**Exercise 1: Control Structures**

**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.

CREATE TABLE customers (

customer\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

balance NUMBER(12, 2),

loan\_interest\_rate NUMBER(5, 2),

IsVIP CHAR(1) DEFAULT 'N'

);

INSERT INTO customers (name, age, balance, loan\_interest\_rate) VALUES

('Alice', 65, 15000, 5.5),

('Bob', 45, 8000, 6.0),

('Charlie', 70, 11000, 4.8),

('Diana', 30, 9500, 5.2);

COMMIT;

BEGIN

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

IF cust\_rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'Y'

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

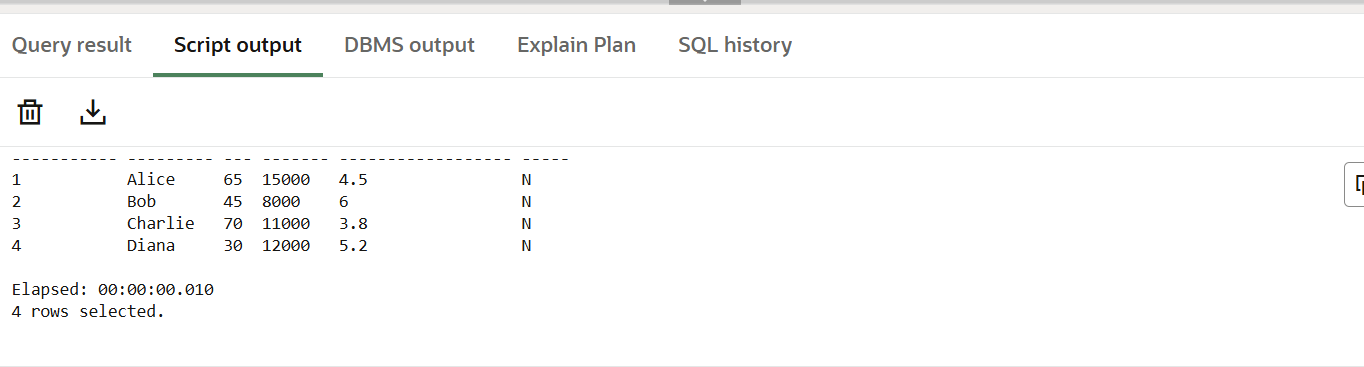
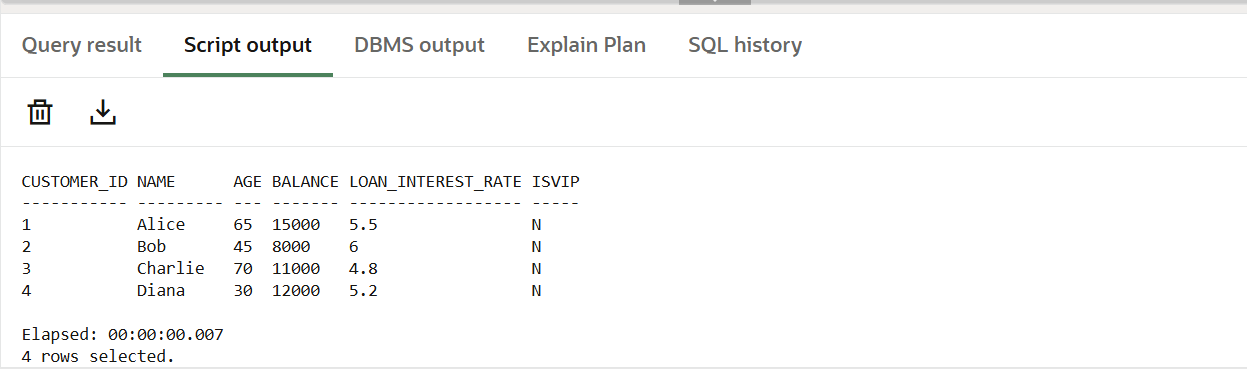
END IF;

END LOOP;

COMMIT;

END;

SELECT name, balance, IsVIP FROM customers;



**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.

DECLARE

CURSOR vip\_cursor IS

SELECT customer\_id, balance

FROM customers;

BEGIN

FOR cust\_rec IN vip\_cursor LOOP

IF cust\_rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'TRUE'

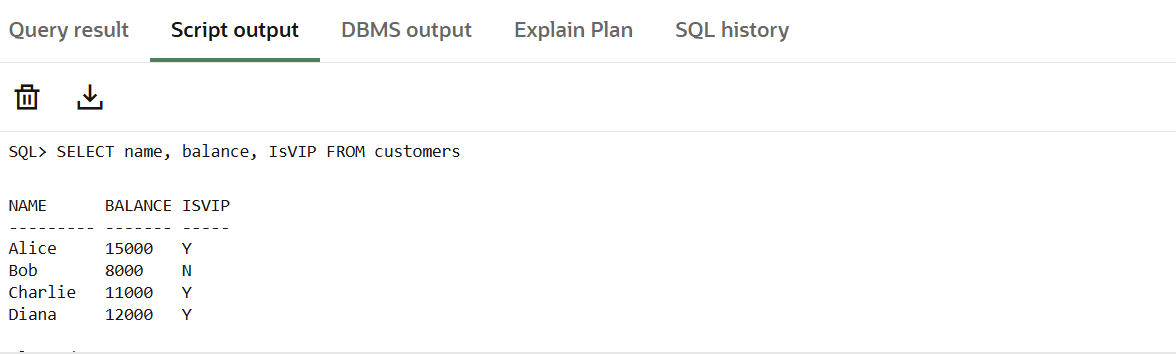
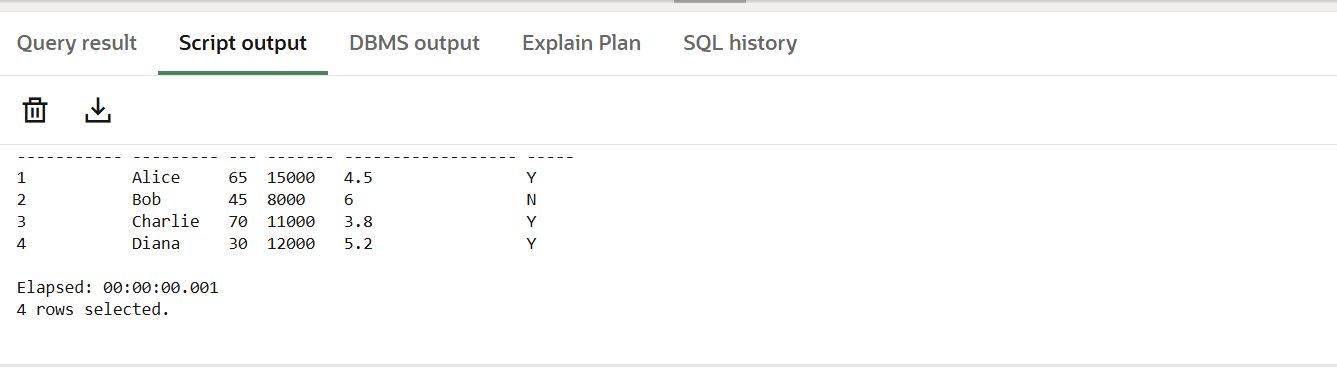
WHERE customer\_id = cust\_rec.customer\_id;

END IF;

END LOOP;

COMMIT;

END;



**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

CREATE TABLE loans (

loan\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES customers(customer\_id),

due\_date DATE

);

INSERT INTO loans (customer\_id, due\_date) VALUES (1, SYSDATE + 10); -- Due soon

INSERT INTO loans (customer\_id, due\_date) VALUES (2, SYSDATE + 35); -- Not due yet

INSERT INTO loans (customer\_id, due\_date) VALUES (3, SYSDATE + 5); -- Due soon

INSERT INTO loans (customer\_id, due\_date) VALUES (4, SYSDATE + 20); -- Due soon

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

FOR loan\_rec IN (

SELECT l.loan\_id,

l.customer\_id,

c.name AS customer\_name,

l.due\_date

FROM loans l

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

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

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || loan\_rec.customer\_name ||

', your Loan ID ' || loan\_rec.loan\_id ||

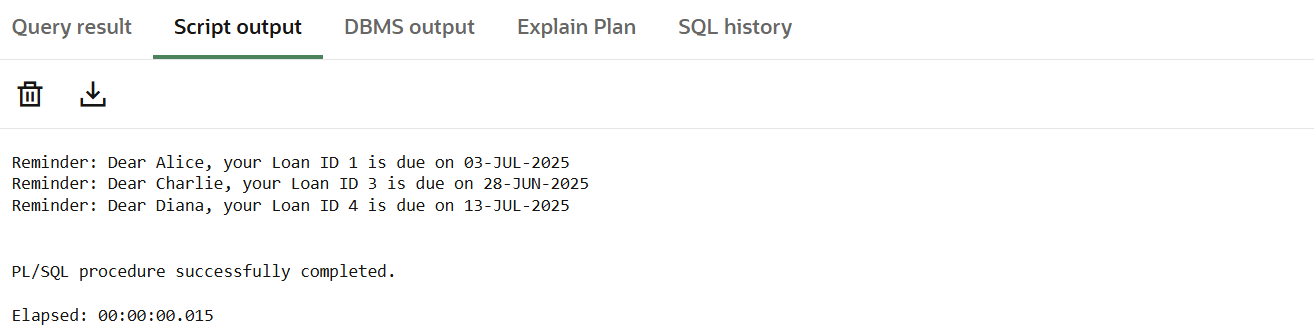
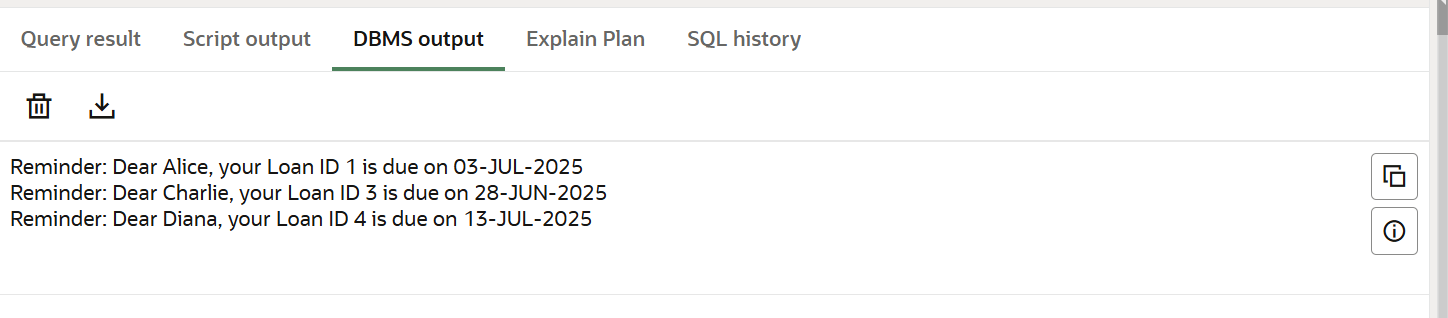
' is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY')

);

END LOOP;

END;

/



**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.

CREATE TABLE accounts (

account\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20), -- e.g., 'savings', 'checking'

balance NUMBER(12, 2)

);

INSERT INTO accounts (customer\_id, account\_type, balance) VALUES (1, 'savings', 1000);

INSERT INTO accounts (customer\_id, account\_type, balance) VALUES (2, 'checking', 2000);

INSERT INTO accounts (customer\_id, account\_type, balance) VALUES (3, 'savings', 3000);

INSERT INTO accounts (customer\_id, account\_type, balance) VALUES (4, 'savings', 500);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

-- Apply 1% interest to all savings accounts

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'savings';

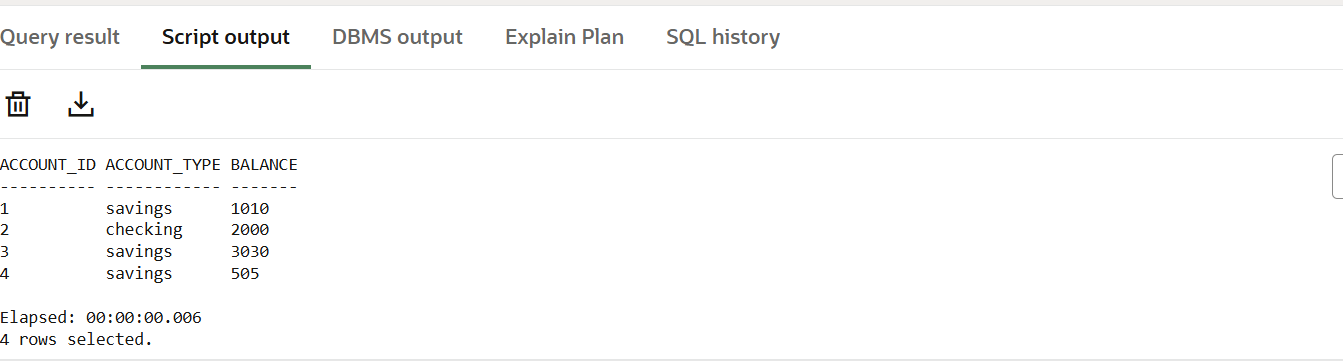
COMMIT;

END;

/

EXEC ProcessMonthlyInterest;

SELECT account\_id, account\_type, balance FROM accounts;



**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 TABLE employees (

emp\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR2(100),

department\_id NUMBER,

salary NUMBER(10, 2)

);

INSERT INTO employees (name, department\_id, salary) VALUES ('John', 101, 50000);

INSERT INTO employees (name, department\_id, salary) VALUES ('Alice', 102, 60000);

INSERT INTO employees (name, department\_id, salary) VALUES ('Bob', 101, 55000);

INSERT INTO employees (name, department\_id, salary) VALUES ('Diana', 103, 62000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

)

AS

BEGIN

UPDATE employees

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

WHERE department\_id = p\_dept\_id;

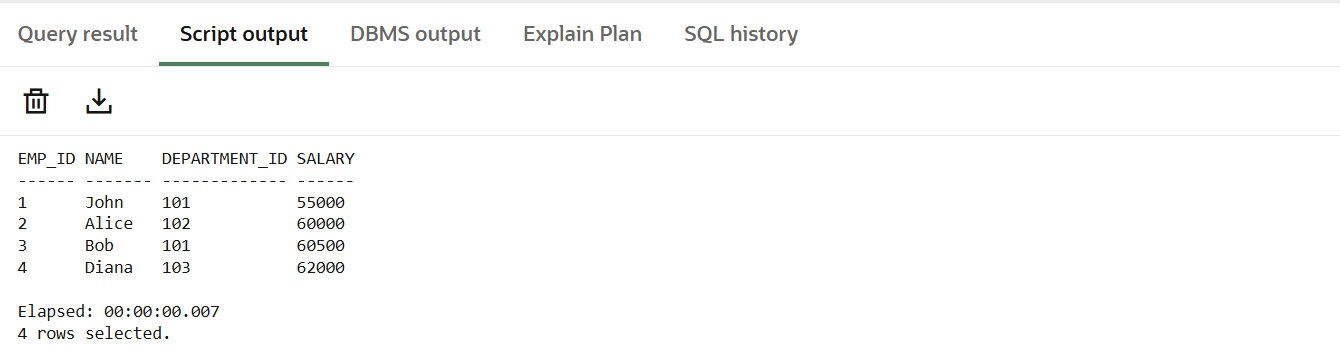
COMMIT;

END;

/

EXEC UpdateEmployeeBonus(101, 10);

SELECT emp\_id, name, department\_id, salary FROM employees;



**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 TABLE accounts (

account\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20),

balance NUMBER(12, 2)

);

INSERT INTO accounts (customer\_id, account\_type, balance) VALUES (1, 'savings', 10000);

INSERT INTO accounts (customer\_id, account\_type, balance) VALUES (2, 'checking', 5000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

AS

v\_from\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE account\_id = p\_from\_account\_id

FOR UPDATE;

-- Check if sufficient balance exists

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Deduct from source account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account\_id;

-- Add to destination account

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account\_id;

COMMIT;

END;

/

-- Transfer 2000 from account 1 to account 2

EXEC TransferFunds(1, 2, 2000);

