**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.**

Begin

FOR customer IN (Select CustomerID, Name, DOB FROM Customers)

LOOP

IF (MONTHS\_BETWEEN (SYSDATE, customer.DOB) / 12) > 60 THEN

UPDATE Loans SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = customer.CustomerID;

dbms\_output.put\_line('1% interest discount given to Customer ID: ' || customer.CustomerID);

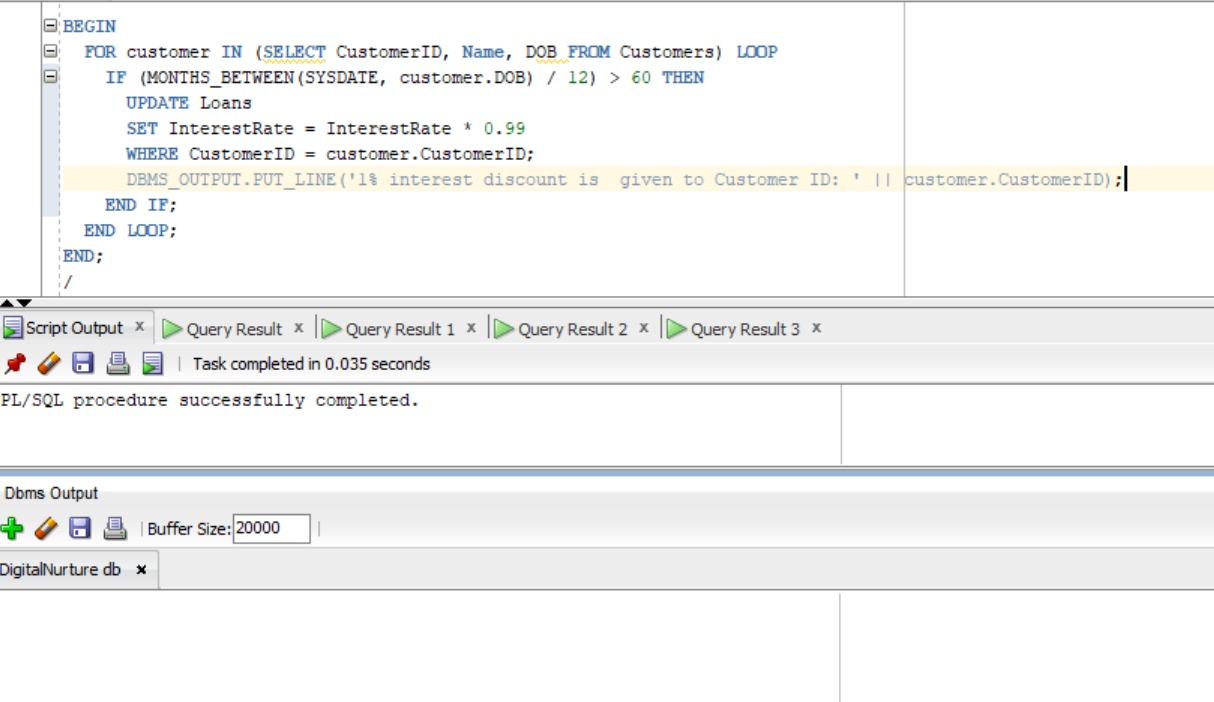
END IF;

END LOOP;

End;

/

**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.**

--ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N';

Begin

FOR c IN (SELECT CustomerID, Balance FROM Customers)

LOOP

IF c.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'Y' WHERE CustomerID = c.CustomerID;

dbms\_output.put\_line('Customer ID ' || c.CustomerID || ' is now marked as VIP.');

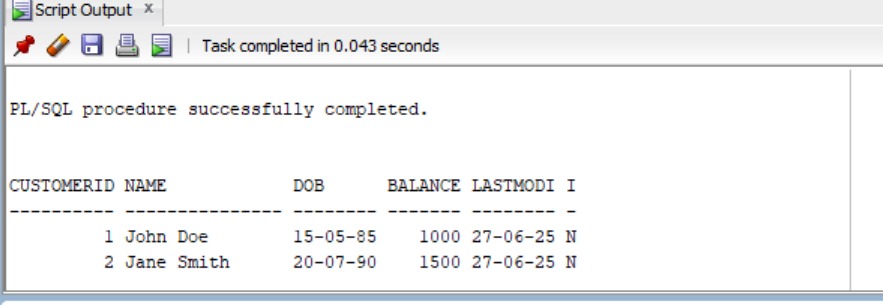
END IF;

END LOOP;

END;

/

Select \* 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.**

set SERVEROUTPUT on;

BEGIN

FOR due IN (Select customer.CustomerID, customer.Name, l.loanID, l.enddate FROM Customers customer

JOIN Loans l on customer.CustomerID = l.CustomerID where l.enddate between sysdate and sysdate+30)

LOOP

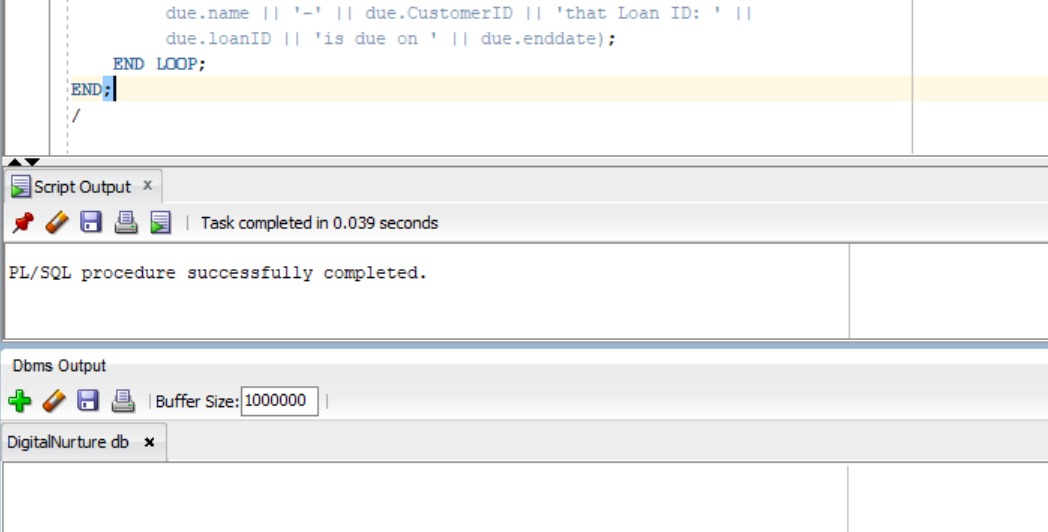
DBMS\_OUTPUT.PUT\_LINE('Remainder for ' ||

due.name || '-' || due.CustomerID || 'that Loan ID: ' ||

due.loanID || 'is due on ' || due.enddate);

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 or replace PROCEDURE ProcessMonthlyInterest IS

BEGIN

update Accounts set Balance = Balance + (Balance \* 0.01) where accounttype = 'Savings';

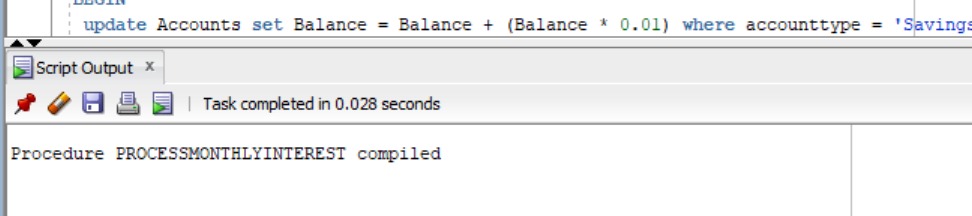
commit;

dbms\_output.put\_line('Monthly Interest applied to all the accounts.');

End;

/

**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\_p IN number) AS

Begin

UPDATE employees set salary = salary + (salary \* bonus\_p) where department = dept\_name;

commit;

dbms\_output.put\_line('Bonus has been applied to '||dept\_name||' department.');

END;

/

DECLARE

e\_dept VARCHAR2(50) := 'HR';

e\_bonus number := 0.02;

BEGIN

UpdateEmployeeBonus(e\_dept, e\_bonus);

FOR employe IN (SELECT EmployeeID, Name, Salary FROM Employees WHERE Department = e\_dept)

LOOP DBMS\_OUTPUT.PUT\_LINE('EmployeeID: ' || employe.EmployeeID || ', Name: ' || employe.Name || ', Updated Salary: ' || employe.Salary);

END LOOP;

dbms\_output.put\_line('Bonus applied successfully');

END;

/

SELECT \* FROM Employees WHERE Department = 'HR';

**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(amt\_from\_id IN number, amt\_to\_id IN number, transfer\_amt IN number ) IS

curr\_bal number;

BEGIN

select Balance INTO curr\_bal FROM Accounts WHERE AccountID = amt\_from\_id;

IF curr\_bal < transfer\_amt THEN

dbms\_output.put\_line(' Not sufficient balance to transfer.');

return;

END IF;

UPDATE Accounts SET Balance = Balance - transfer\_amt, LastModified = SYSDATE

WHERE AccountID = amt\_from\_id;

UPDATE Accounts SET Balance = Balance + transfer\_amt,LastModified = SYSDATE

WHERE AccountID = amt\_to\_id;

DBMS\_OUTPUT.PUT\_LINE('Successfully transfered!');

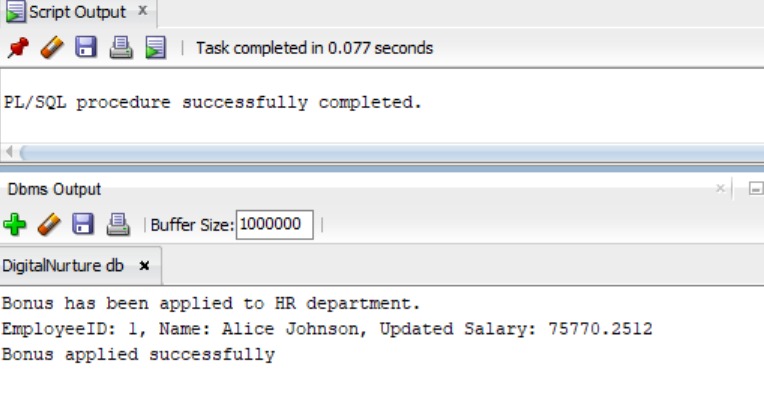
END;

/

BEGIN

TransferFunds(1, 2, 10);

END;

**OUTPUT:-**

