**1) 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.

**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.

**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.

I’ve Created two tables named **“customers”** and **“loans”** and then inserted some dummy values into it and then I’ve done the operations on the tables.

**Table Creation :**

CREATE TABLE customers (

    customerid INT PRIMARY KEY,

    name VARCHAR2(100),

    age INT,

    balance NUMBER(15,2),

    isvip VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE loans (

    loanid INT PRIMARY KEY,

    customerid INT,

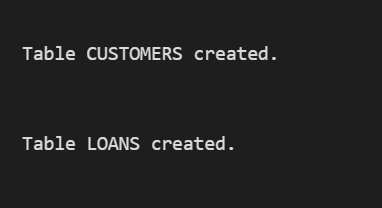
    duedate DATE,

    interestrate NUMBER(5,2),

    FOREIGN KEY (customerid) REFERENCES customers(customerid)

);

**Output:**



I’ve inserted some dummy values into the tables.

**Data Insertion:**

-- Sample customers

INSERT INTO customers (customerid, name, age, balance, isvip) VALUES (1, 'Alice', 25, 5000, 'FALSE');

INSERT INTO customers (customerid, name, age, balance, isvip) VALUES (2, 'Ben', 59, 15000, 'FALSE');

INSERT INTO customers (customerid, name, age, balance, isvip) VALUES (3, 'Charles', 72, 11000, 'FALSE');

INSERT INTO customers (customerid, name, age, balance, isvip) VALUES (4, 'David', 45, 8000, 'FALSE');

-- Sample loans

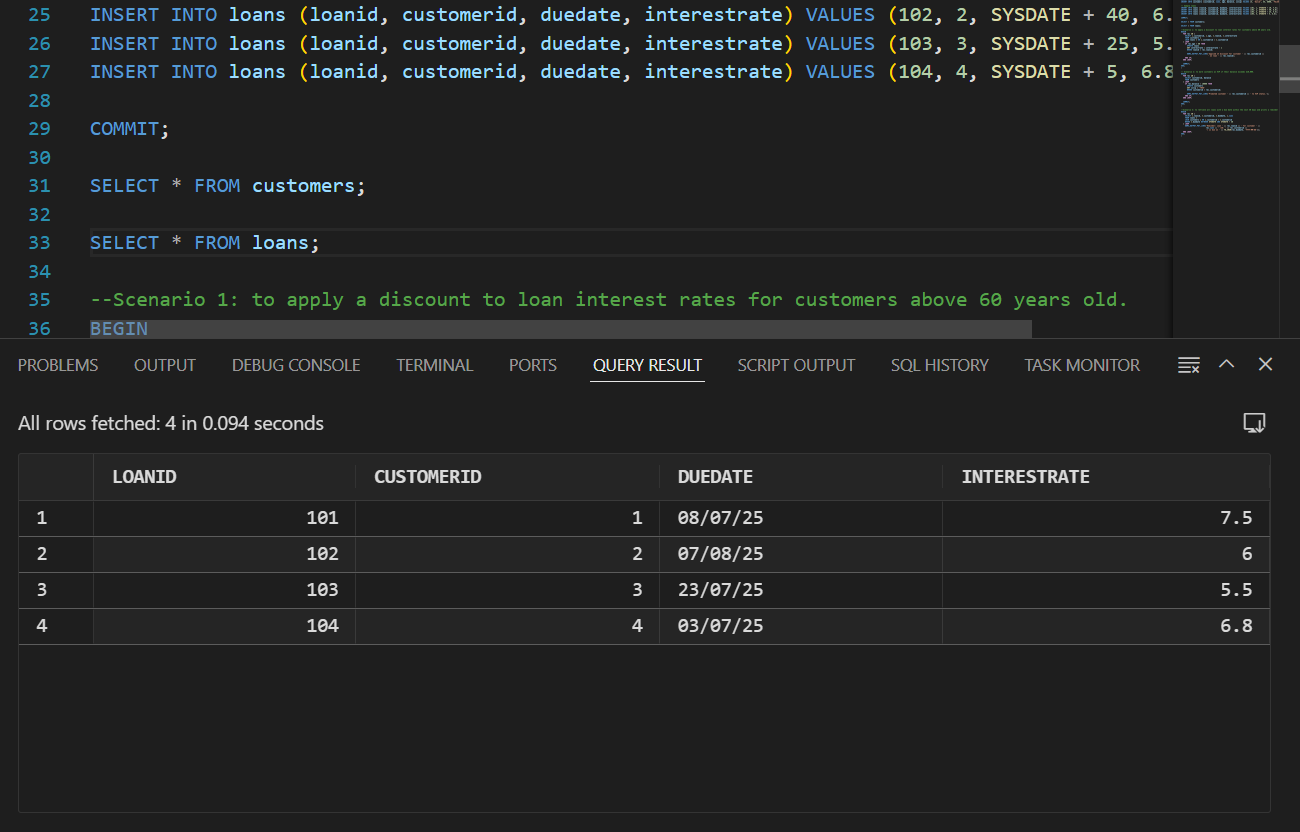
INSERT INTO loans (loanid, customerid, duedate, interestrate) VALUES (101, 1, SYSDATE + 10, 7.5);

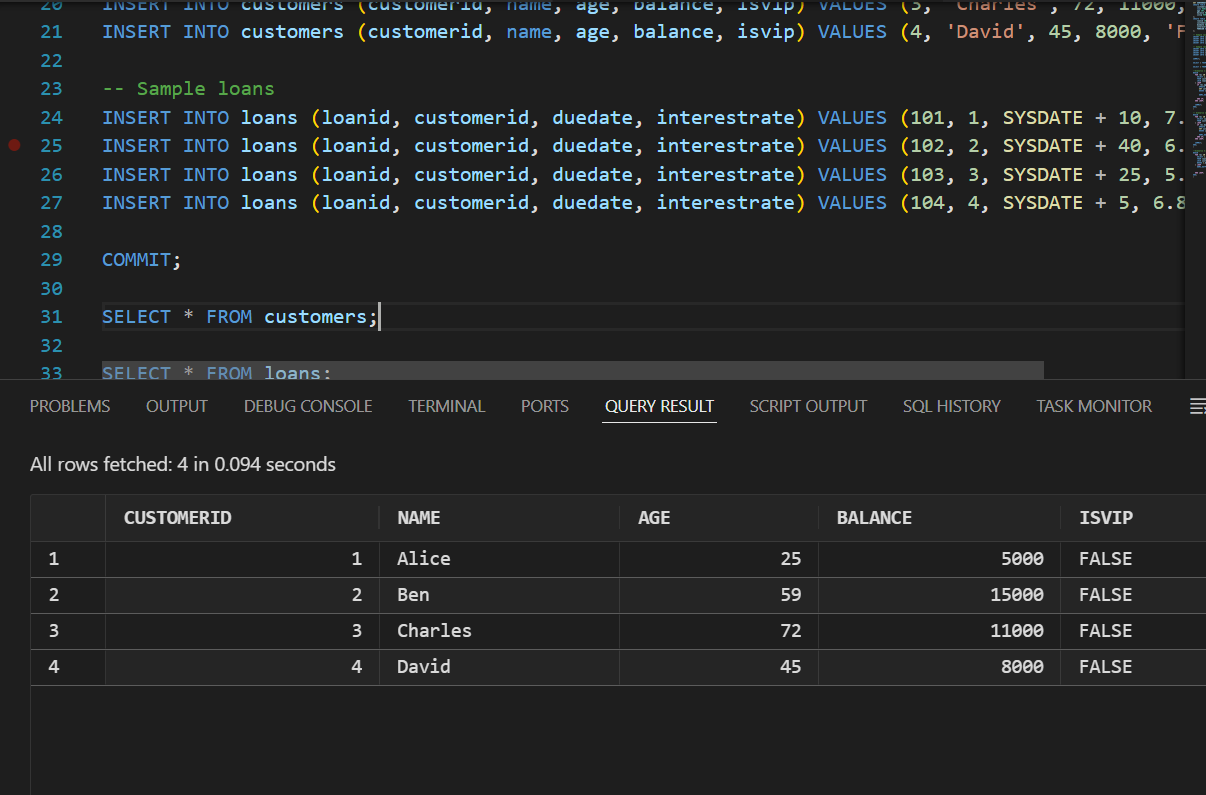
INSERT INTO loans (loanid, customerid, duedate, interestrate) VALUES (102, 2, SYSDATE + 40, 6.0);

INSERT INTO loans (loanid, customerid, duedate, interestrate) VALUES (103, 3, SYSDATE + 25, 5.5);

INSERT INTO loans (loanid, customerid, duedate, interestrate) VALUES (104, 4, SYSDATE + 5, 6.8);

**Output:**





**Scenario 1:**

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:**

BEGIN

  FOR rec IN (

    SELECT c.customerid, c.age, l.loanid, l.interestrate

    FROM customers c

    JOIN loans l ON c.customerid = l.customerid

  ) LOOP

    IF rec.age > 60 THEN

      UPDATE loans

      SET interestrate = interestrate - 1

      WHERE loanid = rec.loanid;

      DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for customer ' || rec.customerid ||

                           ' on loan ' || rec.loanid);

    END IF;

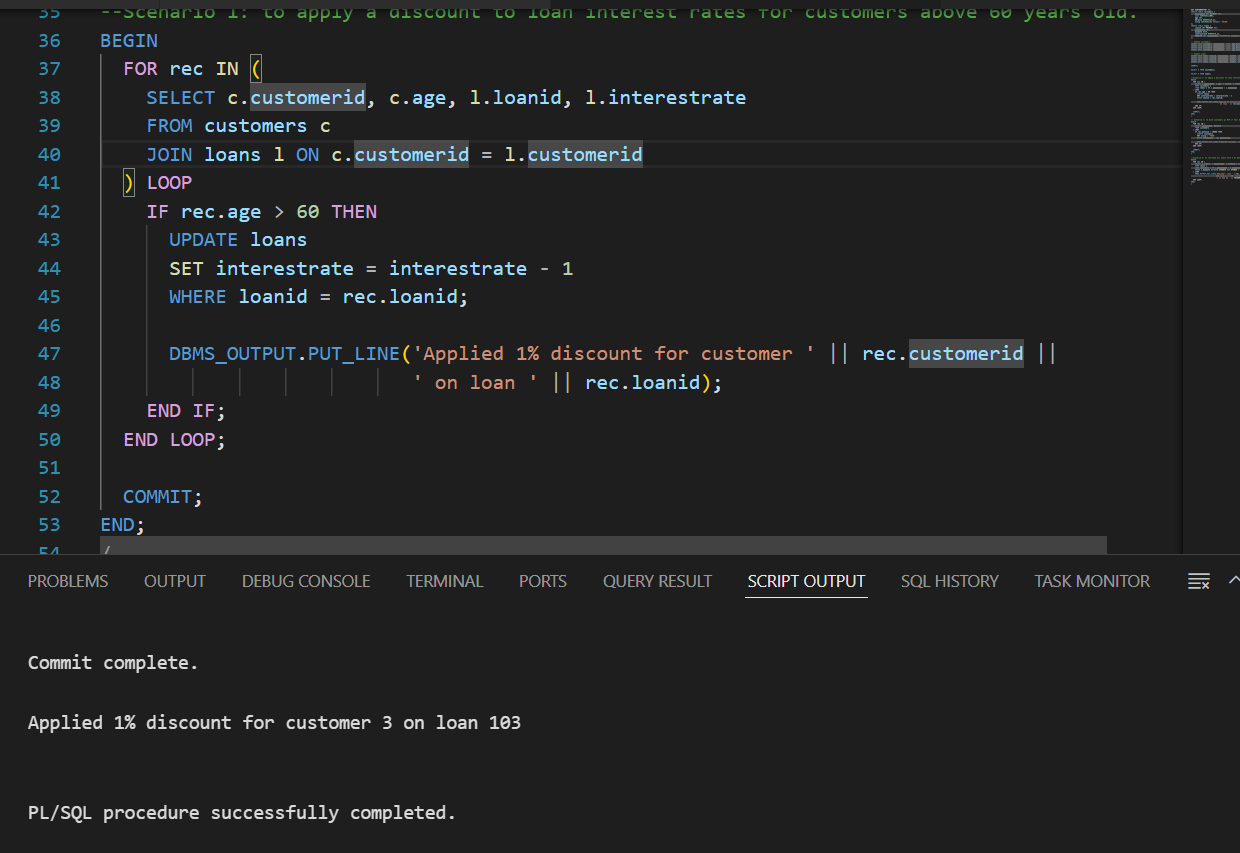
  END LOOP;

COMMIT;

END;

/

**Output:**



**Scenario 2:**

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:**

BEGIN

  FOR rec IN (

    SELECT customerid, balance

    FROM customers

  ) LOOP

    IF rec.balance > 10000 THEN

      UPDATE customers

      SET isvip = 'TRUE'

      WHERE customerid = rec.customerid;

  DBMS\_OUTPUT.PUT\_LINE('Promoted customer ' || rec.customerid || ' to VIP status.');

    END IF;

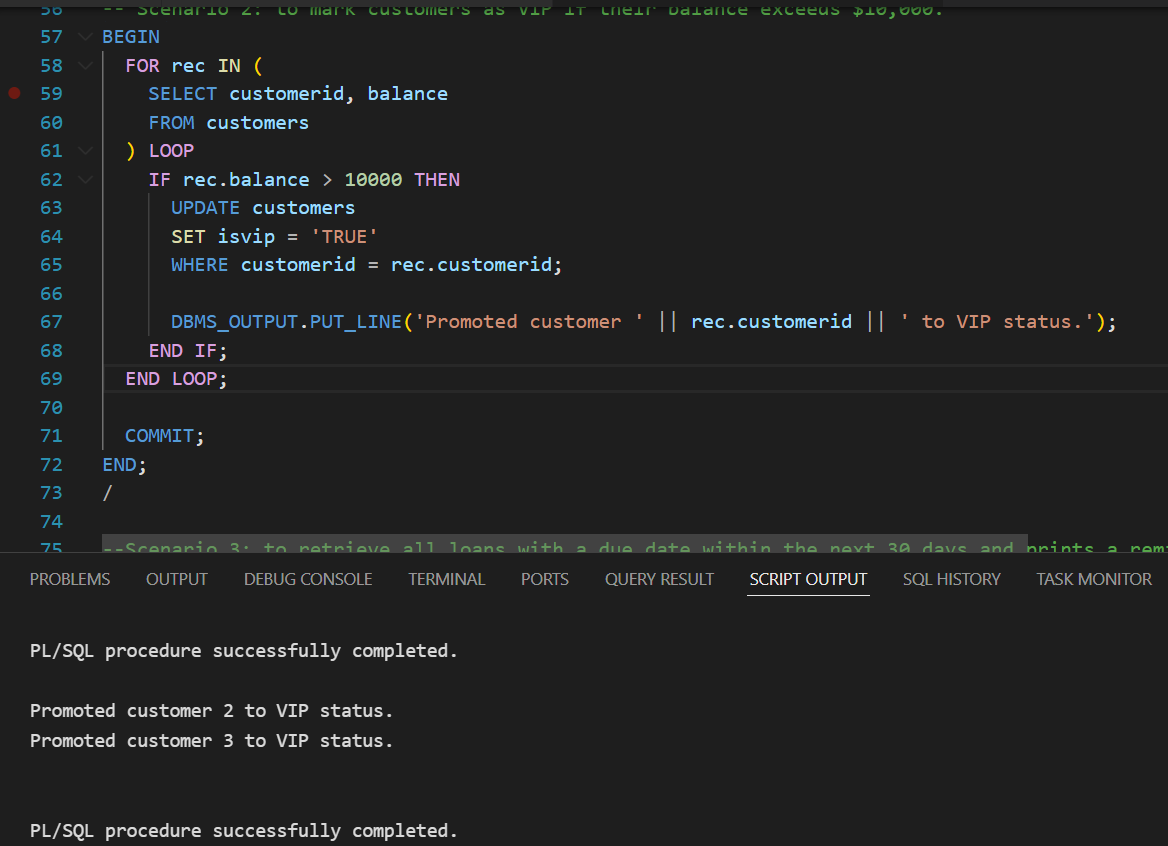
  END LOOP;

COMMIT;

END;

/

**Output:**



**Scenario 3:**

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

  FOR rec IN (

    SELECT l.loanid, l.customerid, l.duedate, c.name

    FROM loans l

    JOIN customers c ON c.customerid = l.customerid

    WHERE l.duedate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.loanid || ' for customer ' ||

                         rec.name || ' (ID: ' || rec.customerid ||

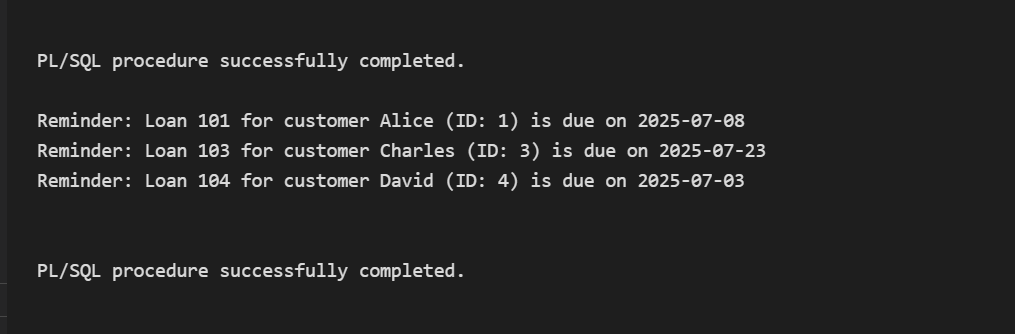
                         ') is due on ' || TO\_CHAR(rec.duedate, 'YYYY-MM-DD'));

  END LOOP;

END;

/

**Output:**



**2) 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.

**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.

**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.

I’ve Created two tables named **“employees”** and **“savings\_accounts”** and then inserted some dummy values into it and then I’ve done the operations on the tables.

**Table Creation :**

CREATE TABLE savings\_accounts (

    accountid INT PRIMARY KEY,

    customerid INT,

    balance NUMBER(15,2)

);

CREATE TABLE employees (

    employeeid INT PRIMARY KEY,

    name VARCHAR2(100),

    department VARCHAR2(50),

    salary NUMBER(15,2)

);

I’ve inserted some dummy values into the tables.

**Data Insertion:**

-- Sample savings accounts

INSERT INTO savings\_accounts (accountid, customerid, balance) VALUES (101, 1, 1000);

INSERT INTO savings\_accounts (accountid, customerid, balance) VALUES (102, 2, 5000);

INSERT INTO savings\_accounts (accountid, customerid, balance) VALUES (103, 3, 3000);

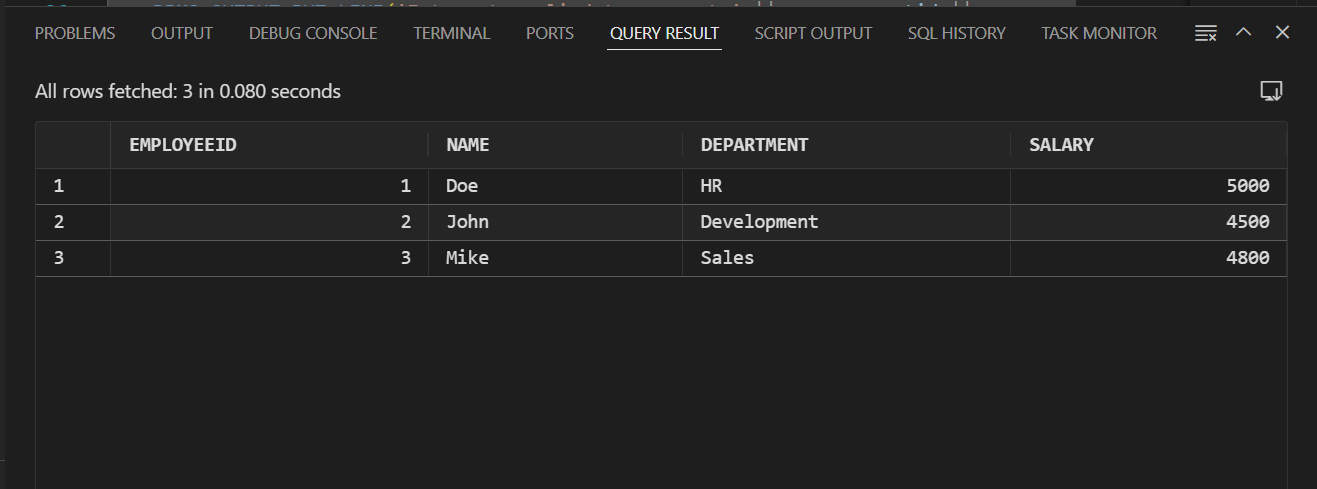
-- Sample employees

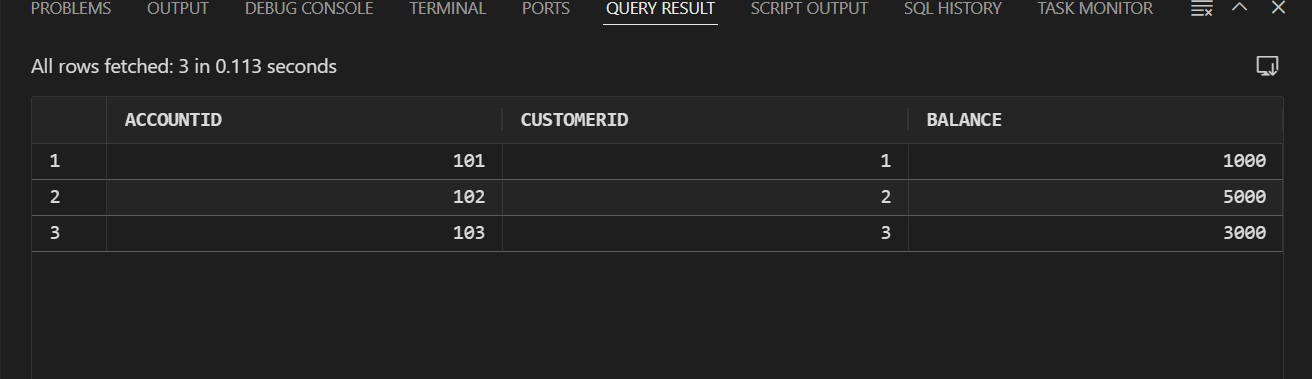
INSERT INTO employees (employeeid, name, department, salary) VALUES (1, 'Doe', 'HR', 5000);

INSERT INTO employees (employeeid, name, department, salary) VALUES (2, 'John', 'Development', 4500);

INSERT INTO employees (employeeid, name, department, salary) VALUES (3, 'Mike', 'Sales', 4800);

**Output:**





**Scenario 1:**

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 IS

BEGIN

  FOR rec IN (SELECT accountid, balance FROM savings\_accounts) LOOP

    UPDATE savings\_accounts

    SET balance = balance + (balance \* 0.01)

    WHERE accountid = rec.accountid;

    DBMS\_OUTPUT.PUT\_LINE('Interest applied to account ' || rec.accountid ||

                         ': new balance = ' ||

                         TO\_CHAR(rec.balance + (rec.balance \* 0.01), '999999.99'));

  END LOOP;

COMMIT;

END;

/

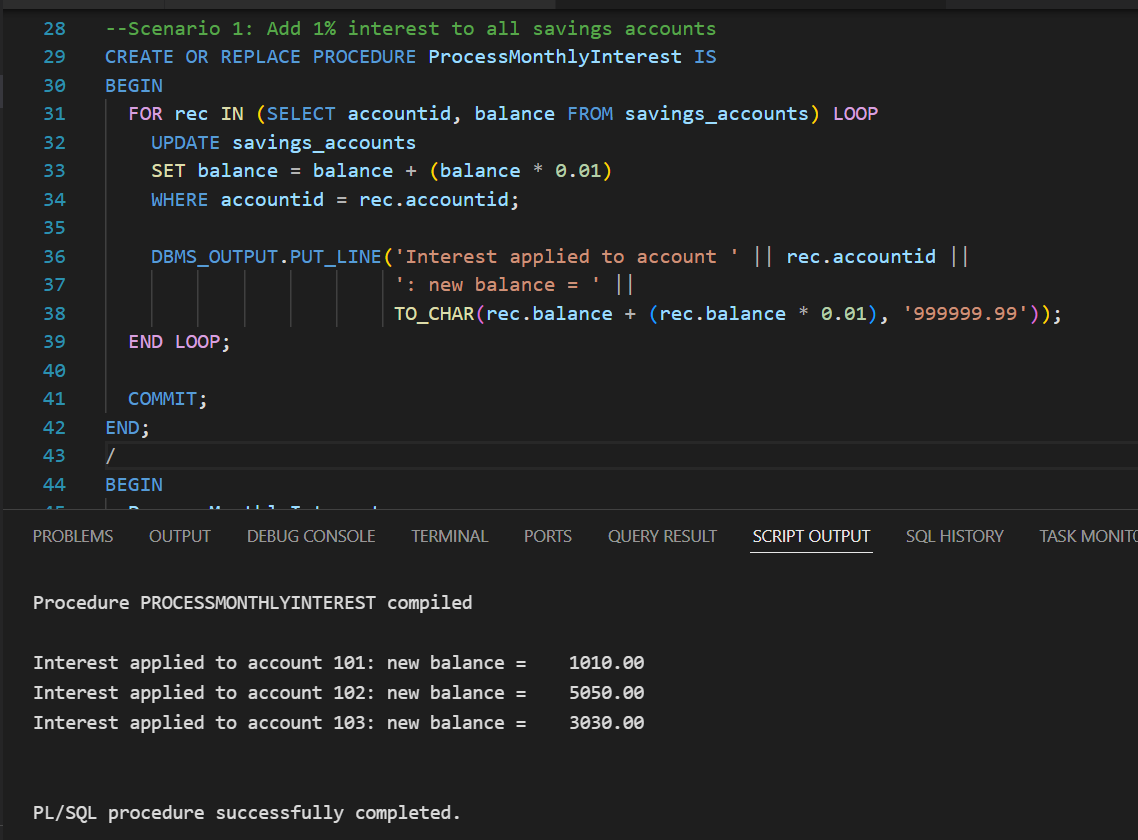
BEGIN

  ProcessMonthlyInterest;

END;

/

**Output:**



**Scenario 2:**

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 (

  p\_department IN VARCHAR2,

  p\_bonus\_pct IN NUMBER

) IS

BEGIN

  FOR rec IN (

    SELECT employeeid, salary FROM employees WHERE department = p\_department

  ) LOOP

    UPDATE employees

    SET salary = salary + (salary \* (p\_bonus\_pct/100))

    WHERE employeeid = rec.employeeid;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to employee ' || rec.employeeid ||

                         ': new salary = ' ||

                         TO\_CHAR(rec.salary + (rec.salary \* (p\_bonus\_pct/100)), '999999.99'));

  END LOOP;

  COMMIT;

END;

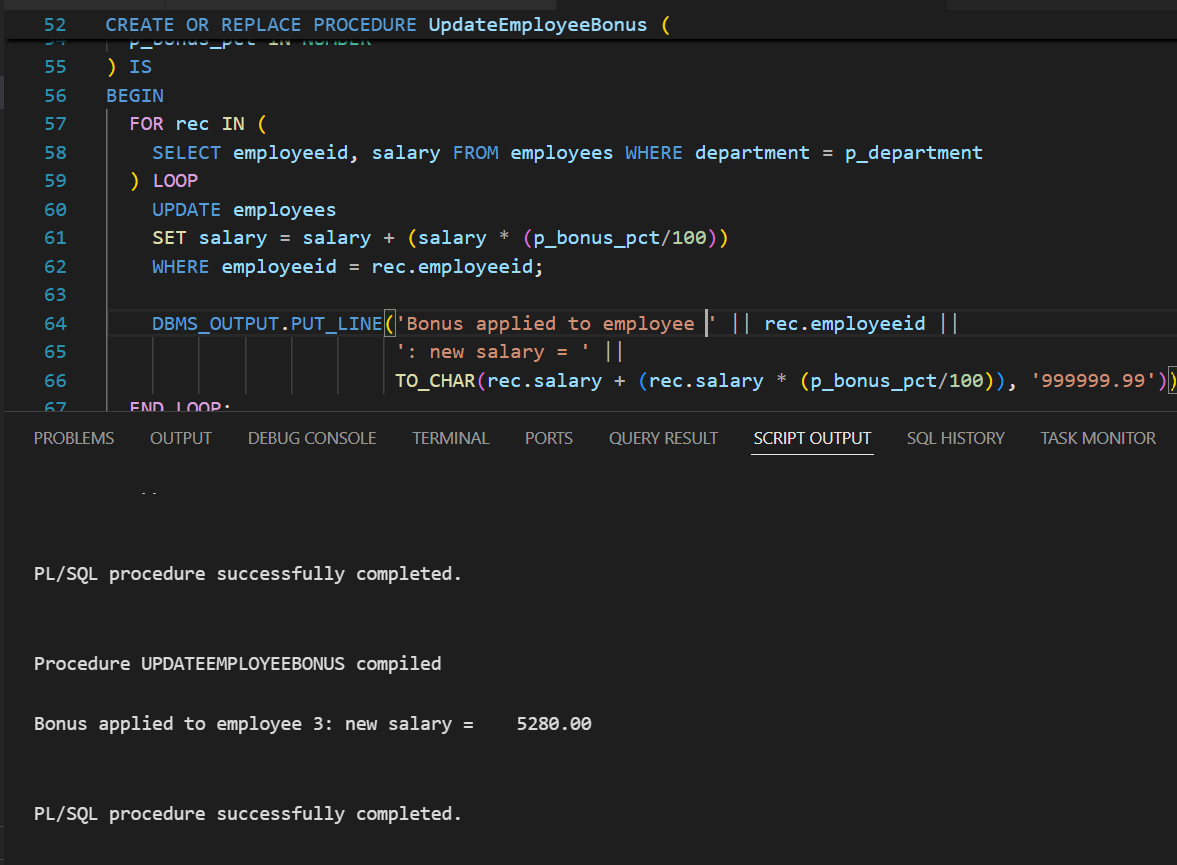
/

BEGIN

  UpdateEmployeeBonus('Sales', 10);

END;

**Output:**

****

**Scenario 3:**

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 (

  p\_source\_account IN INT,

  p\_target\_account IN INT,

  p\_amount IN NUMBER

) IS

  v\_source\_balance NUMBER;

BEGIN

  -- Locking the source account row for update and get balance

  SELECT balance INTO v\_source\_balance

  FROM savings\_accounts

  WHERE accountid = p\_source\_account

  FOR UPDATE;

  -- Checking for sufficient funds

  IF v\_source\_balance < p\_amount THEN

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

  END IF;

  -- Deducting from source account

  UPDATE savings\_accounts

  SET balance = balance - p\_amount

  WHERE accountid = p\_source\_account;

  -- Adding money to target account

  UPDATE savings\_accounts

  SET balance = balance + p\_amount

  WHERE accountid = p\_target\_account;

  DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount ||

                       ' from account ' || p\_source\_account ||

                       ' to account ' || p\_target\_account);

  COMMIT;

END;

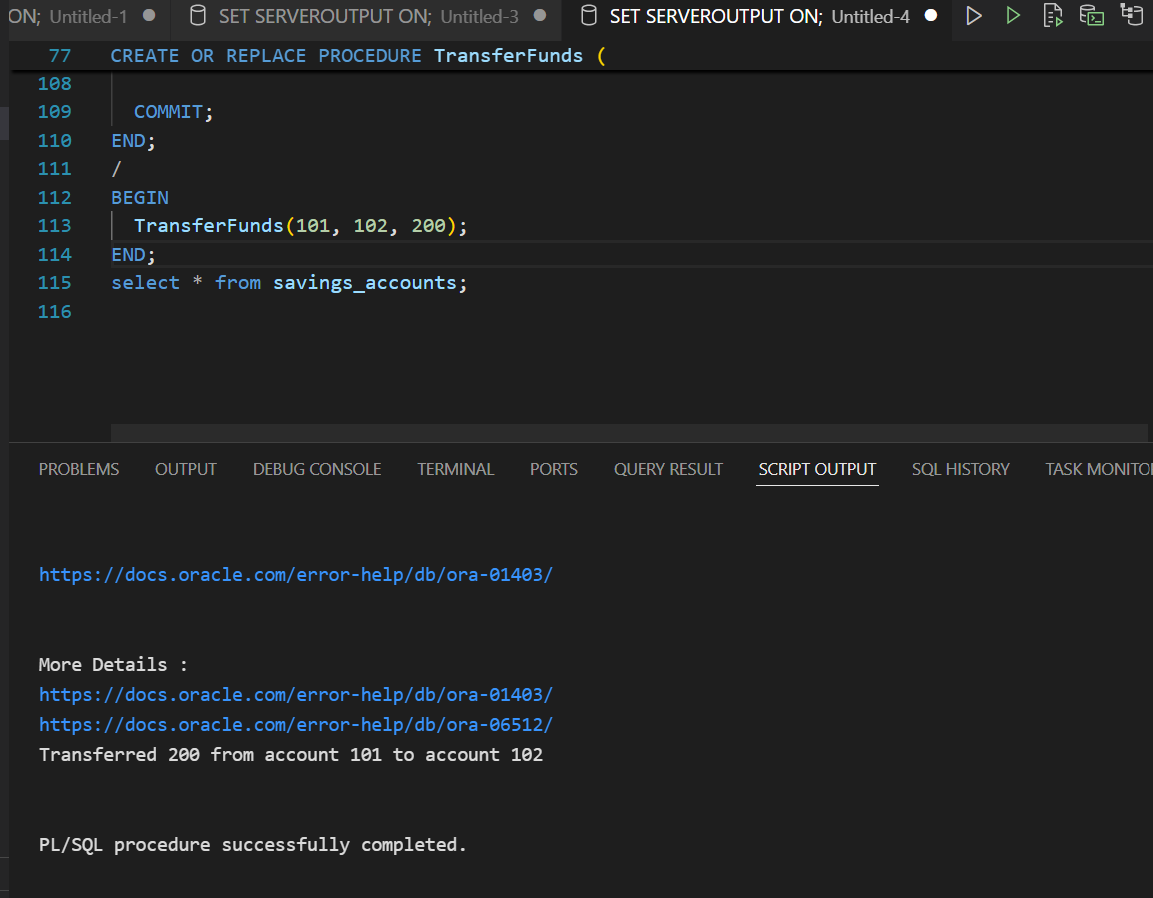
/

BEGIN

  TransferFunds(101, 102, 200);

END;

**Output:**

****