**Exercise 4: 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.

CREATE OR REPLACE FUNCTION CalculateAge (

dob DATE

) RETURN NUMBER IS

age NUMBER;

BEGIN

SELECT EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM dob)

INTO age

FROM DUAL;

RETURN age;

END CalculateAge;

DECLARE

CURSOR customer\_cursor IS

SELECT customerid, name, dob

FROM customers;

age NUMBER;

BEGIN

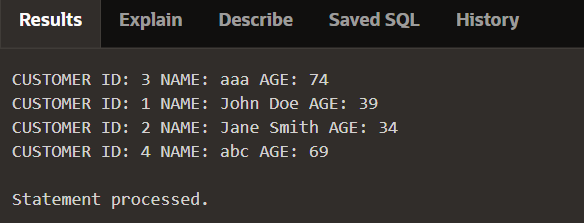
FOR customer IN customer\_cursor LOOP

age:=calculateAge(customer.dob);

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID: '|| customer.customerid || ' NAME: '||customer.name||' AGE: '||age);

END LOOP;

END;



**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.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loan\_amount IN NUMBER,

annual\_interest\_rate IN NUMBER,

total\_months IN NUMBER

) RETURN NUMBER

IS

monthly\_interest\_rate NUMBER;

emi NUMBER;

BEGIN

monthly\_interest\_rate := annual\_interest\_rate / (12 \* 100);

emi := (loan\_amount \* monthly\_interest\_rate \* POWER((1 + monthly\_interest\_rate), total\_months)) /

(POWER((1 + monthly\_interest\_rate), total\_months) - 1);

RETURN emi;

END;

DECLARE

CURSOR loans\_cursor IS

SELECT loanid, loanamount, startdate,enddate,interestrate

FROM loans;

MonthlyInstallment NUMBER;

BEGIN

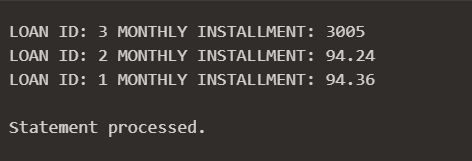
FOR rec IN loans\_cursor LOOP

MonthlyInstallment:=CalculateMonthlyInstallment(rec.loanamount,rec.interestrate,MONTHS\_BETWEEN(rec.EndDate, rec.StartDate));

DBMS\_OUTPUT.PUT\_LINE('LOAN ID: '|| rec.loanid || ' MONTHLY INSTALLMENT: '|| ROUND(MonthlyInstallment,2));

END LOOP;

END;

****

**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.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

cid IN NUMBER,

balance IN NUMBER

) RETURN BOOLEAN IS

current\_balance NUMBER;

BEGIN

SELECT balance INTO current\_balance

FROM accounts

WHERE customerid = cid;

IF current\_balance >= balance THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

DECLARE

CURSOR acc\_cursor IS

SELECT customerid

FROM accounts;

customerid accounts.customerid%TYPE;

amount\_to\_check NUMBER := 500;

check\_balance BOOLEAN;

BEGIN

FOR rec IN acc\_cursor LOOP

customerid := rec.customerid;

check\_balance := HasSufficientBalance(customerid, amount\_to\_check);

IF check\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID: ' || customerid || ' HAS SUFFICIENT BALANCE: TRUE');

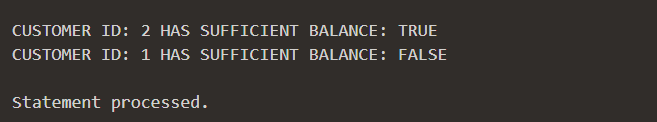
ELSE

DBMS\_OUTPUT.PUT\_LINE('CUSTOMER ID: ' || customerid || ' HAS SUFFICIENT BALANCE: FALSE');

END IF;

END LOOP;

END;

****