-- ============================================================

-- STEP 1: CREATE TABLES

-- ============================================================

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

);

/

-- ============================================================

-- STEP 2: SAMPLE DATA

-- ============================================================

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

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

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

/

-- ============================================================

-- STEP 3: PACKAGE - CustomerManagement

-- ============================================================

CREATE OR REPLACE PACKAGE CustomerManagement IS

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

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

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

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

BEGIN

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

END;

PROCEDURE UpdateCustomer(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

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

END;

END CustomerManagement;

/

-- ============================================================

-- STEP 4: PACKAGE - EmployeeManagement

-- ============================================================

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_date DATE);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_position VARCHAR2, p\_salary NUMBER);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_date DATE) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept, p\_date);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_position VARCHAR2, p\_salary NUMBER) IS

BEGIN

UPDATE Employees

SET Position = p\_position,

Salary = p\_salary

WHERE EmployeeID = p\_id;

END;

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

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

RETURN v\_salary \* 12;

END;

END EmployeeManagement;

/

-- ============================================================

-- STEP 5: PACKAGE - AccountOperations

-- ============================================================

CREATE OR REPLACE PACKAGE AccountOperations IS

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

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

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 GetTotalCustomerBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

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

RETURN NVL(v\_total, 0);

END;

END AccountOperations;

/

-- ============================================================

-- STEP 6: TESTING EXAMPLES (Optional)

-- ============================================================

-- EXEC CustomerManagement.AddCustomer(3, 'Mark Twain', TO\_DATE('1970-01-01','YYYY-MM-DD'), 2000);

-- EXEC EmployeeManagement.HireEmployee(3, 'Lisa Green', 'Analyst', 55000, 'Finance', SYSDATE);

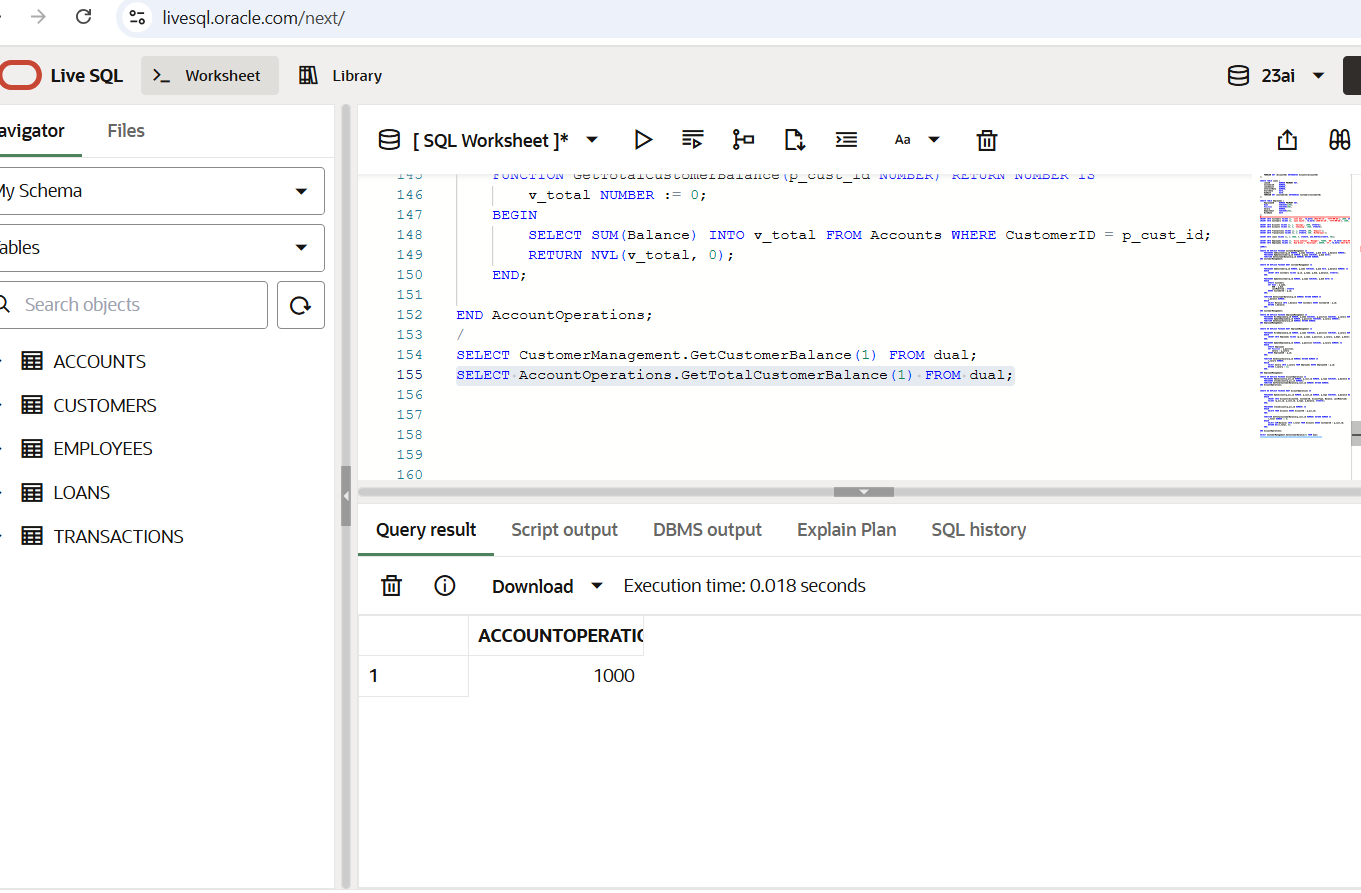
-- EXEC AccountOperations.OpenAccount(3, 1, 'Savings', 3000);

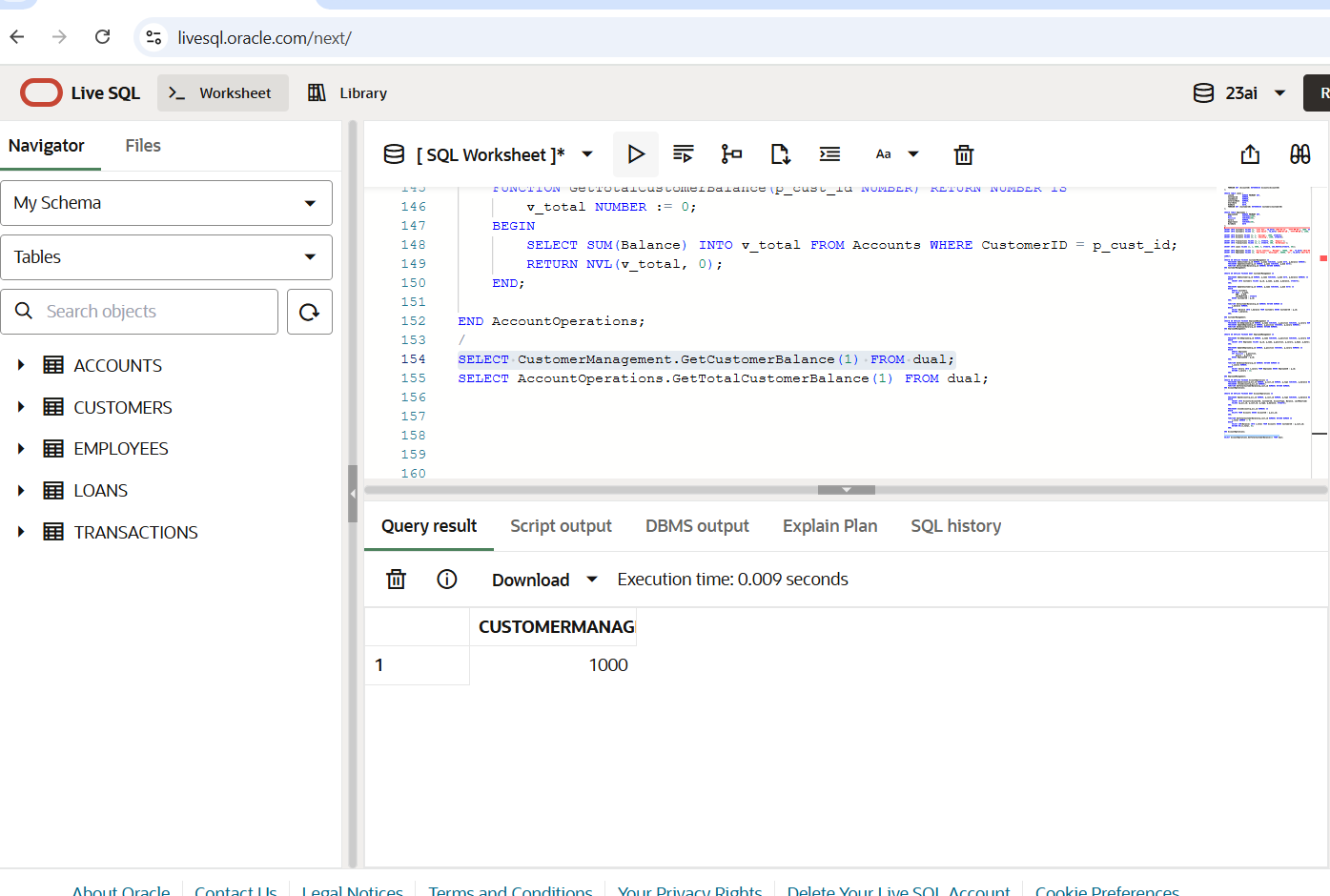
-- SELECT CustomerManagement.GetCustomerBalance(1) FROM dual;

-- SELECT EmployeeManagement.GetAnnualSalary(1) FROM dual;

-- SELECT AccountOperations.GetTotalCustomerBalance(1) FROM dual;

**OUTPUT**

****

****