**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:**

BEGIN

FOR rec IN (SELECT \* FROM customers) LOOP

IF rec.age > 60 THEN

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - 1

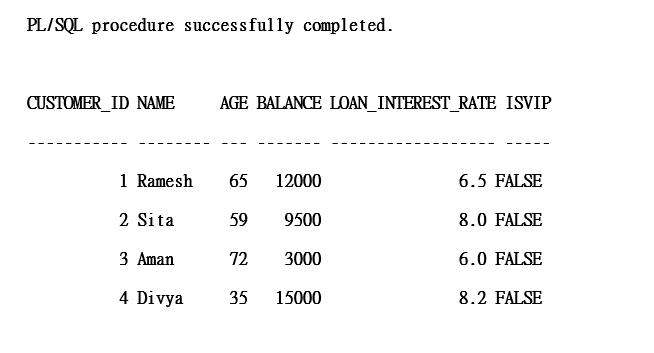
WHERE customer\_id = rec.customer\_id;

END IF;

END LOOP;

END;

/



**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**

BEGIN

FOR rec IN (SELECT \* FROM customers) LOOP

IF rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'TRUE'

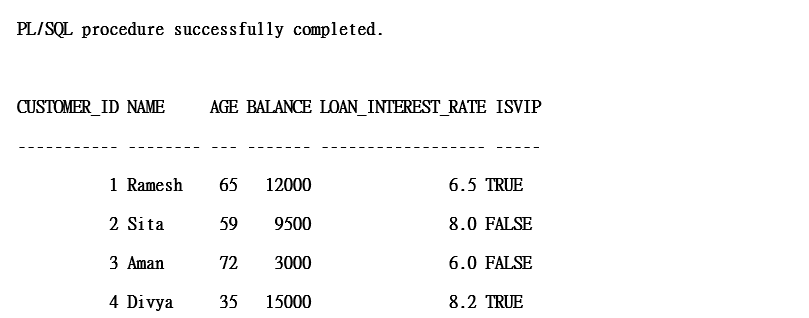
WHERE customer\_id = rec.customer\_id;

END IF;

END LOOP;

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.

**Program**

BEGIN

FOR rec IN (

SELECT l.loan\_id, l.due\_date, c.name

FROM loans l

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

WHERE l.due\_date <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || rec.name ||

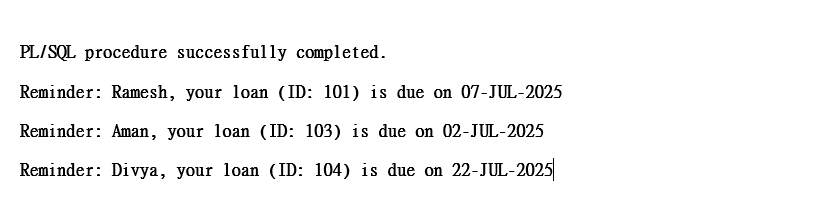
', your loan (ID: ' || rec.loan\_id ||

') is due on ' || TO\_CHAR(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.

**Program:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts';

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(50),

balance NUMBER

);

INSERT INTO savings\_accounts VALUES (1, 'Ramesh', 10000);

INSERT INTO savings\_accounts VALUES (2, 'Sita', 15000);

INSERT INTO savings\_accounts VALUES (3, 'Aman', 5000);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01); -- Adds 1% interest

END;

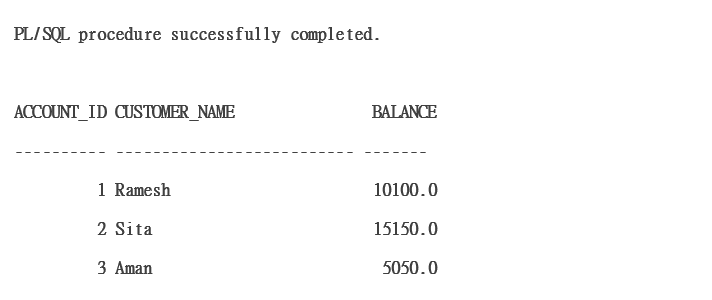
/

BEGIN

ProcessMonthlyInterest;

END;

/

SELECT \* FROM savings\_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.

**Program:**

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

emp\_name VARCHAR2(50),

department VARCHAR2(30),

salary NUMBER

);

INSERT INTO employees VALUES (1, 'John', 'Sales', 30000);

INSERT INTO employees VALUES (2, 'Radha', 'HR', 25000);

INSERT INTO employees VALUES (3, 'Steve', 'Sales', 35000);

INSERT INTO employees VALUES (4, 'Megha', 'Finance', 40000);

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

)

IS

BEGIN

UPDATE employees

SET salary = salary + (salary \* bonus\_percent / 100)

WHERE department = dept\_name;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || dept\_name);

END;

/

SET SERVEROUTPUT ON;

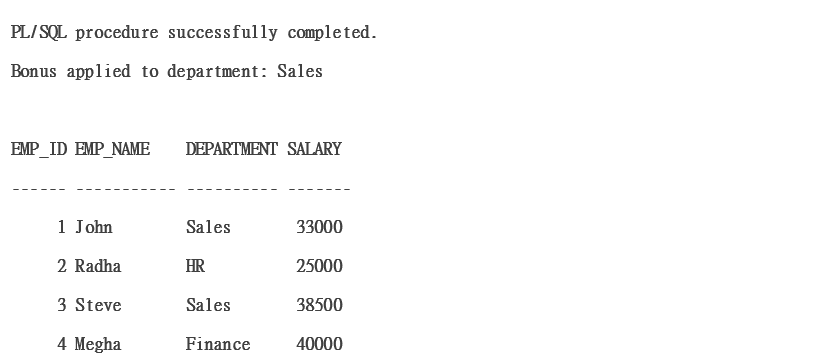
BEGIN

UpdateEmployeeBonus('Sales', 10);

END;

/

SELECT \* 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

**Program**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE bank\_accounts';

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

CREATE TABLE bank\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(50),

balance NUMBER

);

INSERT INTO bank\_accounts VALUES (1, 'Asha', 15000);

INSERT INTO bank\_accounts VALUES (2, 'Rahul', 10000);

INSERT INTO bank\_accounts VALUES (3, 'Meena', 8000);

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

)

IS

insufficient\_funds EXCEPTION;

current\_balance NUMBER;

BEGIN

-- Check current balance of sender

SELECT balance INTO current\_balance

FROM bank\_accounts

WHERE account\_id = from\_account;

IF current\_balance < amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE bank\_accounts

SET balance = balance - amount

WHERE account\_id = from\_account;

UPDATE bank\_accounts

SET balance = balance + amount

WHERE account\_id = to\_account;

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

EXCEPTION

WHEN insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in Account ' || from\_account);

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

/

SET SERVEROUTPUT ON;

BEGIN

TransferFunds(1, 2, 4000); -- Transfer ₹4000 from Asha to Rahul

END;

/

SELECT \* FROM bank\_accounts;

