NAME: NAZIA KARIM KHAN OISHEE

STUDENT ID: 200042137

COURSE CODE: CSE 4308

SUBJECT: DATABASE MANAGEMENT SYSTEMS LAB

Introduction

Our task in this lab was to use PL to conduct sql queries.PL stands for Procedural Language.

Procedural Language for Structured Query Language (PL/SQL) lets us write code in such a way that it can be deployed in the database nearest to data. PL/SQL simplifies application development, optimizes execution,and improves resource utilization in the database. The language is a case-insensitive programming language.

Our task was divided into two parts. Firstly, there were warm-up tasks to get us acquainted with PL/SQL .Secondly, we were to implement PL/SQL to execute given scenario based queries on banking systems.

Task1

(a) Print your student ID.

Solution:

```
SET SERVEROUTPUT ON SIZE 1000000
BEGIN
DBMS_OUTPUT . PUT_LINE ( '200042137');
END ;
/
```

Explanation:

Here I used `SERVEROUTPUT` to print content on the console.We print or show output in PL/SQL using `DBMS_OUTPUT.PUT_LINE`. Whatever we output using the `DBMS_OUTPUT .PUT_LINE` is internally stored inside a buffer in the Shared Global Area (SGA) , a

memory area of size 2000 bytes. Here, `SIZE` sets the size of the buffer. In order to fetch it from that buffer, we need to use `SERVEROUTPUT` for each session.

(b) Take your name as input and print its length.

Solution:

```
SET VERIFY OFF
DECLARE
USERNAME VARCHAR2 (10);
BEGIN
USERNAME := '& username ';
DBMS_OUTPUT . PUT_LINE ( ' Length of username:' || LENGTH(USERNAME));
END ;
/
```

Explanation:

In this task,I declared a variable `USERNAME` and substituted the variable in the interactive console, using an ampersand (&).To determine the length of username I used the LENGTH() function and to suppress echoing of the substitution I used `SET VERIFY OFF`.

(c) Take two numbers as input and print their sum.

Solution:

```
SET VERIFY OFF
DECLARE
NUMBER1 NUMBER;
NUMBER2 NUMBER;
BEGIN
```

```
NUMBER1 := '& NUMBER1 ';
NUMBER2 := '& NUMBER2 ';
DBMS_OUTPUT . PUT_LINE ( 'SUM:' || (NUMBER1+NUMBER2));
END ;
/
```

Explanation:

In this task I took user input for two numbers like the previous task and simply displayed their sum.

(d) Print the current system time in 24-hour format.

Solution:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
D DATE := SYSDATE ;
BEGIN
DBMS_OUTPUT . PUT_LINE ( TO_CHAR (D ,'HH24 :MI :SS') );
END ;
/
```

Explanation:

To capture timestamp I used the SYSDATE built-in function.It returns qualified dates and contains all field elements of a DATE variable or column. Then using the `HH24 :MI :SS` format I converted its format to 24-hour format.

(e) Take a number as input and print whether it is odd or even (with and without CASE statement).

Solution:

With CASE statement:

```
SET SERVEROUTPUT ON SIZE 1000000
SET VERIFY OFF
DECLARE
NUMBER1 NUMBER;
REM NUMBER;
BEGIN
NUMBER1 := '& NUMBER1 ';
REM := MOD(NUMBER1,2);
CASE REM
WHEN 0 THEN
DBMS_OUTPUT . PUT_LINE (NUMBER1 || ' IS EVEN');
ELSE
DBMS_OUTPUT . PUT_LINE ( NUMBER1 || ' IS ODD');
END CASE ;
END ;
/
```

Explanation:

I took the number as an input from the user and checked if the modulus was 0. If the modulus is 0 then the number is even, else odd. Using a case statement I completed this task.

With out CASE statement:

```
SET SERVEROUTPUT ON SIZE 1000000
SET VERIFY OFF
DECLARE
NUMBER1 NUMBER;
BEGIN
NUMBER1 := '& NUMBER1 ';
```

```
IF MOD(NUMBER1,2)=0 THEN
DBMS_OUTPUT.PUT_LINE ( NUMBER1 || ' IS EVEN');
ELSE
DBMS_OUTPUT. PUT_LINE ( NUMBER1 || ' IS ODD');
END IF;
END ;
/
```

Explanation:

My logic for this task was the same as the previous one.But instead of using a case statement I used conditional statements to complete this task.

(f) Write a procedure that takes a number as an argument and prints whether it is a prime number or not.

Solution:

```
CREATE OR REPLACE
PROCEDURE PRIME(NUM IN number)
AS
    FOUND NUMBER :=0;
BEGIN
    FOR i IN 2..(NUM-1) LOOP
        IF(MOD(NUM,i)=0) THEN
            DBMS_OUTPUT . PUT_LINE('NOT PRIME');
            FOUND:=1;
            EXIT;
        END IF;
    END LOOP;
    IF(FOUND=0) THEN
        DBMS_OUTPUT. PUT_LINE('PRIME');
    END IF;

END;
DECLARE
```

```
N NUMBER;
BEGIN
N := '& N';
PRIME(N);
END ;
/
```

Explanation:

A procedure is a module that performs one or more actions. It is similar to function in most of the ways. I created a procedure named `PRIME` that would take a number as a parameter. The parameter mode is IN which means the parameter is read-only and it cannot be changed. The procedure uses a for loop to determine if a number is prime or not and prints the output.

Task2

(a) Write a procedure to find the N richest branches and their details. The procedure will take N as input and print the details up to N branches. If N is greater then the number of branches, then it will print an error message.

Solution:

```
CREATE OR REPLACE
PROCEDURE INFO_OF_N_RichestBranches(NUM IN NUMBER)
AS
    NUMBEROFROWS NUMBER;
BEGIN
    SELECT max(ROWNUM) INTO NUMBEROFROWS FROM (SELECT * FROM branch ORDER BY
assets DESC);
    IF(NUM>NUMBEROFROWS) THEN
        DBMS_OUTPUT.PUT_LINE('N Exceeds the number of records');
        RETURN;
    END IF;
    FOR i IN (SELECT * FROM (SELECT * FROM branch ORDER BY assets DESC) WHERE
```

```
ROWNUM<=NUM) LOOP
        DBMS_OUTPUT.PUT_LINE(i.branch_name || ' ' || i.branch_city || ' ' ||
i.assets);
    END LOOP;

END;
/
DECLARE
    NUM NUMBER;
BEGIN
    NUM:='& NUM';
    INFO_OF_N_RichestBranches(NUM);
END;
/
```

Explanation:

The procedure `INFO_OF_N_RichestBranches` first counts the number of branches and then shows the error message if the given input is greater than the number of branches. If not then prints the details of the first N branches in descending order.

(b) Write a procedure to find the customer status (“Green zone”, “Red zone”). If net loan >net balance, then the status should be “Red zone”, else it should be “Green zone”. The procedure will take the name of the customer as input and print the status.

Solution:

```
CREATE OR REPLACE
PROCEDURE CustomerStatus(CustomerName IN customer.customer_name%TYPE)
AS
    NET_BALANCE NUMBER;
    NET_LOAN NUMBER;
BEGIN
```

```
    SELECT sum(account.balance) INTO NET_BALANCE FROM account,depositor WHERE
depositor.customer_name=CustomerName and
depositor.account_number=account.account_number;
    SELECT sum(loan.amount) INTO NET_LOAN FROM borrower,loan WHERE
borrower.customer_name=CustomerName and borrower.loan_number=loan.loan_number;
    IF((NET_BALANCE)<=(NET_LOAN)) THEN
        DBMS_OUTPUT.PUT_LINE('Green Zone');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Red Zone');
    END IF;
END;
/

BEGIN
    CustomerStatus('Hayes');
END;
/
```

Explanation:

The procedure `CustomerStatus` takes the customer name as input and converts the type of the parameter into the type of the database's customer name.Then calculates the total balance and total loan of the customer. If the balance is greater than the loan amount then the customer is in the green zone, else in the red zone.

(c) Write a function to find the tax amount for each customer. A customer is eligible for tax if their net balance is greater than or equal to 750 (do not consider the loan). And the amount of tax for one is 8% of the net balance.

Solution:

```
CREATE OR REPLACE
FUNCTION TaxAmount(CustomerName customer.customer_name%TYPE)
RETURN NUMBER
```

```
AS
    NET_BALANCE NUMBER;
    TAX NUMBER;
BEGIN
    SELECT sum(account.balance) INTO NET_BALANCE FROM account,depositor WHERE
depositor.customer_name=CustomerName and
depositor.account_number=account.account_number;
    IF((NET_BALANCE)>=750) THEN
        TAX:=0.08*NET_BALANCE;
    ELSE
        Tax:=0;
    END IF;
    RETURN Tax;
END;
/

BEGIN
    DBMS_OUTPUT.PUT_LINE(TaxAmount('Johnson'));
END;
/
```

Explanation:

A function is a module that returns data through its return clause, rather than in an OUT or IN parameter unlike a procedure call.

The function `TaxAmount` takes the customer name as input and converts the type of the parameter into the type of the database's customer name.Then calculates the total balance of the customer. If the balance is greater than 750 then tax will be calculated. Otherwise tax will not be calculated.Finally the function returns the amount of tax.

(d) Write a function to find the customer category.

Solution:

```
CREATE OR REPLACE
FUNCTION CATEGORY(CUSTOMER_NAME customer.customer_name%TYPE)
RETURN VARCHAR2
AS
    NET_BALANCE NUMBER;
    NET_LOAN NUMBER;
    Category VARCHAR2(20);
BEGIN
    SELECT sum(account.balance) INTO NET_BALANCE FROM account,depositor WHERE
depositor.customer_name=CustomerName AND
depositor.account_number=account.account_number;
    SELECT sum(loan.amount) INTO NET_LOAN FROM borrower,loan WHERE
borrower.customer_name=CustomerName AND borrower.loan_number=loan.loan_number;
    IF((NET_BALANCE)>1000 AND (NET_LOAN)<1000) THEN
        Category:='C-A1';
    ELSIF((NET_BALANCE)<500 AND (NET_LOAN)>2000) THEN
        Category:='C-C3';
    ELSE
        Category:='C-B1';
    END IF;
    RETURN Category;
END;
/
BEGIN
    DBMS_OUTPUT.PUT_LINE(CATEGORY('Johnson'));
END;
/
```

Explanation:

The function `CATEGORY` takes the customer name as input and converts the type of the parameter into the type of the database's customer name.Then calculates the total balance and total loan of the customer. Based on the conditions upon total balance and total loan the function determines the category of the customer and returns the string.

Problems I faced during the task:

PL/SQL being a new programming language to me , I struggled a bit to write the syntaxes properly especially while terminating loops and conditional statements. Other than that I faced errors while passing the arguments to the function due to incompatible data types. Later I solved this using the TYPE tool.