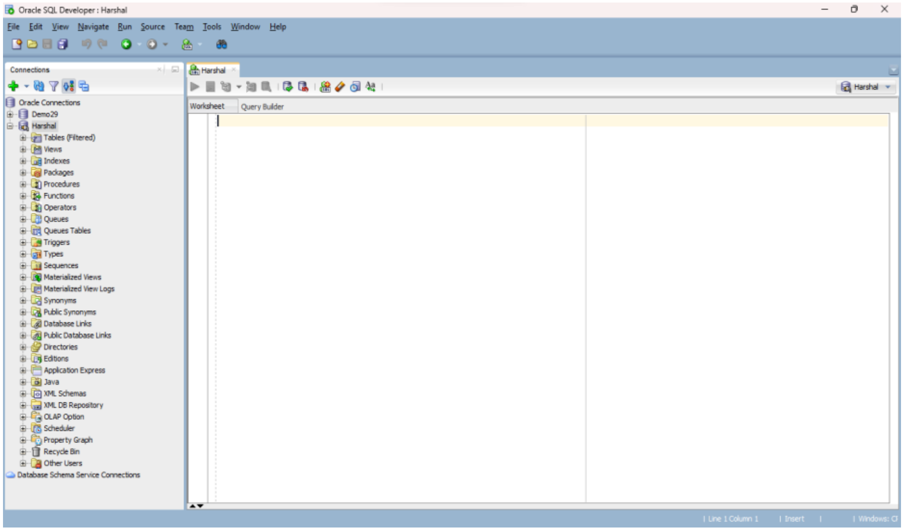
Harshal Sawant
A20538827
ITMD 523 - 05

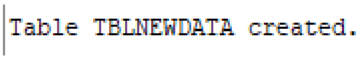
Lab 5 Submittal

Step 1: Launch Oracle SQL Developer or equivalent SQL application IDE.



Step 2: Create and Populate the Table

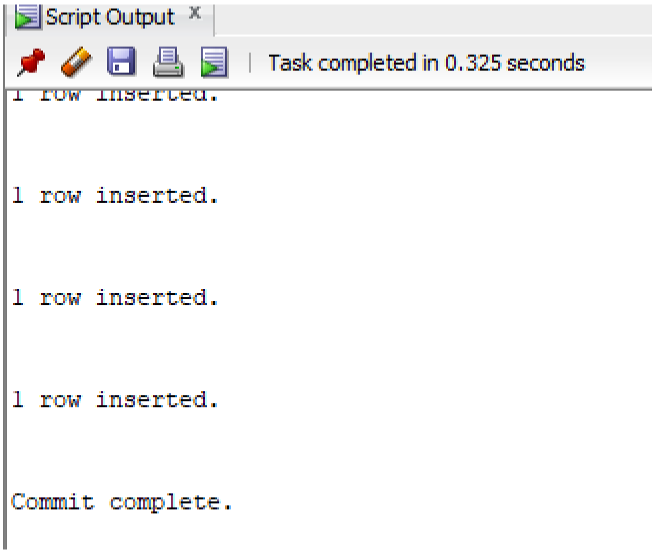
Create Table:



```
Table TBLNEWDATA created.
```

Output of the create table query.

Populating the table:

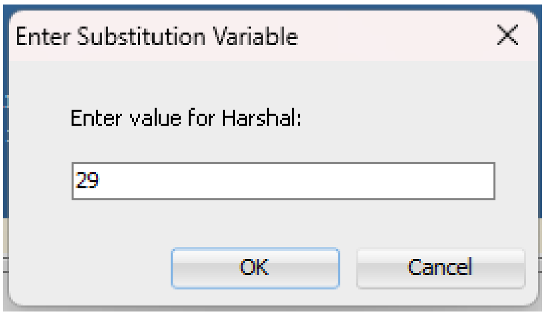


```
1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


Commit complete.
```

Output after adding 6 new entries to the table

Step 3: Create and Run Various PL / SQL Scripts with Your Data Table

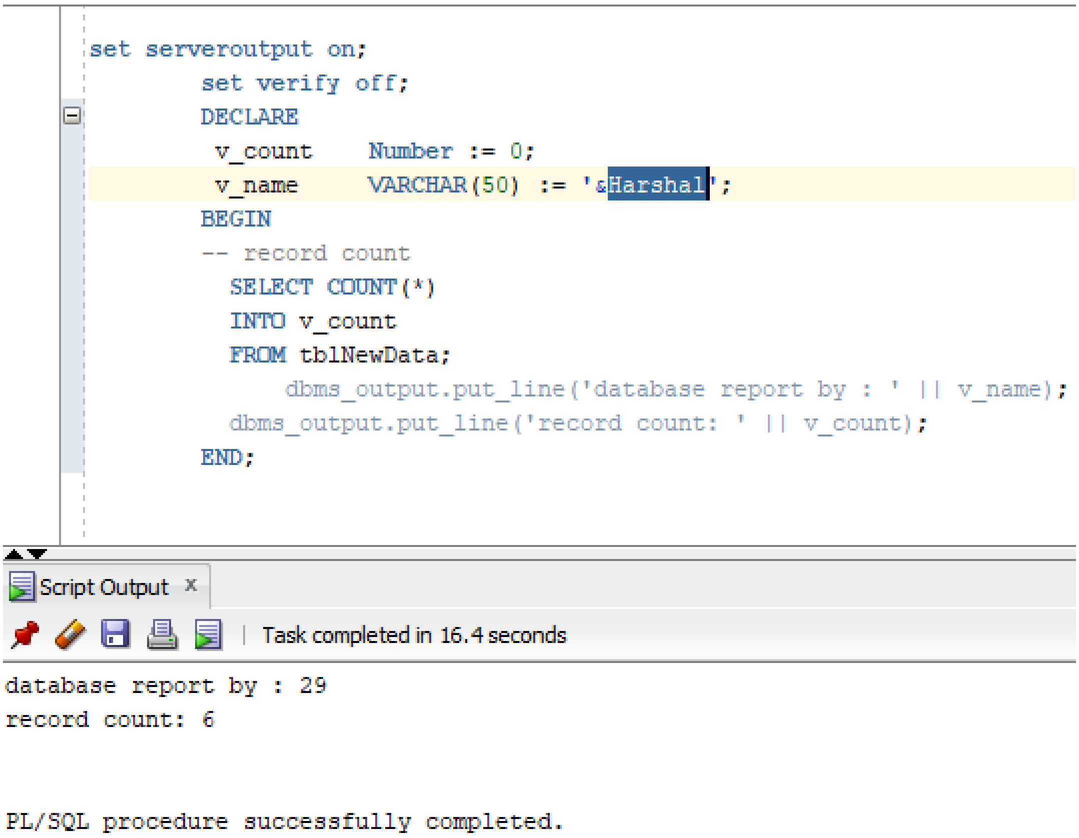
Script 1: **Determine the Count of the Records**



Enter Substitution Variable

Enter value for Harshal:

29



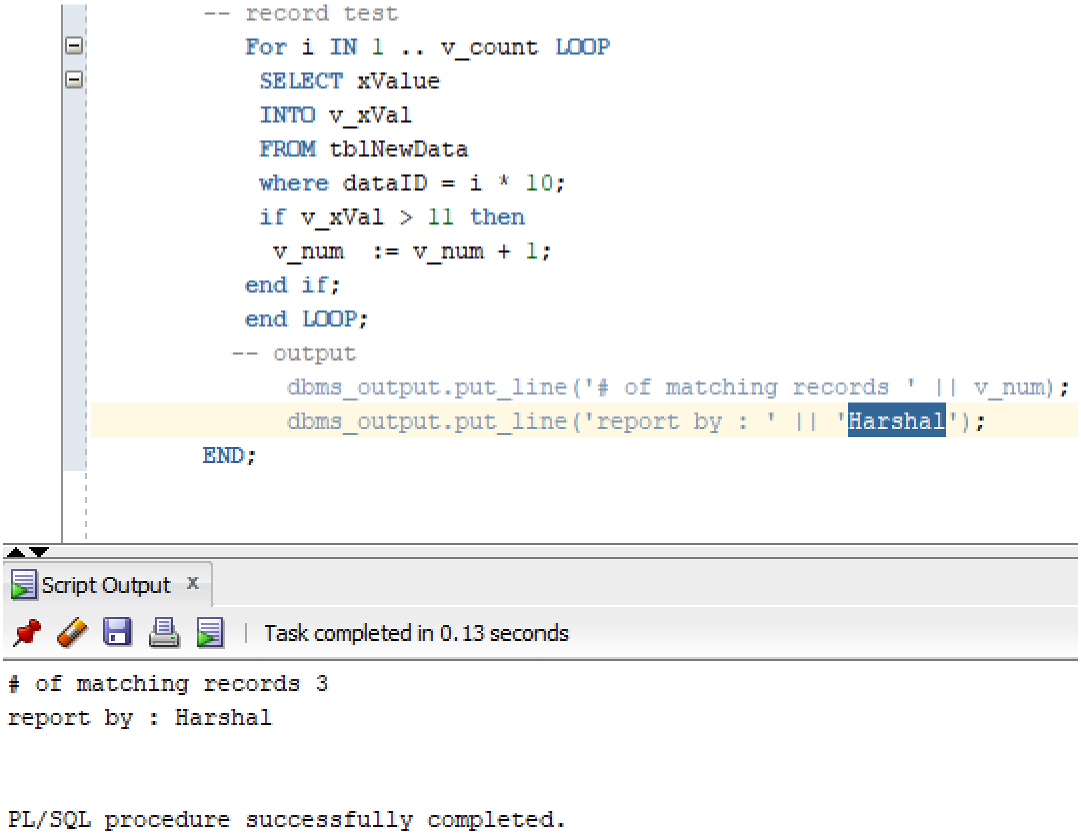
```
set serveroutput on;
        set verify off;
        DECLARE
         v_count     Number := 0;
         v_name      VARCHAR(50)  := '&Harshal';
        BEGIN
        -- record count
          SELECT COUNT(*)
          INTO v_count
          FROM tblNewData;
              dbms_output.put_line('database report by : ' || v_name);
          dbms_output.put_line('record count: ' || v_count);
        END;
```

Task completed in 16.4 seconds

```
database report by : 29
record count: 6


PL/SQL procedure successfully completed.
```

Output of the script where I entered v_name = Harshal

Script 2: **Determine the Count of the Records for Some Criteria**



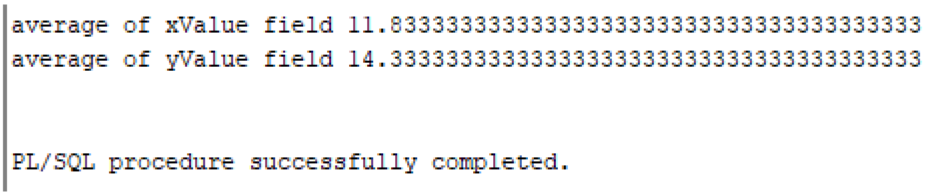
```
        -- record test
           For i IN 1 .. v_count LOOP
            SELECT xValue
            INTO v_xVal
            FROM tblNewData
            where dataID = i * 10;
            if v_xVal > 11 then
             v_num  := v_num + 1;
           end if;
           end LOOP;
          -- output
              dbms_output.put_line('# of matching records ' || v_num);
              dbms_output.put_line('report by : ' || 'Harshal');
        END;
```

Task completed in 0.13 seconds

```
# of matching records 3
report by : Harshal


PL/SQL procedure successfully completed.
```

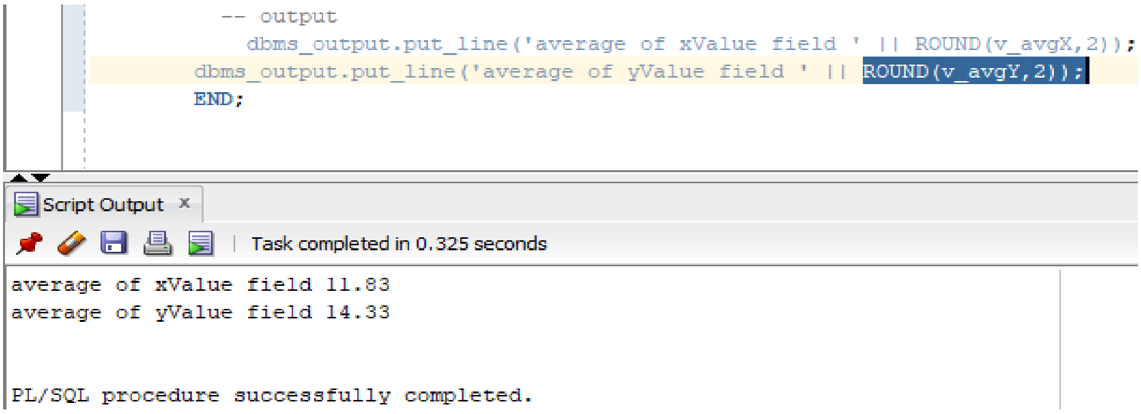
Output of the script where I made the changes where needed.

Script 3: **Determine the Average of the xValues and yValues.**



```
average of xValue field 11.83333333333333333333333333333333333333
average of yValue field 14.33333333333333333333333333333333333333


PL/SQL procedure successfully completed.
```

Normal Output



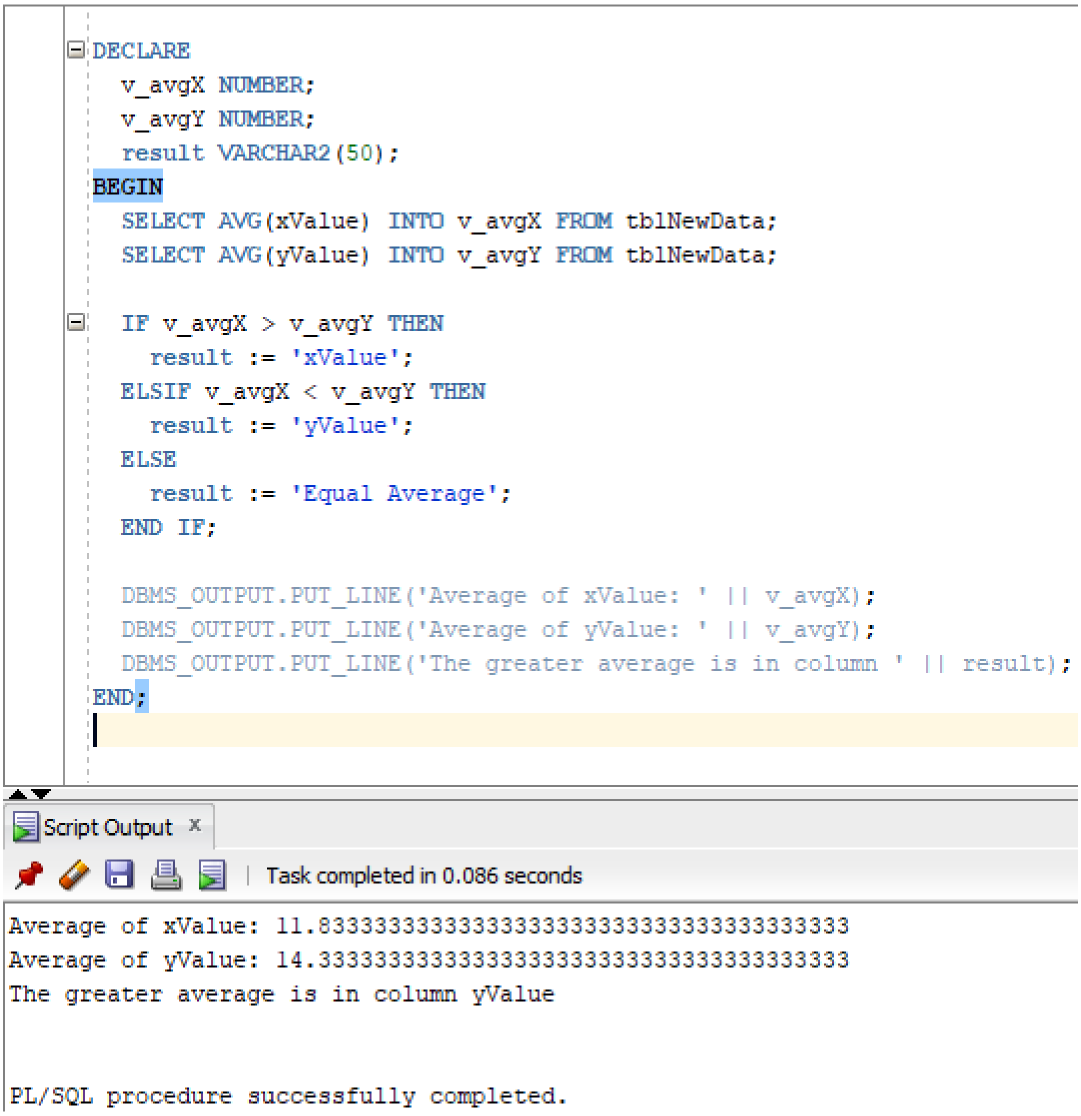
```
          -- output
            dbms_output.put_line('average of xValue field ' || ROUND(v_avgX,2));
        dbms_output.put_line('average of yValue field ' || ROUND(v_avgY,2));
        END;
```

Task completed in 0.325 seconds

```
average of xValue field 11.83
average of yValue field 14.33


PL/SQL procedure successfully completed.
```

Output after using ROUND function with 2 decimal places.

Script 4: **Determine the Greater Average of two Columns**

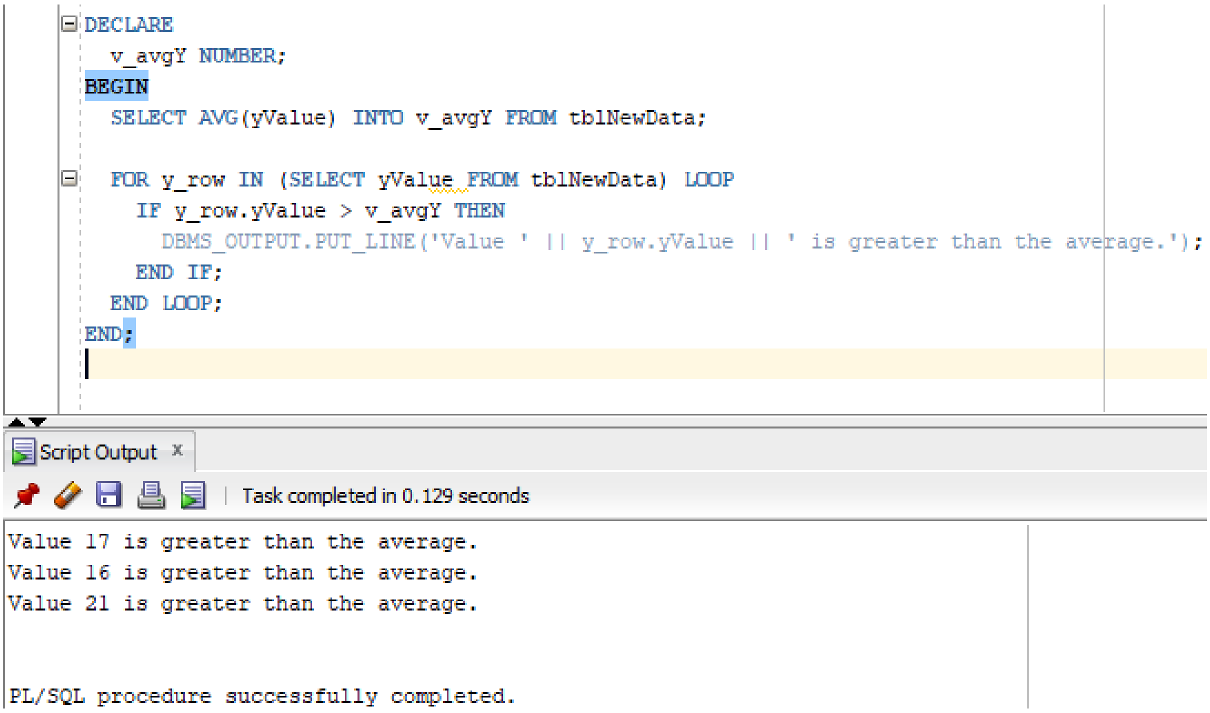


```
DECLARE
  v_avgX NUMBER;
  v_avgY NUMBER;
  result VARCHAR2(50);
BEGIN
  SELECT AVG(xValue) INTO v_avgX FROM tblNewData;
  SELECT AVG(yValue) INTO v_avgY FROM tblNewData;

  IF v_avgX > v_avgY THEN
    result := 'xValue';
  ELSIF v_avgX < v_avgY THEN
    result := 'yValue';
  ELSE
    result := 'Equal Average';
  END IF;

  DBMS_OUTPUT.PUT_LINE('Average of xValue: ' || v_avgX);
  DBMS_OUTPUT.PUT_LINE('Average of yValue: ' || v_avgY);
  DBMS_OUTPUT.PUT_LINE('The greater average is in column ' || result);
END;
```

```
Average of xValue: 11.83333333333333333333333333333333333333
Average of yValue: 14.33333333333333333333333333333333333333
The greater average is in column yValue


PL/SQL procedure successfully completed.
```

Output

Script 5: **Determine the Greater Average of two Columns**



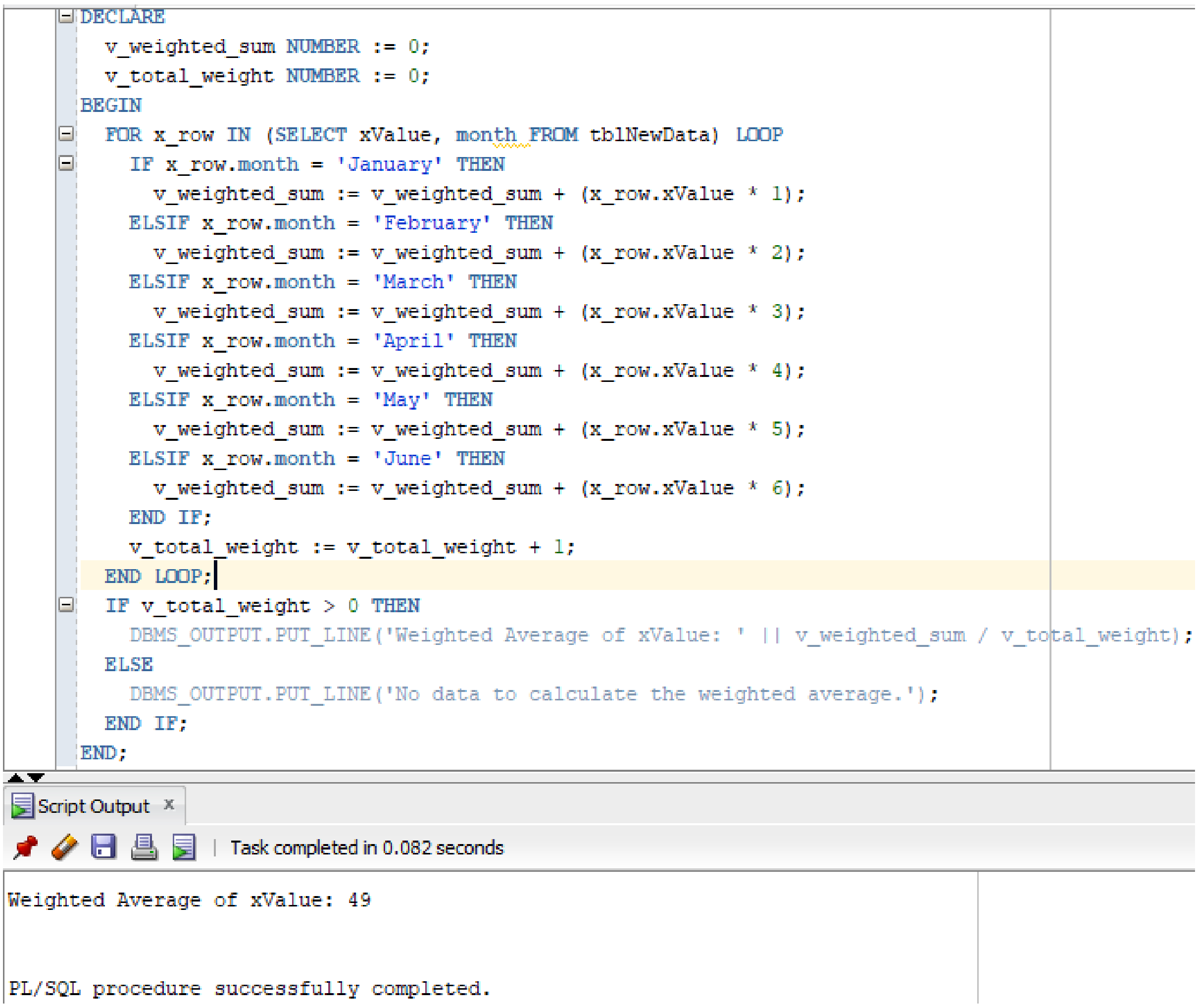
```
DECLARE
  v_avgY NUMBER;
BEGIN
  SELECT AVG(yValue) INTO v_avgY FROM tblNewData;

  FOR y_row IN (SELECT yValue FROM tblNewData) LOOP
    IF y_row.yValue > v_avgY THEN
      DBMS_OUTPUT.PUT_LINE('Value ' || y_row.yValue || ' is greater than the average.');
    END IF;
  END LOOP;
END;
```

Task completed in 0.129 seconds

```
Value 17 is greater than the average.
Value 16 is greater than the average.
Value 21 is greater than the average.


PL/SQL procedure successfully completed.
```

Output

Script 6: Determine the Weighted Average



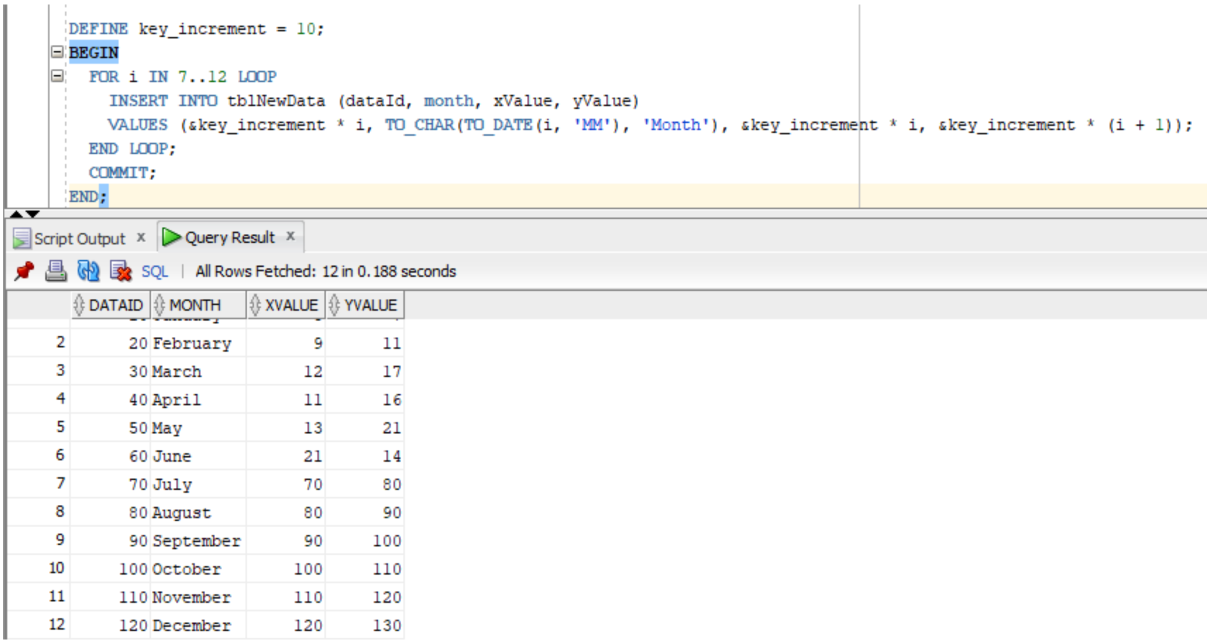
```
DECLARE
  v_weighted_sum NUMBER := 0;
  v_total_weight NUMBER := 0;
BEGIN
  FOR x_row IN (SELECT xValue, month FROM tblNewData) LOOP
    IF x_row.month = 'January' THEN
      v_weighted_sum := v_weighted_sum + (x_row.xValue * 1);
    ELSIF x_row.month = 'February' THEN
      v_weighted_sum := v_weighted_sum + (x_row.xValue * 2);
    ELSIF x_row.month = 'March' THEN
      v_weighted_sum := v_weighted_sum + (x_row.xValue * 3);
    ELSIF x_row.month = 'April' THEN
      v_weighted_sum := v_weighted_sum + (x_row.xValue * 4);
    ELSIF x_row.month = 'May' THEN
      v_weighted_sum := v_weighted_sum + (x_row.xValue * 5);
    ELSIF x_row.month = 'June' THEN
      v_weighted_sum := v_weighted_sum + (x_row.xValue * 6);
    END IF;
    v_total_weight := v_total_weight + 1;
  END LOOP;
  IF v_total_weight > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Weighted Average of xValue: ' || v_weighted_sum / v_total_weight);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No data to calculate the weighted average.');
  END IF;
END;
```

Task completed in 0.082 seconds

```
Weighted Average of xValue: 49


PL/SQL procedure successfully completed.
```

Output

Script 7: Using Substitution Variables

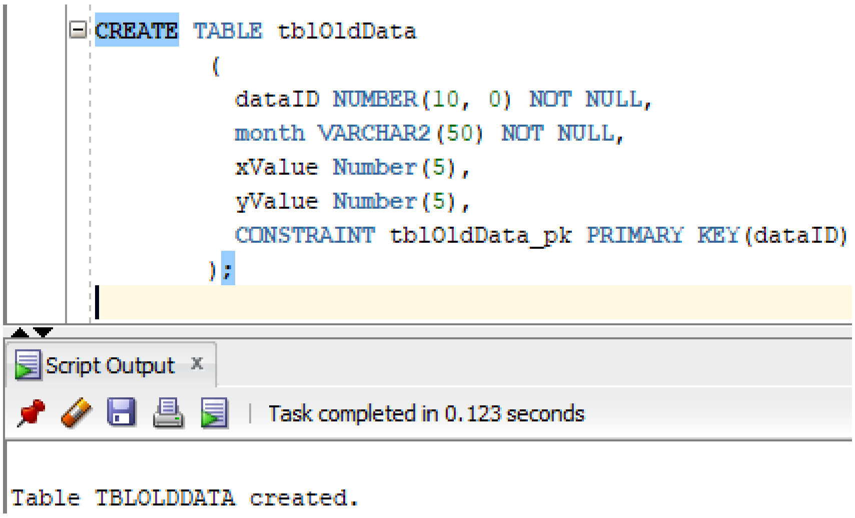


```
DEFINE key_increment = 10;
BEGIN
  FOR i IN 7..12 LOOP
    INSERT INTO tblNewData (dataId, month, xValue, yValue)
    VALUES (&key_increment * i, TO_CHAR(TO_DATE(i, 'MM'), 'Month'), &key_increment * i, &key_increment * (i + 1));
  END LOOP;
  COMMIT;
END;
```

All Rows Fetched: 12 in 0.188 seconds

	DATAID	MONTH	XVALUE	YVALUE
2	20	February	9	11
3	30	March	12	17
4	40	April	11	16
5	50	May	13	21
6	60	June	21	14
7	70	July	70	80
8	80	August	80	90
9	90	September	90	100
10	100	October	100	110
11	110	November	110	120
12	120	December	120	130

Script 8: Create another Table.

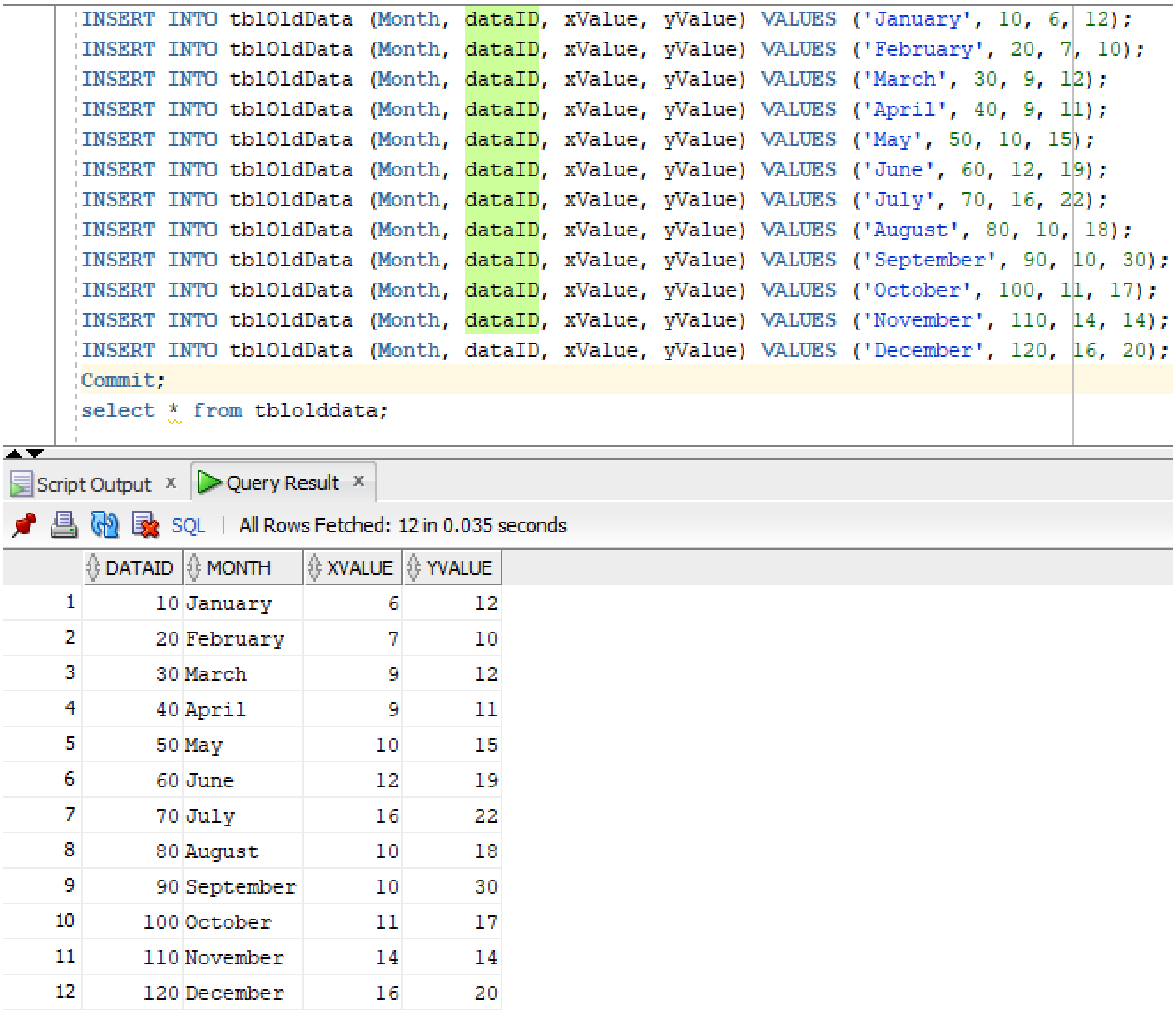


```
CREATE TABLE tblOldData
        (
          dataID NUMBER(10, 0) NOT NULL,
          month VARCHAR2(50) NOT NULL,
          xValue Number(5),
          yValue Number(5),
          CONSTRAINT tblOldData_pk PRIMARY KEY(dataID)
        );
```

Task completed in 0.123 seconds

```
Table TBLOLDDATA created.
```

Output



```
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('January', 10, 6, 12);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('February', 20, 7, 10);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('March', 30, 9, 12);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('April', 40, 9, 11);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('May', 50, 10, 15);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('June', 60, 12, 19);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('July', 70, 16, 22);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('August', 80, 10, 18);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('September', 90, 10, 30);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('October', 100, 11, 17);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('November', 110, 14, 14);
INSERT INTO tblOldData (Month, dataID, xValue, yValue) VALUES ('December', 120, 16, 20);
Commit;
select * from tblolddata;
```

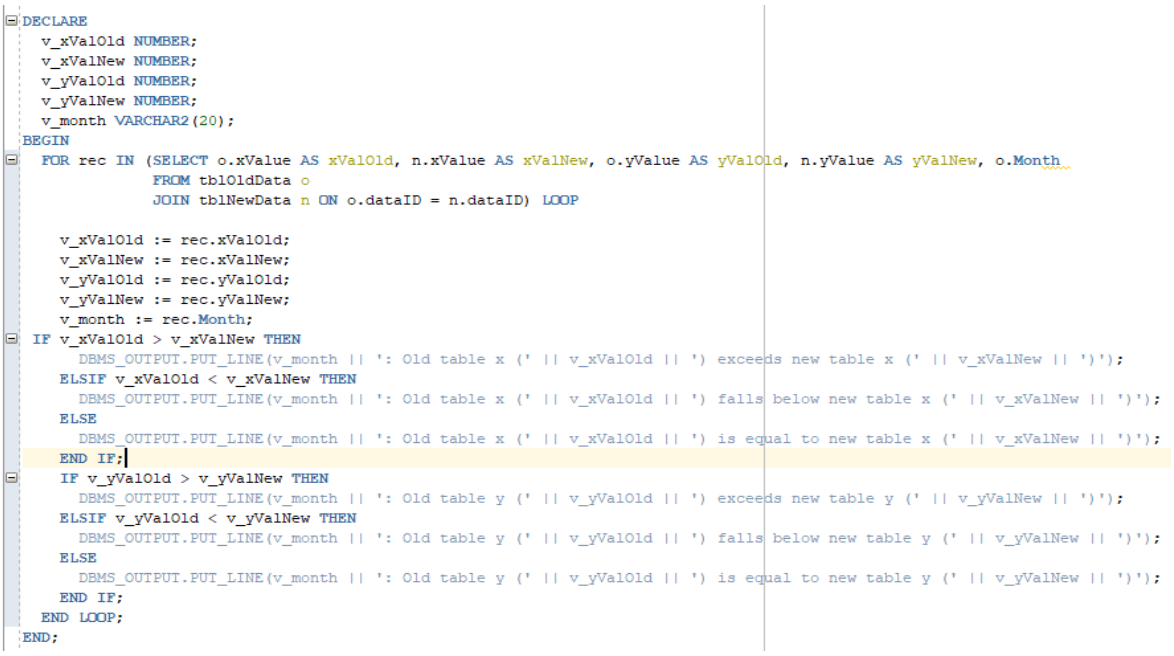
All Rows Fetched: 12 in 0.035 seconds

	DATAID	MONTH	XVALUE	YVALUE
1	10	January	6	12
2	20	February	7	10
3	30	March	9	12
4	40	April	9	11
5	50	May	10	15
6	60	June	12	19
7	70	July	16	22
8	80	August	10	18
9	90	September	10	30
10	100	October	11	17
11	110	November	14	14
12	120	December	16	20

Adding the data

Harshal Sawant
A20538827
ITMD 523 - 05

Script 9: Compare the tables.



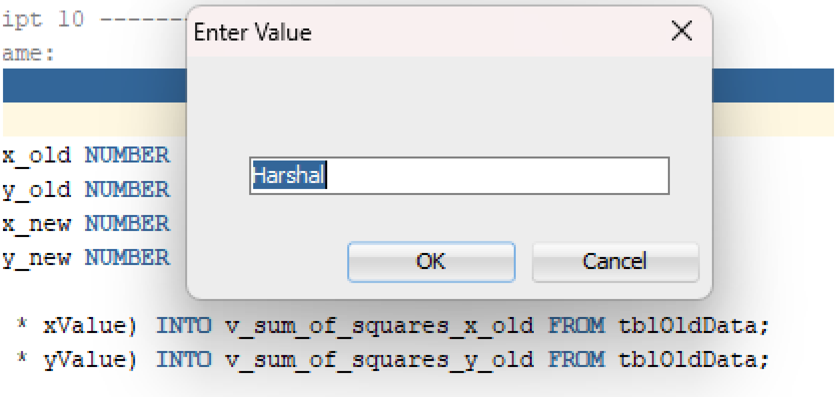
```
DECLARE
  v_xValOld NUMBER;
  v_xValNew NUMBER;
  v_yValOld NUMBER;
  v_yValNew NUMBER;
  v_month VARCHAR2(20);
BEGIN
  FOR rec IN (SELECT o.xValue AS xValOld, n.xValue AS xValNew, o.yValue AS yValOld, n.yValue AS yValNew, o.Month
              FROM tblOldData o
              JOIN tblNewData n ON o.dataID = n.dataID) LOOP

    v_xValOld := rec.xValOld;
    v_xValNew := rec.xValNew;
    v_yValOld := rec.yValOld;
    v_yValNew := rec.yValNew;
    v_month := rec.Month;
 IF v_xValOld > v_xValNew THEN
      DBMS_OUTPUT.PUT_LINE(v_month || ': Old table x (' || v_xValOld || ') exceeds new table x (' || v_xValNew || ')');
    ELSIF v_xValOld < v_xValNew THEN
      DBMS_OUTPUT.PUT_LINE(v_month || ': Old table x (' || v_xValOld || ') falls below new table x (' || v_xValNew || ')');
    ELSE
      DBMS_OUTPUT.PUT_LINE(v_month || ': Old table x (' || v_xValOld || ') is equal to new table x (' || v_xValNew || ')');
    END IF;
    IF v_yValOld > v_yValNew THEN
      DBMS_OUTPUT.PUT_LINE(v_month || ': Old table y (' || v_yValOld || ') exceeds new table y (' || v_yValNew || ')');
    ELSIF v_yValOld < v_yValNew THEN
      DBMS_OUTPUT.PUT_LINE(v_month || ': Old table y (' || v_yValOld || ') falls below new table y (' || v_yValNew || ')');
    ELSE
      DBMS_OUTPUT.PUT_LINE(v_month || ': Old table y (' || v_yValOld || ') is equal to new table y (' || v_yValNew || ')');
    END IF;
  END LOOP;
END;
```

The script

```
January: Old table x (6) exceeds new table x (5)
January: Old table y (12) exceeds new table y (7)
February: Old table x (7) falls below new table x (9)
February: Old table y (10) falls below new table y (11)
March: Old table x (9) falls below new table x (12)
March: Old table y (12) falls below new table y (17)
April: Old table x (9) falls below new table x (11)
April: Old table y (11) falls below new table y (16)
May: Old table x (10) falls below new table x (13)
May: Old table y (15) falls below new table y (21)
June: Old table x (12) falls below new table x (21)
June: Old table y (19) exceeds new table y (14)
July: Old table x (16) falls below new table x (70)
July: Old table y (22) falls below new table y (80)
August: Old table x (10) falls below new table x (80)
August: Old table y (18) falls below new table y (90)
September: Old table x (10) falls below new table x (90)
September: Old table y (30) falls below new table y (100)
October: Old table x (11) falls below new table x (100)
October: Old table y (17) falls below new table y (110)
November: Old table x (14) falls below new table x (110)
November: Old table y (14) falls below new table y (120)
December: Old table x (16) falls below new table x (120)
December: Old table y (20) falls below new table y (130)


PL/SQL procedure successfully completed.
```

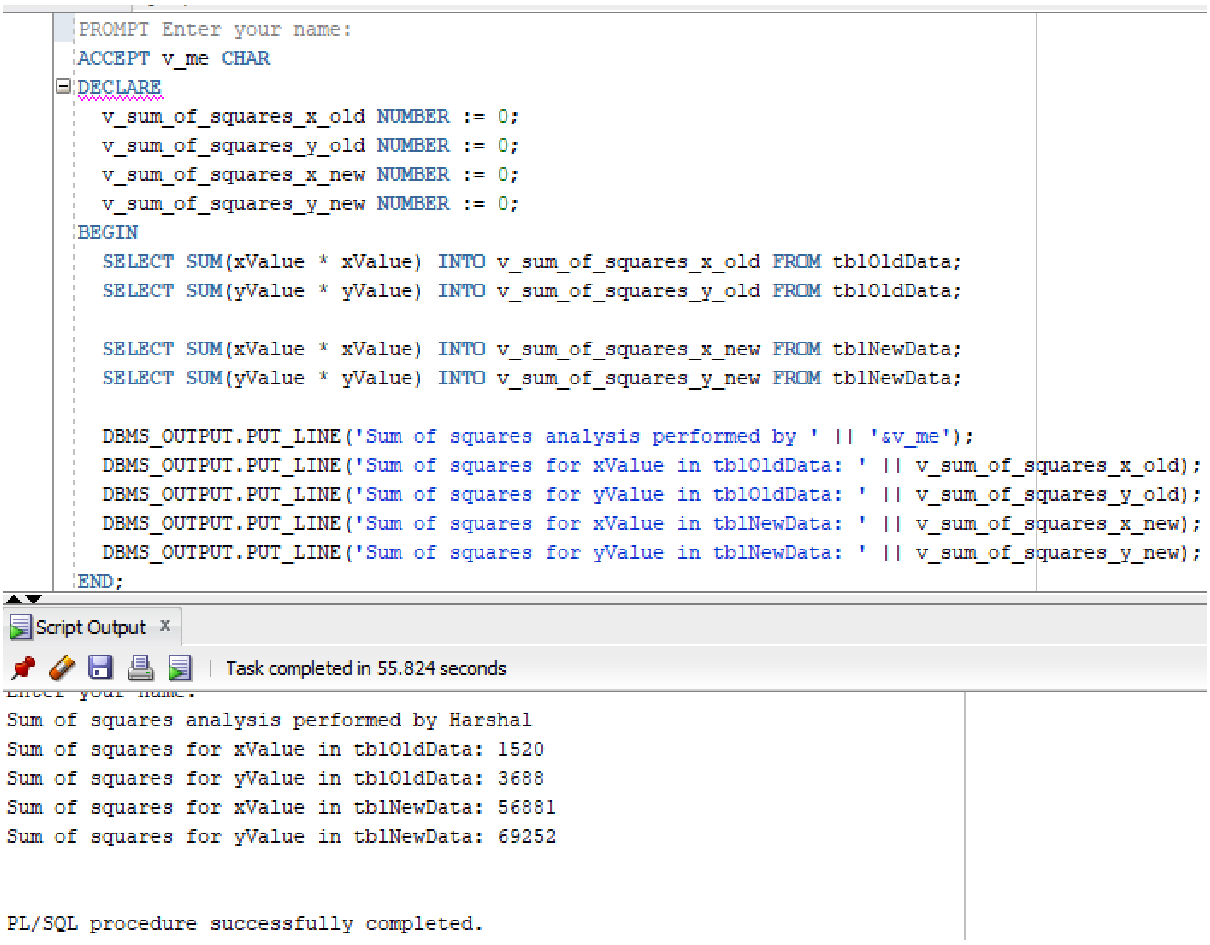
The Output

Script 10: Using Substitution Values.



```
ipt 10 ------
ame:

x_old NUMBER
y_old NUMBER
x_new NUMBER
y_new NUMBER

 * xValue) INTO v_sum_of_squares_x_old FROM tblOldData;
 * yValue) INTO v_sum_of_squares_y_old FROM tblOldData;
```

Accepting name of the analyst at runtime



```
PROMPT Enter your name:
ACCEPT v_me CHAR
DECLARE
  v_sum_of_squares_x_old NUMBER := 0;
  v_sum_of_squares_y_old NUMBER := 0;
  v_sum_of_squares_x_new NUMBER := 0;
  v_sum_of_squares_y_new NUMBER := 0;
BEGIN
  SELECT SUM(xValue * xValue) INTO v_sum_of_squares_x_old FROM tblOldData;
  SELECT SUM(yValue * yValue) INTO v_sum_of_squares_y_old FROM tblOldData;

  SELECT SUM(xValue * xValue) INTO v_sum_of_squares_x_new FROM tblNewData;
  SELECT SUM(yValue * yValue) INTO v_sum_of_squares_y_new FROM tblNewData;

  DBMS_OUTPUT.PUT_LINE('Sum of squares analysis performed by ' || '&v_me');
  DBMS_OUTPUT.PUT_LINE('Sum of squares for xValue in tblOldData: ' || v_sum_of_squares_x_old);
  DBMS_OUTPUT.PUT_LINE('Sum of squares for yValue in tblOldData: ' || v_sum_of_squares_y_old);
  DBMS_OUTPUT.PUT_LINE('Sum of squares for xValue in tblNewData: ' || v_sum_of_squares_x_new);
  DBMS_OUTPUT.PUT_LINE('Sum of squares for yValue in tblNewData: ' || v_sum_of_squares_y_new);
END;
```

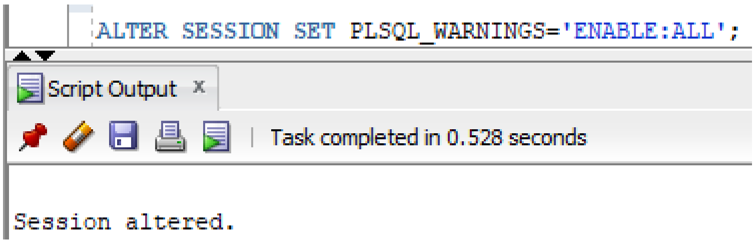
```
Enter your name:
Sum of squares analysis performed by Harshal
Sum of squares for xValue in tblOldData: 1520
Sum of squares for yValue in tblOldData: 3688
Sum of squares for xValue in tblNewData: 56881
Sum of squares for yValue in tblNewData: 69252


PL/SQL procedure successfully completed.
```

Output

Harshal Sawant
A20538827
ITMD 523 - 05

Step 4: PL/SQL Tuning

1)

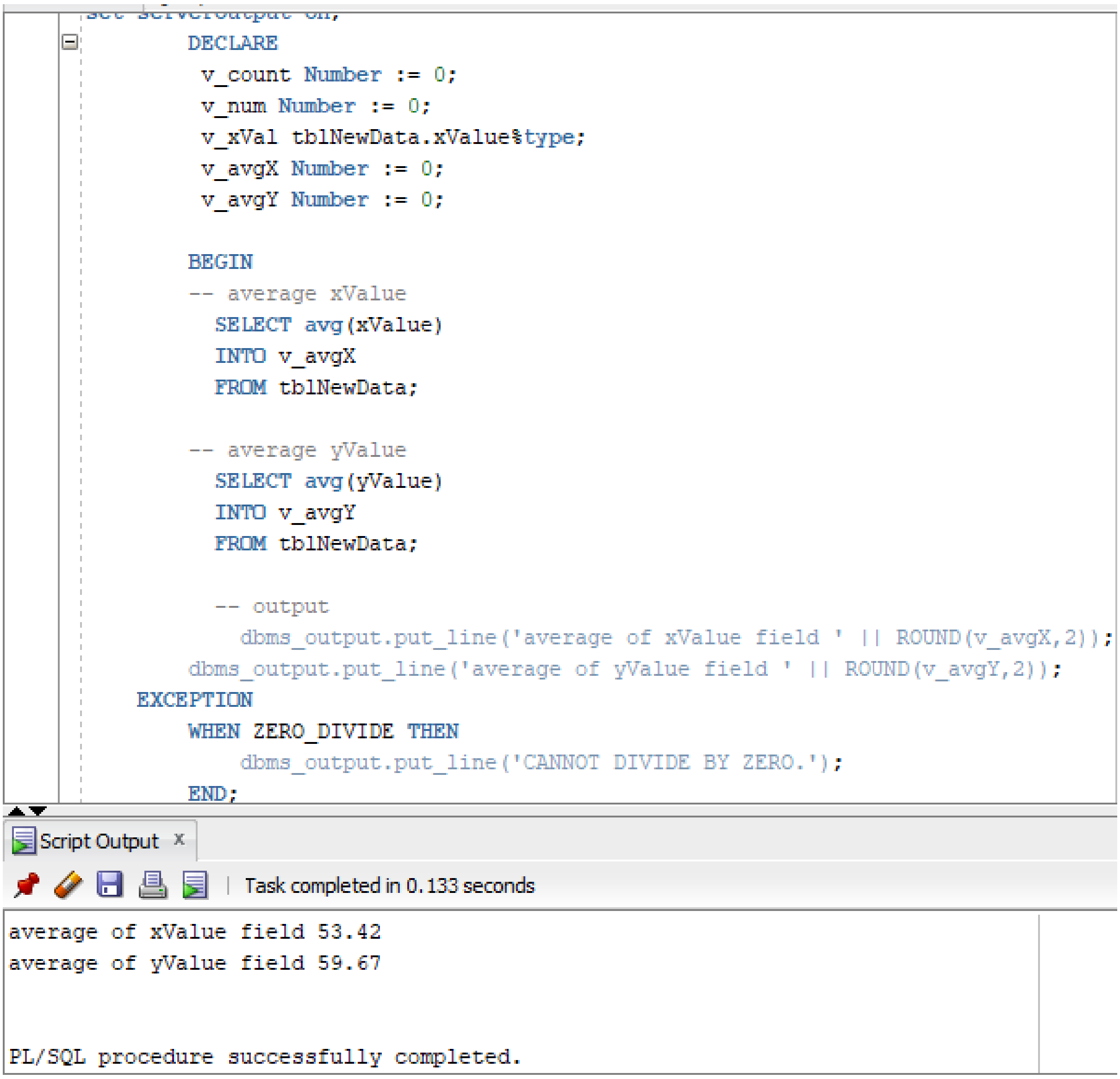


```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:ALL';
```

Task completed in 0.528 seconds

```
Session altered.
```

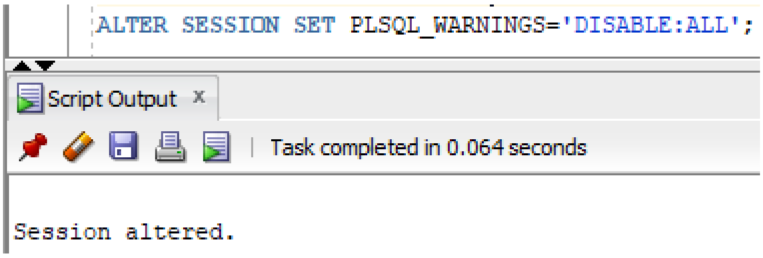
2)



```
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 ' || ROUND(v_avgX,2));
        dbms_output.put_line('average of yValue field ' || ROUND(v_avgY,2));
    EXCEPTION
        WHEN ZERO_DIVIDE THEN
            dbms_output.put_line('CANNOT DIVIDE BY ZERO.');
        END;
```

```
average of xValue field 53.42
average of yValue field 59.67


PL/SQL procedure successfully completed.
```

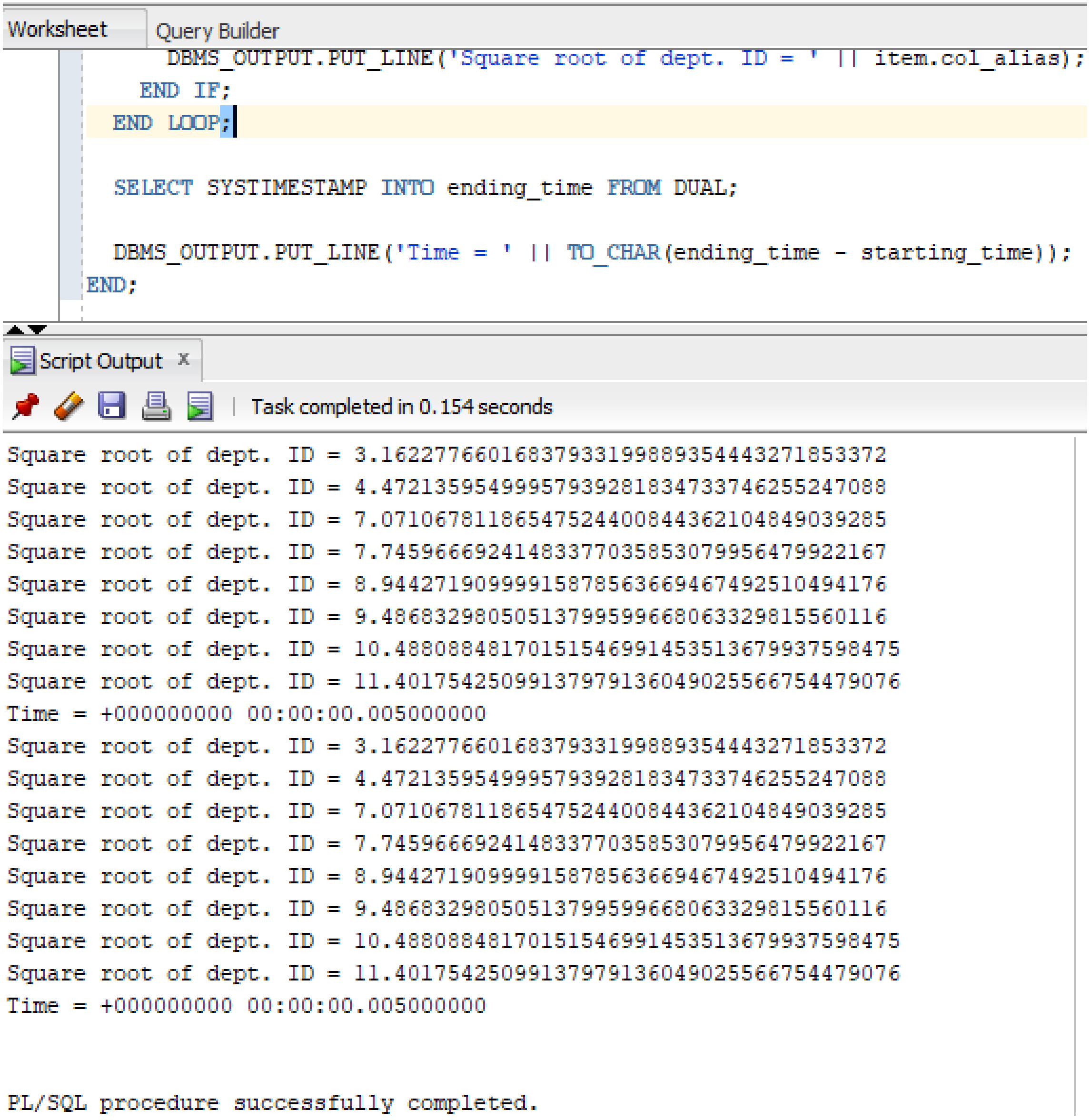
3)



```
ALTER SESSION SET PLSQL_WARNINGS='DISABLE:ALL';
```

```
Session altered.
```

4)

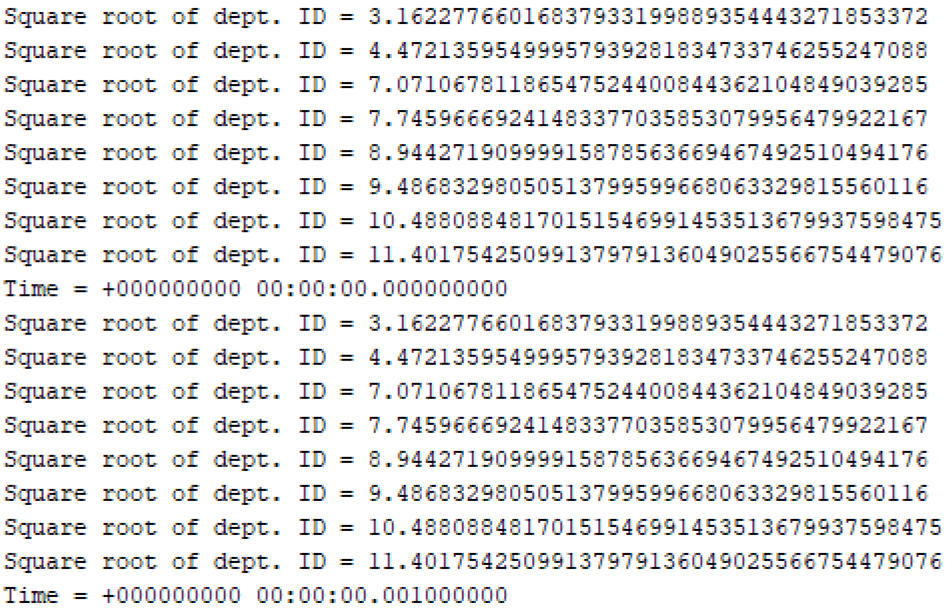


```
        DBMS_OUTPUT.PUT_LINE('Square root of dept. ID = ' || item.col_alias);
      END IF;
    END LOOP;

    SELECT SYSTIMESTAMP INTO ending_time FROM DUAL;

    DBMS_OUTPUT.PUT_LINE('Time = ' || TO_CHAR(ending_time - starting_time));
END;
```

```
Square root of dept. ID = 3.16227766016837933199889354443271853372
Square root of dept. ID = 4.47213595499957939281834733746255247088
Square root of dept. ID = 7.07106781186547524400844362104849039285
Square root of dept. ID = 7.74596669241483377035853079956479922167
Square root of dept. ID = 8.94427190999915878563669467492510494176
Square root of dept. ID = 9.48683298050513799599668063329815560116
Square root of dept. ID = 10.48808848170151546991453513679937598475
Square root of dept. ID = 11.40175425099137979136049025566754479076
Time = +000000000 00:00:00.005000000
Square root of dept. ID = 3.16227766016837933199889354443271853372
Square root of dept. ID = 4.47213595499957939281834733746255247088
Square root of dept. ID = 7.07106781186547524400844362104849039285
Square root of dept. ID = 7.74596669241483377035853079956479922167
Square root of dept. ID = 8.94427190999915878563669467492510494176
Square root of dept. ID = 9.48683298050513799599668063329815560116
Square root of dept. ID = 10.48808848170151546991453513679937598475
Square root of dept. ID = 11.40175425099137979136049025566754479076
Time = +000000000 00:00:00.005000000


PL/SQL procedure successfully completed.
```

Because it applies the SQRT function to the workers table directly and doesn't require an additional subquery, the first portion is more efficient. In general, this direct square root computation for every individual department is more efficient. On the other hand, the second component adds cost and becomes somewhat less efficient by requiring an extra subquery to get unique department IDs. The first portion is typically more efficient because it minimizes needless subqueries, even if the two sections' timing differences are negligible (both reporting 0.005 seconds).

<u>But if you run the query again it gives different output:</u>



```
Square root of dept. ID = 3.16227766016837933199889354443271853372
Square root of dept. ID = 4.47213595499957939281834733746255247088
Square root of dept. ID = 7.07106781186547524400844362104849039285
Square root of dept. ID = 7.74596669241483377035853079956479922167
Square root of dept. ID = 8.94427190999915878563669467492510494176
Square root of dept. ID = 9.48683298050513799599668063329815560116
Square root of dept. ID = 10.48808848170151546991453513679937598475
Square root of dept. ID = 11.40175425099137979136049025566754479076
Time = +000000000 00:00:00.000000000
Square root of dept. ID = 3.16227766016837933199889354443271853372
Square root of dept. ID = 4.47213595499957939281834733746255247088
Square root of dept. ID = 7.07106781186547524400844362104849039285
Square root of dept. ID = 7.74596669241483377035853079956479922167
Square root of dept. ID = 8.94427190999915878563669467492510494176
Square root of dept. ID = 9.48683298050513799599668063329815560116
Square root of dept. ID = 10.48808848170151546991453513679937598475
Square root of dept. ID = 11.40175425099137979136049025566754479076
Time = +000000000 00:00:00.001000000
```

But Efficiency in database queries is often about minimizing the amount of work the database needs to do. The second query is more efficient because it reduces the number of calculations performed by applying the function to a smaller subset of data.

Step 5: Questions and Reflections Concerning this Database Project

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

   ➔ In PL/SQL, EXCEPTION statements are used to deal with mistakes or special circumstances. They can be used in SQL statements to handle SQL exceptions or in exception handlers to capture particular problems (like {WHEN NO_DATA_FOUND}). The `RAISE` statement can also be used to re-raise exceptions or construct custom exceptions. You can handle errors at higher levels of your code thanks to exception propagation. Furthermore, EXCEPTION statements in database triggers manage failures that arise during trigger execution. Robust PL/SQL applications must have proper exception management, which may also provide error handling and response that is gracious.

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

   ➔ Functions, procedures, and packages are separate database objects in PL/SQL, each with a particular function:
   Function:
   A PL/SQL software unit that returns a single value is called a function.

Usually, a computation is carried out and the caller receives the result.
Functions are frequently used in SQL expressions to obtain and compute data.
Examples of these computations include finding the square root of an integer and converting units.

Procedure:

A procedure is a unit of PL/SQL code that executes one or more operations.
Unlike functions, it doesn't return a result; instead, it's utilized for its side effects, which include managing errors, creating reports, and altering database information.
Procedures are frequently used to generate reusable code, encapsulate business logic, and manipulate data.

Package:

A package serves as a container to hold variables, processes, and associated functions together into a single entity.
It facilitates improved code management and offers modular organization.
Code reusability and maintainability are improved through the usage of packages, which are used to distribute and maintain code among other applications or modules.

3) Distinguish between Oracle date types RRRR and YYYY.

➔ When displaying the year in Oracle date formats, {YYYY} presents it as a four-digit number without any modifications. It indicates the date's actual year. Nevertheless, `RRRR} accounts for two-digit year ambiguity while formatting the year as four digits as well. Based on the current date and system settings, Oracle reads two-digit years (e.g., '00' to '49') as 2000-2049 and (e.g., '50' to '99') as 1950-1999 when using `RRRR}. When working with dates where the century is ambiguous, {RRRR~ is frequently recommended to avoid misinterpreting dates close to the turn of the century.

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

➔ No, you cannot utilize substitution variables directly in a PL/SQL function declaration. Substitution variables are not a component of PL/SQL syntax; instead, they are a feature of SQL*Plus or SQL Developer tools, used for scripting or interactive input.
In PL/SQL, function definitions need a well-defined structure and certain data types for arguments. A function's arguments can be defined, but they are typed explicitly and cannot be changed using substitution variables.
When calling a function, substitution variables can be used to supply input values; however, these values must be supplied as arguments to the function and cannot be used directly in the function declaration.

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

➔ **For loops**:

When you know ahead of time the precise amount of iterations, you should use for loops. They are perfect when you want to run a piece of code a certain number of times or within a specific range of values. For loops are more suitable, for instance, for iterating through an array's items or carrying out a certain action a predetermined number of times.

Example:

```
FOR i IN 1..10 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

While loops:

Conversely, while loops are employed when the precise number of iterations is unknown beforehand or when you need to iterate depending on a condition that could change while the loop is being executed. While loops keep running as long as the given condition is still true. When specific conditions are met, they can be used for activities like processing items in a list or reading data till the end of a file.

Example:

```
DECLARE
  countdown NUMBER := 10;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Countdown to liftoff:');
  WHILE countdown >= 1 LOOP
    DBMS_OUTPUT.PUT_LINE(countdown);
    countdown := countdown - 1;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Liftoff!');
END;
```