

1. Reading done.

2.

Query that lists all pairs (SNO, PNO) in all real world scenario's unlike Dates':

```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris' OR P.CITY is null OR S.CITY is null;
```

Condition #1 – Both S.City not null and P.City are not null Date's query works just fine.  
Both S.city and P.city are known should work. Here S.city is 'Paris' and P.city is 'Paris', which should return nothing as FALSE and FALSE = FALSE.

The screenshot shows a database query tool interface. On the left, a tree view displays the database structure for 'JVALLES', including tables like COURSE, ENROLLED, MEMBEROF, P, S, STUDENT, and STUDENTGROUP. The 'P' table has columns PNO and CITY, and the 'S' table has columns SNO and CITY. The main window shows a 'Query Builder' tab with the following SQL query:

```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris' OR P.CITY is null OR S.CITY is null;
```

Below the query, a 'Script Output' window shows the results of the query. It indicates 'All Rows Fetched: 0 in 0.06 seconds'. The results are displayed in a table with columns SNO and PNO. The table is empty, showing no results.

At the bottom, there are two tabs for the query results. The left tab, titled 'S', shows a single row with SNO 'S1' and CITY 'Paris'. The right tab, titled 'P', shows a single row with PNO 'P1' and CITY 'Paris'.

Condition #1 – Both S.City not null and P.City are not null

Both S.city and P.city are known should work. Here S.city is 'Paris' and P.city is 'London', which should return (S1,P1)

The screenshot shows the SQL Developer interface. On the left, the 'Connections' pane shows the 'JVALLES' database. The 'Tables (Filtered)' section is expanded, showing tables like COURSE, ENROLLED, MEMBEROF, P, S, STUDENT, and STUDENTGROUP. The 'Query Builder' pane on the right shows the following SQL query:

```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris' OR P.CITY is null OR S.CITY is null;
```

The 'Script Output' and 'Query Result' panes show the execution results. The 'Query Result' pane displays a single row of data:

SNO	PNO
1 S1	P1

Condition #2 –Both S.City and P.City are null, Date's query will fail.

For the statement to be true, P.city cannot be null, since one of the expressions has to evaluate to true. In this case, P.city is unknown in the first comparison S.city <> P.city and in the second, P.city <> 'Paris'. When P.city is unknown in both expressions the result is unknown, thus Date's query fails. That is why I added P.City is NULL and S.city is NULL.

The screenshot shows the SQL Developer interface. On the left, the 'Connections' pane shows the 'JVALLES' database. The 'Tables (Filtered)' section is expanded, showing tables like COURSE, ENROLLED, MEMBEROF, P, S, STUDENT, and STUDENTGROUP. The 'Query Builder' pane on the right shows the following SQL query:

```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris' OR P.CITY is null OR S.CITY is null;
```

The 'Script Output' and 'Query Result' panes show the execution results. The 'Query Result' pane displays a single row of data:

SNO	PNO
1 S1	P1

Columns	Data	Model	Constraints	Grants	Statistics	Triggers
SNO	CITY					
1 S1	(null)					

Columns	Data	Model	Constraints	Grants	Statistics	Triggers
P...	CITY					
1 P1	(null)					

Condition #3 - S.city is null, P.city not null, Date's query fails.

Unknown or True = True

Unknown or False = Unknown

If S.city is null but P.city is not , but evaluates to False, the query returns unknown. In my example, P.city = 'Paris' and S.city is NULL in the real world this would return (S1, P1) since S.City is not the same as 'Paris'. Therefore, we must account for that, so I've added S.City is null.

Connections	Start Page	JVALLES	P
Connections	Worksheet	Query Builder	
JVALLES	<pre>SELECT S.SNO, P.PNO FROM S, P WHERE S.CITY &lt;&gt; P.CITY OR P.CITY &lt;&gt; 'Paris' OR P.CITY is null OR S.CITY is null;</pre>		
Tables (Filtered)	Script Output	Query Result	
COURSE	All Rows Fetched: 1 in 0.046 seconds		
ENROLLED			
MEMBEROF			
P			
S			
STUDENT			
STUDENTGROUP			
Views			
Editioning Views			

SNO	PNO
1 S1	P1

Columns	Data	Model	Constraints	Grants	Statistics	Triggers
SNO	CITY					
1 S1	null					

Columns	Data	Model	Constraints	Grants	Statistics	Triggers
P...	CITY					
1 P1	Paris					

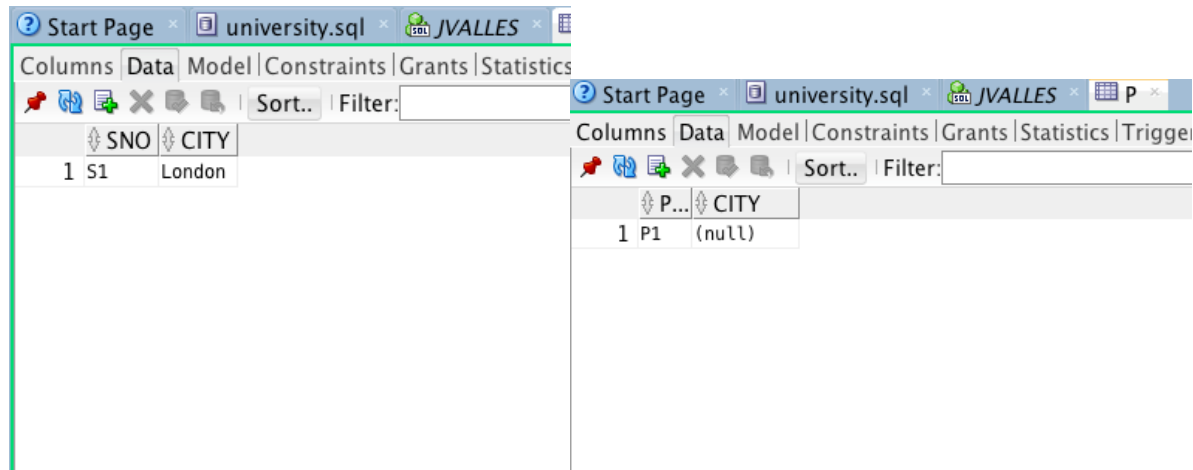
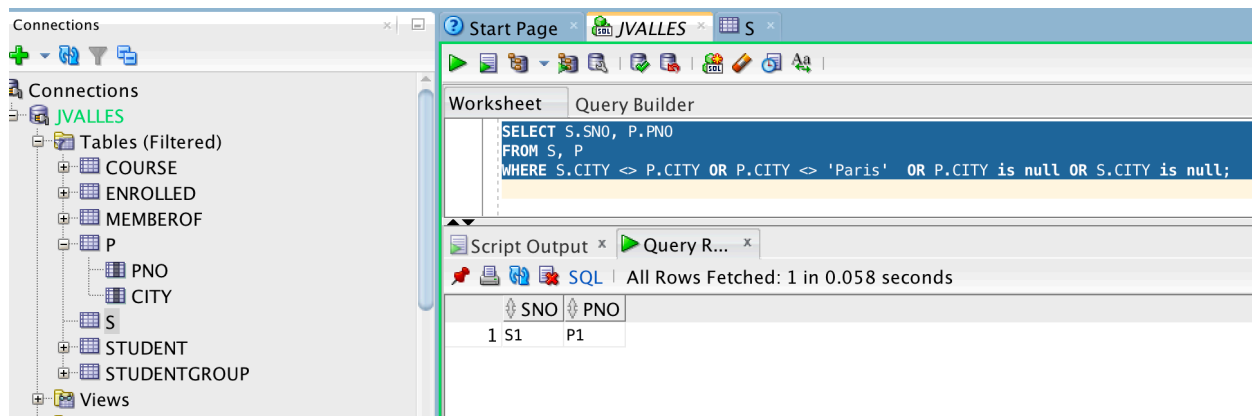
Condition #4 S.city is not null, P.city is null , Date's query fails.

True or Unknown = True

False or Unkown = Unkown

Unkown or Unkown = Unknown

P.city is unknown in the first comparison S.city <> P.city and in the second, P.city <> 'Paris' also evaluates to unknown. When P.city is unknown both expressions result in unknown, thus Date's query fails. That is why P.City is NULL. If P.city and S.city are both NULL, they are not equal. NULL in the "real world" means the value is not 'Paris' and is not equal to S.city. Therefore, by adding P.city is NULL, the WHERE condition evaluates to true.



3.

3-value logic truth table.

	A	B	C	D	E	F	G	H
1	<b>A</b>	<b>B</b>	<b>not(A)</b>	<b>not(B)</b>	<b>A or B</b>	<b>not(A or B)</b>	<b>not(A) and not(B)</b>	
2	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	
3	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	
4	TRUE	UNKNOWN	FALSE	UNKNOWN	TRUE	FALSE	FALSE	
5	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	
6	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	
7	FALSE	UNKNOWN	TRUE	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	
8	UNKNOWN	TRUE	UNKNOWN	FALSE	TRUE	FALSE	FALSE	
9	UNKNOWN	FALSE	UNKNOWN	TRUE	UNKNOWN	UNKNOWN	UNKNOWN	
10	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	
11								

not(A or B) is equivalent to not(A) and not (B) for all truth-values A and B. The law still holds for 3-valued logic.

4.

a.

```
SELECT sid, lastname, firstname, name, founded
from (student left join (memberof inner join studentgroup on groupid = GID)
on sid = studentid);
```

The screenshot shows the SQL Developer interface. On the left, the 'Connections' pane shows a tree view of the database schema, including tables like COURSE, ENROLLED, MEMBEROF, P, S, STUDENT, STUDENTGROUP, and TRUTHS. The main window displays a query in the 'Query Builder' tab. The query is:   
 SELECT sid, lastname, firstname, name, founded  
 from (student left join (memberof inner join studentgroup on groupid = GID)  
 on sid = studentid);  
 The 'Script Output' and 'Query Result' tabs are active. The 'Query Result' tab shows 16 rows of data. The columns are SID, LASTNAME, FIRSTNAME, NAME, and FOUNDED. The data is as follows:

SID	LASTNAME	FIRSTNAME	NAME	FOUNDED
1	8871 Snowdon	Jonathan	(null)	(null)
2	11035 Winter	Abigail	HerCDM	2003
3	14662 Patel	Deepa	(null)	(null)
4	14998 Degroff	Jarvis	(null)	(null)
5	19992 Starck	Jason	(null)	(null)
6	32105 Johnson	Peter	DeFrag	2004
7	32105 Johnson	Peter	Robototics Society	1998
8	32105 Johnson	Peter	HerCDM	2003
9	39077 Pollard	Joya	(null)	(null)
10	57923 Kubik	Dwayne	(null)	(null)
11	58992 Skelly	Trinity	(null)	(null)
12	60973 Krol	Angelo	(null)	(null)
13	75234 Patel	Prakash	DeFrag	2004
14	75234 Patel	Prakash	Robototics Society	1998
15	90421 Brennigan	Marcus	(null)	(null)
16	93321 Snowdon	Jennifer	HerCDM	2003

Groups that have no members is Computer Science Society -2 , I removed from Member of Table for testing purposes.

The screenshot shows the SQL Developer interface. On the left, the 'Connections' pane shows a tree view of the database schema. The main window displays a query in the 'Query Builder' tab. The query is:   
 SELECT sid, lastname, firstname, name, founded  
 from (student left join (memberof inner join studentgroup on groupid = GID)  
 on sid = studentid);  
 The 'Script Output' and 'Query Result' tabs are active. The 'Query Result' tab shows 7 rows of data. The columns are STUDENTID, GROUPID, and JOINED. The data is as follows:

STUDENTID	GROUPID	JOINED
1	75234	42 2015
2	11035	221 2016
3	93321	221 2015
4	75234	101 2015
5	32105	42 2017
6	32105	221 2017
7	32105	101 2017

b.

```
SELECT SID, LastName, FirstName
FROM student
WHERE SID IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND student.SID = enrolled.STUDENTID AND
COURSE.DEPARTMENT IN ('DC'))
AND SID IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND student.SID = enrolled.STUDENTID AND
COURSE.DEPARTMENT IN ('GAM'))
AND SID NOT IN
(SELECT studentid
FROM enrolled , course
WHERE COURSE.CID = ENROLLED.COURSEID AND COURSE.DEPARTMENT IN 'CSC');
```

Worksheet

```

SELECT SID, LastName, FirstName
FROM student
WHERE SID IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND student.SID = enrolled.STUDENTID AND COURSE.DEPARTMENT IN ('DC'))
AND SID NOT IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND student.SID = enrolled.STUDENTID AND COURSE.DEPARTMENT IN ('GAM'))
AND SID NOT IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND COURSE.DEPARTMENT IN ('CSC'));

```

Script Output x Query Result x

SQL | All Rows Fetched: 2 in 0.019 seconds

SID	LASTNAME	FIRSTNAME
1 90421	Brennigan	Marcus
2 60973	Krol	Angelo

90421 is enrolled in 8772 (GAM) and 2987 (DC) and 60973 is enrolled in 8772 (GAM) and 2987 (DC).

Columns Data Model Constraints Grants Statistics Triggers

STUDENTID	COURSEID	QUARTER	YEAR
1	11035	1020 Fall	2017
2	11035	1092 Fall	2016
3	75234	3201 Winter	2017
4	8871	1092 Fall	2018
5	39077	1092 Fall	2018
6	14998	9219 Winter	2018
7	32105	1020 Winter	2018
8	19992	3201 Winter	2018
9	60973	8772 Spring	2018
10	90421	8772 Spring	2018
11	90421	2987 Spring	2018
12	60973	2987 Spring	2018

Columns Data Model Constraints Grants Statistics Triggers Flashback D

CID	COURSENAME	DEPARTMENT	COURSENR
1	1020 Theory of Computation	CSC	489
2	1092 Cryptology	CSC	440
3	3201 Data Analysis	IT	223
4	9219 Databases Technologies	CSC	453
5	3111 Theory of Computation	CSC	389
6	8772 History of Games	GAM	206
7	2987 Topics in Digital Cinema	DC	270

c.

List all students that:

Are a member of Computer Science Society and have not taken at least one CSC class

look for null in year

Or List all members of Computer Science Society that have taken a class after their group join date.

Worksheet

```

SELECT DISTINCT s.sid, s.LastName, s.FirstName
FROM student s INNER JOIN memberof m on s.sid = m.StudentID
INNER JOIN studentgroup sg on sg.gid = m.GroupID
LEFT JOIN (
SELECT * FROM enrolled e inner join course c on e.CourseID = c.cid AND c.DEPARTMENT = 'CSC'
) ec on ec.StudentID = s.SID
WHERE (m.joined < ec.year OR ec.year is null) AND sg.NAME = 'Computer Science Society';

```

Script Output x Query Result x

SQL | All Rows Fetched: 3 in 0.111 seconds

SID	LASTNAME	FIRSTNAME
1 11035	Winter	Abigail
2 32105	Johnson	Peter
3 75234	Patel	Prakash

Abigal took 1020 (CSC) in 2017 and 1092 (CSC) in 2019. She joined the CSS in 2016.

Johnson took 1020 (CSC) in 2018, but join CSS in 2017.

Patel took 3201 (IT) 2017.

The screenshot shows two tables in a database. The left table, MEMBEROF, has columns STUDENTID, GROUPID, and JOINED. The right table, ENROLLED, has columns STUDENTID, COURSEID, QUARTER, and YEAR.

STUDENTID	GROUPID	JOINED
1	75234	42
2	11035	2
3	93321	221
4	75234	2
5	32105	2
6	32105	221
7	32105	101

STUDENTID	COURSEID	QUARTER	YEAR
1	11035	1020 Fall	2017
2	11035	1092 Fall	2019
3	75234	3201 Winter	2017
4	8871	1092 Fall	2018
5	39077	1092 Fall	2018
6	14998	9219 Winter	2018
7	32105	1020 Winter	2018
8	19992	3201 Winter	2018
9	60973	8772 Spring	2018
10	90421	8772 Spring	2018
11	90421	2987 Spring	2018
12	60973	2987 Spring	2018

d.

List groups that only have undergraduate members

```
SELECT GID, Name, career, FirstName
FROM (student JOIN (memberof JOIN studentgroup on groupId = gid) on sid = studentid
)
WHERE career = 'UGRD';
```

The screenshot shows a SQL query in the Query Builder window. The query is: `SELECT GID, Name FROM (student JOIN (memberof join studentgroup on groupId = gid) on sid = studentid ) WHERE career = 'UGRD';` The Script Output window shows the results of the query.

GID	NAME
1	221 HerCDM
2	101 Robototics Society
3	2 Computer Science Society
4	42 DeFrag
5	2 Computer Science Society

The screenshot shows two tables in a database. The left table, MEMBEROF, has columns STUDENTID, GROUPID, and JOINED. The right table, STUDENT, has columns LASTNAME, FIRSTNAME, SID, SSN, CAREER, PROGRAM, CITY, and STARTED.

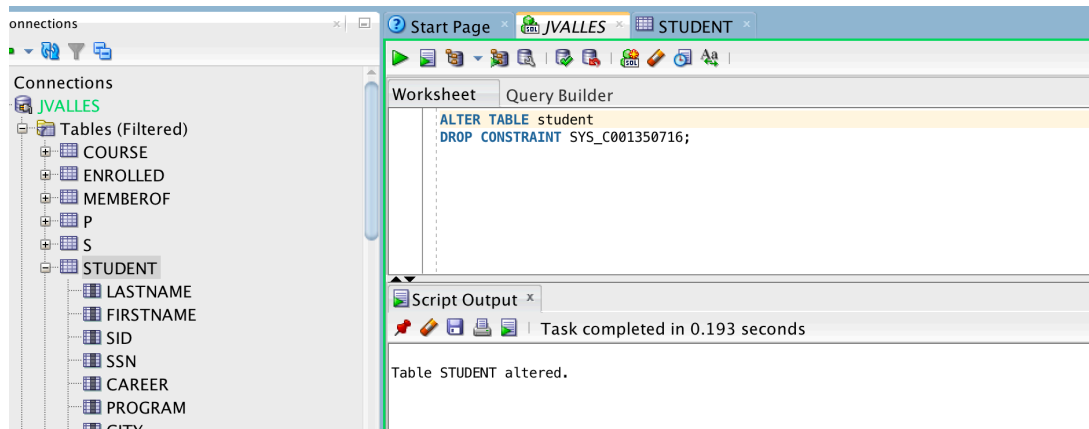
STUDENTID	GROUPID	JOINED
1	75234	42
2	11035	221
3	93321	221
4	75234	2
5	32105	2
6	32105	221
7	32105	101

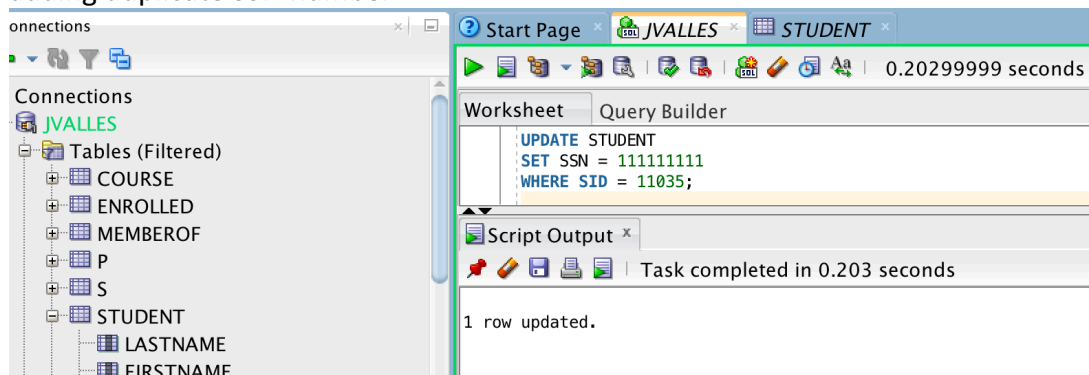
LASTNAME	FIRSTNAME	SID	SSN	CAREER	PROGRAM	CITY	STARTED
1	Brennigan	Marcus	90421 987654321	UGRD	INFO-TECH	Evanston	2015
2	Patel	Deepa	14662	(null) GRD	COMP-SCI	Evanston	2016
3	Snowdon	Jonathan	8871 123123123	GRD	INFO-SYS	Springfield	2017
4	Starck	Jason	19992 789789789	UGRD	INFO-SYS	Springfield	2014
5	Johnson	Peter	32105 123456789	UGRD	COMP-SCI	Chicago	2017
6	Winter	Abigail	11035 111111111	GRD	PHD	Chicago	2016
7	Patel	Prakash	75234	(null) UGRD	COMP-SCI	Chicago	2014
8	Snowdon	Jennifer	93321 321321321	GRD	COMP-SCI	Springfield	2015
9	Degroff	Jarvis	14998 113311331	GRD	INFO-TECH	Evanston	2016
10	Kubik	Dwayne	57923 979797979	UGRD	COMP-SCI	Springfield	2018
11	Skelly	Trinity	58992 55522555	GRD	PHD	Springfield	2018
12	Krol	Angelo	60973	(null) UGRD	COMP-SCI	Springfield	2013
13	Pollard	Joya	39077	(null) GRD	COMP-SCI	Springfield	2017

e.

## Removing table constraint (unique)



## adding duplicate SSN number



The screenshot shows the 'Data' tab of the SQL Developer interface, displaying the contents of the 'STUDENT' table. The table has 13 rows and 8 columns: LASTNAME, FIRSTNAME, SID, SSN, CAREER, PROGRAM, CITY, and STARTED.

	LASTNAME	FIRSTNAME	SID	SSN	CAREER	PROGRAM	CITY	STARTED
1	Brennigan	Marcus	98421	987654321	UGRD	INFO-TECH	Evanston	2015
2	Patel	Deepa	14662	(null)	GRD	COMP-SCI	Evanston	2016
3	Snowdon	Jonathan	8871	123123123	GRD	INFO-SYS	Springfield	2017
4	Starck	Jason	19992	111111111	UGRD	INFO-SYS	Springfield	2014
5	Johnson	Peter	32105	123456789	UGRD	COMP-SCI	Chicago	2017
6	Winter	Abigail	11035	111111111	GRD	PHD	Chicago	2016
7	Patel	Prakash	75234	(null)	UGRD	COMP-SCI	Chicago	2014
8	Snowdon	Jennifer	93321	321321321	GRD	COMP-SCI	Springfield	2015
9	Degroff	Jarvis	14998	113311331	GRD	INFO-TECH	Evanston	2016
10	Kubik	Dwayne	57923	979797979	UGRD	COMP-SCI	Springfield	2018
11	Skelly	Trinity	58992	555222555	GRD	PHD	Springfield	2018
12	Krol	Angelo	60973	(null)	UGRD	COMP-SCI	Springfield	2013
13	Pollard	Joya	39077	(null)	GRD	COMP-SCI	Springfield	2017

```
SELECT SID, LastName, FirstName
FROM student S2
WHERE 1 =
(SELECT COUNT(*) FROM STUDENT S1 WHERE S2.SSN = S1.SSN
)
OR S2.SSN IS NULL;
```



The screenshot shows a database management interface with a 'Connections' pane on the left, a 'Query Builder' pane in the center, and a 'Script Output' / 'Query Result' pane on the right. The 'Connections' pane shows a tree view of tables including COURSE, ENROLLED, MEMBEROF, P, S, STUDENT, and STUDENTGROUP. The 'Query Builder' pane contains the following SQL query:

```
SELECT SID, LastName, FirstName
FROM student S2
WHERE 1 =
(SELECT COUNT(*) FROM STUDENT S1 WHERE S2.SSN = S1.SSN
)
OR S2.SSN IS NULL;
```

The 'Script Output' pane shows the query results in a table with 11 rows. The columns are SID, LASTNAME, and FIRSTNAME. The results are as follows:

	SID	LASTNAME	FIRSTNAME
1	90421	Brennigan	Marcus
2	14662	Patel	Deepa
3	8871	Snowdon	Jonathan
4	32105	Johnson	Peter
5	75234	Patel	Prakash
6	93321	Snowdon	Jennifer
7	14998	Degroff	Jarvis
8	57923	Kubik	Dwayne
9	58992	Skelly	Trinity
10	60973	Krol	Angelo
11	39077	Pollard	Joya

Count the number of instances where  $s2.ssn = s1.ssn$  or where  $s2.ssn$  is null and return those records.

Abigail and Jason are the only ones with the same SSN 11111111, the rest do not. So list includes those with null SSN (unknown) and unique.