Hands on session

PLSQL Excerises

**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.

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.

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.

**Objective**

Modularize logic using stored procedures for reusability and separation of business rules.

**Steps to implement**

* Step 1: Create Tables
* savings\_accounts: For interest calculation
* employees: For salary bonus updates
* accounts: For fund transfer between accounts
* Step 2: Insert Sample Data
* Add multiple records into each table for realistic testing.
* Step 3: Scenario 1 – Monthly Interest on Savings Accounts
* Use ProcessMonthlyInterest
* Step 4: Scenario 2 – Bonus for Employees
* Use UpdateEmployeeBonus(p\_department\_id, p\_bonus\_percent)
* Scenario 3 – Transfer Funds Between Accounts
* Use TransferFunds(p\_from\_account, p\_to\_account, p\_amount)
* Step 6: Call Each Procedure

**Code**

SET SERVEROUTPUT ON;

BEGIN EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts'; EXCEPTION WHEN OTHERS THEN NULL; END;

-- Step 1: Create Tables

-- Table: Savings Accounts

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

-- Table: Employees

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

emp\_name VARCHAR2(100),

salary NUMBER,

department\_id NUMBER

);

-- Table: Accounts (for fund transfer)

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

-- Step 2: Insert Sample Data

-- Savings Accounts

INSERT INTO savings\_accounts VALUES (1, 101, 1000);

INSERT INTO savings\_accounts VALUES (2, 102, 5000);

INSERT INTO savings\_accounts VALUES (3, 103, 7500);

-- Employees

INSERT INTO employees VALUES (1, 'Alice', 30000, 10);

INSERT INTO employees VALUES (2, 'Bob', 35000, 10);

INSERT INTO employees VALUES (3, 'Charlie', 40000, 20);

-- Accounts

INSERT INTO accounts VALUES (101, 201, 10000);

INSERT INTO accounts VALUES (102, 202, 2000);

COMMIT;

-- Step 3: Stored Procedure - ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT account\_id, balance FROM savings\_accounts

) LOOP

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_id = acc.account\_id;

END LOOP;

COMMIT;

END;

/

-- Call procedure

BEGIN

ProcessMonthlyInterest;

DBMS\_OUTPUT.PUT\_LINE('✅ Monthly interest applied.');

END;

/

-- View updated savings accounts

SELECT \* FROM savings\_accounts;

-- Step 4: Stored Procedure - UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE employees

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

WHERE department\_id = p\_department\_id;

COMMIT;

END;

/

-- Call procedure (e.g., 10% bonus to department 10)

BEGIN

UpdateEmployeeBonus(10, 10);

DBMS\_OUTPUT.PUT\_LINE('✅ Bonus updated for department 10.');

END;

/

-- View updated employee salaries

SELECT \* FROM employees;

-- Step 5: Stored Procedure - TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Get source balance

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_account;

IF v\_balance >= p\_amount THEN

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

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

ELSE

RAISE\_APPLICATION\_ERROR(-20001, '❌ Insufficient balance in source account.');

END IF;

END;

/

-- Call procedure (valid transfer)

BEGIN

TransferFunds(101, 102, 3000);

END;

/

-- Check balances

SELECT \* FROM accounts;

-- Call procedure (invalid transfer, should fail)

BEGIN

TransferFunds(102, 101, 10000);

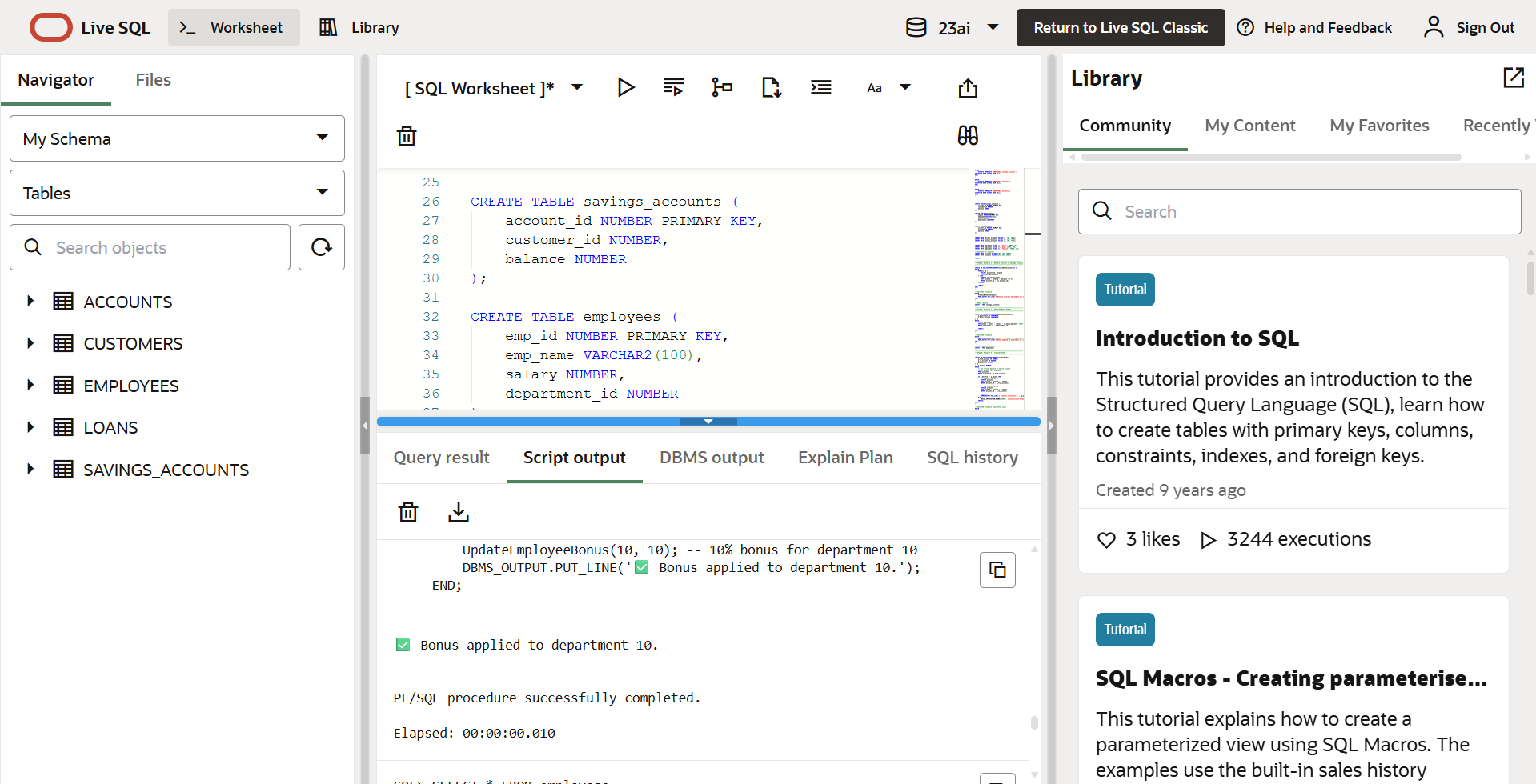
END;

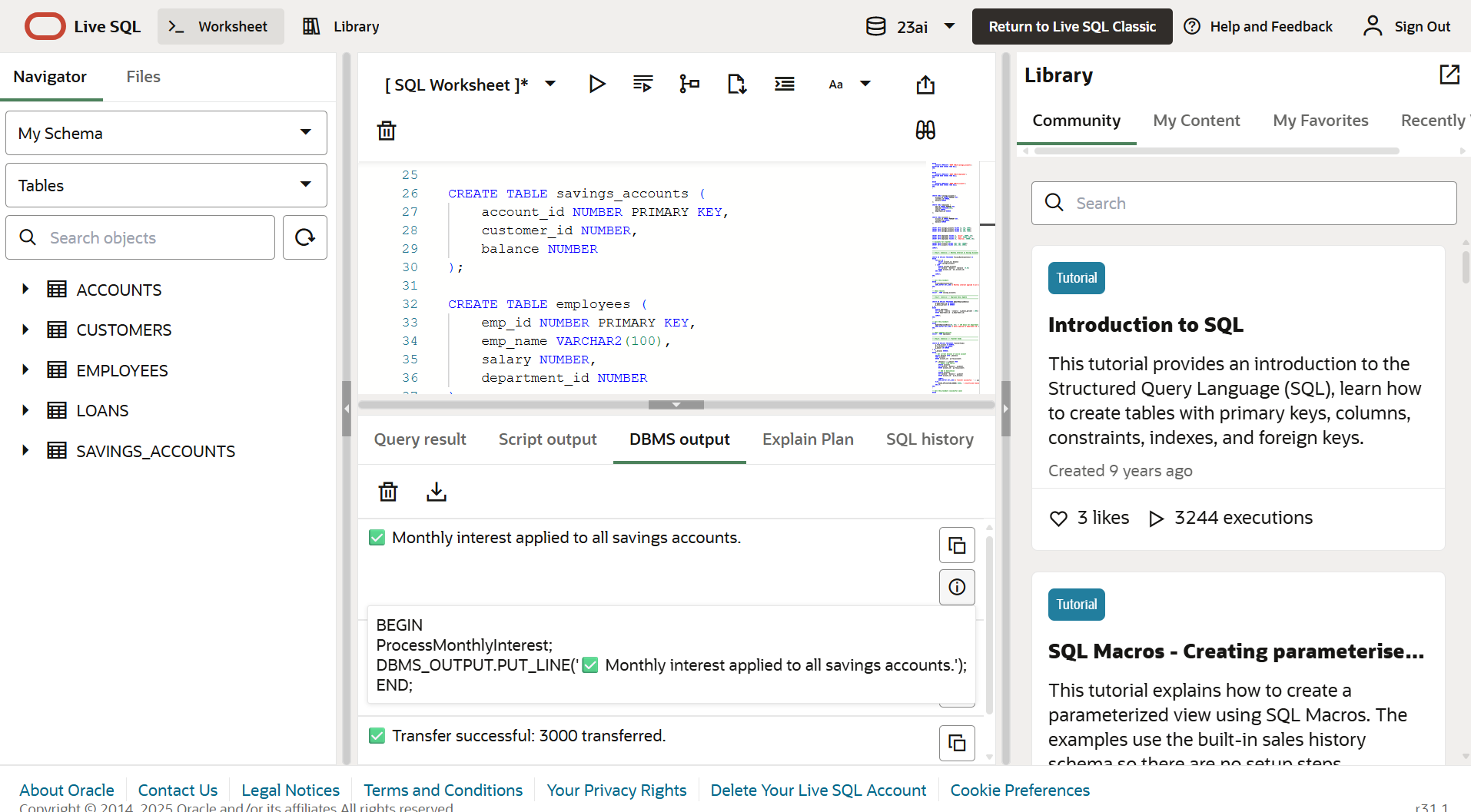
/

-- Final Output Message

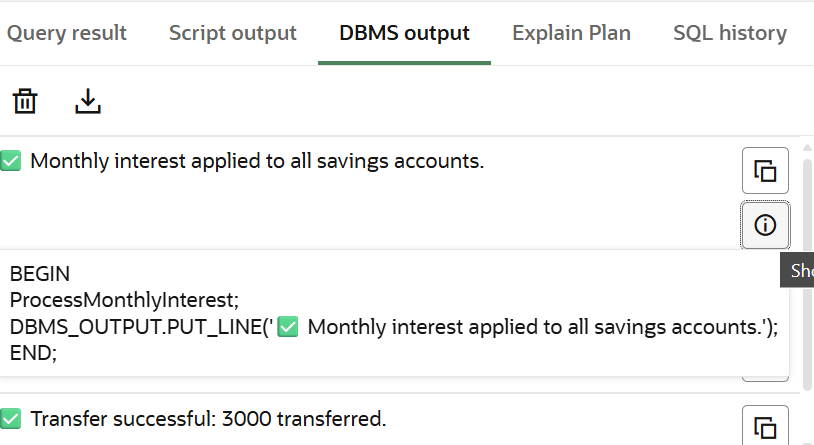
SELECT '✅ Exercise 3: All procedures executed successfully!' AS status FROM dual;

**Output**

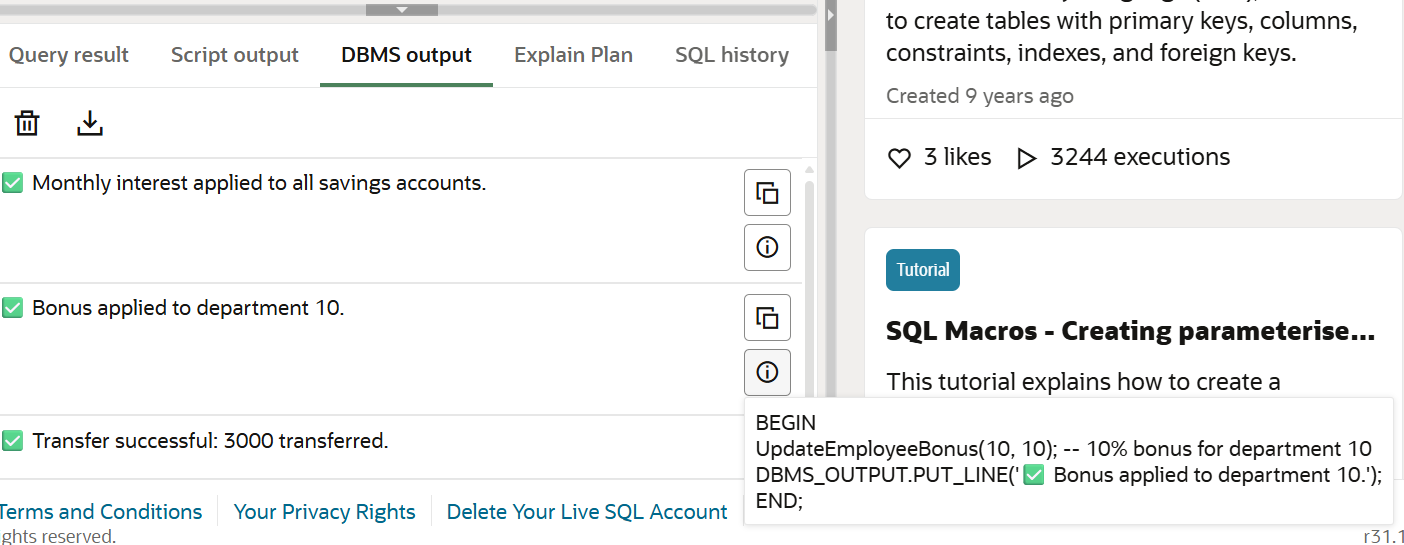




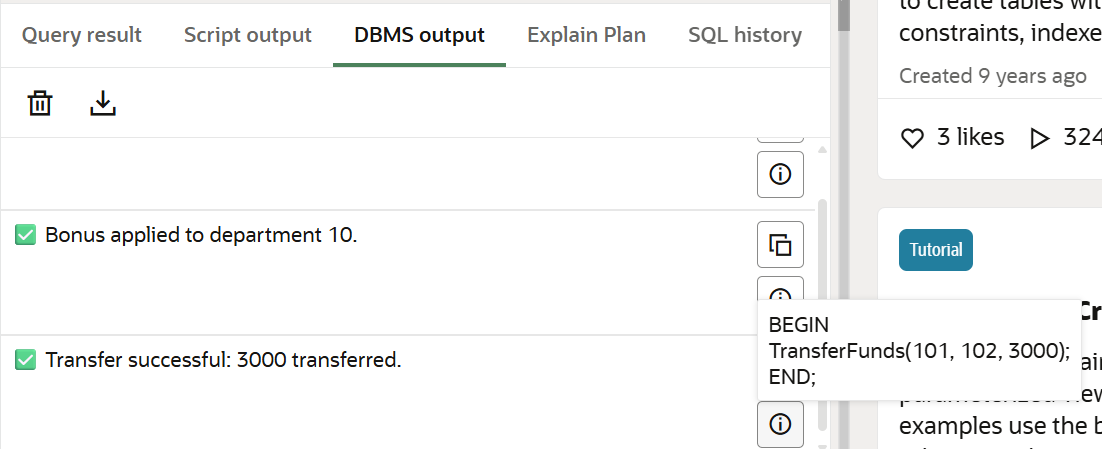
DBMS Output



Scenario 1



Scenario 2



Scenario 3