PLSQL\_Exercises

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.

**Program Code:**

DECLARE

CURSOR cur\_senior\_customers IS

SELECT l.LoanID, l.InterestRate, c.CustomerID

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60;

BEGIN

FOR rec IN cur\_senior\_customers LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to Loan ID ' || rec.LoanID ||

' for Customer ID ' || rec.CustomerID);

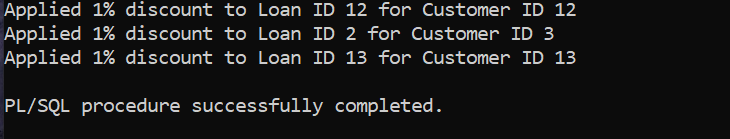
END LOOP;

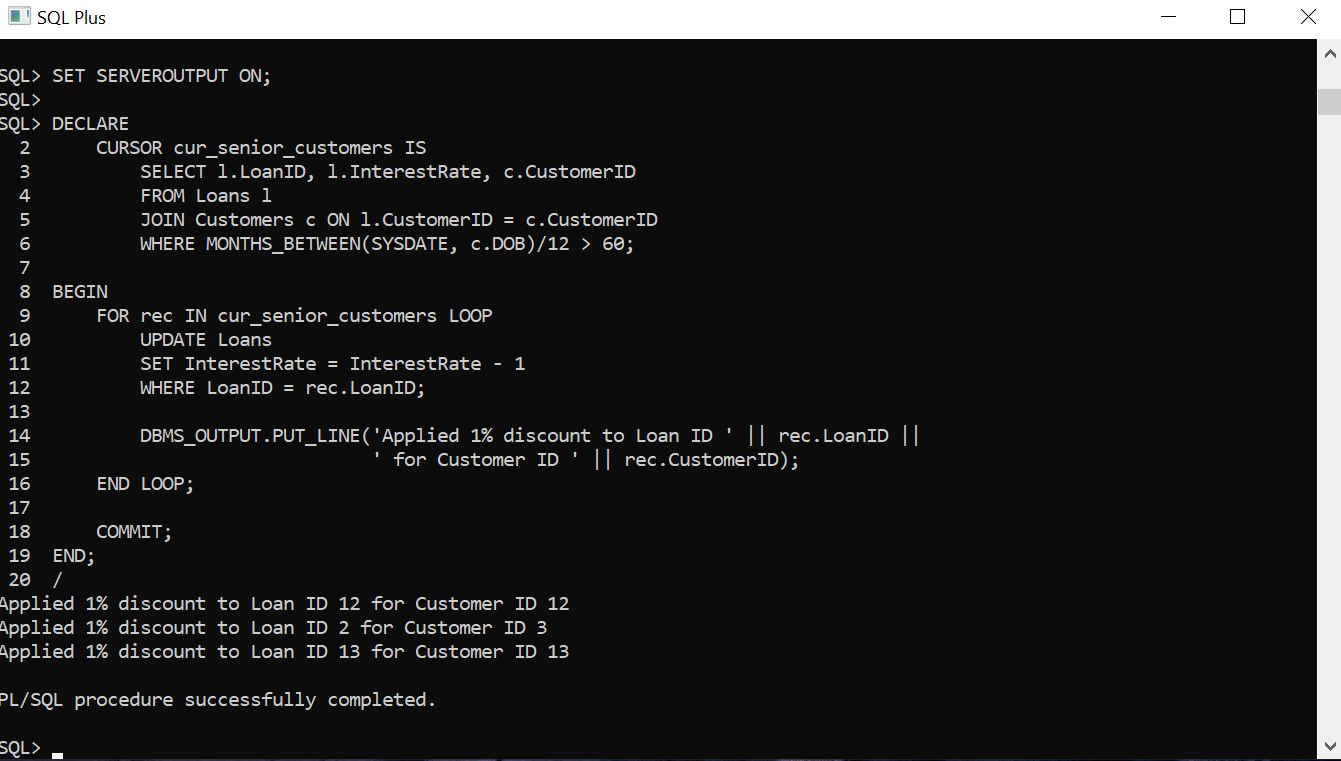
COMMIT;

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.

**Program code:**

DECLARE

CURSOR vip\_customers IS

SELECT CustomerID, Balance FROM Customers

WHERE Balance > 10000;

BEGIN

FOR rec IN vip\_customers LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.CustomerID ||

' promoted to VIP (Balance: $' || rec.Balance || ')');

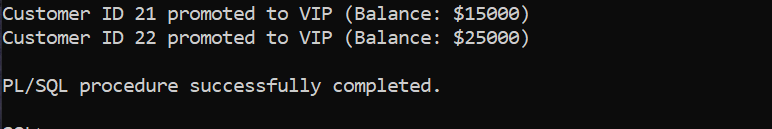
END LOOP;

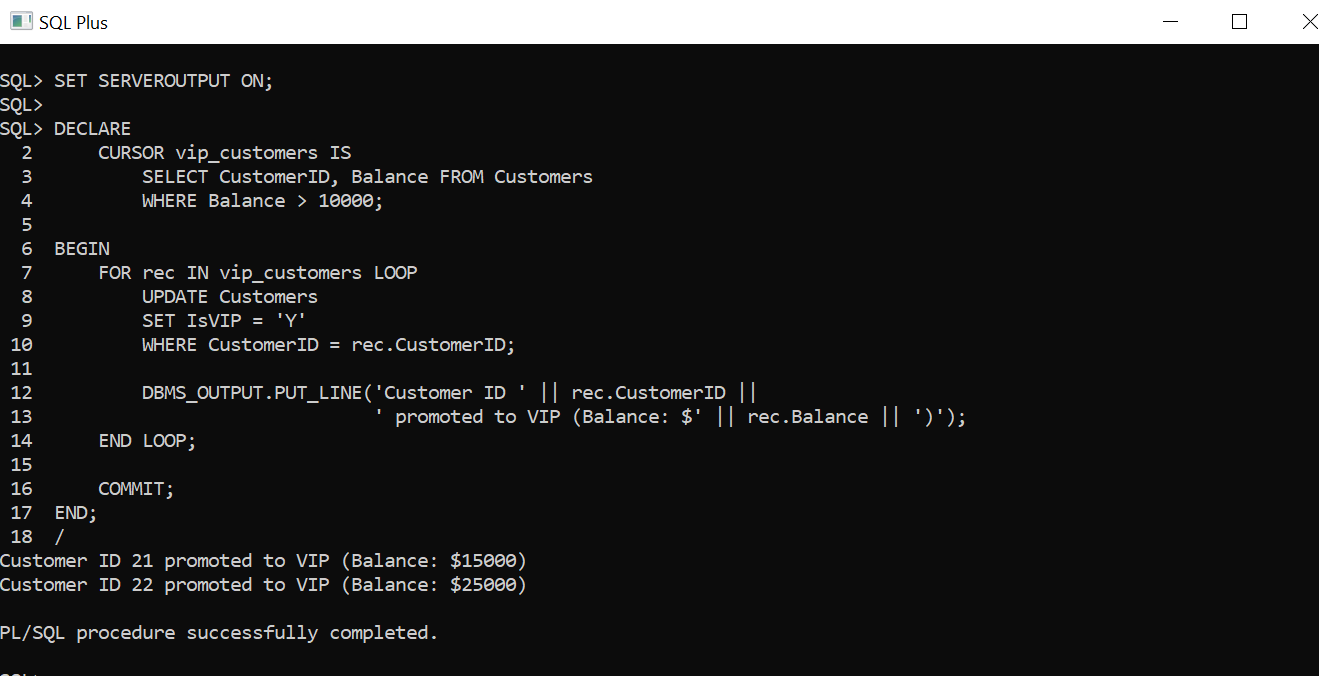
COMMIT;

END;

/

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.

**Program code:**

DECLARE

CURSOR due\_soon\_loans IS

SELECT l.LoanID, l.EndDate, c.Name, c.CustomerID

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN due\_soon\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for Customer ' || rec.Name ||

' (Customer ID: ' || rec.CustomerID ||

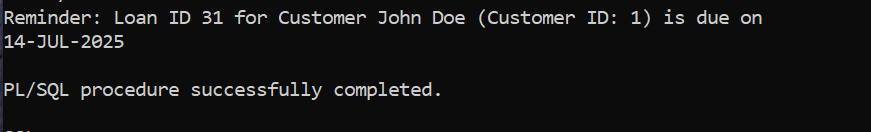
') is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

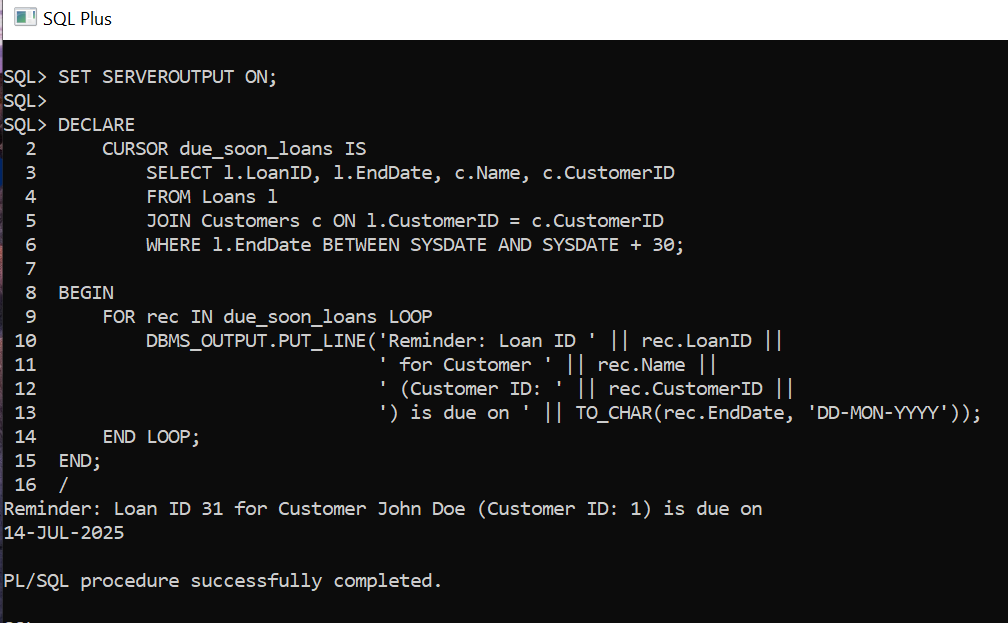
END LOOP;

END;

/

Output:





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.

**Program Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

-- Apply 1% interest to all savings accounts

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END;

/

To Run the Procedure:

SET SERVEROUTPUT ON;

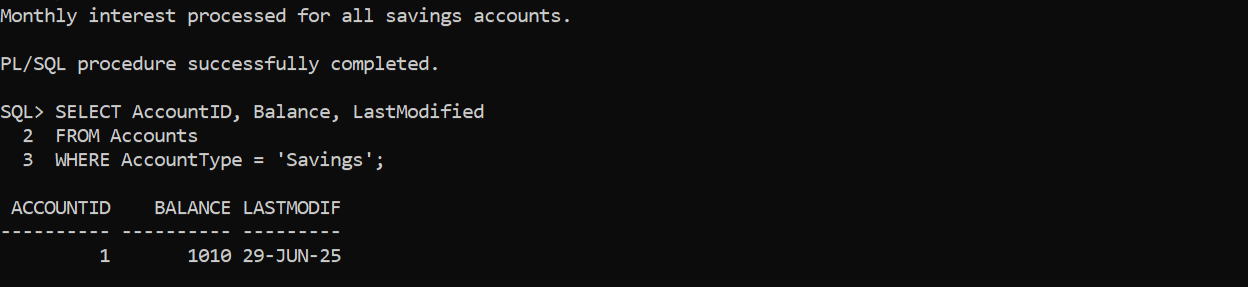
BEGIN

ProcessMonthlyInterest;

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.

**Program code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

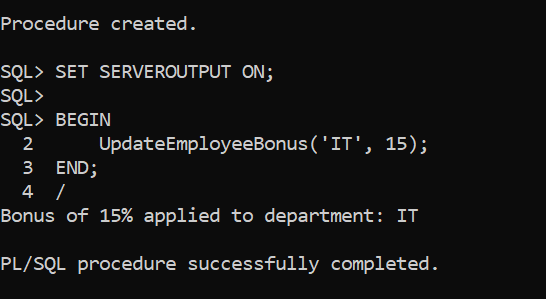
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to department: ' || p\_department);

END;

/

**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.

**Program code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

v\_from\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_seq.NEXTVAL, p\_from\_account\_id, SYSDATE, p\_amount, 'Debit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_seq.NEXTVAL, p\_to\_account\_id, SYSDATE, p\_amount, 'Credit');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_from\_account\_id || ' to Account ' || p\_to\_account\_id);

END;

/

**OUTPUT:**

