**Exercise 1: Control Structure**

**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.

**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.

**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.

**Scenario 1:**

--Scenario 1- Apply 1% discount to age>60

BEGIN

FOR rec IN (

SELECT c.cus\_id, l.loanid

FROM CUSTOMERS1 c

JOIN LOANS1 l ON c.cus\_id = l.cus\_id

WHERE c.age > 60

) LOOP

UPDATE LOANS1

SET INT\_RATE = INT\_RATE - 1

WHERE loanid = rec.loanid;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to customer ID: ' || rec.cus\_id);

END LOOP;

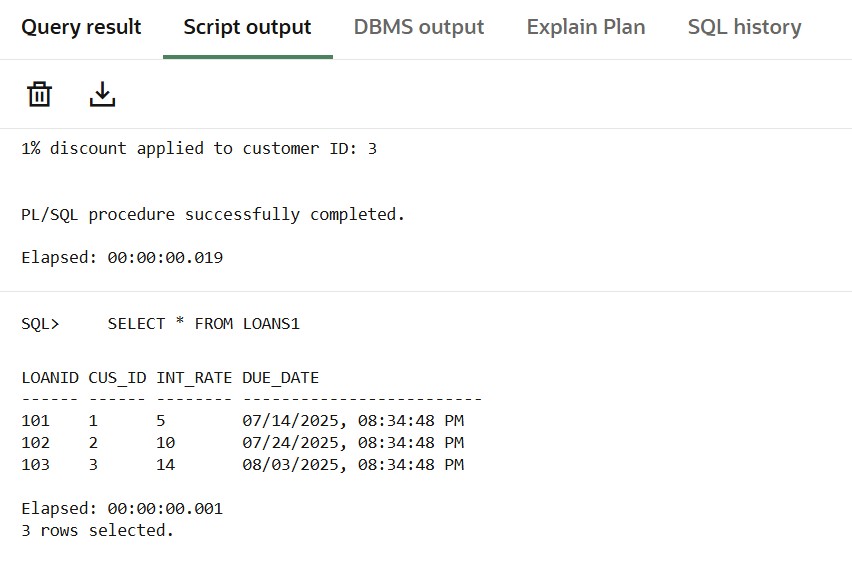
COMMIT;

END;

/

SELECT \* FROM LOANS1;

**Output Scenario1:**



**Scenario 2:**

--Scenario 2-Promote to vip if balance>10000

BEGIN

FOR rec IN(SELECT \* FROM CUSTOMERS1)LOOP

IF rec.balance>10000 THEN

UPDATE CUSTOMERS1

SET IsVIP='TRUE'

WHERE cus\_id=rec.cus\_id;

DBMS\_OUTPUT.PUT\_LINE(rec.name || 'promoted to vip');

END IF;

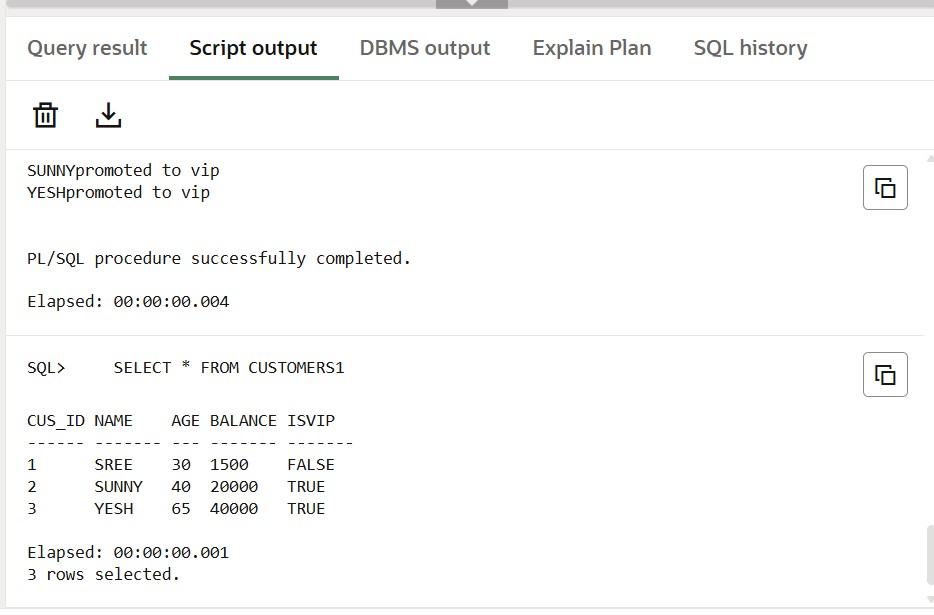
END LOOP;

END;

/

SELECT \* FROM CUSTOMERS1;

**Output Scenario2:**



**Scenario 3:**

--scenario 3 -Send reminders for loans due in next 30 days

BEGIN

FOR rec IN(

SELECT l.loanid,c.name,l.DUE\_DATE

FROM LOANS1 l

JOIN CUSTOMERS1 c ON l.cus\_id=c.cus\_id

WHERE l.DUE\_DATE<=SYSDATE+30

)LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || rec.name || 'has a loan due on ' || TO\_CHAR(rec.DUE\_DATE,'dd-mm-yyyy'));

END LOOP;

END;

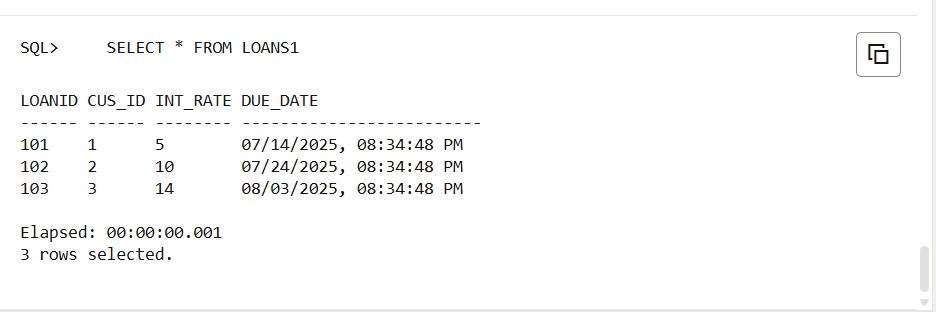
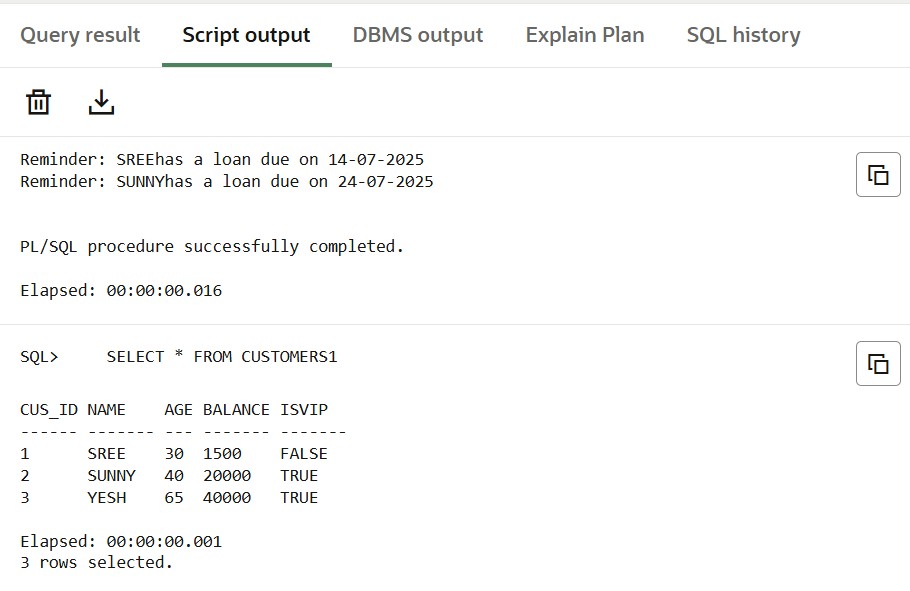
/

--view updated results

SELECT \* FROM CUSTOMERS1;

SELECT \* FROM LOANS1;

**Output Scenario3:**



**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.

**Scenario 1:**

--Scenario 1: Apply 1% Interest to All Savings Accounts

CREATE or REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE savings\_acc

SET balance=balance\*1.01;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('monthly interset applied.');

END;

/

BEGIN

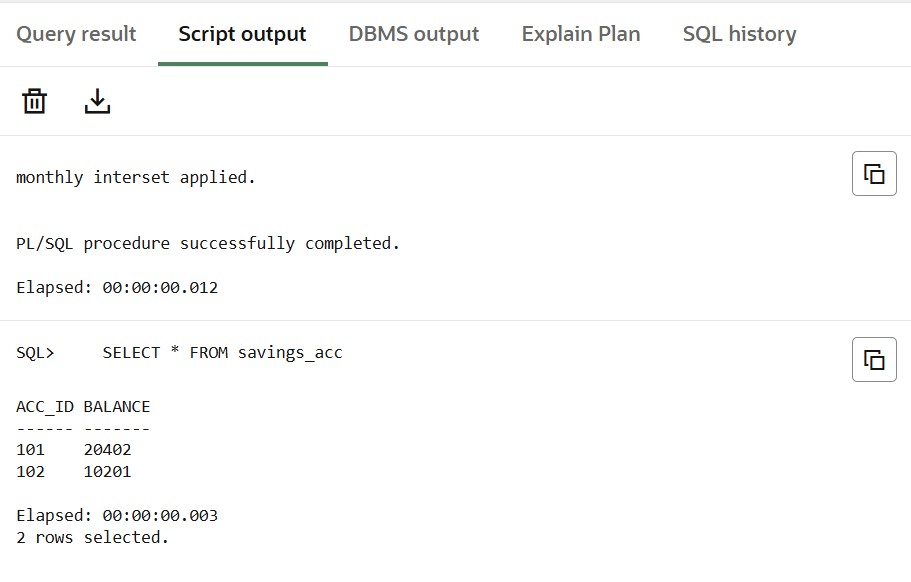
ProcessMonthlyInterest;

END;

/

SELECT \* FROM savings\_acc;

**Output Scenario1:**



**Scenario 2:**

-- Scenario 2: Bonus for Employees by Dept

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN employees.dept\_id%TYPE, p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE employees

SET salary = salary \* (1 + p\_bonus\_pct / 100)

WHERE dept\_id = p\_dept\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonuses applied to dept: ' || p\_dept\_id);

END UpdateEmployeeBonus;

/

-- Test it for department 11 with a 10% bonus

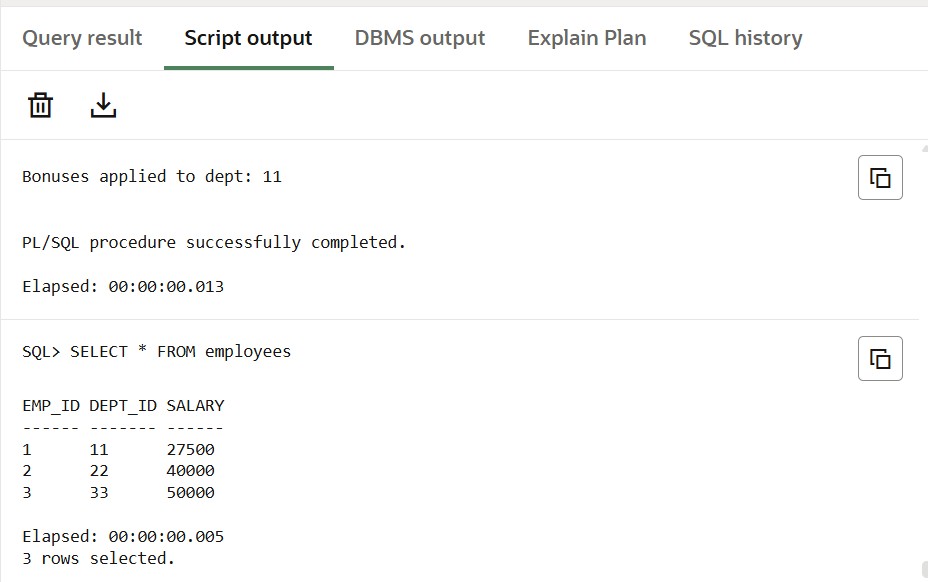
BEGIN

UpdateEmployeeBonus(11, 10);

END;

/

SELECT \* FROM employees; **Output Scenario2:**



**Scenario 3:**

-- Scenario 3: Transfer Funds with Validation

CREATE OR REPLACE PROCEDURE TransferFunds ( p\_from\_acct IN acc\_balances.acc\_id%TYPE, p\_to\_acct IN acc\_balances.acc\_id%TYPE, p\_amount IN NUMBER

) AS v\_balance NUMBER; BEGIN

-- Fetch current balance

SELECT balance INTO v\_balance

FROM acc\_balances

WHERE acc\_id = p\_from\_acct;

-- Validate

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in account ' || p\_from\_acct);

END IF;

-- Debit and credit

UPDATE acc\_balances

SET balance = balance - p\_amount

WHERE acc\_id = p\_from\_acct;

UPDATE acc\_balances

SET balance = balance + p\_amount

WHERE acc\_id = p\_to\_acct;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from ' || p\_from\_acct || ' to ' || p\_to\_acct);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END TransferFunds;

/

-- Before transfer

SELECT \* FROM acc\_balances;

BEGIN

TransferFunds(101, 103, 5000);

END;

/

-- After transfer

SELECT \* FROM acc\_balances; **Output Scenario3:**

