**Exercise 1: Control Structures**

**Prerequisites:**

CREATE TABLE customer (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER,

balance NUMBER(10, 2),

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

INSERT INTO customer (customer\_id, name, age, balance)

VALUES (101, 'Rahul Kumar', 66, 150000.00),

(102, 'Arpit Raj', 47, 8900.00),

(103, 'Parth Kothari', 80, 12000.00),

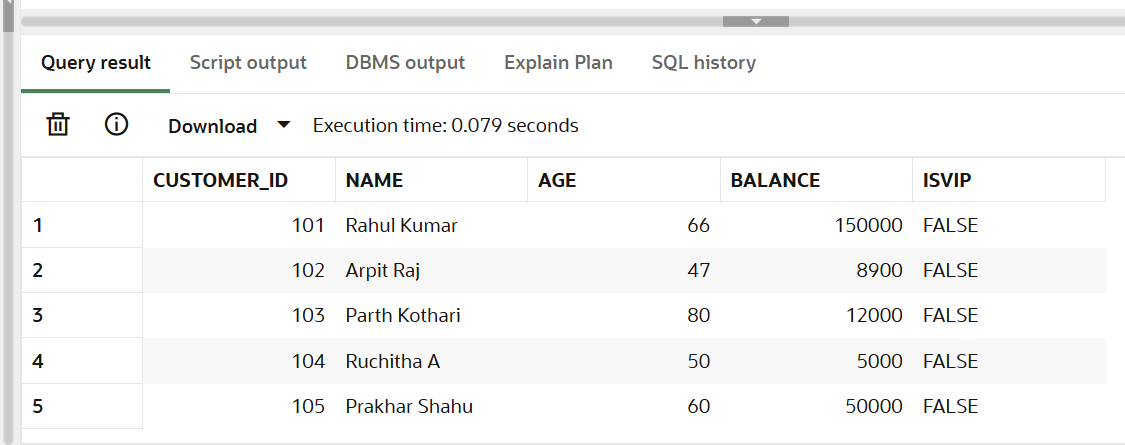
(104, 'Ruchitha A', 50, 5000.00),

(105, 'Prakhar Shahu', 60, 50000.00)

;

SELECT \* FROM customer;

**Output:**

****

CREATE TABLE loan (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER REFERENCES customer(customer\_id),

interest\_rate NUMBER(5, 2),

due\_date DATE

);

INSERT INTO loan (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (201, 101, 7.50, SYSDATE + 10),

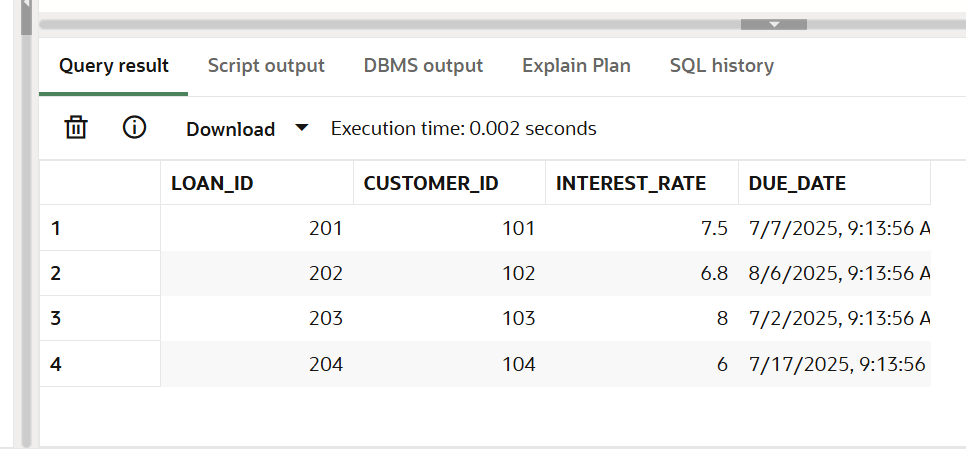
(202, 102, 6.80, SYSDATE + 40),

(203, 103, 8.00, SYSDATE + 5),

(204, 104, 6.00, SYSDATE + 20);

SELECT \* FROM loan;

**Output:**

****

**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.

BEGIN

FOR rec IN (SELECT customer\_id, interest\_rate

FROM loan

WHERE customer\_id IN (SELECT customer\_id FROM customer WHERE age > 60))

LOOP

UPDATE loan

SET interest\_rate = interest\_rate - 0.01

WHERE customer\_id = rec.customer\_id;

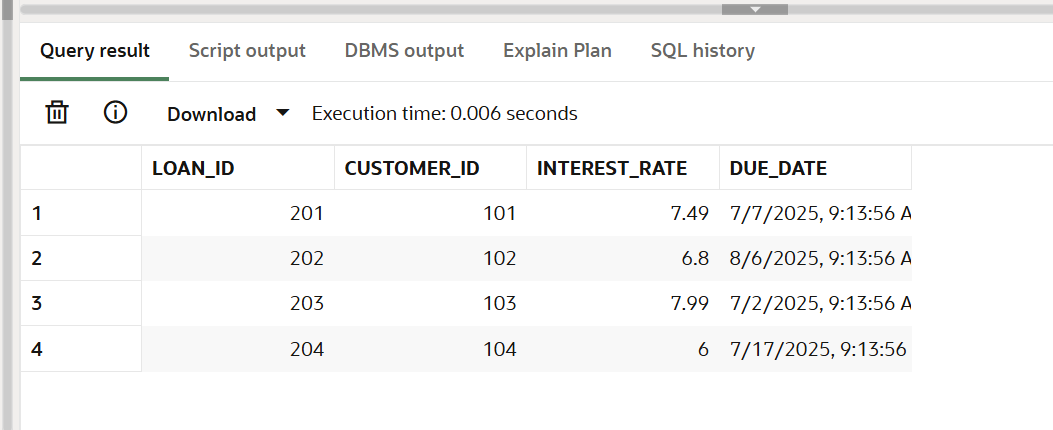
END LOOP;

COMMIT;

END;

SELECT \* FROM loan;

**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.

BEGIN

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

LOOP

UPDATE customer

SET IsVIP = 'TRUE'

WHERE customer\_id = rec.customer\_id;

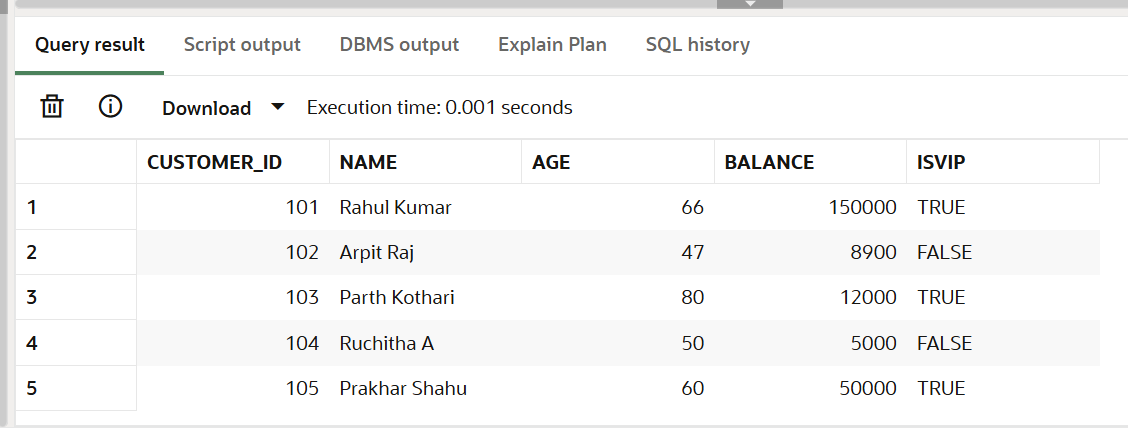
END LOOP;

COMMIT;

END;

SELECT \* FROM customer;

**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.

DECLARE

v\_due\_date loan.due\_date%TYPE;

v\_customer\_id loan.customer\_id%TYPE;

BEGIN

FOR rec IN (

SELECT customer\_id, due\_date

FROM loan

WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

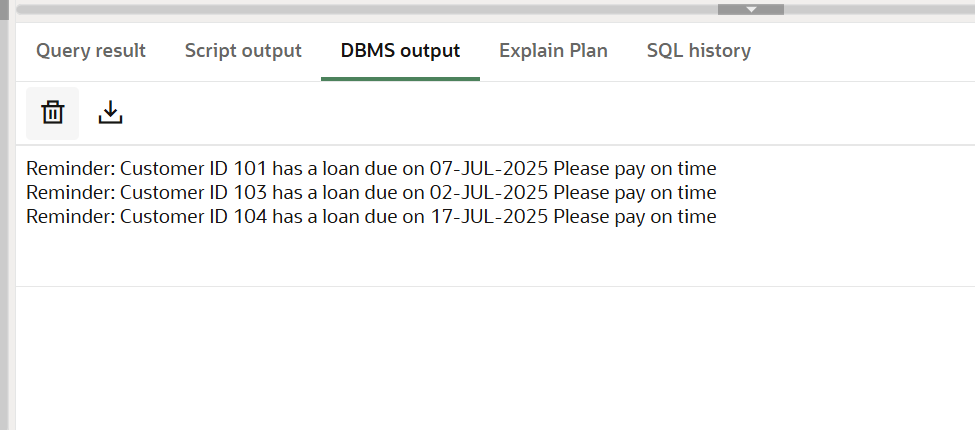
DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID ' || rec.customer\_id ||

' has a loan due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY')|| ' Please pay on time');

END LOOP;

END;

**Output:**

****

**Exercise 3: Stored Procedures**

**Prerequisites:**

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20),

balance NUMBER(10, 2)

);

INSERT INTO accounts (account\_id, customer\_id, account\_type, balance) VALUES

(1, 101, 'Savings', 10000.00),

(2, 102, 'Checking', 5000.00),

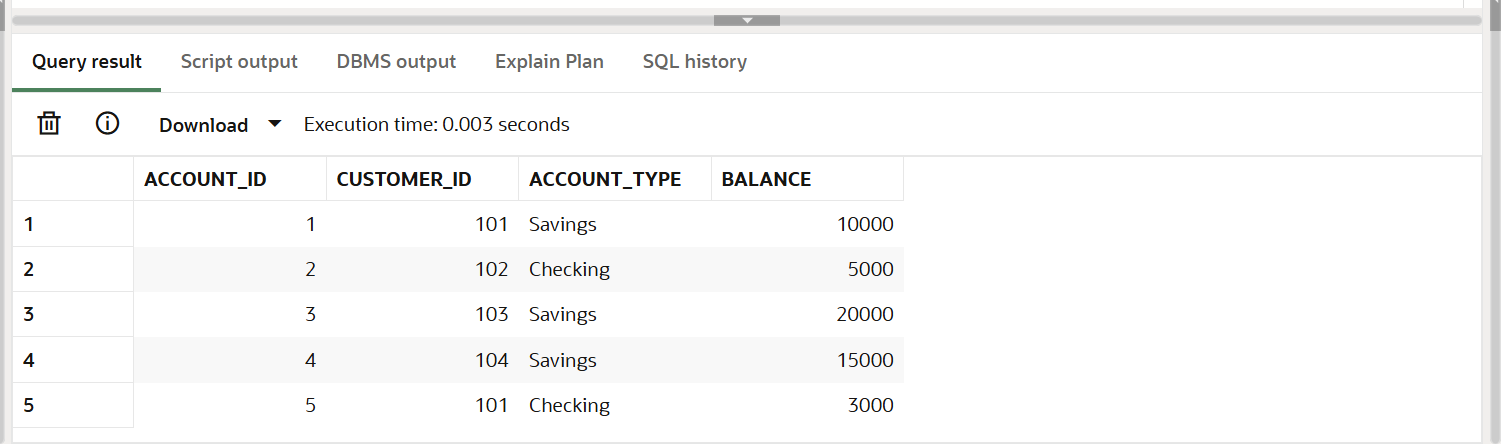
(3, 103, 'Savings', 20000.00),

(4, 104, 'Savings', 15000.00),

(5, 101, 'Checking', 3000.00);

SELECT \* FROM accounts;

**Output:**

****

CREATE TABLE employee (

employee\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

department VARCHAR2(30),

salary NUMBER(10, 2)

);

INSERT INTO employee (employee\_id, name, department, salary) VALUES

(1, 'Rahul', 'IT', 70000),

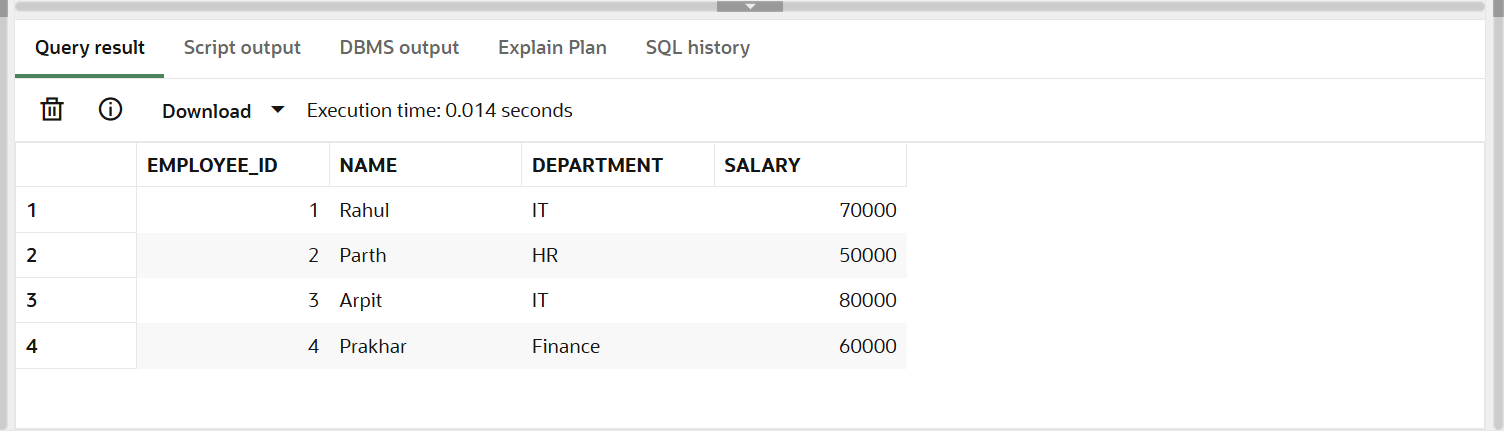
(2, 'Parth', 'HR', 50000),

(3, 'Arpit', 'IT', 80000),

(4, 'Prakhar', 'Finance', 60000);

SELECT \* FROM employee;

**Output:**



**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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

v\_old\_balance NUMBER;

v\_new\_balance NUMBER;

BEGIN

FOR acc IN (

SELECT account\_id, balance

FROM accounts

WHERE LOWER(account\_type) = 'savings'

) LOOP

v\_old\_balance := acc.balance;

v\_new\_balance := acc.balance + (acc.balance \* 0.01);

UPDATE accounts

SET balance = v\_new\_balance

WHERE account\_id = acc.account\_id;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.account\_id ||

' | Old Balance: ' || TO\_CHAR(v\_old\_balance, '99999.99') ||

' | New Balance: ' || TO\_CHAR(v\_new\_balance, '99999.99'));

END LOOP;

COMMIT;

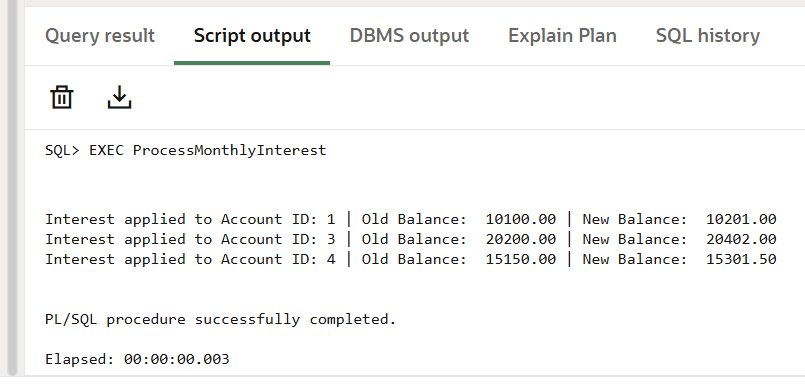
END;

/

SET SERVEROUTPUT ON;

EXEC ProcessMonthlyInterest;

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) AS

v\_old\_salary NUMBER;

v\_new\_salary NUMBER;

BEGIN

FOR emp IN (

SELECT employee\_id, salary

FROM employee

WHERE LOWER(department) = LOWER(dept\_name)

) LOOP

v\_old\_salary := emp.salary;

v\_new\_salary := emp.salary + (emp.salary \* bonus\_percent / 100);

UPDATE employee

SET salary = v\_new\_salary

WHERE employee\_id = emp.employee\_id;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.employee\_id ||

' | Old Salary: ' || TO\_CHAR(v\_old\_salary, '99999.99') ||

' | New Salary: ' || TO\_CHAR(v\_new\_salary, '99999.99'));

END LOOP;

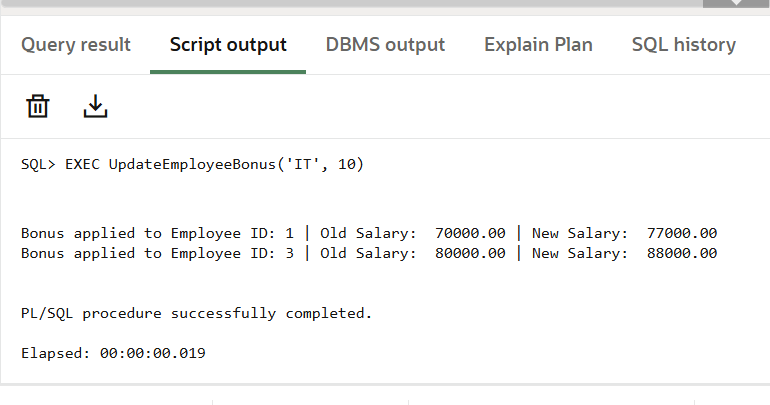
COMMIT;

END;

/

EXEC UpdateEmployeeBonus('IT', 10);

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account\_id IN NUMBER,

to\_account\_id IN NUMBER,

amount IN NUMBER

) AS

insufficient\_balance EXCEPTION;

v\_balance NUMBER;

BEGIN

-- Checking the balance of source account

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = from\_account\_id

FOR UPDATE;

IF v\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

-- Deducting the balance from source account

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = from\_account\_id;

-- Adding balance to destination account

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = to\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Transferred Amount ' || amount ||

' from Account ' || from\_account\_id ||

' to Account ' || to\_account\_id);

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in account ID ' || from\_account\_id);

ROLLBACK;

WHEN NO\_DATA\_FOUND THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

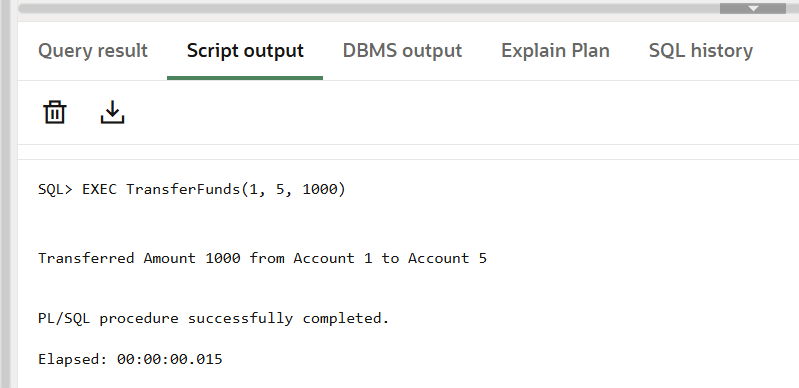
ROLLBACK;

END;

/

EXEC TransferFunds(1, 5, 1000);

**Output:**

****