**PL/SQL**

***Exercise 1: Control Structures***

**Q1> 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.**

🡺Code

DECLARE

c\_id customers.customerId%type;

c\_dob customers.dob%type;

c\_age number;

c\_loanRate loans.interestRate%type;

c\_newLoanRate loans.interestRate%type;

BEGIN

for i in (select c.customerId, c.dob from customers c) LOOP

c\_id := i.customerId;

c\_dob := i.dob;

c\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, c\_dob)/12);

if(c\_age > 60) then

select l.interestRate into c\_loanRate

from loans l

where l.customerId = c\_id;

c\_newLoanRate := c\_loanRate – (c\_loanRate \* 0.01);

update loans

set interestRate = c\_newLoanRate

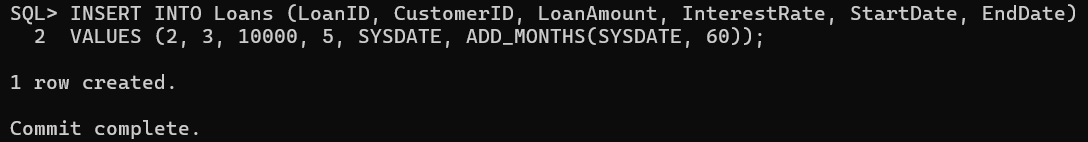
where customerId = c\_id;

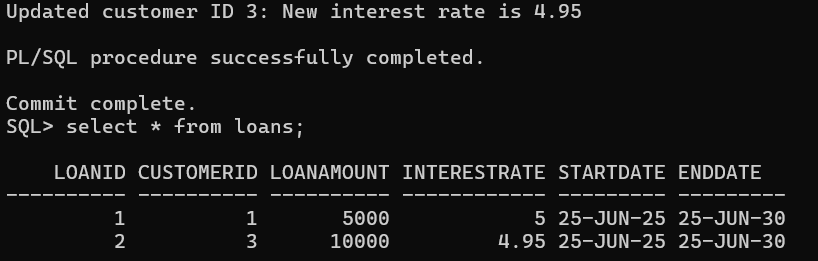
dbms\_output.put\_line('Updated customer ID ' || c\_id || ': New interest rate is ' || c\_newLoanRate);

end if;

end loop;

END;

/



**Q2> 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.**

🡺code

DECLARE

    c\_id customers.customerId%type;

    c\_balance accounts.balance%type;

BEGIN

    for i IN (select a.customerId, a.balance from accounts a) LOOP

        c\_id := i.customerId;

        c\_balance := i.balance;

        if (c\_balance > 10000) then

            update customers

            set IsVIP = 'TRUE'

            where customerId = c\_id;

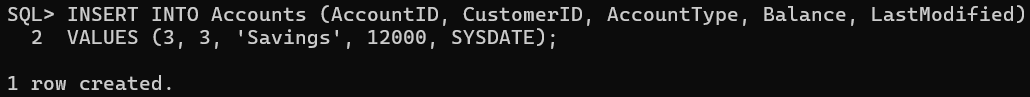
            dbms\_output.put\_line('Customer ID ' || c\_id || ' marked as VIP');

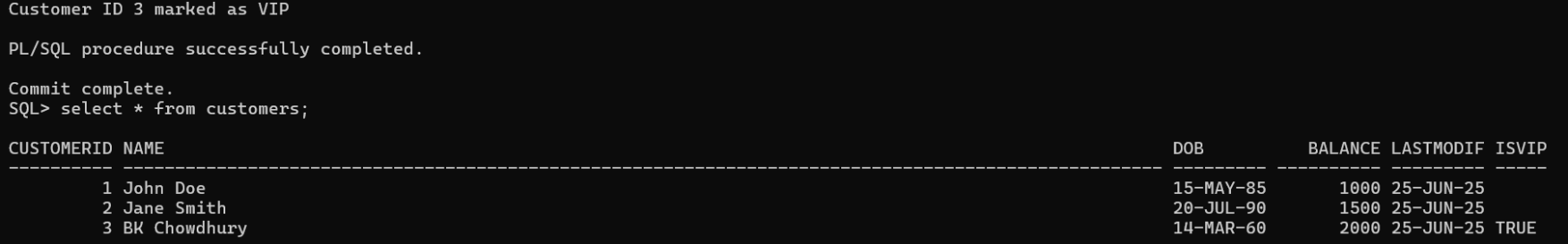
        end if;

    end loop;

END;

/





**Q3>Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.**

🡺code

DECLARE

    c\_id loans.customerId%type;

    c\_loanAmt loans.loanAmount%type;

    c\_dueDate loans.endDate%type;

BEGIN

    for i in (select customerId, loanAmount, endDate from loans where endDate BETWEEN SYSDATE AND SYSDATE+30) LOOP

        c\_id := i.customerId;

        c\_loanAmt := i.loanAmount;

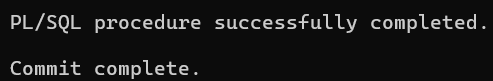
        c\_dueDate := i.endDate;

        dbms\_output.put\_line('Reminder: Customer ID ' || c\_id || ', your loan of $' || ' is due on ' || to\_Char(c\_dueDate, 'DD/MM/YYYY'));

    end loop;

END;

/



(no entry found with dues)

**Exercise 3: Stored Procedures**

**Q1>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.**

**🡺code**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

    UPDATE Accounts

    SET Balance = Balance + (Balance\*0.01)

    WHERE AccountType = 'Savings';

    COMMIT;

    dbms\_output.put\_line('Interest of 1% applied to savings account!');

EXCEPTION

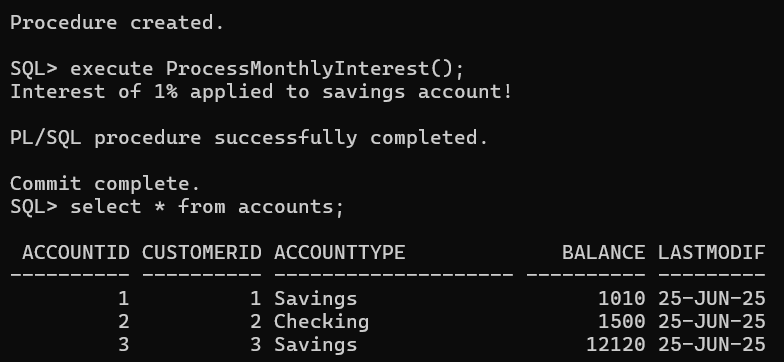
    when others then

        rollback;

        dbms\_output.put\_line('error! '|| SQLERRM);

END ProcessMonthlyInterest;

/



**Q2>Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.**

**🡺code**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    dept IN varchar2,

    bonusPer IN number) AS

BEGIN

    UPDATE Employees

    SET salary = salary + (salary\*bonusPer)

    WHERE department = dept;

    COMMIT;

    dbms\_output.put\_line('Salaries updated for dept ' || dept);

EXCEPTION

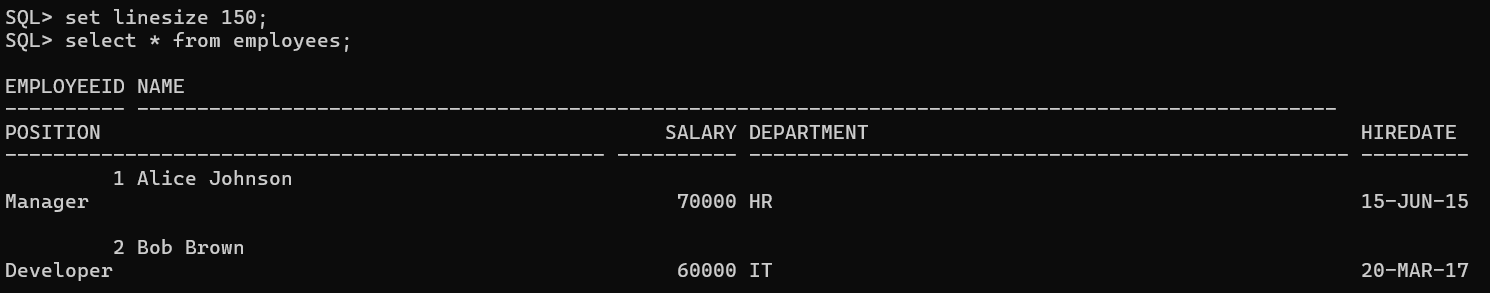
    WHEN others then

        rollback;

        dbms\_output.put\_line('error! ' || SQLERRM);

END UpdateEmployeeBonus;

/





Q3> **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.**

**🡺code**

CREATE OR REPLACE PROCEDURE TransferFunds(

    src\_id IN number,

    target\_id IN number,

    amount IN number) AS

    src\_balance accounts.balance%type;

BEGIN

    select balance into src\_balance from accounts

    where accountId = src\_id;

    if src\_balance < amount then

        dbms\_output.put\_line('insufficient balance!');

        return;

    end if;

    update accounts

    set balance = balance - amount

    where accountId = src\_id;

    update accounts

    set balance = balance + amount

    where accountId = target\_id;

    COMMIT;

    dbms\_output.put\_line('transaction successfull');

EXCEPTION

    when others then

        rollback;

        dbms\_output.put\_line('Error! ' || SQLERRM);

END TransferFunds;

/

