ITMD 523 *Advanced Topics in Data Management* HW 6

Student Name **Sivaranjani Prabasankar** Section

Instructor ***Luke Papademas*** Due Date 04/07/2019

Part	1	2	3	4	TOTAL	Score
Maximum Points	25 points	25 points	25 points	25 points	**100** points	

Textbook Reading Assignment Thoroughly read Week 1 - 10 course lecture notes.

Part 1 Concepts, Topics, Glossary Terms, Topics in Data Management

(1) (Topics in SQL: Subqueries)

Explain the difference between a regular subquery and a correlated subquery. Provide an example of each of a regular subquery and a correlated subquery.

CORRELATED SUBQUERY

- A correlated subquery is a subquery that uses the values of the outer query.
- In other words, it depends on the outer query for its values.
- This dependency, a correlated subquery cannot be executed independently as a simple subquery.
- Moreover, a correlated subquery is executed repeatedly, once for each row evaluated by the outer query. The correlated subquery is also known as a repeating subquery.

Example

```
SELECT   companyname, city FROM   customers
  WHERE  100000 <
     (SELECT SUM (unitprice * quantity) FROM orders
       INNER JOIN orderdetails ON orderdetails.orderid = orders.orderid
        WHERE orders.customerid = customers.customerid);
```

REGULAR SUBQUERY

- A subquery is a SQL query within a query.
- Subqueries are nested queries that provide data to the enclosing query.
- Subqueries can return individual values or a list of records
- Subqueries must be enclosed with parenthesis. There is no general syntax; subqueries are regular queries placed inside parenthesis.
- Subqueries can be used in different ways and at different locations inside a query.

Example

```
SELECT ProductName FROM Product
  WHERE Id IN
    (SELECT ProductId FROM OrderItem WHERE Quantity > 100);
```

Student Name **Sivaranjani Prabasankar** Section _____

(2) (Topics in SQL: Set Operations)

Suppose that a PRODUCT table contains two attributes, PROD_CODE and VEND_CODE . Those two attributes have values of ABC , 125 , DEF , 124 , GHI , 124 and JKL , 123 , respectively. The VENDOR table contains a single attribute, VEND_CODE , with values 123 , 124 , 125 and 127 , respectively.
(The VEND_CODE attribute in the PRODUCT table is a foreign key to the VEND_CODE in the VENDOR table.)

Given that information, what would be the query output for:

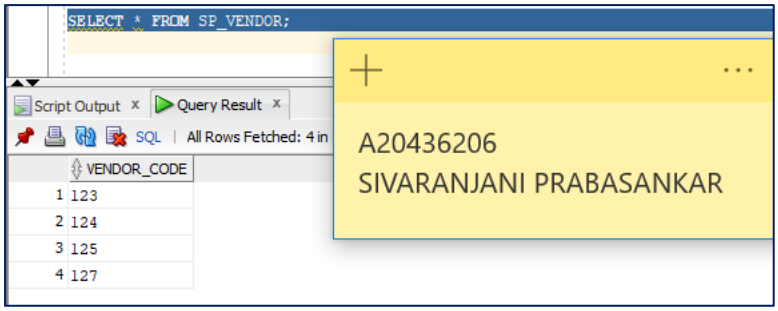
(a) a UNION query based on these two tables

(b) a UNION ALL query based on these two tables

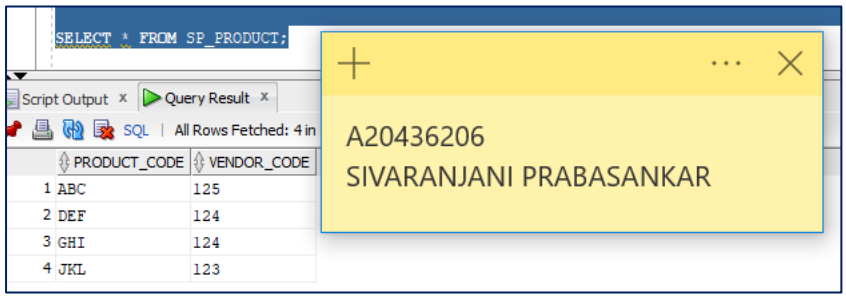
(c) an INTERSECT query based on these two tables

(d) a MINUS query based on these two tables.

```
SELECT * FROM SP_VENDOR;
```

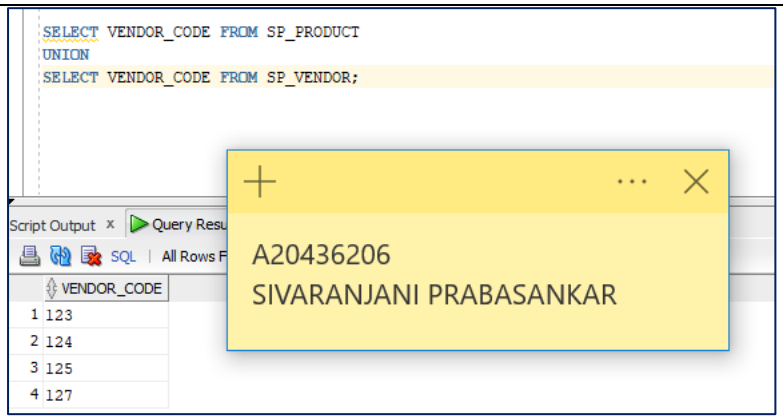



```
SELECT * FROM SP_PRODUCT;
```



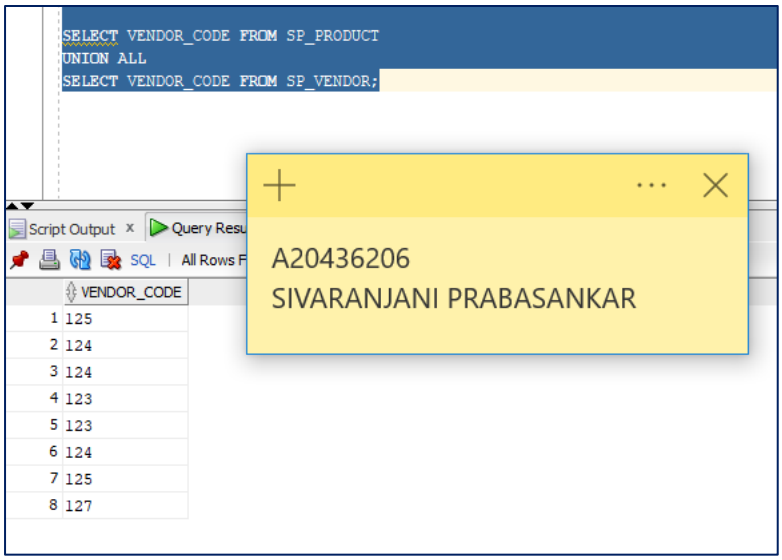
A) UNION

```
SELECT VENDOR_CODE FROM SP_PRODUCT
UNION
SELECT VENDOR_CODE FROM SP_VENDOR;
```



B) UNION ALL

SELECT VENDOR_CODE FROM SP_PRODUCT
UNION ALL
SELECT VENDOR_CODE FROM SP_VENDOR;



C) INTERSECT

SELECT VENDOR_CODE FROM SP_PRODUCT
INTERSECT
SELECT VENDOR_CODE FROM SP_VENDOR;

```
SELECT VENDOR_CODE FROM SP_PRODUCT
INTERSECT
SELECT VENDOR_CODE FROM SP_VENDOR;
```

Script Output | Query Resu | SQL | All Rows F

	VENDOR_CODE
1	123
2	124
3	125

A20436206
SIVARANJANI PRABASANKAR

D) MINUS

SELECT VENDOR_CODE FROM SP_VENDOR
MINUS
SELECT VENDOR_CODE FROM SP_PRODUCT;

```
SELECT VENDOR_CODE FROM  SP_VENDOR
MINUS
SELECT VENDOR_CODE FROM SP_PRODUCT;
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in

	VENDOR_CODE
1	127

A20436206
SIVARANJANI PRABASANKAR

SELECT VENDOR_CODE FROM SP_PRODUCT
MINUS
SELECT VENDOR_CODE FROM SP_VENDOR;

```
SELECT VENDOR_CODE FROM  SP_PRODUCT
MINUS
SELECT VENDOR_CODE FROM SP_VENDOR;
```

Script Output | Query Result | SQL | All Rows Fetched: 0 in

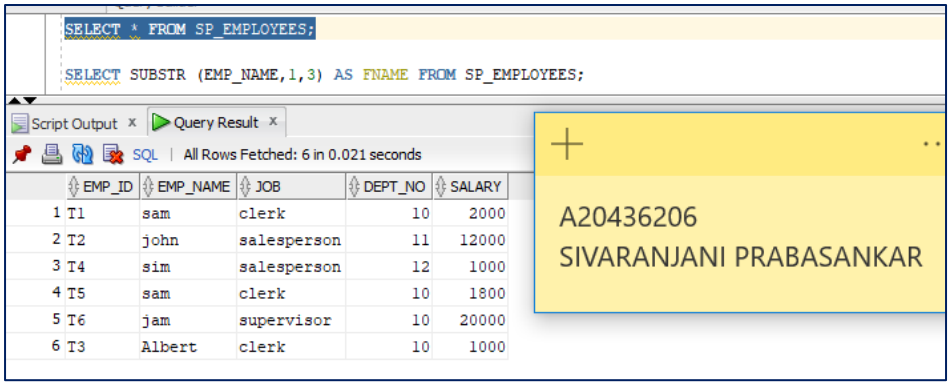
VENDOR_CODE

A20436206
SIVARANJANI PRABASANKAR

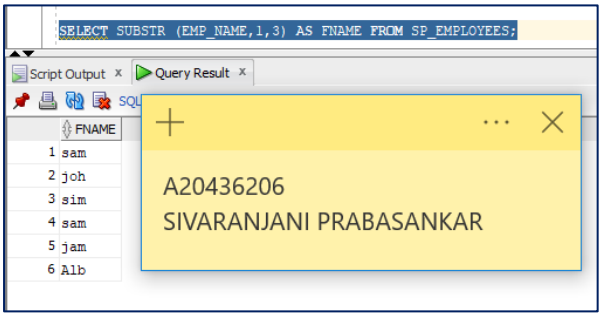
(3) (SQL String Functions)

What string function should you use to list the first three characters of a company's EMP_LNAME values? Give an example, using a table named EMPLOYEE .

SELECT * FROM SP_EMPLOYEES;



SELECT SUBSTR (EMP_NAME,1,3) AS FNAME FROM SP_EMPLOYEES;



Part 2 PL / SQL Concepts - Advanced Topics in Data Management

(1) (Creating Functions in PL /SQL)

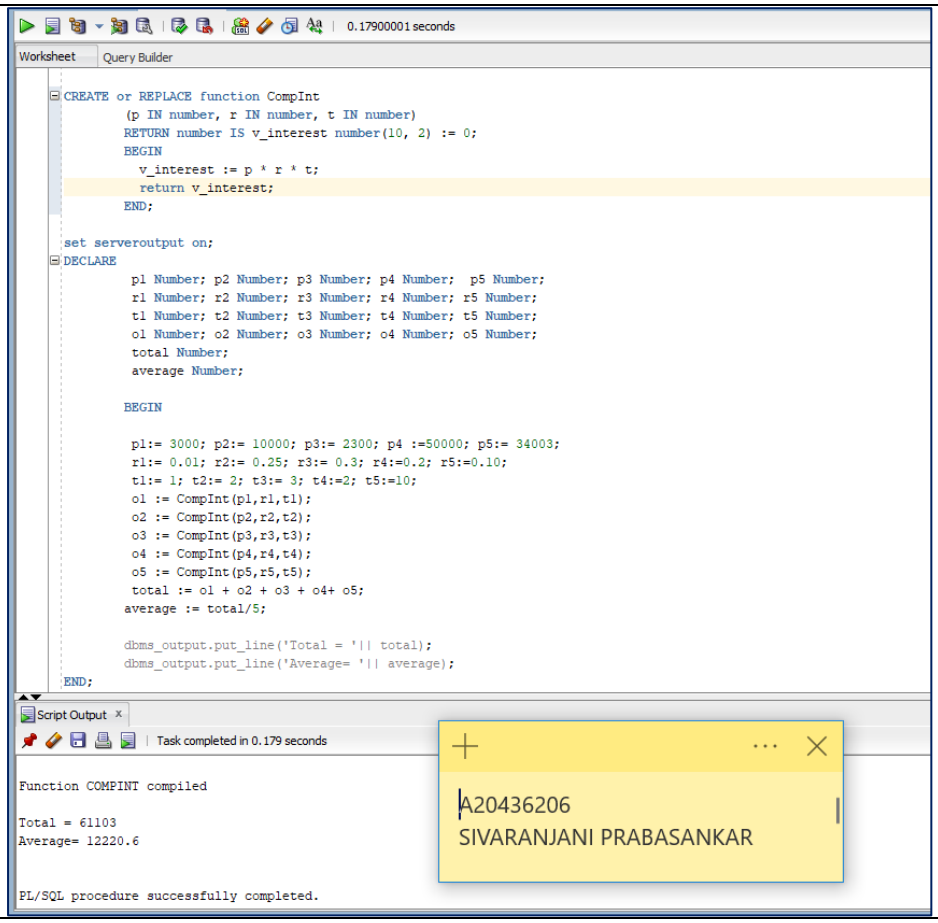
Examine the PL / SQL function given below. This function receives three parameters (principal, rate and time) and returns the simple interest earnings associated with these parameters.

```
--function definition
CREATE or REPLACE function CompInt
(p IN number, r IN number, t IN number)
RETURN number IS v_interest number(10, 2) := 0;

BEGIN
  v_interest := p * r * t;
  return v_interest;
END;
```

Create a script that will test your function with five different and valid values of the principal, rate and time. That is, incorporate 5 separate function calls in your

script for sample principal, rate and time amounts. Your script is also to display the average interest earnings of your five separate function calls.
Assume that the rate is an annual percentage and that the time is expressed in years.



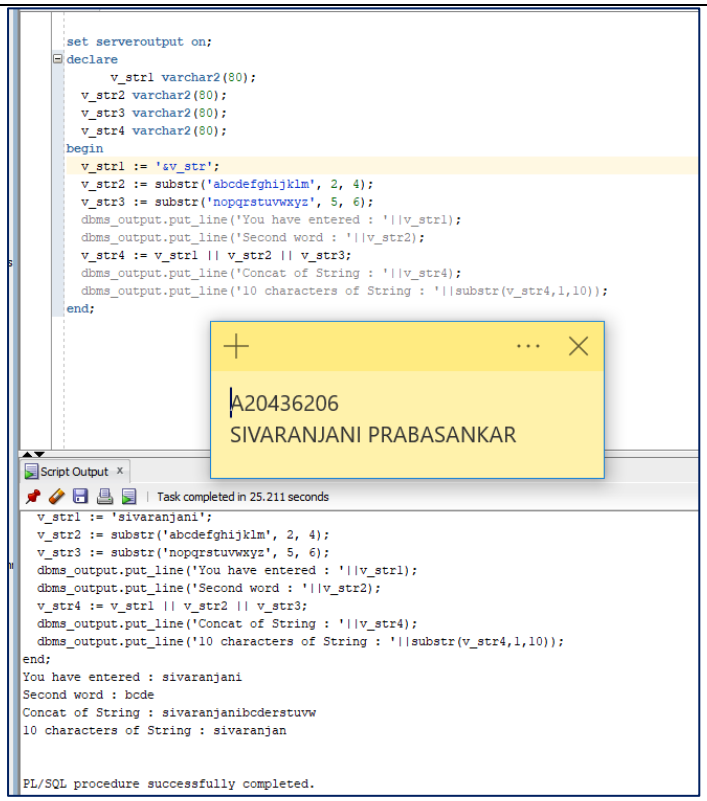
(2) (Simple Encryption - Secret Code Words)

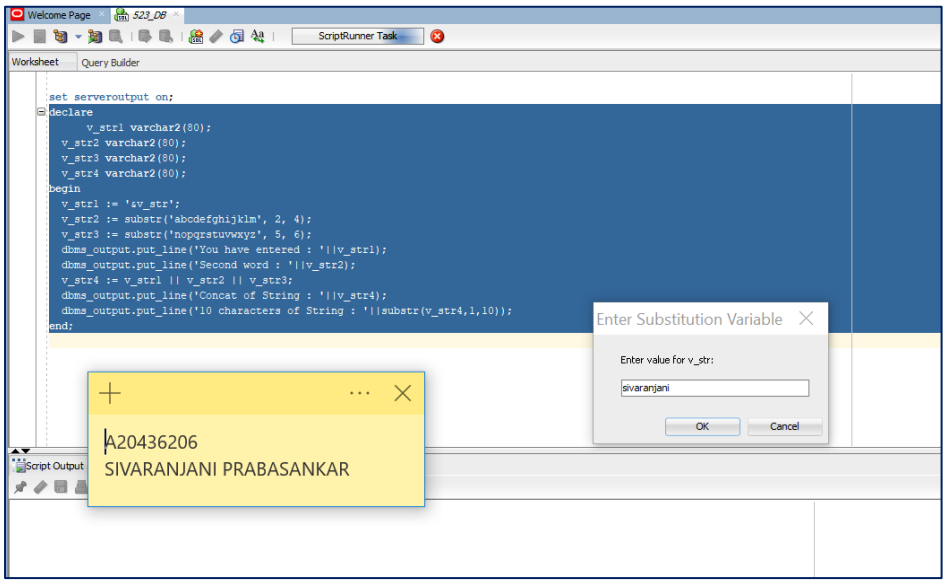
A database specialist wishes to create some secret code words using a simple PL / SQL script given below.

```
set serveroutput on;
declare
  v_str1 varchar2(80);
  v_str2 varchar2(80);
  v_str3 varchar2(80);
  v_str4 varchar2(80);
begin
  v_str1 := '&v_str';
  v_str2 := substr('abcdefghijklm', 2, 4);
  v_str3 := substr('nopqrstuvwxyz', 5, 6);
  dbms_output.put_line(v_str1);
  dbms_output.put_line(v_str2);
```

```
  v_str4 := v_str1 || v_str2 || v_str3;
  dbms_output.put_line(v_str4);
end;
```

Keeping the same script statements above, alter the above code such that a single ten - character secret code word is generated.





Part 3 Data Analytics - Advanced Topics in Data Management

(1) (Exponential Moving Average)

Often used in the realm of stock price analysis, the exponential moving average (EMA) is a weighted average of the last *n* prices. Here the weighting decreases exponentially with each previous price.

Calculation Formula:

$$EMA(n) = EMA(n-1) + ((2 / (n + 1)) \times (P(n) - EMA(n-1)))$$

where EMA(n) is the current EMA value
EMA(n−1) is the previous EMA value
n is the number of values to average
2 / (n + 1) is the exponential smoothing multiplier
P(n) is the current price

http://tradingsim.com/blog/exponential-moving-average-ema-technical-indicator/

For this *n* = 6 bar exponential moving average data with prices of 25.35 , 41.58 , 72.49 , etc. compute EMA(n) using the traditional average of the prices below as EMA(n−1) .

Month	prices
10	$25.35
11	$41.58
12	$72.49
13	$33.16
14	$33.95
15	$43.97

Formula used : EMA(n−1) + ((2 / (n + 1)) × (P(n) − EMA(n−1))

Month	Prices	Formula & Calculation	EMA
10	$25.35	$25.35	$25.35
11	$41.58	EMA (6−1) + ((2 / (6 + 1)) × (P(6) − EMA(6−1)) 25.35+ [0.286 * (41.58 – 25.35)] = 25.35+4.64178	$29.99
12	$72.49	EMA (6−1) + ((2 / (6 + 1)) × (P(6) − EMA(6−1)) 29.99+ [0.286 * (72.49 – 29.99)] = 29.99+12.155	$42.15
13	$33.16	EMA (6−1) + ((2 / (6 + 1)) × (P(6) − EMA(6−1)) 42.15+ [0.286 * (33.16 – 42.15)] = 42.15-2.57114	$39.58
14	$33.95	EMA (6−1) + ((2 / (6 + 1)) × (P(6) − EMA(6−1)) 39.58+ [0.286 * (33.95 – 39.58)] = 39.58-1.61018	$37.97
15	$43.97	EMA (6−1) + ((2 / (6 + 1)) × (P(6) − EMA(6−1)) 37.97+ [0.286 * (43.97 – 37.97)] = 37.97+1.716	$39.69

(2) (Forecasting with SQL - Linear Regression)

Assume that the following table has been created.

```
--DROP TABLE my_data;
CREATE TABLE my_data AS
SELECT LEVEL id,
  floor(dbms_random.value(0, 100)) field_x,
floor(dbms_random.value(0, 1000)) field_y
FROM dual
CONNECT BY LEVEL <= 100;

SELECT * FROM my_data;
```

Execute the above code and observe the table of data that has been created.

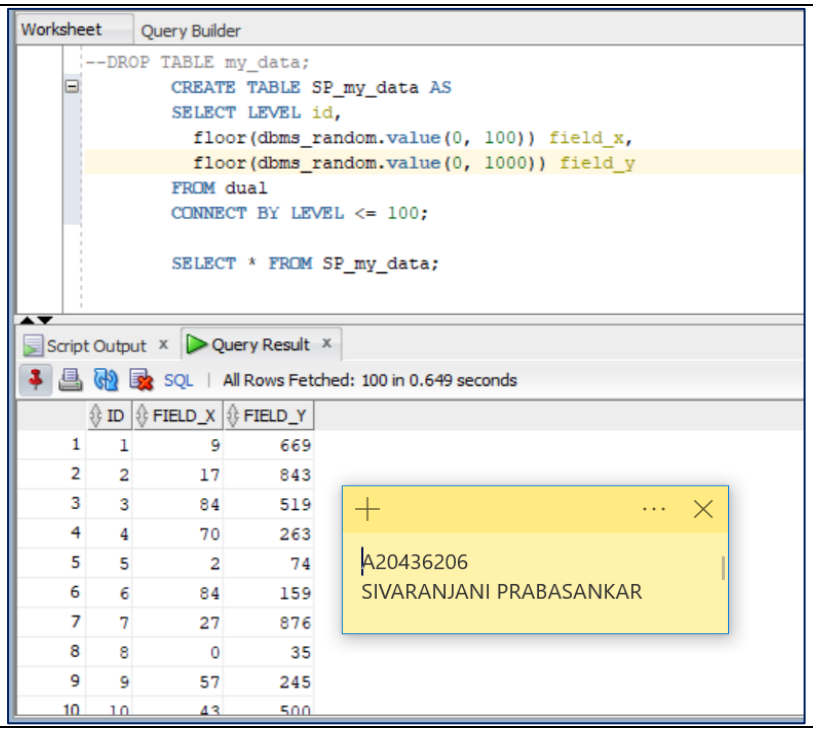
Review the article that is listed at the provided Web site:

Exporting Oracle Data to a Microsoft Excel File

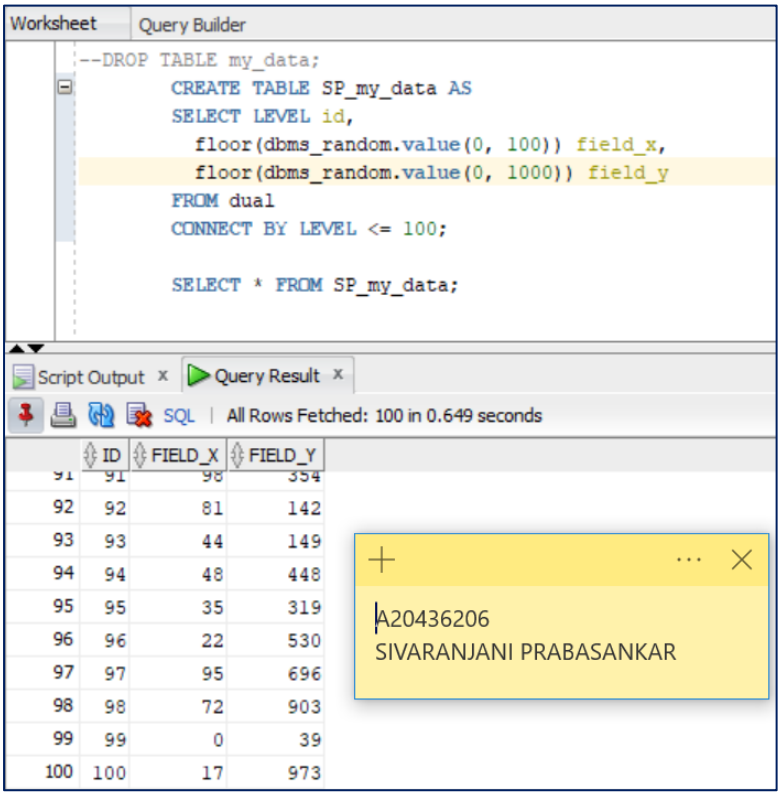
http://docs.oracle.com/cd/E17781_01/server.112/e18804/impexp.htm#BABHFHGH

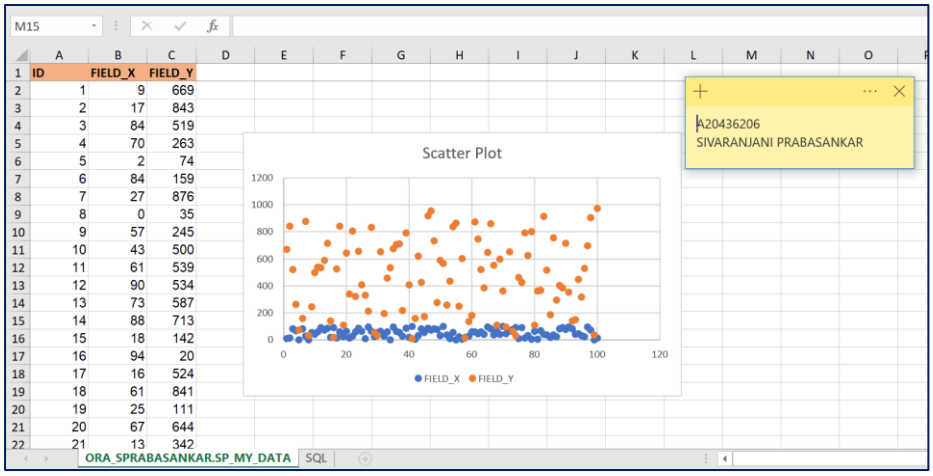
Export your table data to MS Excel and then construct a Scatter Plot based on your data, using the Excel path on the Ribbon: [Insert] -> [Charts] group -> [Insert Scatter Plot] and select a type of plot from the gallery

Then write five separate facts that describe your plot. The facts could involve properties of a data graph such as clustering, curving, shaping, etc.

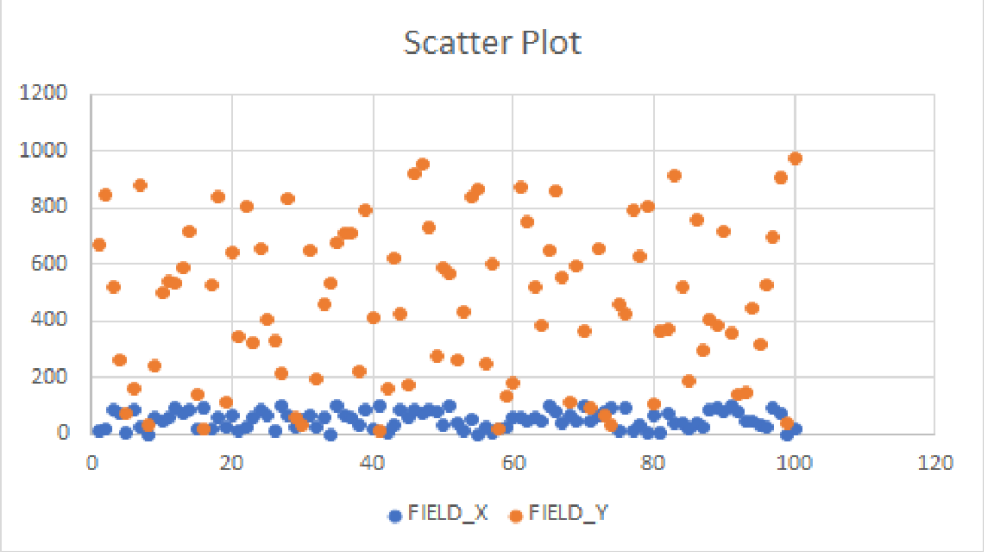


Student Name **Sivaranjani Prabasankar** Section





Export_Mydata_SP.xlsx



Interpreting Scatter Plot

- In the above graph we see that the data points for field x are less scattered than field y
- Linearity is observed from the scatter plot for field x as it is less scattered
- Whereas field Y are lightly correlated and bonding is too less.
- As field Y are scattered more there could be more possibilities for outliers for Y
- The variance seems constant for field x and less constant for field y
- If we need to interpret we may require transformations like LogY, SQRT Y or inverse of Y for better analysis.

Part 4 Data Design Concepts - Advanced Topics in Data Management

(1) (Oracle Blobs and Clobs)

Research Oracle Blob and Clob data types. Expound on five factual items that relate to each of Blob and Clob. Also examine if it is possible to convert a Blob to a Clob.

BLOB

- BLOB stores values as LOB (Large OBject) in bitstreams.
- It is a binary string with 2,147,483,647 characters long.
- BLOB datatype stores unstructured binary large objects.
- BLOB objects can be thought of as bitstreams with no character set semantics.
- BLOB length is generally defined in K, M and G
- BLOB is primarily intended to hold non-traditional data, such as voice or mixed media.

CLOB

- CLOB stores values as LOB (Large OBject) in character steams.
- It can also have 2,147,483,64 characters in length.
- The CLOB datatype stores single-byte and multibyte character data.
- Both fixed-width and variable-width character sets are supported, and both use the database character set.
- Generally, a Clob is used to store a unicode or character-based data. Such data can be large documents.
- Length is specified in Characters.

Similarities between BLOB & CLOB

⇒ Multiple LOB columns allowed in a single table
⇒ Random data access supported
⇒ Better table space management
⇒ Allowed in user-defined object types
⇒ Easily passed to procedures and external calls

Below PL/SQL function script will convert a BLOB into a CLOB. The convert function accepts a BLOB as input and returns a CLOB datatype.

dbms_lob.converttoclob
utl_raw.cast_to_varchar2

(2) (SQL Syntax - Difference Between HAVING and WHERE Clauses)

Using the keywords HAVING, WHERE and GROUP BY, fill the blanks to complete each sentence or phrase that distinguish the differences between the HAVING and WHERE clauses.

- The **WHERE** clause selects rows before grouping whereas the **HAVING** clause selects rows after grouping.

- The **WHERE** clause cannot contain aggregate functions whereas the **HAVING** clause can contain aggregate functions.

- The **WHERE** clause specifies the criteria that individual records must meet to be selected by a query.

- The HAVING clause cannot be used without the **GROUPBY** clause.

Next, describe the columns and records that would be returned by the following SQL statement.

```
SELECT SalesOrderID,
SUM(UnitPrice * OrderQty) AS TotalPrice
FROM Sales.SalesOrderDetail
WHERE LineTotal > 200
GROUP BY SalesOrderID
HAVING SUM(UnitPrice * OrderQty) > 5000
```

⇒ Above query will return SalesOrderID, Total Price column.
⇒ SalesOrderID will be returned directly from table whereas Total Price will be calculated using by multiplying each value in two columns like UnitPrice and OrderQty from table and summing up the multiplied values.

It will return records which meets below criteria

1) Line total should be more than 200
2) Total Price more than 5000
3) It will group the records by using SalesOrderID.