**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Code:**

DECLARE

    CURSOR cur\_senior IS

        SELECT customerid

        FROM   CUSTOMERS

        WHERE  TRUNC(MONTHS\_BETWEEN(SYSDATE, dob)/12) > 60;

BEGIN

    FOR c IN cur\_senior LOOP

        UPDATE LOANS

        SET    interestrate = interestrate - 1      --1% discount

        WHERE  customerid = c.customerid;

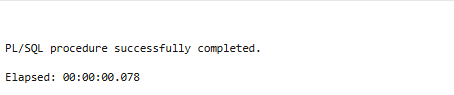
    END LOOP;

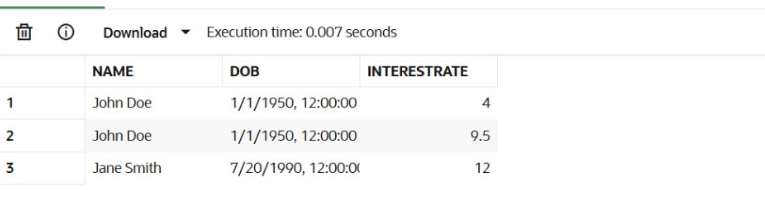
    COMMIT;

END;

/

**Output:**





**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Code:**

first **add it**:

ALTER TABLE Customers ADD IsVIP CHAR(1);

DECLARE

    CURSOR cur\_highbal IS

        SELECT customerid

        FROM   customers

        WHERE  balance > 10000;

BEGIN

    FOR c IN cur\_highbal LOOP

        UPDATE customers

        SET    isvip = 'Y'

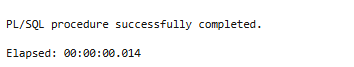
        WHERE  customerid = c.customerid;

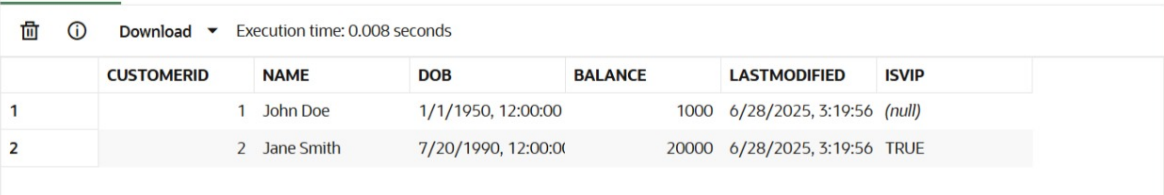
    END LOOP;

    COMMIT;

END;

**Output:**

****



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Code:**

DECLARE

    CURSOR cur\_due IS

        SELECT loanid,

               customerid,

               enddate

        FROM   Loans

        WHERE  enddate BETWEEN TRUNC(SYSDATE)

                              AND TRUNC(SYSDATE) + 30;

    v\_name  customers.name%TYPE;

BEGIN

    FOR r IN cur\_due LOOP

        SELECT name

        INTO   v\_name

        FROM   customers

        WHERE  customerid = r.customerid;

        DBMS\_OUTPUT.PUT\_LINE(

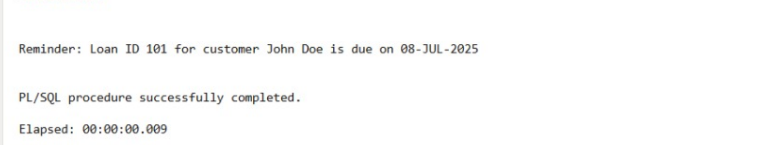
           'Reminder → '|| ': Loan ID' || r.loanid ||'for customer'|| v\_name

           ' is due on ' || TO\_CHAR(r.enddate,'DD-Mon-YYYY'));

    END LOOP;

END;

**Output:**



**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

    l\_rows\_updated INTEGER;

BEGIN

    UPDATE accounts

       SET balance = balance \* 1.01

     WHERE UPPER(accounttype) = 'SAVINGS'

     RETURNING COUNT(\*) INTO l\_rows\_updated;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE(l\_rows\_updated || ' savings accounts updated with 1% interest.');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

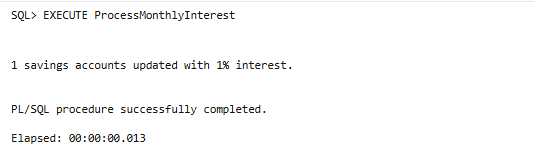
        RAISE;

END ProcessMonthlyInterest;

/

EXECUTE ProcessMonthlyInterest

**Output:**

****

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus

(

    p\_department IN employees.department%TYPE,

    p\_bonus\_pct     IN NUMBER

)

IS

    l\_rows\_updated INTEGER;

BEGIN

    UPDATE employees

       SET salary = salary \* (1 + p\_bonus\_pct/100)

     WHERE department = p\_department

     RETURNING COUNT(\*) INTO l\_rows\_updated;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE(

        l\_rows\_updated || ' employee(s) received a ' ||

        p\_bonus\_pct    || '% bonus in department ' || p\_department

    );

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

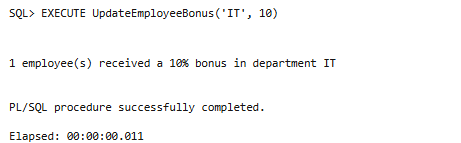
        RAISE;

END UpdateEmployeeBonus;

/

EXECUTE UpdateEmployeeBonus('IT', 10);

**Output:**

****

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds

(

    p\_from\_acct  IN accounts.accountid%TYPE,

    p\_to\_acct    IN accounts.accountid%TYPE,

    p\_amount     IN NUMBER

)

IS

    e\_insufficient\_funds EXCEPTION;

    l\_from\_balance       accounts.balance%TYPE;

BEGIN

    IF p\_amount <= 0 THEN

        RAISE\_APPLICATION\_ERROR( -20001,'Transfer amount must be positive.');

    END IF;

    -- Lock the source row first to prevent race conditions

    SELECT balance

      INTO l\_from\_balance

      FROM accounts

     WHERE accountid = p\_from\_acct

       FOR UPDATE;

    IF l\_from\_balance < p\_amount THEN

        RAISE e\_insufficient\_funds;

    END IF;

    UPDATE accounts

       SET balance = balance - p\_amount

     WHERE accountid = p\_from\_acct;

    UPDATE accounts

       SET balance = balance + p\_amount

     WHERE accountid = p\_to\_acct;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE(

        '₹' || p\_amount || ' transferred from account id:(' || p\_from\_acct ||

        ') to account id(' || p\_to\_acct ||')'

    );

EXCEPTION

    WHEN e\_insufficient\_funds THEN

        ROLLBACK;

        RAISE\_APPLICATION\_ERROR(-20001, 'Your error message here');

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

        RAISE\_APPLICATION\_ERROR(-20003, 'One or both account IDs not found.');

    WHEN OTHERS THEN

        ROLLBACK;

        RAISE;

END TransferFunds;

/

EXECUTE TransferFunds(1,2,500);

**Output:**

