**PL/SQL Programming**

**Tables**

CREATE TABLE customers (

    customer\_id     NUMBER PRIMARY KEY,

    name            VARCHAR2(50),

    age             NUMBER,

    balance         NUMBER

);

CREATE TABLE loans (

    loan\_id         NUMBER PRIMARY KEY,

    customer\_id     NUMBER,

    due\_date        DATE,

    amount          NUMBER,

    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE savings\_accounts (

    account\_id      NUMBER PRIMARY KEY,

    customer\_id     NUMBER,

    balance         NUMBER,

    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE employees (

    emp\_id          NUMBER PRIMARY KEY,

    name            VARCHAR2(50),

    department      VARCHAR2(50),

    salary          NUMBER

);

CREATE TABLE accounts (

    account\_id      NUMBER PRIMARY KEY,

    customer\_id     NUMBER,

    balance         NUMBER,

    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers VALUES (1, 'Ram', 35, 12000);

INSERT INTO customers VALUES (2, 'Raju', 42, 9500);

INSERT INTO customers VALUES (3, 'Rani', 28, 18000);

INSERT INTO customers VALUES (4, 'Bob', 60, 15000);

INSERT INTO customers VALUES (5, 'Mary', 50, 10500);

INSERT INTO loans VALUES (101, 1, TO\_DATE('2025-12-31', 'YYYY-MM-DD'), 5000);

INSERT INTO loans VALUES (102, 2, TO\_DATE('2025-11-15', 'YYYY-MM-DD'), 3000);

INSERT INTO loans VALUES (103, 3, TO\_DATE('2025-10-20', 'YYYY-MM-DD'), 7000);

INSERT INTO loans VALUES (104, 4, TO\_DATE('2025-09-05', 'YYYY-MM-DD'), 2000);

INSERT INTO loans VALUES (105, 5, TO\_DATE('2026-01-10', 'YYYY-MM-DD'), 4500);

INSERT INTO savings\_accounts VALUES (201, 1, 3000);

INSERT INTO savings\_accounts VALUES (202, 2, 1500);

INSERT INTO savings\_accounts VALUES (203, 3, 8000);

INSERT INTO savings\_accounts VALUES (204, 4, 5000);

INSERT INTO savings\_accounts VALUES (205, 5, 6000);

INSERT INTO employees VALUES (301, 'Ram', 'Finance', 40000);

INSERT INTO employees VALUES (302, 'Raju', 'Loans', 35000);

INSERT INTO employees VALUES (303, 'Rani', 'Customer Service', 32000);

INSERT INTO employees VALUES (304, 'Bob', 'Accounts', 38000);

INSERT INTO employees VALUES (305, 'Mary', 'HR', 36000);

INSERT INTO accounts VALUES (401, 1, 10000);

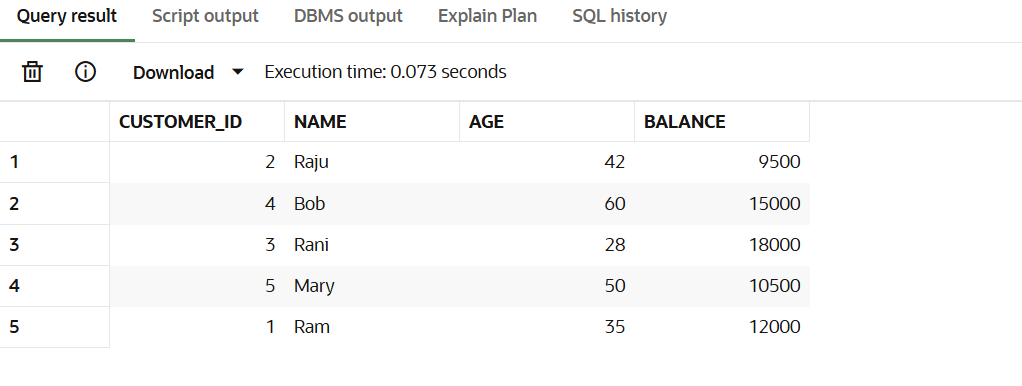
INSERT INTO accounts VALUES (402, 2, 8000);

INSERT INTO accounts VALUES (403, 3, 12000);

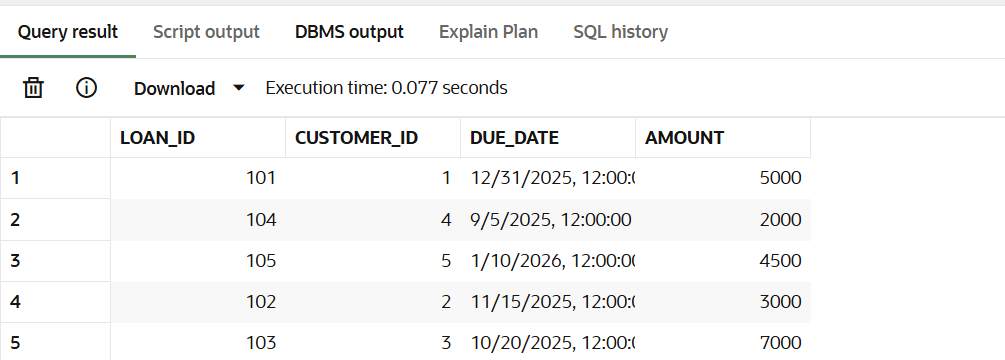
INSERT INTO accounts VALUES (404, 4, 9000);

INSERT INTO accounts VALUES (405, 5, 11000);

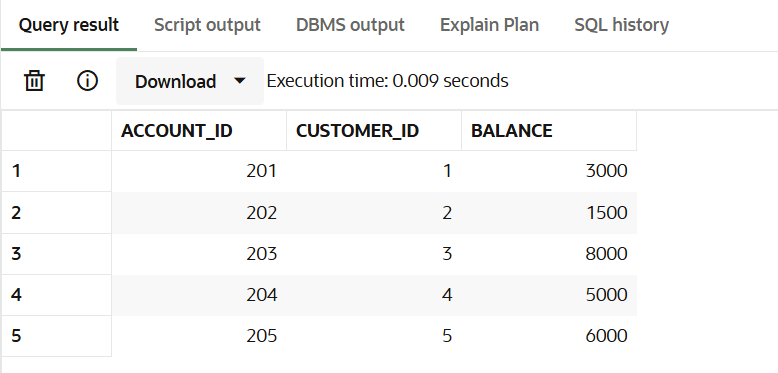
**Customers Table**



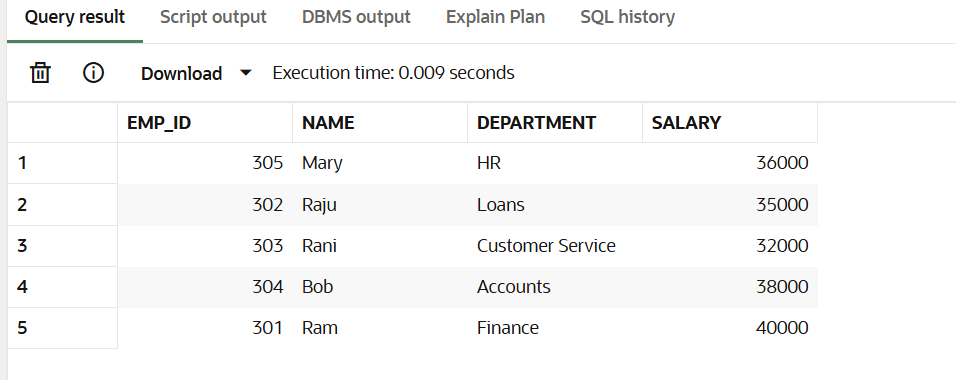
**Loans Table:**



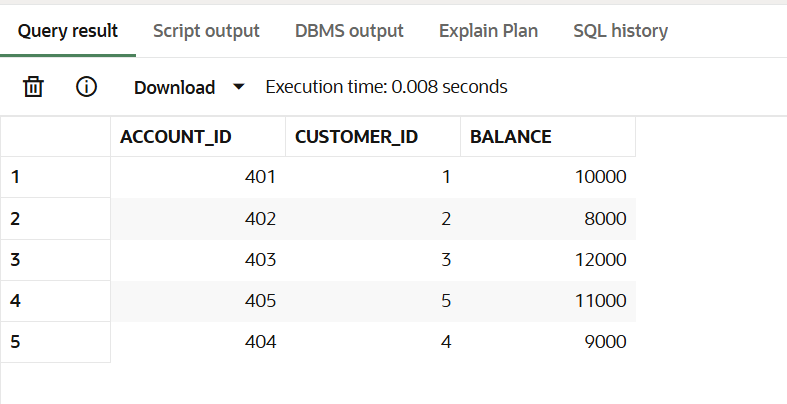
**Savings\_Accounts Table:**

****

**Employees Table:**

****

**Accounts Table:**

****

**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.

ALTER TABLE loans ADD (interest\_rate NUMBER(5,2));

UPDATE loans SET interest\_rate = 7.5 WHERE loan\_id = 101;

UPDATE loans SET interest\_rate = 8.0 WHERE loan\_id = 102;

UPDATE loans SET interest\_rate = 6.8 WHERE loan\_id = 103;

UPDATE loans SET interest\_rate = 7.2 WHERE loan\_id = 104;

UPDATE loans SET interest\_rate = 7.9 WHERE loan\_id = 105;

BEGIN

    FOR cust IN (

        SELECT c.customer\_id, c.age, l.loan\_id, l.interest\_rate

        FROM customers c

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

        WHERE c.age > 60

    )

    LOOP

        UPDATE loans

        SET interest\_rate = interest\_rate - 1

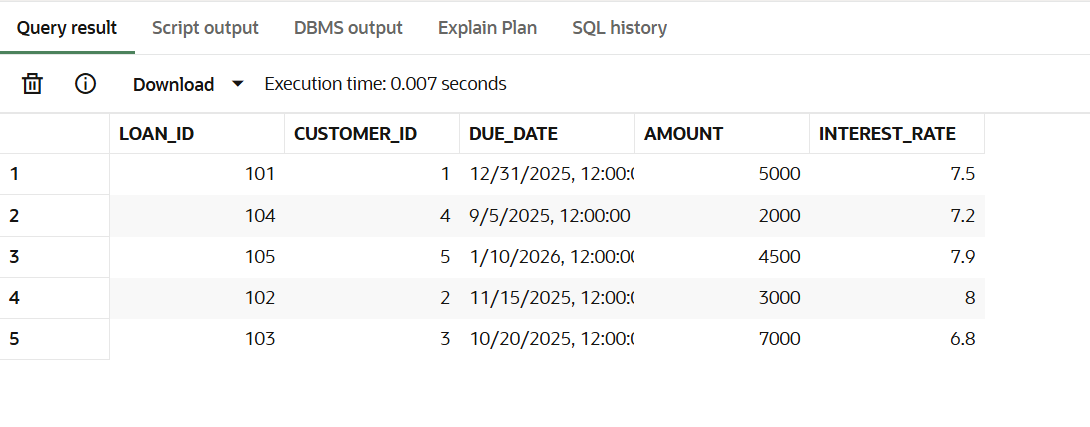
        WHERE loan\_id = cust.loan\_id;

    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 VARCHAR2(5) DEFAULT 'FALSE';

BEGIN

    FOR cust IN (

        SELECT customer\_id, balance

        FROM customers

        WHERE balance > 10000

    )

    LOOP

        UPDATE customers

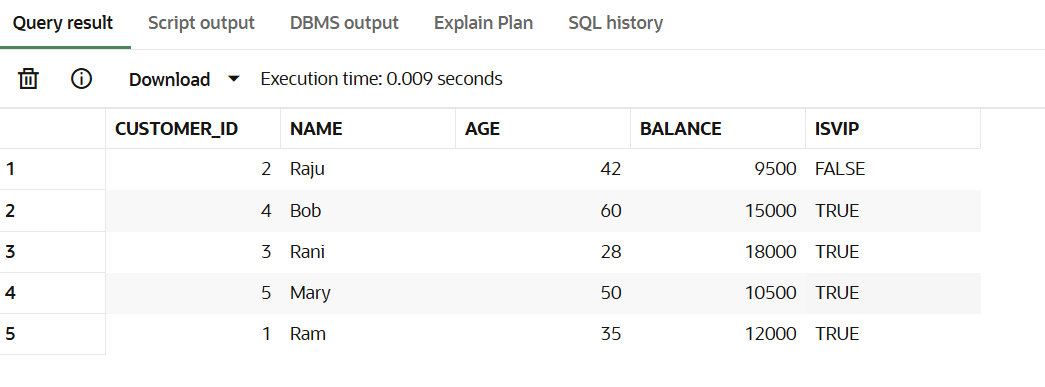
        SET isvip = 'TRUE'

        WHERE customer\_id = cust.customer\_id;

    END LOOP;

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.

SELECT \* FROM customers;

BEGIN

    FOR r IN (

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

        FROM customers c

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

        --WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

    )

    LOOP

       DBMS\_OUTPUT.PUT\_LINE('Dear ' || r.name ||

                     ', your loan (Loan ID: ' || r.loan\_id ||

                     ') of amount $' || r.amount ||

                     ' is due on ' || TO\_CHAR(r.due\_date, 'DD-MON-YYYY') ||

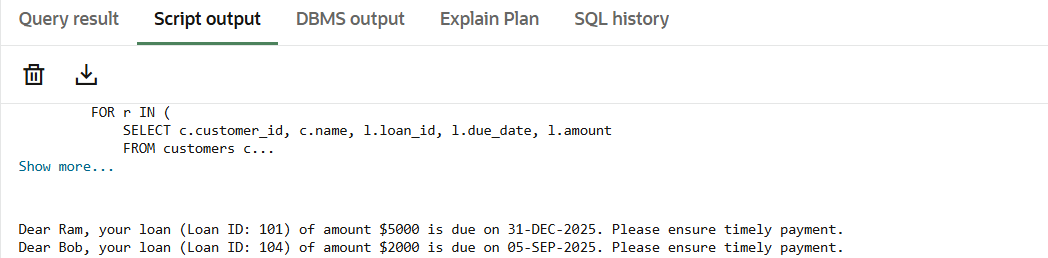
                     '. Please ensure timely payment.');

    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.

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

    v\_new\_balance NUMBER;

BEGIN

    FOR acc IN (SELECT account\_id, balance FROM savings\_accounts) LOOP

        v\_new\_balance := acc.balance + (acc.balance \* 0.01);

        UPDATE savings\_accounts

        SET balance = v\_new\_balance

        WHERE account\_id = acc.account\_id;

    END LOOP;

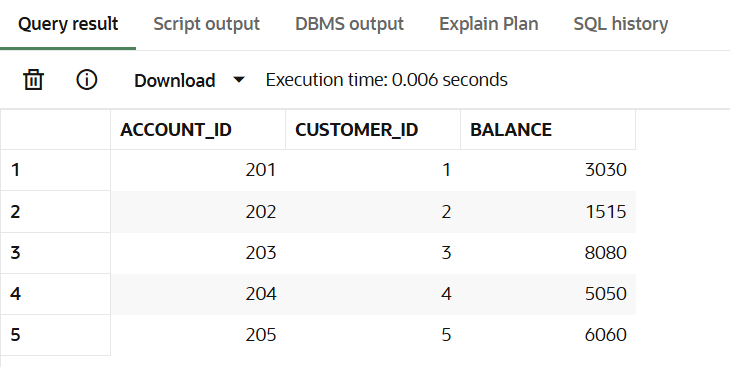
END;

/

**Calling Procedure**

EXEC ProcessMonthlyInterest;

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.

SET SERVEROUTPUT ON;

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;

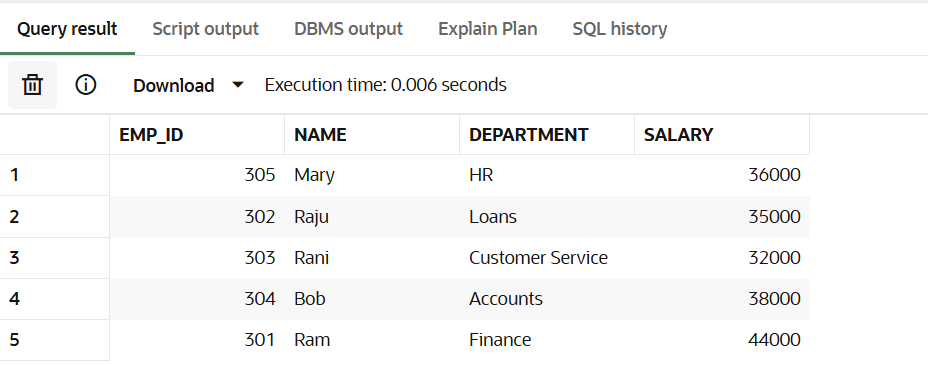
END;

/

**Calling Procedure:**

EXEC UpdateEmployeeBonus('Finance', 10);

OUPTUT:



**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 (

    p\_source\_account\_id  IN NUMBER,

    p\_dest\_account\_id    IN NUMBER,

    p\_amount             IN NUMBER

) IS

    v\_balance NUMBER;

BEGIN

    SELECT balance INTO v\_balance

    FROM accounts

    WHERE account\_id = p\_source\_account\_id;

    IF v\_balance < p\_amount THEN

        DBMS\_OUTPUT.PUT\_LINE('Insufficient balance.');

        RETURN;

    END IF;

    UPDATE accounts

    SET balance = balance - p\_amount

    WHERE account\_id = p\_source\_account\_id;

    UPDATE accounts

    SET balance = balance + p\_amount

    WHERE account\_id = p\_dest\_account\_id;

EXCEPTION

    WHEN OTHERS THEN

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

END;

/

**Calling Procedure:**

EXEC TransferFunds(401, 402, 2000);

SELECT \* FROM accounts WHERE account\_id IN (401, 402);

OUTPUT:

