**Name: Sarvesh Kaushik**

**Lab: 5**

**Instructor: *Luke Papademas***

**Database Schema:**

CREATE TABLE tblNewData

(

dataID NUMBER(10, 0) NOT NULL,

month VARCHAR2(50) NOT NULL,

xValue Number(5),

yValue Number(5),

CONSTRAINT tblNewData\_pk PRIMARY KEY(dataID)

);

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (10, 'January', 5, 7);

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (20, 'February', 9, 11);

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (30, 'March', 12, 17);

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (40, 'April', 11, 16);

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (50, 'May', 13, 21);

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (60, 'June', 21, 14);

commit;

**Determine the Count of the Records**

* + - Run the following script to determine the count of your records.
    - Place your own name in the appropriate area where the v\_name variable is initialized.

set serveroutput on;

set verify off;

DECLARE

v\_count Number := 0;

v\_name VARCHAR(50) := '&Sarvesh';

BEGIN

--record count

SELECT COUNT(\*)

INTO v\_count

FROM tblNewData;

dbms\_output.put\_line('database report by : ' || v\_name);

dbms\_output.put\_line('record count: ' || v\_count);

END;

Output:

A picture containing graphical user interface

Description automatically generated

**Determine the Count of the Records for Some Criteria**

set serveroutput on;

DECLARE

v\_count Number := 0;

v\_num Number := 0;

v\_xVal tblNewData.xValue%type;

BEGIN

--record count

SELECT COUNT(\*)

INTO v\_count

FROM tblNewData;

--record test

For i IN 1 .. v\_count LOOP

SELECT xValue

INTO v\_xVal

FROM tblNewData

where dataID = i \* 10;

if v\_xVal > 11 then

v\_num := v\_num + 1;

end if;

end LOOP;

--output

dbms\_output.put\_line('# of matching records ' || v\_num);

dbms\_output.put\_line('report by : ' || 'Sarvesh Kaushik');

END;

OUTPUT:

Graphical user interface, application, Word

Description automatically generated

**Determine the Average of the xValues and yValues**

Run the following script to determine the average of the xValue and the average of the yValue fields. Then, adjust the script to round the output values to two decimal points.

set serveroutput on;

DECLARE

v\_count Number := 0;

v\_num Number := 0;

v\_xVal tblNewData.xValue%type;

v\_avgX Number := 0;

v\_avgY Number := 0;

BEGIN

--average xValue

SELECT Round(avg(xValue),2)

INTO v\_avgX

FROM tblNewData;

--average yValue

SELECT Round(avg(yValue),2)

INTO v\_avgY

FROM tblNewData;

--output

dbms\_output.put\_line('average of xValue field ' || v\_avgX);

dbms\_output.put\_line('average of yValue field ' || v\_avgY);

END;

OUTPUT:

Graphical user interface

Description automatically generated with low confidence

**Determine the Greater Average of two Columns**

Using if statement logic, create a script that will determine the greater average between the xValue and yValue data columns.

set serveroutput on;

DECLARE

v\_count Number := 0;

v\_num Number := 0;

v\_xVal tblNewData.xValue%type;

v\_avgX Number := 0;

v\_avgY Number := 0;

BEGIN

--average xValue

SELECT Round(avg(xValue),2)

INTO v\_avgX

FROM tblNewData;

--average yValue

SELECT Round(avg(yValue),2)

INTO v\_avgY

FROM tblNewData;

--compare and outpur

if v\_avgX > v\_avgY then

dbms\_output.put\_line('The Greatest Average is ' || v\_avgX);

else

dbms\_output.put\_line('The Greatest Average is ' || v\_avgY);

end if;

END;

Graphical user interface, application

Description automatically generated

**Determine the Greater Average of two Columns**

Using if statement logic, create a script that will determine those yValue data values that exceed the average yValue amount.

set serveroutput on;

DECLARE

v\_count Number := 0;

v\_num Number := 0;

v\_yVal tblNewData.yValue%type;

v\_avgY Number := 0;

BEGIN

--record count

SELECT COUNT(\*)

INTO v\_count

FROM tblNewData;

--average yValue

SELECT Round(avg(yValue),2)

INTO v\_avgY

FROM tblNewData;

--comparing each yvalue

dbms\_output.put\_line('YValues greater than its Average');

For i IN 1 .. v\_count LOOP

SELECT yValue

INTO v\_yVal

FROM tblNewData

where dataID = i \* 10;

if v\_yVal > v\_avgY then

dbms\_output.put\_line('' || v\_yVal);

end if;

end LOOP;

END;



**Determine the Weighted Average**

Using looping techniques, create a script that will determine the weighted average of the xValue data values. If you use if logic instead of looping techniques, then a goto <<Label>> statement pair would have to be used.

set serveroutput on;

DECLARE

v\_count Number := 6;

P\_sum Number := 0;

P\_track Number := 0;

P\_countnum Number := 0;

W\_avg Number := 0;

v\_xVal tblNewData.xValue%type;

v\_avgY Number := 0;

BEGIN

--Calculating Average

For i IN 1 .. v\_count LOOP

SELECT xValue

INTO v\_xVal

FROM tblNewData

where dataID = i \* 10;

P\_track := i\*v\_xVal;

P\_sum := P\_sum+P\_track;

P\_countnum := P\_countnum+i;

end LOOP;

W\_avg:=(P\_sum/ P\_countnum);

dbms\_output.put\_line('The Weighted Average of xValue is ' || W\_avg);

END;

A picture containing table

Description automatically generated

**Using Substitution Variables**

Use PL / SQL substitution variables to write 6 more records to the database table. Place your own choices for the xValue numbers and yValue numbers but use the months July through December, for the individual month values. Use 70 through 120 , in increments of 10 , as the key field values

Consider the following PL/SQL query using the substitution variable to get the variable values of the x and y and inserting into the table.

set serveroutput on;

set verify off;

DECLARE

v\_xVal Number(2):= '& enter';

v\_yVal Number(2):= '& enter';

BEGIN

--record count

INSERT INTO tblNewData(dataID, month, xValue, yValue)

VALUES (70, 'July',v\_xVal, v\_yVal);

END;

Similar code will follow for all the remaining months from August to December by changing the values field.

Graphical user interface, application

Description automatically generated

**Create Another Table**

Create another table named tblOldData and use the following structure and data values.

**CREATE TABLE tblOldData**

**(**

**dataID NUMBER(10, 0) NOT NULL,**

**month VARCHAR2(50) NOT NULL,**

**xValue Number(5),**

**yValue Number(5),**

**CONSTRAINT tblOldData\_pk PRIMARY KEY(dataID)**

**);**

**PL/SQL QUERY:**

set serveroutput on;

DECLARE

v\_count Number := 0;

v\_num Number := 0;

v\_xValNew tblNewData.xValue%type;

v\_xValOld tblOldData.xValue%type;

BEGIN

--record count

SELECT COUNT(\*)

INTO v\_count

FROM tblNewData;

--comparing xvalues from the both tables

For i IN 1 .. v\_count LOOP

SELECT xValue

INTO v\_xValNew

FROM tblNewData

where dataID = i \* 10;

SELECT xValue

INTO v\_xValOld

FROM tblOldData

where dataID = i \* 10;

IF v\_xValOld >= v\_xValNew THEN

dbms\_output.put\_line('old table x exceeds or equals new table x');

ELSE

dbms\_output.put\_line('old table x falls below new table x');

END IF;

end LOOP;

END;

Graphical user interface, application, Word

Description automatically generated

**Using Substitution Variables**

set serveroutput on;

DECLARE

v\_count Number := 0;

v\_num Number := 0;

v\_xValNew tblNewData.xValue%type;

v\_yValNew tblNewData.yValue%type;

v\_xValOld tblOldData.xValue%type;

v\_yValOld tblOldData.yValue%type;

My\_Name VARCHAR(50) := '&Sarvesh';

V\_sqXNew Number := 0;

V\_sqYNew Number := 0;

V\_sqXOld Number := 0;

V\_sqYOld Number := 0;

V\_sumxNew Number := 0;

V\_sumyNew Number := 0;

V\_sumxOld Number := 0;

V\_sumyOld Number := 0;

BEGIN

--record count

SELECT COUNT(\*)

INTO v\_count

FROM tblNewData;

--Squaring and Adding

For i IN 1 .. v\_count LOOP

SELECT xValue

INTO v\_xValNew

FROM tblNewData

where dataID = i \* 10;

SELECT yValue

INTO v\_yValNew

FROM tblNewData

where dataID = i \* 10;

SELECT yValue

INTO v\_yValOld

FROM tblOldData

where dataID = i \* 10;

SELECT xValue

INTO v\_xValOld

FROM tblOldData

where dataID = i \* 10;

-- Sq of new x

V\_sqXNew := v\_xValNew\*v\_xValNew;

V\_sumxNew := V\_sumxNew + V\_sqXNew;

-- sq of newy

V\_sqYNew := v\_xValNew\*v\_xValNew;

V\_sumyNew := V\_sumyNew + V\_sqYNew;

--sqof old X

V\_sqXOld := v\_xValOld\*v\_xValOld;

V\_sumxOld := V\_sumxOld + V\_sqXOld;

--sqof old Y

V\_sqYOld := v\_yValOld\*v\_yValOld;

V\_sumyOld := V\_sumyOld + V\_sqYOld;

end LOOP;

dbms\_output.put\_line('Sum of Squares of X Values in new table is ' || V\_sumxNew );

dbms\_output.put\_line('Sum of Squares of Y Values in new table is ' || V\_sumyNew );

dbms\_output.put\_line('Sum of Squares of X Values in old table is ' || V\_sumxOld );

dbms\_output.put\_line('Sum of Squares of X Values in old table is ' || V\_sumyOld );

dbms\_output.put\_line('analysis performed by ' || My\_Name);

END;

OUTPUT:

Shape

Description automatically generated with medium confidence

**Questions and Reflections Concerning this Database Project**

***When and Where should EXCEPTION statements be used in a PL - SQL block statement?***

Exceptions should always be used in the Begin and end statement in the PL/SQL Query. Whenever there are chances of errors while executing specific statements in PL/SQL we introduce the EXCEPTIONS.

In PL/SQL, we primarily care about the two types of the Exceptions,

System defined and User defined,

PL/SQL allows you to create customized exception according to the needs that falls under the Customized Exceptions.

Consider the following syntax for the EXCEPTIONS.

DECLARE

name\_exception EXCEPTION;

BEGIN

IF condition THEN

RAISE name\_exception;

END IF;

EXCEPTION

WHEN name\_exception THEN

execute statement;

END;

***When using PL - SQL, differentiate between a function, a procedure, and a Package. Point when each of these entities may be used.***

Functions really plays an important role in the PL/SQL as name suggests functions are used to perform the specific operations in PL/SQL and return the result at the end.

Procedure on the other hand is useful in returning the multiple values instead of one. It is effective in hiding the queries from external users. It achieves data security through data hiding.

A package can be considered as a schema object that logically groups the different subprograms, variables, and different types. Moreover, In the broader vie package is the collection of the different functions and procedures. As we call individual procedures and functions by using their name, similarly we can approach the packages to utilize the functions defined within.

***Distinguish between Oracle date types RRRR and YYYY .?***

YYYY and RRRR are a format to represent the year formats.

When we see the current year and want to represent the same in the four-digit format then in that case we will be primarily utilizing the YYYY format.

In the case of RRRR format we can get either 4 digit, or 2-digit year format based upon current requirements. RRRR will give user the year depends on the current database year that is already predefined in the database.

For Example:

to\_char(to\_date('86-01-01','rrrr-mm-dd'),'yyyy')

Above function will return the year as an 1986 from the database, where as the below function will return the result as 0086.

to\_char(to\_date('86-01-01','yyyy-mm-dd'),'yyyy')

***Can substitution variables be used in a function definition? Support your answer.***

The Substitution variable can be utilized in the function definition.

Consider the following example.

set serveroutput on;

CREATE OR REPLACE FUNCTION sumetest

RETURN number IS

v\_count number(2) := 0;

test\_cahar varchar(50) := '& Name';

BEGIN

SELECT COUNT(\*)

INTO v\_count

FROM tblNewData;

dbms\_output.put\_line('analysis performed by ' || test\_cahar);

RETURN v\_count ;

END;

set serveroutput on;

DECLARE

c number(2);

BEGIN

c:=sumetest();

dbms\_output.put\_line('The count is ' || c);

END;

Output:

Function SUMETEST compiled

analysis performed by Sarvesh

The count is 12

PL/SQL procedure successfully completed.

According to the above example we can use the substitution variables to source the values in the function and utilize this values at function call.

***When should for loops be used as opposed to using while loops? Support your answer with examples.***

The For loops can be Primarily used when the condition is specified and the number of the iterations are known. Moreover, The while loop can be utilized if the condition is specified and the number of the iterations is unknown.

For i IN 1 .. v\_count LOOP

end loop;

WHILE condition

LOOP

statements;

END LOOP;

While loop will be primarily used when you are not sure exactly how many times you are going to execute the body statements.