**FUNCTIONS.**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**SOLUTION:**

**Query:**

create or replace function CalculateAge(c\_dob in customers.dob%type)

return number

is

age number;

begin

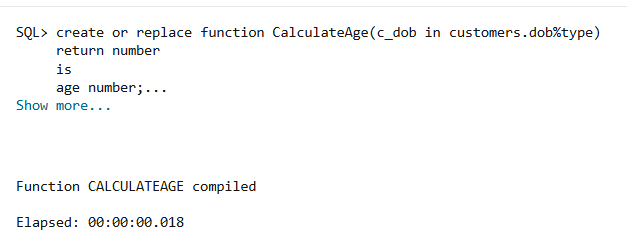
    age:=months\_between(sysdate,c\_dob)/12;

    dbms\_output.put\_line('AGE CALCULATED SUCCESSFULLY...');

    return age;

end;

**Output:**



**Query:**

declare

age number;

c\_dob customers.dob%type;

begin

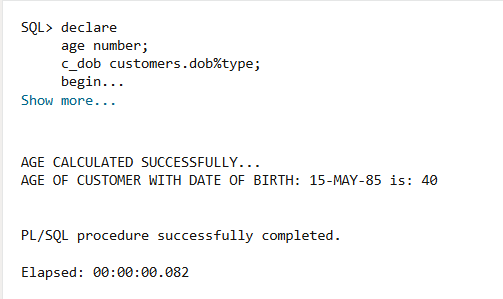
    select dob into c\_dob from customers where customerid=1;

    age:=CalculateAge(c\_dob);

    dbms\_output.put\_line('AGE OF CUSTOMER WITH DATE OF BIRTH: '||c\_dob||' is: '||floor(age));

end;

**Output:**



**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + Question: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**SOLUTION:**

**Query:**

create or replace function CalculateMonthlyInstallment(loan\_amount in loans.loanamount%type,l\_interest in loans.interestrate%type,l\_year in number)

return number

as

monthly\_installment number;

monthly\_rate number;

n\_months number;

begin

    monthly\_rate:=l\_interest/12/100;

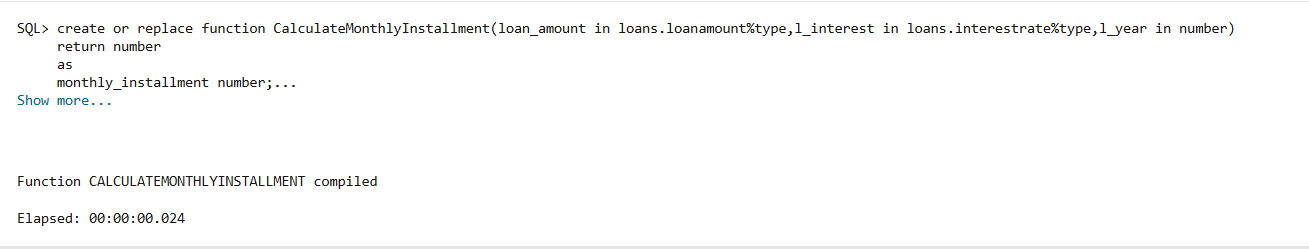
    n\_months:=l\_year\*12;

    monthly\_installment:=loan\_amount\*monthly\_rate\*POWER((1+monthly\_rate),n\_months)/(POWER((1+monthly\_rate),n\_months)-1);

    return monthly\_installment;

end;

**Output:**



**Query:**

declare

monthly\_installment number;

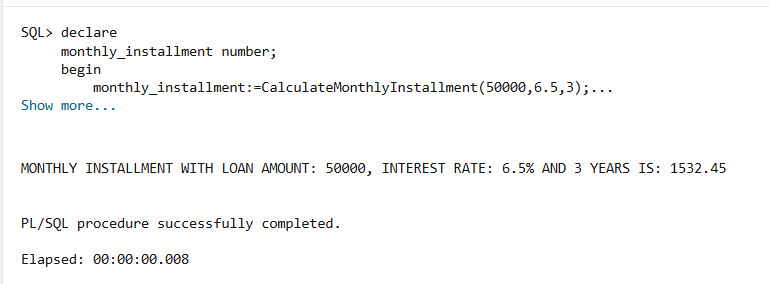
begin

    monthly\_installment:=CalculateMonthlyInstallment(50000,6.5,3);

    dbms\_output.put\_line('MONTHLY INSTALLMENT WITH LOAN AMOUNT: 50000, INTEREST RATE: 6.5% AND 3 YEARS IS: '||round(monthly\_installment,2));

end;

**Output:**



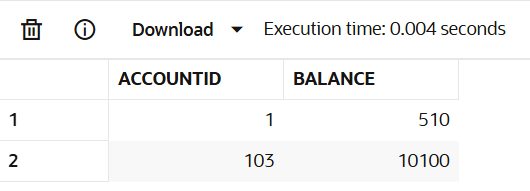
**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + Question: Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

SOLUTION:

Query: select accountid,balance from accounts where accountid=1 or accountid=103;

Output:



Query:

create or replace function HasSufficientBalance(a\_id in accounts.accountid%type,amount number)

return boolean

as

issufficient Boolean;

a\_balance number;

begin

    select balance into a\_balance from accounts where accountid=a\_id;

    if a\_balance>=amount then

    issufficient:=true;

    else

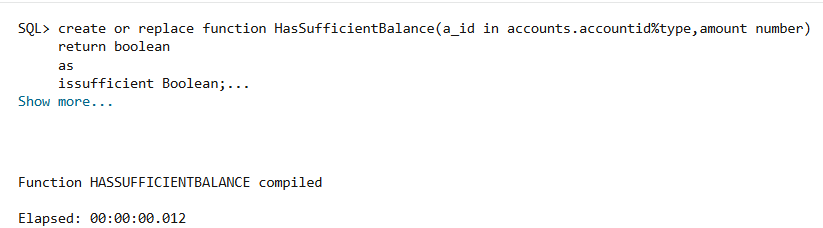
    issufficient:=false;

    end if;

    return issufficient;

end;

Output:



Query:

declare

issufficient Boolean;

begin

    issufficient:=HasSufficientBalance(1,1000);

    IF issufficient THEN

        dbms\_output.put\_line('ACCOUNTID: 1 HAS SUFFICIENT BALANCE');

    ELSE

        dbms\_output.put\_line('ACCOUNTID: 1 DOES NOT HAVE SUFFICIENT BALANCE');

    END IF;

    issufficient:=HasSufficientBalance(103,10000);

    IF issufficient THEN

        dbms\_output.put\_line('ACCOUNTID: 103 HAS SUFFICIENT BALANCE');

    ELSE

        dbms\_output.put\_line('ACCOUNTID: 103 DOES NOT HAVE SUFFICIENT BALANCE');

    END IF;

end;

Output:

