**EXERCISE 4 – FUNCTIONS**

**SCENARIO 1 - CALCULATE THE AGE OF CUSTOMERS FOR ELIGIBILITY CHECKS.**

**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 IN DATE ) RETURN NUMBER IS

curr\_date DATE := SYSDATE;

age NUMBER;

BEGIN

age := FLOOR(MONTHS\_BETWEEN(curr\_date, dob) / 12);

RETURN age;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END;

/

**-- calling the function**

DECLARE

cust\_id NUMBER;

dob DATE;

age NUMBER;

BEGIN

cust\_id:=&cust\_id;

SELECT DOB INTO dob FROM Customers WHERE CustomerID = cust\_id;

age:= CalculateAge(dob);

DBMS\_OUTPUT.PUT\_LINE('The age of the customer with ID ' || cust\_id || ' is: ' || age);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || cust\_id || ' does not exist.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

**SCENARIO 2 - THE BANK NEEDS TO COMPUTE THE MONTHLY INSTALLMENT FOR A LOAN.**

**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\_amt IN NUMBER,

interest\_rate IN NUMBER,

duration IN NUMBER

) RETURN NUMBER IS

monthly\_interset NUMBER;

total\_payments NUMBER;

monthly\_installment NUMBER;

BEGIN

monthly\_interset := interest\_rate / 12 / 100;

total\_payments := duration \* 12;

IF monthly\_interset = 0 THEN

monthly\_installment := loan\_amt / total\_payments;

ELSE

monthly\_installment := (loan\_amt \* monthly\_interset \* POWER(1 + monthly\_interset, total\_payments)) /

(POWER(1 + monthly\_interset, total\_payments) - 1);

END IF;

RETURN monthly\_installment;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END;

/

**-- calling the function**

DECLARE

loan\_amt NUMBER;

interest\_rate NUMBER;

loan\_duration NUMBER;

monthly\_installment NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('The monthly installment for: ');

FOR rec IN (

SELECT LoanID, LoanAmount, InterestRate,(MONTHS\_BETWEEN(EndDate, StartDate) / 12) AS Duration

FROM Loans ORDER BY LoanId

) LOOP

loan\_amt := rec.LoanAmount;

interest\_rate := rec.InterestRate;

loan\_duration := rec.Duration;

monthly\_installment := CalculateMonthlyInstallment(loan\_amt, interest\_rate, loan\_duration);

DBMS\_OUTPUT.PUT\_LINE( ' Loan ID ' || rec.LoanID || ' is: ' || TO\_CHAR(monthly\_installment, 'FM9999999990.00')

);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

**SCENARIO 3 - CHECK IF A CUSTOMER HAS SUFFICIENT BALANCE BEFORE MAKING A TRANSACTION.**

**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(

acc\_id IN NUMBER,

amt IN NUMBER

) RETURN BOOLEAN IS balance NUMBER;

BEGIN

SELECT Balance INTO balance FROM Accounts

WHERE AccountID = acc\_id;

IF balance >= amt THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN FALSE;

END;

/

**-- calling the function**

DECLARE

acc\_id NUMBER := 1;

amt NUMBER := 500;

sufficient BOOLEAN;

BEGIN

sufficient := HasSufficientBalance(acc\_id, amt);

IF sufficient THEN

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || acc\_id || ' has sufficient balance to send ' || amt);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || acc\_id || ' does not have sufficient balance to send '|| amt);

END IF;

END;

/