**PROJECT PL / SQL - Retrieving Data with Select Statements**

**Objective** To design and write scripts with PL / SQL .

***PROJECT DESCRIPTION***

Create a table, populate the table and then run PL / SQL that will analyze the data according to the instructions rendered below.

***Information about this Project***

This lab exercise entails creating PL / SQL scripts.

***Steps to Complete this Project***

**STEP 1**  **Open Oracle SQL Developer or Equivalent SQL IDE on Your Computer**

Launch Oracle SQL Developer or equivalent SQL application IDE.

**STEP 2**  **Create and Populate the Table(s)**

Create the SQL table(s) given below.

**[ Create Table ]**

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)

);

**[ Populate Table ]**

Add these records to the above table.

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;

**PROJECT : PL / SQL - Retrieving Data with Select Statements**

**STEP 3**  **Create and Run Various PL / SQL Scripts with Your Data Table**

**[ Script 1 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) := '&Place Your Name Here';**

**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;**

**[ Script 2 Determine the Count of the Records for Some Criteria ]**

Run the following script to determine the count of your records that have an xValue that exceeds a certain value. Place your own name in the appropriate area in the script, where the output placeholder is present.

**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 : ' || 'your name here');**

**END;**

**PROJECT : PL / SQL - Retrieving Data with Select Statements**

**[ Script 3 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 avg(xValue)**

**INTO v\_avgX**

**FROM tblNewData;**

**-- average yValue**

**SELECT avg(yValue)**

**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;**

**[ Script 4 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.

**[ Script 5 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.

**[ Script 6 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.

**PROJECT : PL / SQL - Retrieving Data with Select Statements**

Use the following weights:

***Month Weight***

January 1

February 2

March 3

April 4

May 5

June 6

**[ Script 7 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.

**[ Script 8 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)**

**);**

**PROJECT : PL / SQL - Retrieving Data with Select Statements**

**[ Table Name: tblOldData ]**

***Month key value xValue yValue***

January 10 6 12

February 20 7 10

March 30 9 12

April 40 9 11

May 50 10 15

June 60 12 19

July 70 16 22

August 80 10 18

September 90 10 30

October 100 11 17

November 110 14 14

December 120 16 20

**[ Script 9 Compare the Tables ]**

Write a script that individually compares the magnitudes of each pair of xValues and yValues from both the new and old tables.

Here we can merely test the individual pairs of the xValue and the yValue entries on a row by row basis of each record.

For the first rows ( January ) this would involve:

an xValue from tblNewData ( for example xValue = 5 from January )

an xValue from tblOldData ( for example xValue = 6 from January )

If we assume that SELECT INTO statements read these values as v\_xValOld and v\_xValNew from the two tables then something like the following pseudo - code would suffice.

**IF v\_xValOld >= v\_xValNew THEN**

**Print 'old table x exceeds or equals new table x'**

**ELSE**

**Print 'old table x falls below new table x'**

**END IF**

**PROJECT : PL / SQL - Retrieving Data with Select Statements**

Using a loop we can then perform row by row comparisons of both the x and y values from both tables in this fashion.

**[ Script 10 Using Substitution Variables ]**

Perform sum of squares analysis on both tables by adding the squares of both the xValues and yValues for each table. That is, for each table, square the xValue entries and the yValue entries and sum the squares.

Use a substitution variable here only to input your own name when running the script. For example, you can input your name from substitution variable v\_me and say at the end of the sum of squares analysis use a statement similar to:

**dbms\_output.put\_line('analysis performed by ' || v\_me);**

No need to use substitution variables here to read any x or y values.

**STEP 3**  **Submit Your Scripts**

Submit your scripts and screen snapshots of the output of each script.

**STEP 4**  **PL/SQL Tuning**

Performing the data analysis for this lab via the use of PL/SQL programming, should also be followed with PL/SQL tuning. Execute the following procedures, etc., and place a snapshot of the results in your lab submittal document.

1. ALTER SESSION SET PLSQL\_WARNINGS='ENABLE:ALL';
2. Add the following exception for Script No. 3:

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DMBS\_OUTPUT.PUT\_LINE('CANNOT DIVIDE BY ZERO.');

(3) ALTER SESSION SET PLSQL\_WARNINGS='DISABLE:ALL';

(4) Run the following code that features nested queries. Which query was more efficient?

**SET SERVEROUTPUT on;**

**DECLARE**

**starting\_time TIMESTAMP WITH TIME ZONE;**

**ending\_time TIMESTAMP WITH TIME ZONE;**

**BEGIN**

**-- Invokes SQRT for every row of employees table:**

**SELECT SYSTIMESTAMP INTO starting\_time FROM DUAL;**

**FOR item IN (**

**SELECT DISTINCT(SQRT(department\_id)) col\_alias**

**FROM employees**

**ORDER BY col\_alias**

**)**

**LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Square root of dept. ID = ' || item.col\_alias);**

**END LOOP;**

**SELECT SYSTIMESTAMP INTO ending\_time FROM DUAL;**

**DBMS\_OUTPUT.PUT\_LINE('Time = ' || TO\_CHAR(ending\_time - starting\_time));**

**-- Invokes SQRT for every distinct department\_id of employees table:**

**SELECT SYSTIMESTAMP INTO starting\_time FROM DUAL;**

**FOR item IN (**

**SELECT SQRT(department\_id) col\_alias**

**FROM (SELECT DISTINCT department\_id FROM employees)**

**ORDER BY col\_alias**

**)**

**LOOP**

**IF item.col\_alias IS NOT NULL THEN**

**DBMS\_OUTPUT.PUT\_LINE('Square root of dept. ID = ' || item.col\_alias);**

**END IF;**

**END LOOP;**

**SELECT SYSTIMESTAMP INTO ending\_time FROM DUAL;**

**DBMS\_OUTPUT.PUT\_LINE('Time = ' || TO\_CHAR(ending\_time - starting\_time));**

**END;**

**/**

**STEP 5**  **Questions and Reflections Concerning this Database Project**

Now that you have completed this lab project, review the questions below to reflect on the procedures and settings that you utilized as you followed the steps to complete the project. Place your responses in your lab submittal document.

**(1)** When and where should **EXCEPTION** statements be used in a PL - SQL block

statement?

**(2)** When using PL - SQL , differentiate between a function, a procedure and a

Package. Point when each of these entities may be used.

**(3)** Distinguish between Oracle date types RRRR and YYYY .

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

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