

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 \_\_\_\_\_

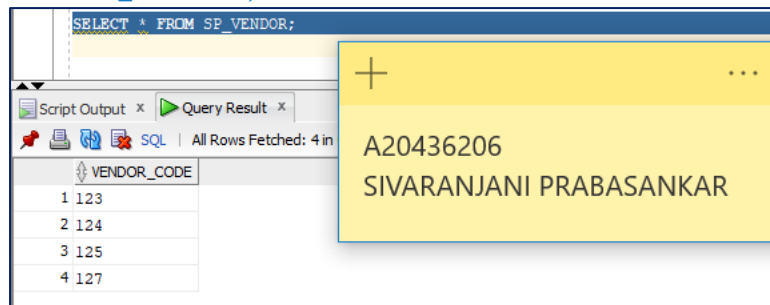
**(2) ( Topics in SQL: Set Operations )**

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:

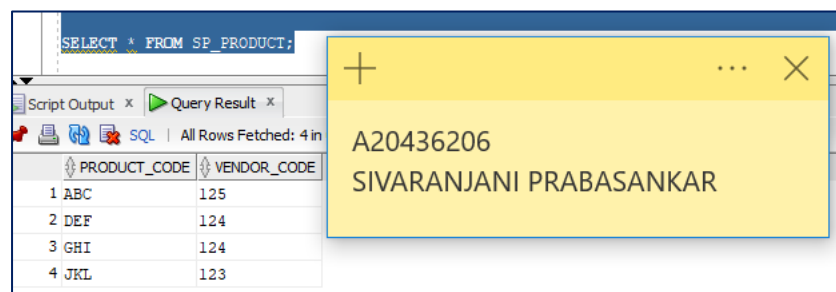
- (a) a UNION query based on these two tables
- (b) a UNION ALL query based on these two tables
- (c) an INTERSECT query based on these two tables
- (d) a MINUS query based on these two tables.

SELECT \* FROM SP\_VENDOR;



	VENDOR_CODE
1	123
2	124
3	125
4	127

SELECT \* FROM SP\_PRODUCT;



	PRODUCT_CODE	VENDOR_CODE
1	ABC	125
2	DEF	124
3	GHI	124
4	JKL	123

**A) UNION**

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

Student Name **Sivaranjani Prabasankar**

Section \_\_\_\_\_

The screenshot shows a SQL query window with the following text:

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

Below the query window, a yellow tooltip displays the results:

VENDOR_CODE
1 123
2 124
3 125
4 127

## B) UNION ALL

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

The screenshot shows a SQL query window with the following text:

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

Below the query window, a yellow tooltip displays the results:

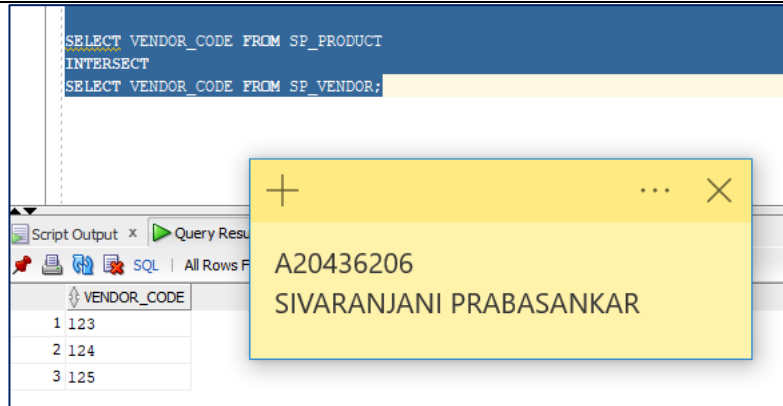
VENDOR_CODE
1 125
2 124
3 124
4 123
5 123
6 124
7 125
8 127

## C) INTERSECT

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

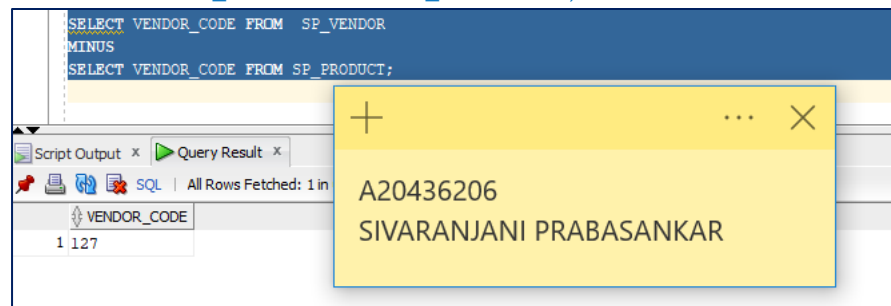
Student Name **Sivaranjani Prabasankar**

Section \_\_\_\_\_

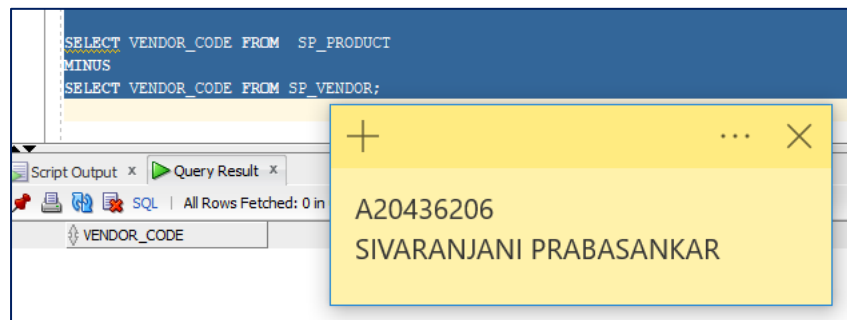


## D) MINUS

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

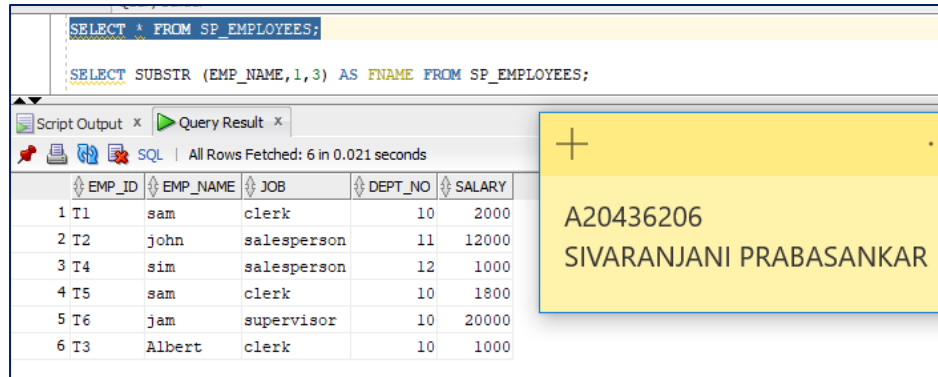


```
SELECT VENDOR_CODE FROM SP_PRODUCT  
MINUS  
SELECT VENDOR_CODE FROM SP_VENDOR;
```

**(3) ( SQL String Functions )**

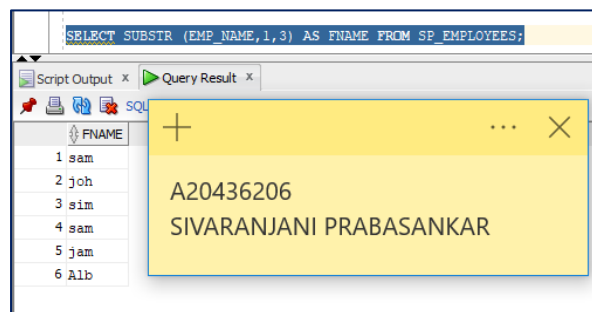
What string function should you use to list the first three characters of a company's EMP\_LNAME values? Give an example, using a table named EMPLOYEE .

SELECT \* FROM SP\_EMPLOYEES;



EMP_ID	EMP_NAME	JOB	DEPT_NO	SALARY
1 T1	sam	clerk	10	2000
2 T2	john	salesperson	11	12000
3 T4	sim	salesperson	12	1000
4 T5	sam	clerk	10	1800
5 T6	jam	supervisor	10	20000
6 T3	Albert	clerk	10	1000

SELECT SUBSTR (EMP\_NAME,1,3) AS FNAME FROM SP\_EMPLOYEES;



FNAME
1 sam
2 joh
3 sim
4 sam
5 jam
6 Alb

## 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

Student Name **Sivaranjani Prabasankar**

Section \_\_\_\_\_

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.

The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL script in the 'Query Builder' tab. The script defines a function 'CompInt' and a procedure that calls it five times with different parameters, calculates a total, and then calculates an average. The 'Script Output' window at the bottom shows the results of the execution, including the total and average values. A yellow sticky note is placed over the output window, displaying the student's ID and name.

```

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;

set serveroutput on;
DECLARE
    p1 Number; p2 Number; p3 Number; p4 Number; p5 Number;
    r1 Number; r2 Number; r3 Number; r4 Number; r5 Number;
    t1 Number; t2 Number; t3 Number; t4 Number; t5 Number;
    o1 Number; o2 Number; o3 Number; o4 Number; o5 Number;
    total Number;
    average Number;

BEGIN

    p1:= 3000; p2:= 10000; p3:= 2300; p4 :=50000; p5:= 34003;
    r1:= 0.01; r2:= 0.25; r3:= 0.3; r4:=0.2; r5:=0.10;
    t1:= 1; t2:= 2; t3:= 3; t4:=2; t5:=10;
    o1 := CompInt(p1,t1,t1);
    o2 := CompInt(p2,t2,t2);
    o3 := CompInt(p3,t3,t3);
    o4 := CompInt(p4,t4,t4);
    o5 := CompInt(p5,t5,t5);
    total := o1 + o2 + o3 + o4+ o5;
    average := total/5;

    dbms_output.put_line('Total = '|| total);
    dbms_output.put_line('Average= '|| average);

END;
  
```

Script Output

Task completed in 0.179 seconds

Function COMPINT compiled

Total = 61103

Average= 12220.6

PL/SQL procedure successfully completed.

A20436206  
SIVARANJANI PRABASANKAR

## (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);
  
```

Student Name Sivaranjani Prabasankar

Section \_\_\_\_\_

```
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.

```
set serveroutput on;  
declare  
  v_str1 varchar2(80);  
  v_str2 varchar2(80);  
  v_str3 varchar2(80);  
  v_str4 varchar2(80);  
begin  
  v_str1 := 'sv_str';  
  v_str2 := substr('abdefghijklm', 2, 4);  
  v_str3 := substr('nopqrstuvwxyz', 5, 6);  
  dbms_output.put_line('You have entered : '||v_str1);  
  dbms_output.put_line('Second word : '||v_str2);  
  v_str4 := v_str1 || v_str2 || v_str3;  
  dbms_output.put_line('Concat of String : '||v_str4);  
  dbms_output.put_line('10 characters of String : '||substr(v_str4,1,10));  
end;
```

Script Output x

Task completed in 25.211 seconds

```
v_str1 := 'sivaranjani';  
v_str2 := substr('abdefghijklm', 2, 4);  
v_str3 := substr('nopqrstuvwxyz', 5, 6);  
dbms_output.put_line('You have entered : '||v_str1);  
dbms_output.put_line('Second word : '||v_str2);  
v_str4 := v_str1 || v_str2 || v_str3;  
dbms_output.put_line('Concat of String : '||v_str4);  
dbms_output.put_line('10 characters of String : '||substr(v_str4,1,10));  
end;
```

You have entered : sivaranjani  
Second word : bode  
Concat of String : sivaranjaniboderstuvw  
10 characters of String : sivaranjan

PL/SQL procedure successfully completed.

```
set serveroutput on;  
declare  
  v_str1 varchar2(80);  
  v_str2 varchar2(80);  
  v_str3 varchar2(80);  
  v_str4 varchar2(80);  
begin  
  v_str1 := 'sv_str';  
  v_str2 := substr('abdefghijklm', 2, 4);  
  v_str3 := substr('nopqrstuvwxyz', 5, 6);  
  dbms_output.put_line('You have entered : '||v_str1);  
  dbms_output.put_line('Second word : '||v_str2);  
  v_str4 := v_str1 || v_str2 || v_str3;  
  dbms_output.put_line('Concat of String : '||v_str4);  
  dbms_output.put_line('10 characters of String : '||substr(v_str4,1,10));  
end;
```

Script Output x

Task completed in 25.211 seconds

```
v_str1 := 'sivaranjani';  
v_str2 := substr('abdefghijklm', 2, 4);  
v_str3 := substr('nopqrstuvwxyz', 5, 6);  
dbms_output.put_line('You have entered : '||v_str1);  
dbms_output.put_line('Second word : '||v_str2);  
v_str4 := v_str1 || v_str2 || v_str3;  
dbms_output.put_line('Concat of String : '||v_str4);  
dbms_output.put_line('10 characters of String : '||substr(v_str4,1,10));  
end;
```

You have entered : sivaranjani  
Second word : bode  
Concat of String : sivaranjaniboderstuvw  
10 characters of String : sivaranjan

PL/SQL procedure successfully completed.

Student Name Sivaranjani Prabasankar

Section \_\_\_\_\_

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

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



Student Name Sivaranjani Prabasankar

Section \_\_\_\_\_

**(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](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.

The screenshot displays the Oracle SQL Developer interface. The top pane, titled 'Query Builder', contains the following SQL code:

```
--DROP TABLE my_data;  
CREATE TABLE SP_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 SP_my_data;
```

The bottom pane, titled 'Query Result', shows the output of the query. It includes a toolbar with icons for saving, printing, and other actions, along with the text 'All Rows Fetched: 100 in 0.649 seconds'. Below this, a table displays the first 10 rows of data:

	ID	FIELD_X	FIELD_Y
1	1	9	669
2	2	17	843
3	3	84	519
4	4	70	263
5	5	2	74
6	6	84	159
7	7	27	876
8	8	0	35
9	9	57	245
10	10	43	500

A yellow tooltip is visible over the table, displaying the text 'A20436206' and 'SIVARANJANI PRABASANKAR'.

Student Name **Sivaranjani Prabasankar**

Section \_\_\_\_\_

Worksheet Query Builder

```
--DROP TABLE my_data;
CREATE TABLE SP_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;

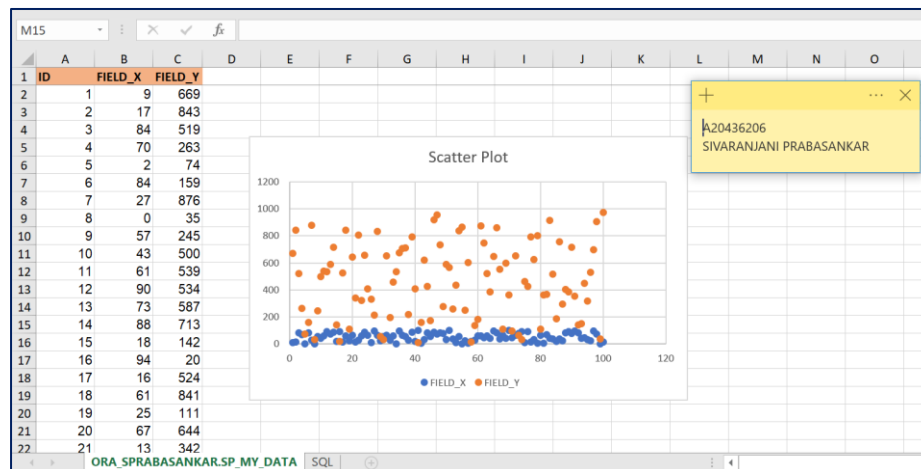
SELECT * FROM SP_my_data;
```

Script Output x Query Result x

SQL | All Rows Fetched: 100 in 0.649 seconds

ID	FIELD_X	FIELD_Y
91	98	354
92	81	142
93	44	149
94	48	448
95	35	319
96	22	530
97	95	696
98	72	903
99	0	39
100	17	973

A20436206  
SIVARANJANI PRABASANKAR

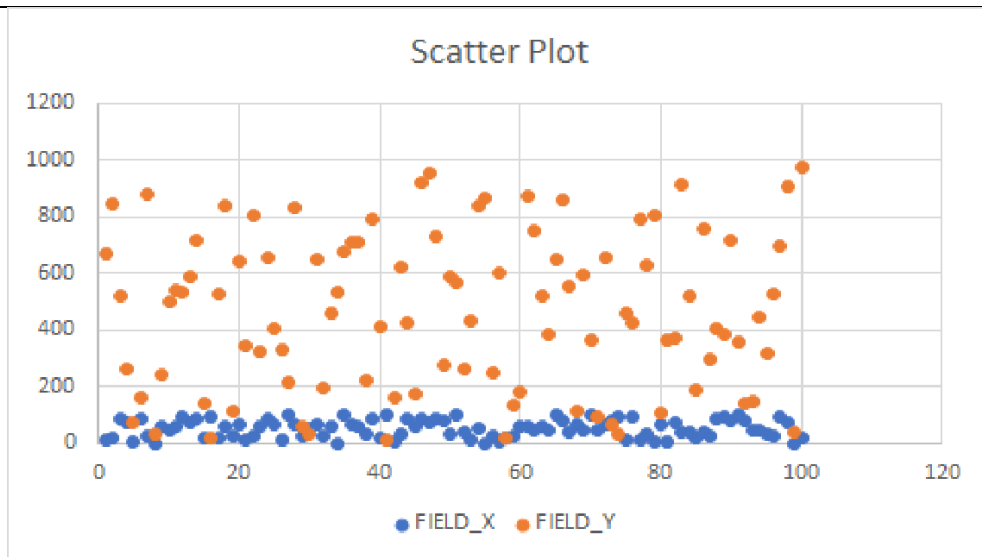


Export\_Mydata\_SP.xls

x

Student Name Sivaranjani Prabasankar

Section \_\_\_\_\_

**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 streams.
- 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.

Student Name **Sivaranjani Prabasankar**

Section \_\_\_\_\_

**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.converttocablob  
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.