**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.

**CODE:**

**-- create**

CREATE TABLE CUSTOMERS (

CUSTOMER\_ID INT PRIMARY KEY,

NAME VARCHAR(100),

AGE INT,

LOAN\_INTEREST\_RATE DECIMAL(5,2)

);

**-- insert**

INSERT INTO CUSTOMERS (CUSTOMER\_ID, NAME, AGE, LOAN\_INTEREST\_RATE) VALUES

(1, 'Anita', 62, 8.50),

(2, 'Ravi', 58, 9.00),

(3, 'Leela', 70, 7.25),

(4, 'John', 45, 8.75),

(5, 'Meera', 65, 9.20);

**-- fetch**

UPDATE CUSTOMERS

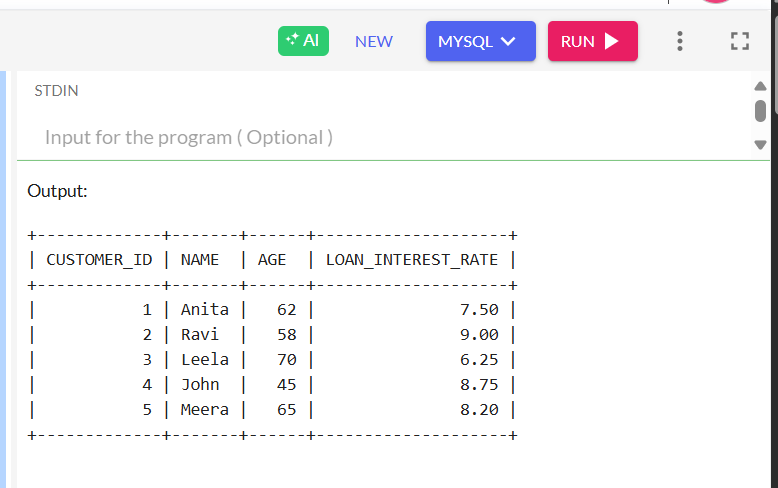
SET LOAN\_INTEREST\_RATE = LOAN\_INTEREST\_RATE - 1

WHERE AGE > 60;

**-- select query to view table**

SELECT \* FROM CUSTOMERS;

**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.

**CODE:**

**-- Create the table**

CREATE TABLE CUSTOMERS (

    CUSTOMER\_ID NUMBER PRIMARY KEY,

    NAME VARCHAR2(100),

    BALANCE NUMBER(10,2),

    ISVIP CHAR(1) DEFAULT 'N'

);

**-- Insert sample records**

INSERT INTO CUSTOMERS VALUES (1, 'Arun', 8000, 'N');

INSERT INTO CUSTOMERS VALUES (2, 'Sneha', 12000, 'N');

INSERT INTO CUSTOMERS VALUES (3, 'Vikram', 15000, 'N');

INSERT INTO CUSTOMERS VALUES (4, 'Meena', 9500, 'N');

COMMIT;

**-- VIP Promotion Logic**

BEGIN

    FOR cust IN (

        SELECT CUSTOMER\_ID

        FROM CUSTOMERS

        WHERE BALANCE > 10000

    ) LOOP

        UPDATE CUSTOMERS

        SET ISVIP = 'Y'

        WHERE CUSTOMER\_ID = cust.CUSTOMER\_ID;

        DBMS\_OUTPUT.PUT\_LINE('Customer ' || cust.CUSTOMER\_ID || ' promoted to VIP.');

    END LOOP;

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

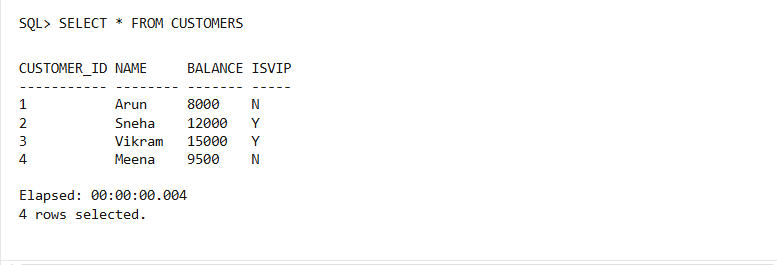
END;

/

**-- Show final result**

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.

**CODE**

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

**-- Create CUSTOMERS table**

CREATE TABLE CUSTOMERS (

    CUSTOMER\_ID NUMBER PRIMARY KEY,

    NAME VARCHAR2(100)

);

**-- Create LOANS table**

CREATE TABLE LOANS (

    LOAN\_ID NUMBER PRIMARY KEY,

    CUSTOMER\_ID NUMBER,

    DUE\_DATE DATE,

    AMOUNT NUMBER(10,2),

    FOREIGN KEY (CUSTOMER\_ID) REFERENCES CUSTOMERS(CUSTOMER\_ID)

);

**-- Insert customers**

INSERT INTO CUSTOMERS VALUES (1, 'Arun');

INSERT INTO CUSTOMERS VALUES (2, 'Sneha');

INSERT INTO CUSTOMERS VALUES (3, 'Vikram');

**-- Insert loans with various due dates**

INSERT INTO LOANS VALUES (101, 1, SYSDATE + 10, 5000);

INSERT INTO LOANS VALUES (102, 2, SYSDATE + 25, 7500);

INSERT INTO LOANS VALUES (103, 3, SYSDATE + 40, 10000); -- not due in 30 days

COMMIT;

-- PL/SQL block to send reminders

BEGIN

    FOR rec IN (

        SELECT C.NAME, L.DUE\_DATE, L.AMOUNT

        FROM LOANS L

        JOIN CUSTOMERS C ON C.CUSTOMER\_ID = L.CUSTOMER\_ID

        WHERE L.DUE\_DATE <= SYSDATE + 30

    ) LOOP

        DBMS\_OUTPUT.PUT\_LINE(

            'Reminder: Dear ' || rec.NAME || ', your loan of ₹' || rec.AMOUNT ||

            ' is due on ' || TO\_CHAR(rec.DUE\_DATE, 'DD-Mon-YYYY') || '. Please make arrangements to pay.'

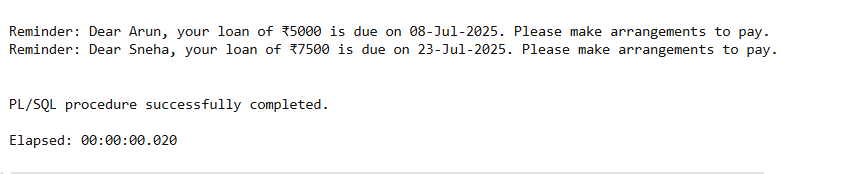
        );

    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.

**CODE**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE SAVINGS\_ACCOUNTS';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE SAVINGS\_ACCOUNTS (

ACCOUNT\_ID NUMBER PRIMARY KEY,

CUSTOMER\_ID NUMBER,

BALANCE NUMBER(10,2),

ACCOUNT\_TYPE VARCHAR2(20)

);

INSERT INTO SAVINGS\_ACCOUNTS VALUES (101, 1, 10000.00, 'SAVINGS');

INSERT INTO SAVINGS\_ACCOUNTS VALUES (102, 2, 5000.00, 'SAVINGS');

INSERT INTO SAVINGS\_ACCOUNTS VALUES (103, 3, 20000.00, 'CURRENT');

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SAVINGS\_ACCOUNTS

SET BALANCE = BALANCE \* 1.01

WHERE ACCOUNT\_TYPE = 'SAVINGS';

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed successfully.');

EXCEPTION

WHEN OTHERS THEN

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

END;

/

BEGIN

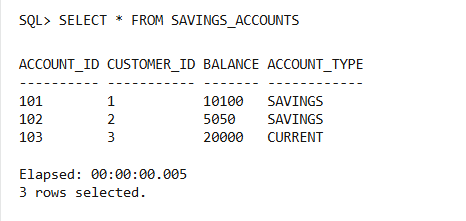
ProcessMonthlyInterest;

END;

/

SELECT \* FROM SAVINGS\_ACCOUNTS;

**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.

**CODE**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE EMPLOYEES';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE EMPLOYEES (

EMP\_ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

DEPARTMENT VARCHAR2(50),

SALARY NUMBER(10, 2)

);

INSERT INTO EMPLOYEES VALUES (1, 'Aarav', 'IT', 50000);

INSERT INTO EMPLOYEES VALUES (2, 'Sneha', 'HR', 45000);

INSERT INTO EMPLOYEES VALUES (3, 'Ravi', 'IT', 52000);

INSERT INTO EMPLOYEES VALUES (4, 'Meera', 'Finance', 60000);

COMMIT;

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 of ' || bonus\_percent || '% applied to department: ' || dept\_name);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

END;

/

BEGIN

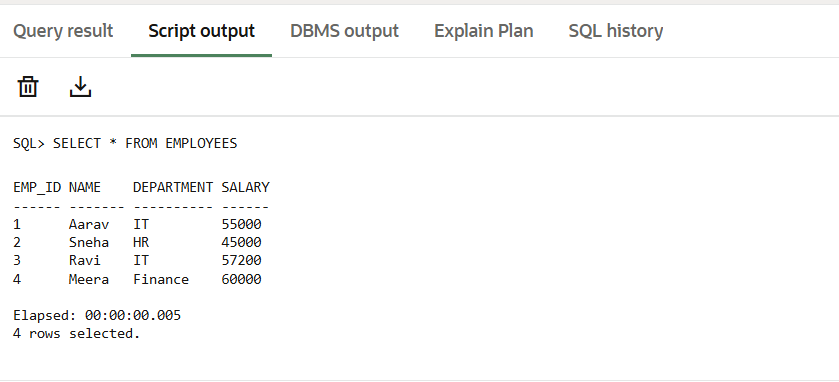
UpdateEmployeeBonus('IT', 10); -- 10% bonus to IT department

END;

/

SELECT \* FROM EMPLOYEES;

**OUTPUT**

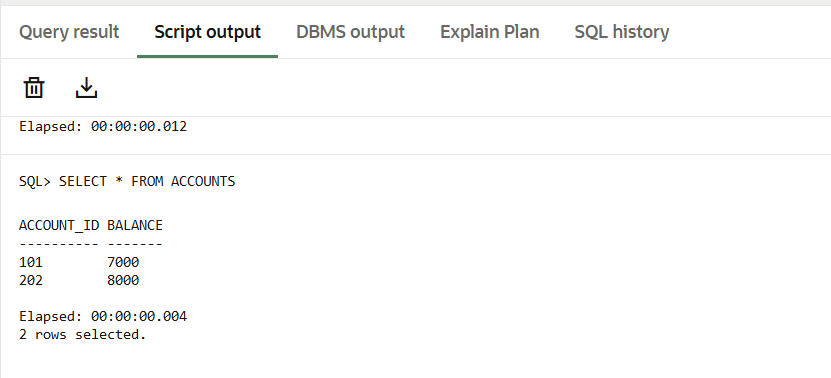
****

**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.

1. BEGIN
2. EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS';
3. EXCEPTION
4. WHEN OTHERS THEN NULL;
5. END;
6. /
7. CREATE TABLE ACCOUNTS (
8. ACCOUNT\_ID NUMBER PRIMARY KEY,
9. BALANCE NUMBER(10, 2)
10. );
11. INSERT INTO ACCOUNTS VALUES (101, 10000); -- Source account
12. INSERT INTO ACCOUNTS VALUES (202, 5000);  -- Destination account
13. COMMIT;
14. CREATE OR REPLACE PROCEDURE TransferFunds (
15. source\_id IN NUMBER,
16. dest\_id IN NUMBER,
17. transfer\_amount IN NUMBER
18. ) IS
19. source\_balance ACCOUNTS.BALANCE%TYPE;
20. BEGIN
21. SELECT BALANCE INTO source\_balance
22. FROM ACCOUNTS
23. WHERE ACCOUNT\_ID = source\_id;
24. IF source\_balance < transfer\_amount THEN
25. DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');
26. ELSE
27. UPDATE ACCOUNTS
28. SET BALANCE = BALANCE - transfer\_amount
29. WHERE ACCOUNT\_ID = source\_id;
30. UPDATE ACCOUNTS
31. SET BALANCE = BALANCE + transfer\_amount
32. WHERE ACCOUNT\_ID = dest\_id;
33. COMMIT;
34. DBMS\_OUTPUT.PUT\_LINE('Transfer of ₹' || transfer\_amount || ' from Account ' || source\_id || ' to Account ' || dest\_id || ' completed.');
35. END IF;
36. EXCEPTION
37. WHEN NO\_DATA\_FOUND THEN
38. DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs not found.');
39. ROLLBACK;
40. WHEN OTHERS THEN
41. DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);
42. ROLLBACK;
43. END;
44. /
45. BEGIN
46. TransferFunds(101, 202, 3000);
47. END;
48. /
49. SELECT \* FROM ACCOUNTS;

**OUTPUT**

****