**Cognizant Deep Skilling - Digital Nurture 4.0**

**PL/SQL Exercises**

**Exercise 1: Control Structures**

**customers table**

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER,

balance NUMBER(10,2),

isVIP CHAR(1),

interest\_rate NUMBER(5,2)

);

**loans table**

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

due\_date DATE,

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

);

**Sample data**

INSERT INTO customers VALUES (1, 'Anitha', 65, 12000, 'N', 8.5);

INSERT INTO customers VALUES (2, 'Balaji', 45, 5000, 'N', 9.0);

INSERT INTO customers VALUES (3, 'Chandru', 70, 20000, 'N', 7.5);

INSERT INTO customers VALUES (4, 'Devi', 30, 3000, 'N', 10.0);

INSERT INTO loans VALUES (101, 1, SYSDATE + 10);

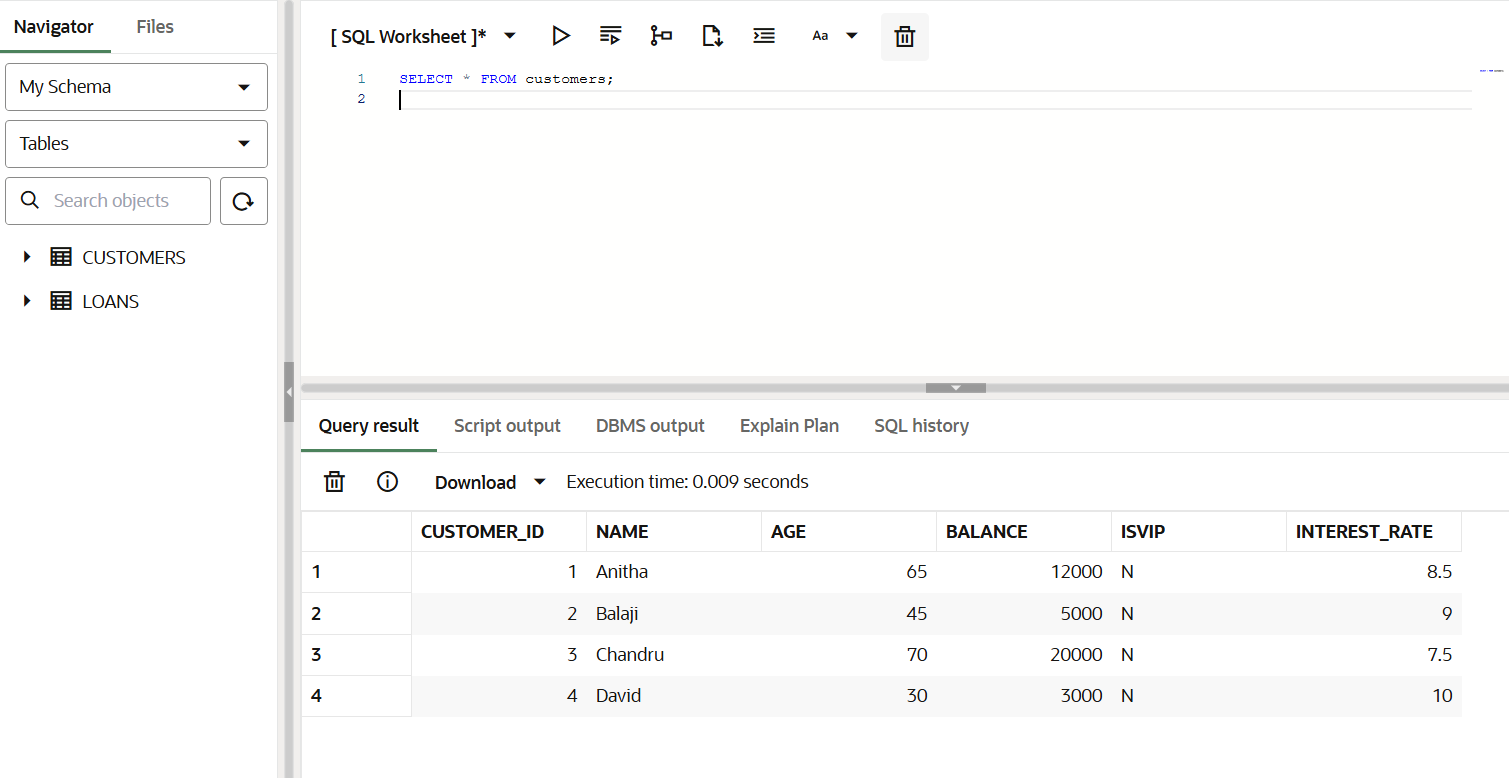
INSERT INTO loans VALUES (102, 2, SYSDATE + 40);

INSERT INTO loans VALUES (103, 3, SYSDATE + 20);

INSERT INTO loans VALUES (104, 4, SYSDATE + 5);

COMMIT;

**SELECT \* FROM customers;**



**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**PL/SQL Block:**

BEGIN

FOR rec IN (SELECT customer\_id, interest\_rate FROM customers WHERE age > 60) LOOP

UPDATE customers

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = rec.customer\_id;

END LOOP;

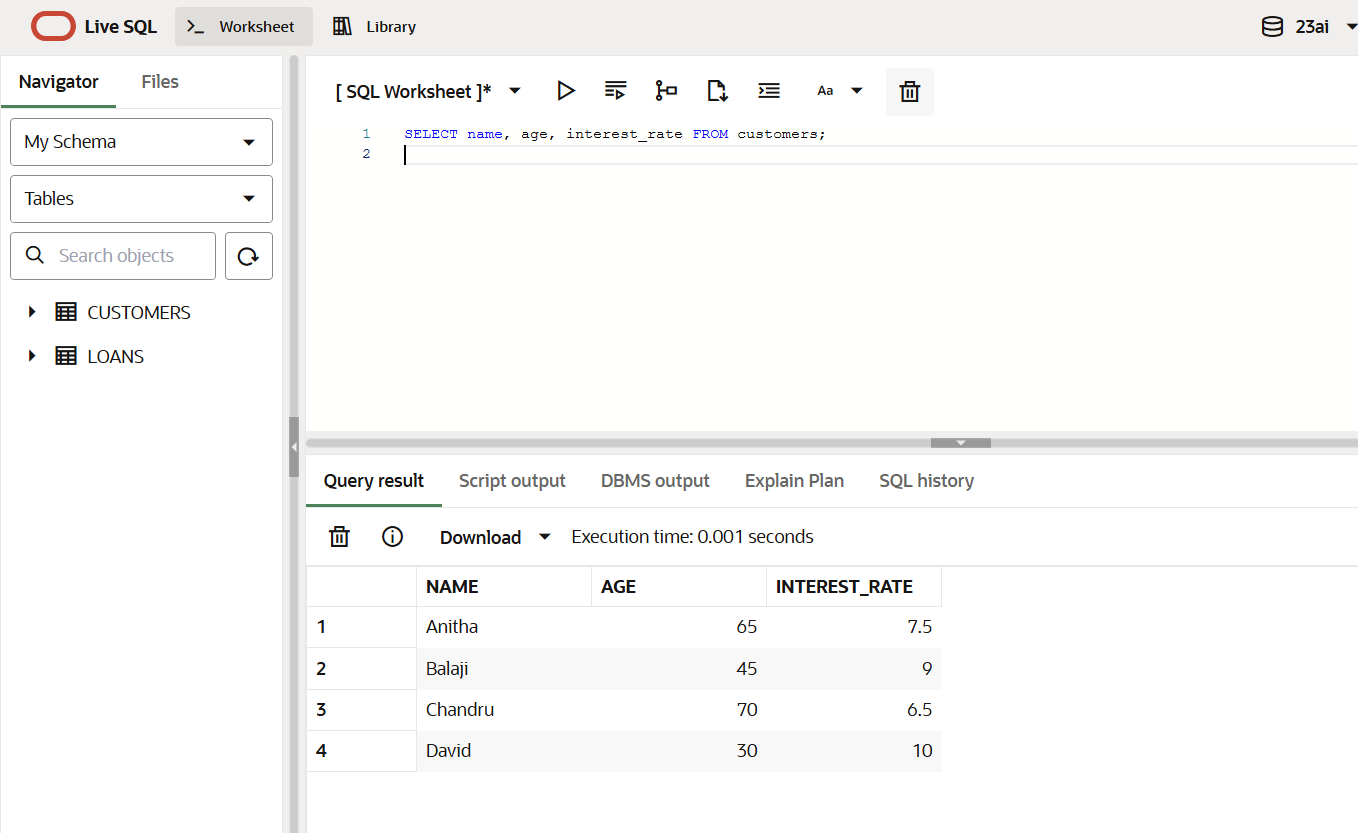
COMMIT;

END;

/

**SELECT name, age, interest\_rate FROM customers;**

**Output:**



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**PL/SQL Block:**

BEGIN

FOR rec IN (SELECT customer\_id FROM customers WHERE balance > 10000) LOOP

UPDATE customers

SET isVIP = 'Y'

WHERE customer\_id = rec.customer\_id;

END LOOP;

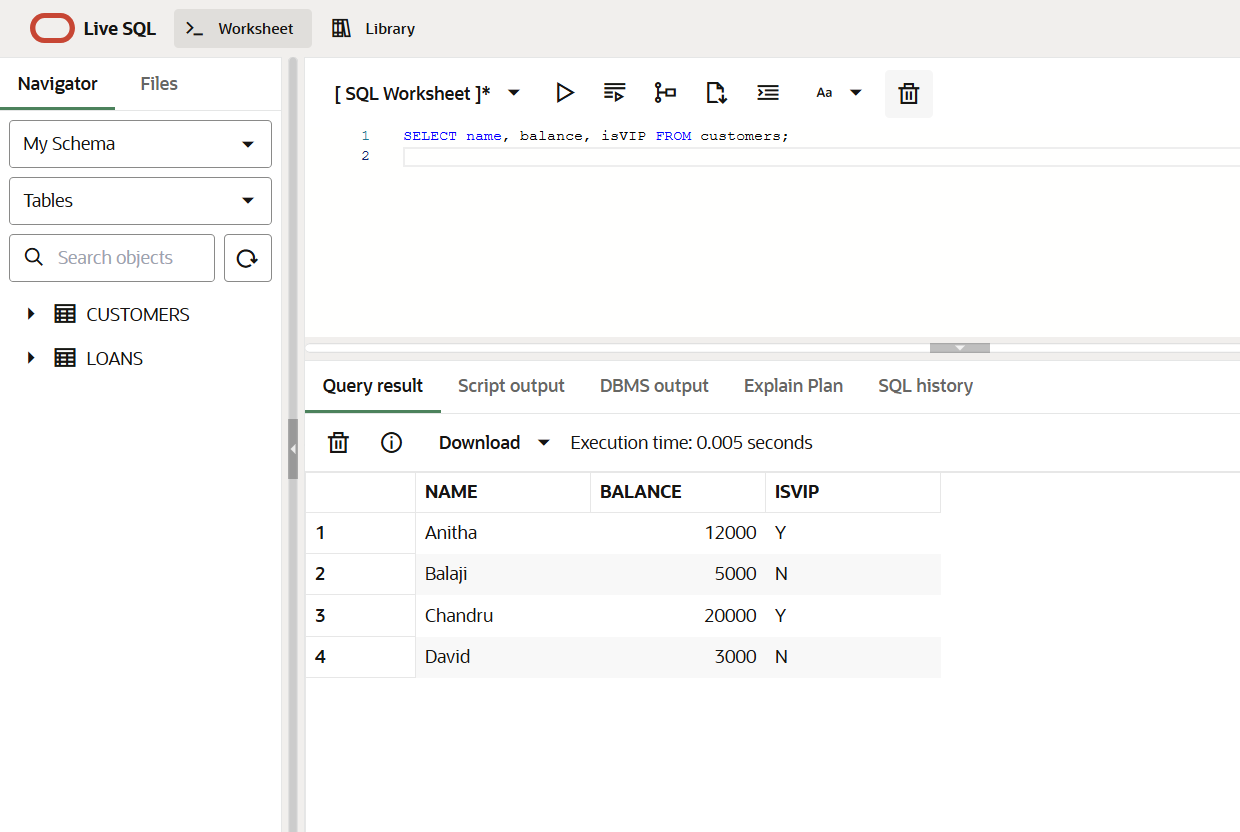
COMMIT;

END;

/

**SELECT name, balance, isVIP FROM customers;**

**Output:**



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**PL/SQL Block:**

DECLARE

v\_name VARCHAR2(50);

v\_due\_date DATE;

BEGIN

FOR rec IN (

SELECT c.name, l.due\_date

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

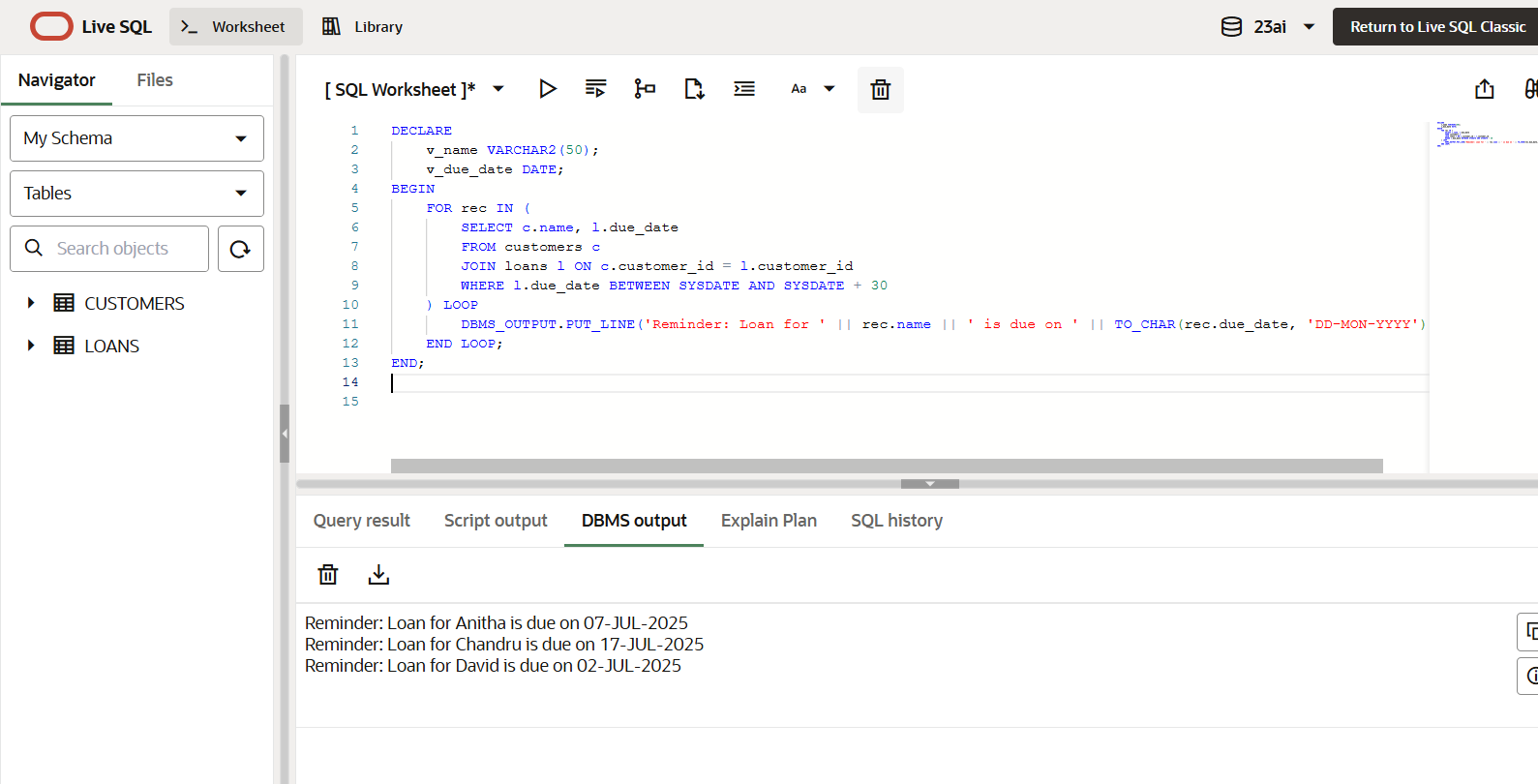
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for ' || rec.name || ' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output:**



**Exercise 3: Stored Procedures**

**Savings accounts table**

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(50),

balance NUMBER(10, 2)

);

**Employees table**

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

emp\_name VARCHAR2(50),

department\_id NUMBER,

salary NUMBER(10, 2)

);

**Accounts table**

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(50),

balance NUMBER(10, 2)

);

**Sample Data**

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

INSERT INTO savings\_accounts VALUES (2, 'Bob', 15000);

INSERT INTO employees VALUES (101, 'John', 10, 50000);

INSERT INTO employees VALUES (102, 'Jane', 10, 55000);

INSERT INTO employees VALUES (103, 'Mark', 20, 60000);

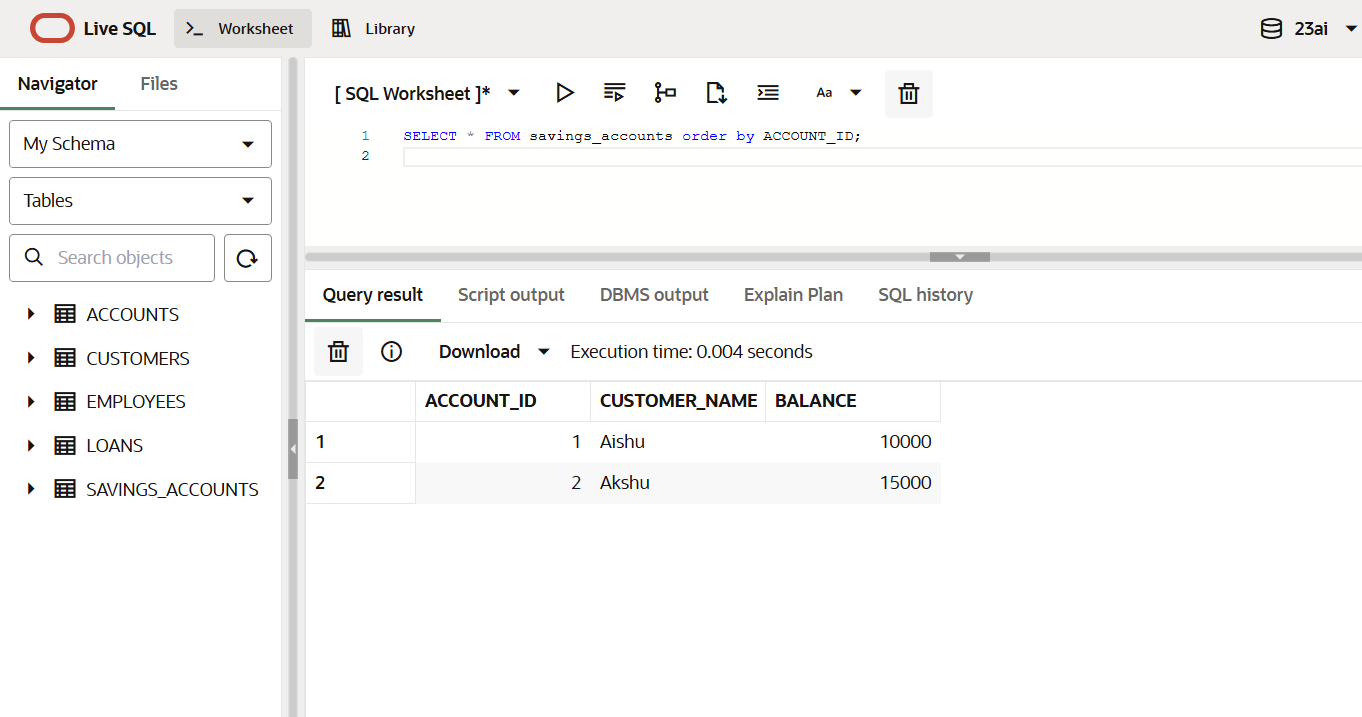
INSERT INTO accounts VALUES (201, 'Alice', 8000);

INSERT INTO accounts VALUES (202, 'Alice', 3000);

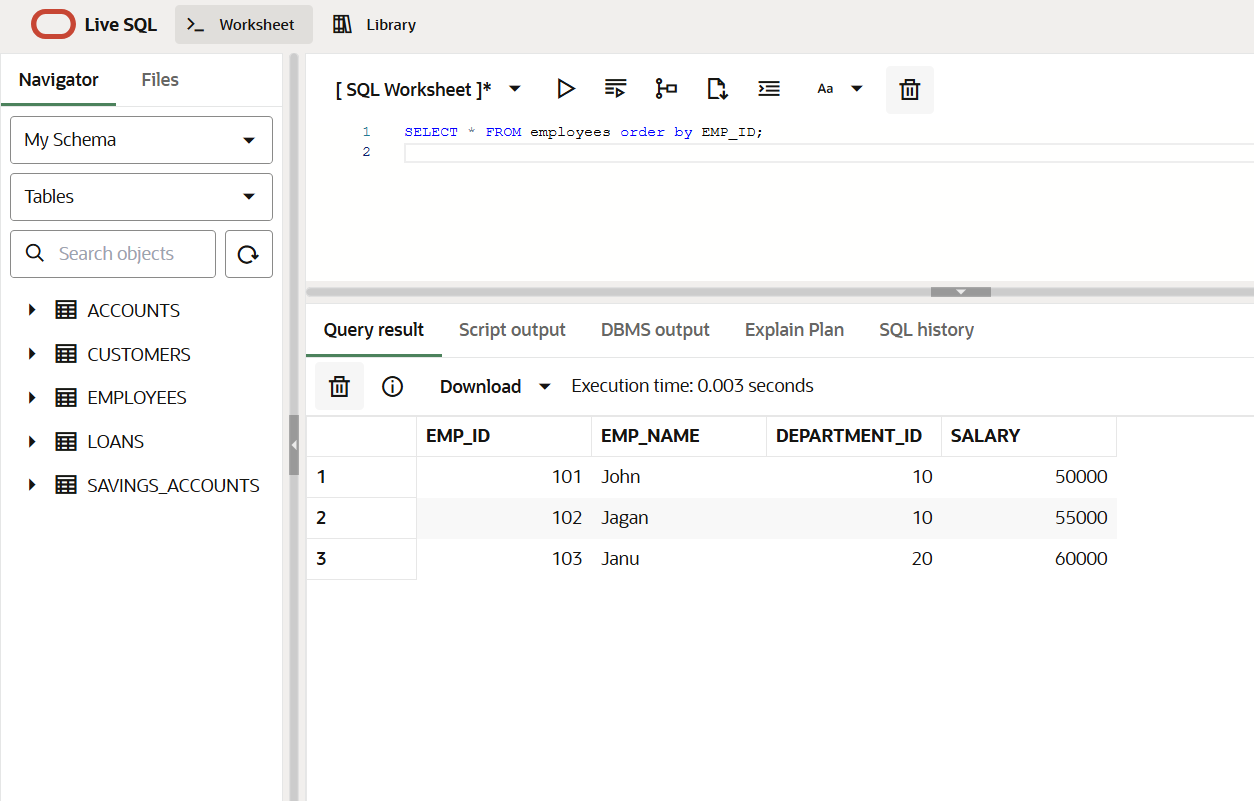
INSERT INTO accounts VALUES (203, 'Bob', 5000);

COMMIT;

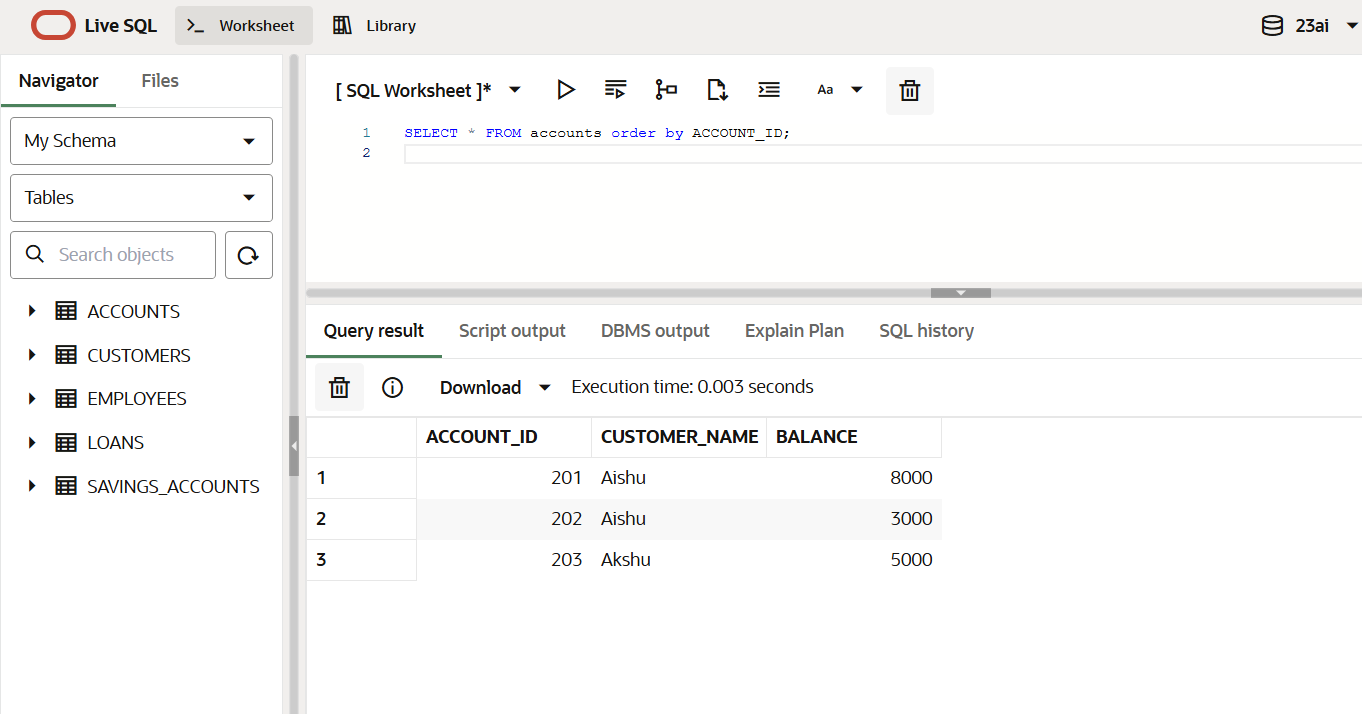
**SELECT \* FROM savings\_accounts order by ACCOUNT\_ID;**



**SELECT \* FROM employees ORDER BY EMP\_ID;**



**SELECT \* FROM accounts ORDER BY ACCOUNT\_ID;**



**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.

**PL/SQL Block:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01);

COMMIT;

END;

/

BEGIN

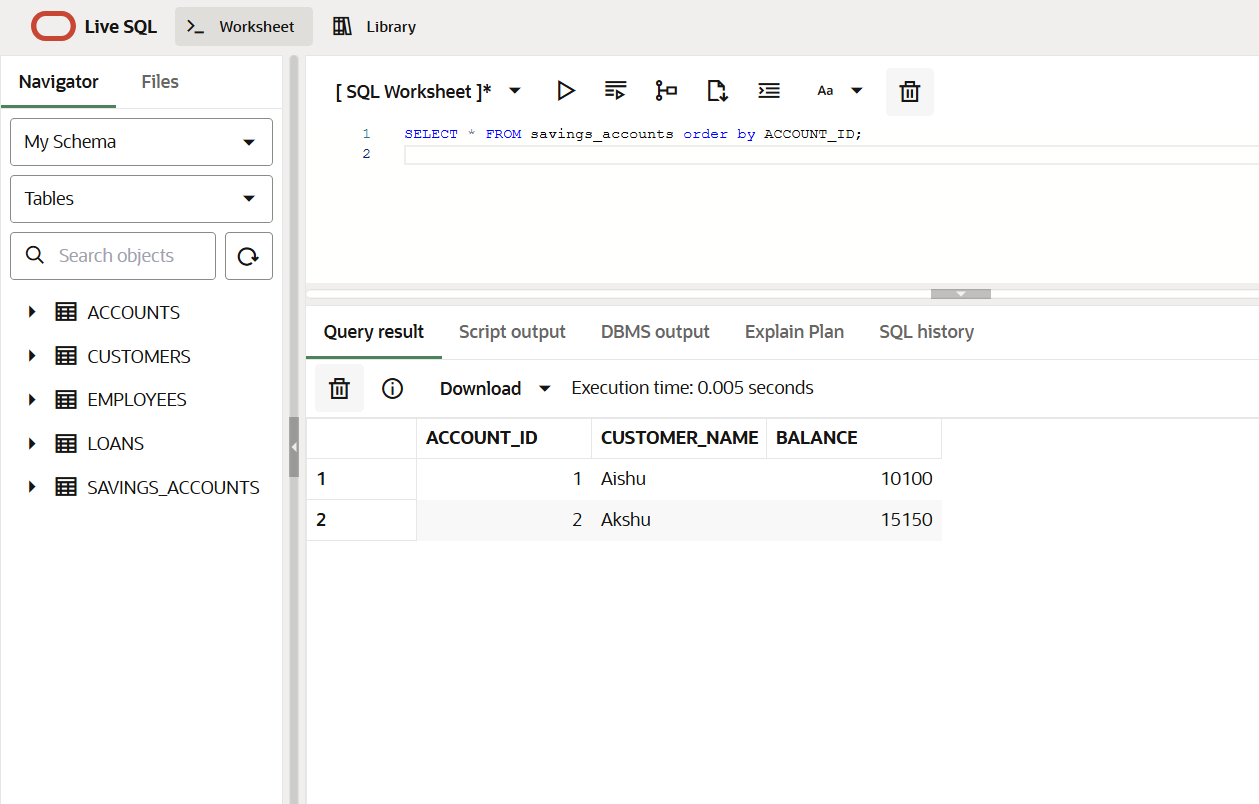
ProcessMonthlyInterest;

END;

/

**SELECT \* FROM savings\_accounts order by ACCOUNT\_ID;**

**Output:**



**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.

**PL/SQL Block:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE employees

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

WHERE department\_id = dept\_id;

COMMIT;

END;

/

BEGIN

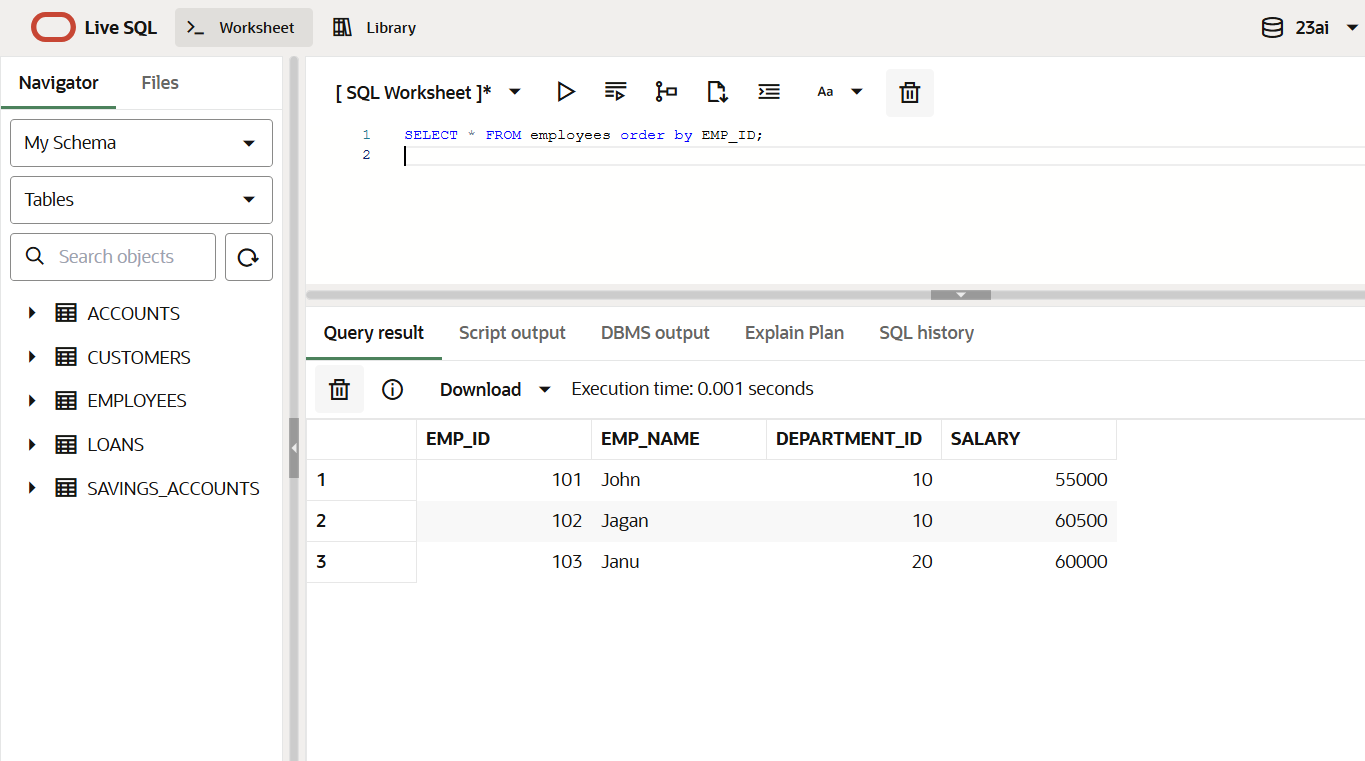
UpdateEmployeeBonus(10, 10);

END;

/

**SELECT \* FROM employees ORDER BY EMP\_ID;**

**Output:**



**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.

**PL/SQL Block:**

CREATE OR REPLACE PROCEDURE TransferFunds (

source\_acc\_id IN NUMBER,

target\_acc\_id IN NUMBER,

amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM accounts WHERE account\_id = source\_acc\_id;

IF v\_balance < amount THEN

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

ELSE

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = source\_acc\_id;

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = target\_acc\_id;

COMMIT;

END IF;

END;

/

BEGIN

TransferFunds(201, 202, 1000);

END;

/

**SELECT \* FROM accounts ORDER BY ACCOUNT\_ID;**

**Output:**

