Islamic University of Technology

Lab 09

CSE 4308 - DBMS Lab

Submitted To :

Zannatun Naim Sristy
Lecturer, CSE Department
Islamic University of Technology

Submitted By :

Rhidwan Rashid
ID : 200042149
Prog. : SWE
Dept. : CSE

Warm up codes:

(a)Printing student id:

Working code:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('200042149');
END;
/
```

b)Taking name as input and printing its length:

Working code:

```
DECLARE
  myNAME VARCHAR2(10);
begin
  myNAME := '&NAME';
  DBMS_OUTPUT.PUT_LINE(LENGTH(myNAME));
end;
/
```

c)Taking two numbers as input and printing their sum:

Working code:

```
DECLARE
  num1 NUMBER;
  num2 NUMBER;
begin
  num1 := '&num1';
  num2 := '&num2';
  DBMS_OUTPUT.PUT_LINE(num1 + num2);
end;
/
```

d)Printing current system time in 24-hour format:

Working code:

```
DECLARE
  d DATE := SYSDATE;
begin
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(d,'HH24:MI:SS'));
end;
/
```

e)Taking a number as input and printing whether it is odd or even:

Working code without case:

```
DECLARE
  num NUMBER;
begin
  num := '&number';
  IF MOD(num,2) = 1 THEN
    DBMS_OUTPUT.PUT_LINE('This is an odd number');
  ELSE
    DBMS_OUTPUT.PUT_LINE('This is an even number');
  END IF;
end;
/
```

Working code with case:

```
DECLARE
  num NUMBER;
begin
  num := '&number';
  CASE MOD(num,2)
    WHEN 1 THEN
      DBMS_OUTPUT.PUT_LINE('This is an odd number');
    ELSE
      DBMS_OUTPUT.PUT_LINE('This is an even number');
  END CASE;
end;
/
```

f)Determining if a number is prime with procedure:

Working code:

```
CREATE OR REPLACE
PROCEDURE PRIME_CHECK(NUM IN NUMBER, RESULT OUT VARCHAR2)
AS
begin
  RESULT := 'Prime';
  FOR i IN 2 .. (NUM/2)
```

```
    loop
      if MOD(NUM, i) = 0 then
        RESULT := 'Not Prime';
      exit;
      end if;
    end loop;
end;
/

--procedure check--
DECLARE
  num NUMBER;
  res VARCHAR2(20);
begin
  num := '&number';
  PRIME_CHECK(num, res);
  DBMS_OUTPUT.PUT_LINE(res);
end;
/
```

Explanation for warm up tasks: The procedural language has three sections, 'declare', 'begin', 'end'. In declare block variables are declared and values are assigned if they have any. In the 'begin' block, the logic of the code is written. The code ends with an 'end' block. The warm up tasks were simple codes with declaring variables, printing stuffs and basic conditional statements. For creating a procedure, we have to create a procedure first as shown in the code. Procedures are created where the 'declare' block was supposed to be. After creating the procedure, in the 'begin' block, the logic for the procedure is written. Then the code ends like normal code. To use this procedure, we have to write another code block with normal 'declare', 'begin' and 'end' blocks. Then, the procedure is called inside the 'begin' block and executed when the code is run.

Problems: As we learned a totally new language, I faced some common problems like assigning with '=' sign. I had a tough time

debugging the codes as there is no way to know where errors occurred. The if-else statement and mod operation also gave some difficulties. In short, common problems that occur while switching to a new language.

Main Tasks:

(1)Task 1:

Analysis: The problem requires writing a procedure that will find N richest branches and their details. If N is greater than total branch the procedure will also show an error message.

Working code:

```
CREATE OR REPLACE
PROCEDURE find_branches(n in number)
AS
ROW NUMBER(5);
BEGIN
    SELECT MAX(ROWNUM) INTO ROW
    FROM (SELECT * FROM branch ORDER BY assets DESC);

    IF(n>ROW) THEN
        DBMS_OUTPUT . PUT_LINE ('Input exceeds number of entries');
        RETURN;
    END IF;

    FOR i IN (SELECT * FROM (SELECT * FROM branch ORDER BY assets DESC)
WHERE ROWNUM<=n) LOOP
        DBMS_OUTPUT . PUT_LINE (i.branch_name || ' ' || i.branch_city || '
' || i.assets);
    END LOOP;

END;
/

DECLARE
    NUM NUMBER(5);
BEGIN
```

```
    NUM := '& number';
    FIND_BRANCHES(NUM);

END;
/
```

Explanation: First, the procedure is declared as shown in the code. Then, in the 'begin' block, a sql statement is written to find the number of branches and store it in the 'ROW' variable. Then, the input is compared with 'ROW', if the input is greater than total branches it will print an error message and end the procedure with a return statement. Then a for loop is conducted in another sql statement which gives a list of all branches based on their assets in descending order and print their details.

(2)Task 2:

Analysis: The problem requires writing a procedure that finds customer status('green zone', 'red zone') given the name of the customer. If a customer's net loan is greater than net balance then the customer is in the red zone. On the other hand, if a customer's net loan is less than net balance then the customer is in the green zone.

Working code:

```
CREATE OR REPLACE
PROCEDURE customer_status(nam in varchar2)
AS
balance number;
loan_ammount number;
BEGIN
    SELECT account.balance INTO balance
    FROM depositor, account
    WHERE depositor.customer_name = nam and depositor.account_number =
account.account_number;

    SELECT loan.amount INTO loan_ammount
    FROM borrower, loan
```

```
    WHERE borrower.customer_name = nam and borrower.loan_number =
loan.loan_number;

    IF((balance) => (loan_ammount)) THEN
        DBMS_OUTPUT.PUT_LINE('Red Zone');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Green Zone');
    END IF;
END;
/

DECLARE
    name VARCHAR2(15);
BEGIN
    name := '&Name';
    customer_status(name);
END;
/
```

Explanation: As before, first a procedure is created which takes name as input as shown in the code. Then in the 'begin' block, two sql statements are written. First one finds the account balance given a name and stores it in the balance variable, the second one finds the loan amount given a name and stores it in loan_amount variable. Then comparing balance and loan amount the status is printed. The procedure is called as shown in the code.

(3)Task 3:

Analysis: The problem requires writing a function to calculate tax for each customer. A customer is eligible for tax if his/her met balance is greater or equal to 750 and the amount of tax is 8% of his/her net balance.

Working code:

```
create or replace
FUNCTION calculatetax(name VARCHAR2)
return NUMBER
```

```
AS
balance NUMBER;
tax number;
BEGIN
    SELECT MAX(account.balance) INTO balance
    FROM depositor, account
    WHERE depositor.customer_name = name and depositor.account_number =
account.account_number;

    IF((balance)>=750) THEN
        tax := (0.08*balance);
    ELSE
        tax := 0;
    END IF;

    RETURN tax;
END;
/

DECLARE
    name VARCHAR2(15);
BEGIN
    name := '&Name';

    DBMS_OUTPUT.PUT_LINE(calculatetax(name));
END;
/
```

Explanation: Declaring a function is the same as declaring a procedure. The only difference is that after declaring a function, one needs to write a return type for it. In the 'begin' block, a sql statement is written to find the account balance of the customer and store it in the balance variable. Then, the balance is checked if it is greater or equal to 750. If the statement is true then, tax is calculated and stored in the tax variable, if false, then the zero value is assigned to the tax variable. Then finally the tax variable is returned.

(4)Task 4:

Analysis: The problem requires writing a function that will find different categories of customers. If a customer has a balance greater than 1000 and total loan less than 500 then the customer category is 'C-A1'. If a customer has a balance greater than 2000 and total loan less than 1000 then the customer category is 'C-C3'. If a customer is neither 'C-A1' nor 'C-C3' then the customer category is 'C-B1'.

Working code:

```
create or replace
FUNCTION customer_category(name varchar2)
RETURN VARCHAR2
AS
balance NUMBER;
loan_ammount number;
category VARCHAR2(10);

BEGIN
    SELECT MAX(account.balance) INTO balance
    FROM depositor, account
    WHERE depositor.customer_name = name and depositor.account_number =
account.account_number;

    SELECT MAX(loan.amount) INTO loan_ammount
    FROM borrower, loan
    WHERE borrower.customer_name = name and borrower.loan_number =
loan.loan_number;

    IF((balance) > 1000 AND (loan_ammount) < 1000) THEN
        category := 'C-A1';
    ELSIF((balance)<500 AND (loan_ammount)>2000) THEN
        category := 'C-C3';
    ELSE
        category :=' C-B1';
    END IF;
    RETURN category;
END;
/
```

```
DECLARE
    name VARCHAR2(15);
BEGIN
    name := '& Customer_Name';
    DBMS_OUTPUT.PUT_LINE(customer_category(name));
END;
/
```

Explanation: The function 'customer_catagory' takes name as a parameter and returns a varchar type value. In the 'begin' block, two sql statements are written to find account balance and loan amount of a customer and stored into the balance and the loan_amount variable. Then, they are checked if they meet the conditions given and satisfying a condition the value of the category variable is set. In the end, the category variable is returned.

Overall problems: Debugging was very difficult. Writing sql statements and passing their values in a variable was also hard to understand. Didn't understand how the for loop works at first. These tasks were comparatively more harder than the warm up tasks.