**Exercise 1: Control Structures**

**Scenario 1:**

set serveroutput on;

declare

cursor cur\_senior\_loans is

select l.loan\_id,

l.interest\_rate

from customers c

join loans l on l.customer\_id = c.customer\_id

where c.age > 60;

begin

-- automatic loop through the result set

for rec in cur\_senior\_loans loop

update loans

set interest\_rate = round(rec.interest\_rate \* 0.99, 2) -- 1 % off

where loan\_id = rec.loan\_id;

dbms\_output.put\_line(

'loan ' || rec.loan\_id || ' discounted to ' ||

round(rec.interest\_rate \* 0.99, 2)

);

end loop;

commit;

exception

when others then

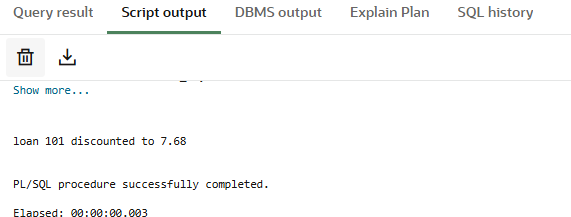
rollback;

raise;

end;

/

**OUTPUT:**



**Scenario 2:**

set serveroutput on;

create table customer\_accounts(

  customer\_id number primary key,

  balance     number,

  isvip       varchar2(5)

);

insert into customer\_accounts values (1, 15000, 'false');

insert into customer\_accounts values (2, 8000 , 'false');

commit;

begin

  for rec in (

      select customer\_id, balance

      from   customer\_accounts

      where  balance > 10000

  ) loop

      update customer\_accounts

      set    isvip = 'true'

      where  customer\_id = rec.customer\_id;

      dbms\_output.put\_line(

         'customer '||rec.customer\_id||' promoted to vip'

      );

  end loop;

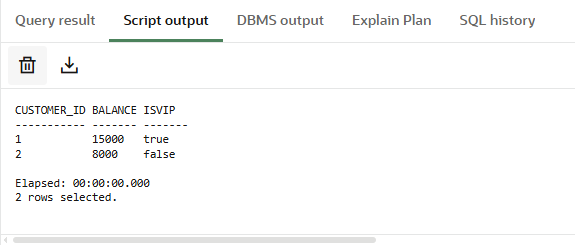
  commit;

end;

/

select \* from customer\_accounts;

**OUTPUT:**



**Scenario:3**

set serveroutput on;

begin

    for rec in (

        select customer\_name,

               due\_date

        from   loan\_due

        where  due\_date between trunc(sysdate)

                         and     trunc(sysdate)

    ) loop

        dbms\_output.put\_line(

            'reminder: ' || rec.customer\_name ||

            ', your loan is due on ' ||

            to\_char(rec.due\_date, 'dd-mon-yyyy')

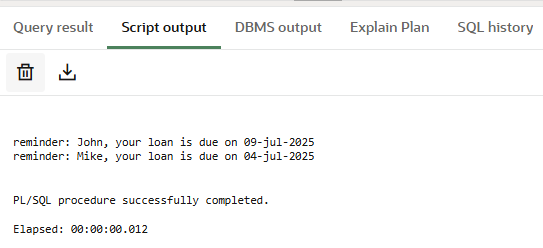
        );

    end loop;

end;

/

**OUTPUT:**



**Exercise 3: Stored Procedures**

**Scenario 1:**

set serveroutput on;

create table savings\_accounts (

    account\_id    number primary key,

    balance       number(10,2),

    account\_type  varchar2(20)

);

insert into savings\_accounts values (1, 10000, 'SAVINGS');

insert into savings\_accounts values (2, 5000,  'SAVINGS');

insert into savings\_accounts values (3, 7000,  'CURRENT'); -- not savings

commit;

create or replace procedure ProcessMonthlyInterest is

    c\_rate constant number := 1.01;

begin

    update savings\_accounts

    set    balance = round(balance \* c\_rate, 2)

    where  upper(account\_type) = 'SAVINGS';

    dbms\_output.put\_line(sql%rowcount || ' savings accounts updated.');

    commit;

exception

    when others then

        rollback;

        raise;

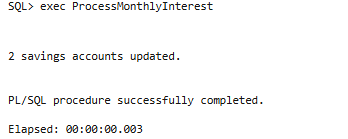
end;

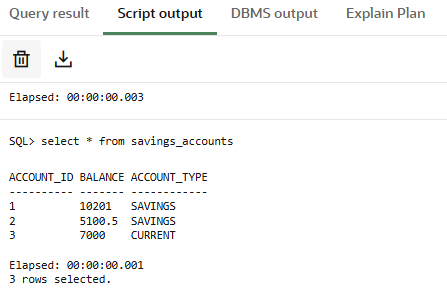
/

exec ProcessMonthlyInterest;

select \* from savings\_accounts;

**OUTPUT:**





**Scenario :2**

set serveroutput on;

create table employees (

    emp\_id       number primary key,

    emp\_name     varchar2(50),

    department   varchar2(50),

    salary       number(10,2)

);

insert into employees values (1, 'Alice', 'IT',     50000);

insert into employees values (2, 'Bob',   'HR',     45000);

insert into employees values (3, 'Carol', 'IT',     60000);

insert into employees values (4, 'David', 'SALES',  40000);

commit;

create or replace procedure UpdateEmployeeBonus(

    p\_dept    in varchar2,

    p\_bonus   in number

) is

    v\_factor number := 1 + (p\_bonus / 100);

begin

    update employees

    set    salary = round(salary \* v\_factor, 2)

    where  upper(department) = upper(p\_dept);

    dbms\_output.put\_line(sql%rowcount || ' employee(s) in ' || upper(p\_dept) || ' received a ' || p\_bonus || '% bonus.');

    commit;

exception

    when others then

        rollback;

        raise;

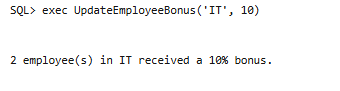
end;

/

exec UpdateEmployeeBonus('IT', 10);

select \* from employees;

**OUTPUT:**



**Scenario:3**

set serveroutput on;

create table bank\_accounts (

    account\_id   number primary key,

    customer\_id  number,

    balance      number(10,2)

);

insert into bank\_accounts values (101, 1, 10000.00); -- sender

insert into bank\_accounts values (102, 1, 5000.00);  -- receiver

insert into bank\_accounts values (103, 2, 2000.00);  -- other account

commit;

-- 3. create the stored procedure

create or replace procedure TransferFunds(

    p\_from\_account\_id in number,

    p\_to\_account\_id   in number,

    p\_amount          in number

) is

    v\_from\_balance number;

begin

    select balance into v\_from\_balance

    from bank\_accounts

    where account\_id = p\_from\_account\_id

    for update;

    if v\_from\_balance < p\_amount then

        raise\_application\_error(-20001, 'Insufficient balance in source account.');

    end if;

    update bank\_accounts

    set balance = balance - p\_amount

    where account\_id = p\_from\_account\_id;

    update bank\_accounts

    set balance = balance + p\_amount

    where account\_id = p\_to\_account\_id;

    commit;

    dbms\_output.put\_line('Transferred ₹' || p\_amount ||

                         ' from account ' || p\_from\_account\_id ||

                         ' to account ' || p\_to\_account\_id);

exception

    when others then

        rollback;

        dbms\_output.put\_line('Transfer failed: ' || sqlerrm);

end;

/

exec TransferFunds(101, 102, 2000);

select \* from bank\_accounts;

**OUTPUT:**

