James Valles
CSC 453
Assignment – 3

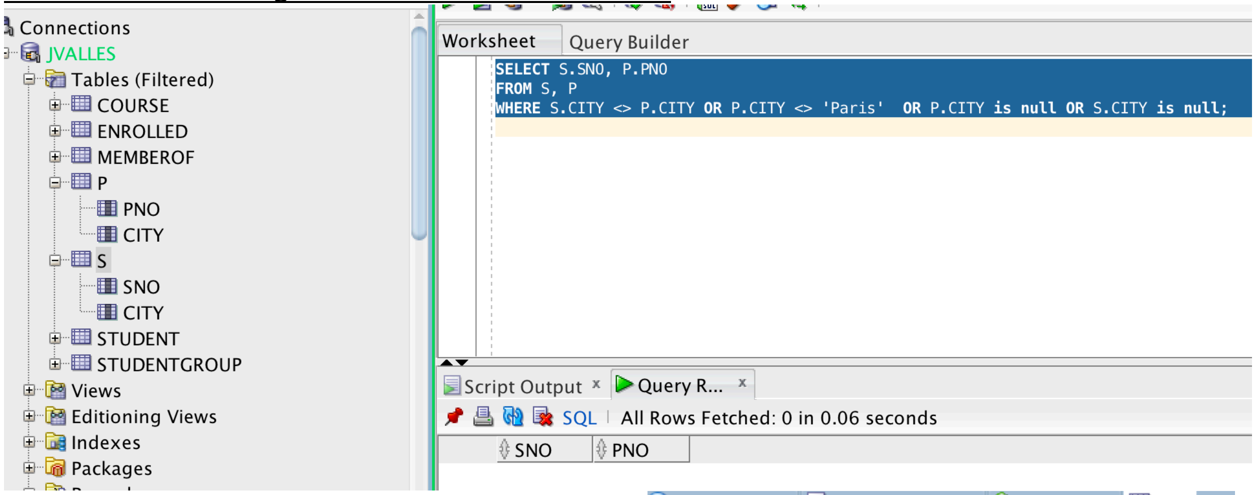
1. Reading done.

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;

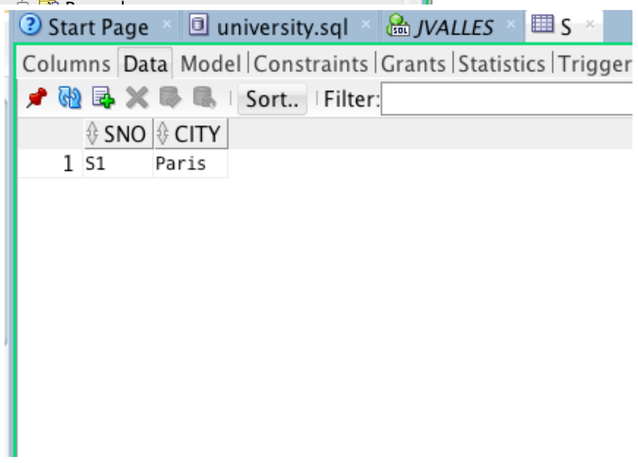
<u>Condition #1 –</u> Both S.City not null and P.City are not null Date's query works just fine.
<u>Both S.city and P.city are known should work. Here S.city is 'Paris' and P.city is 'Paris', which should return nothing as FALSE and FALSE = FALSE.</u>



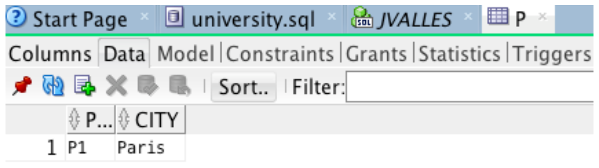
```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris'  OR P.CITY is null OR S.CITY is null;
```

All Rows Fetched: 0 in 0.06 seconds

SNO	PNO

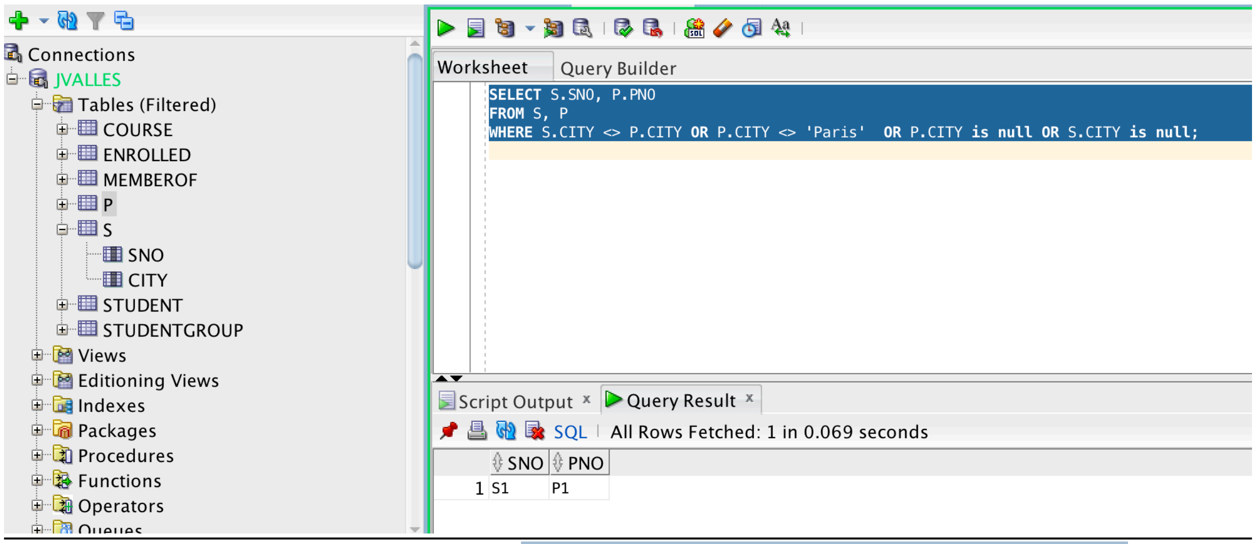


	SNO	CITY
1	S1	Paris



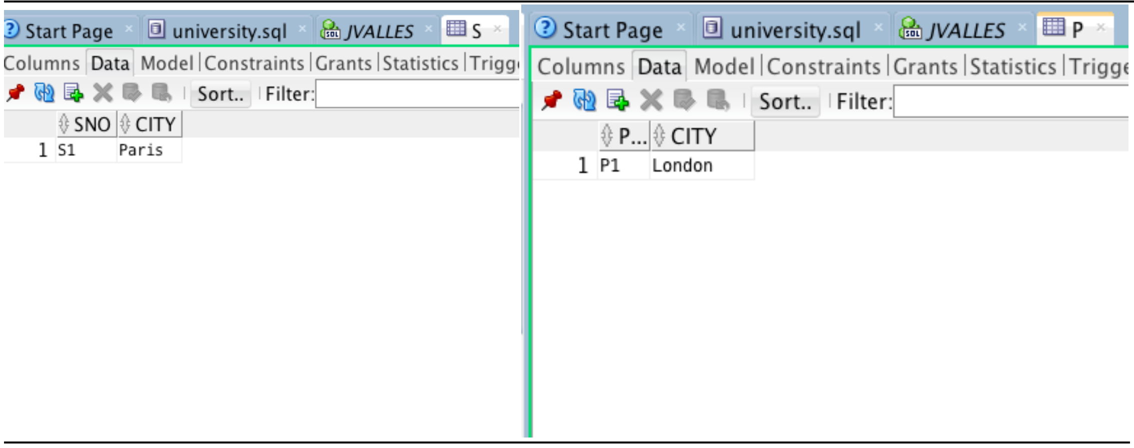
	P...	CITY
1	P1	Paris

<u>Condition #1 –</u> Both S.City not null and P.City are not null
<u>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)</u>



```
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;
```

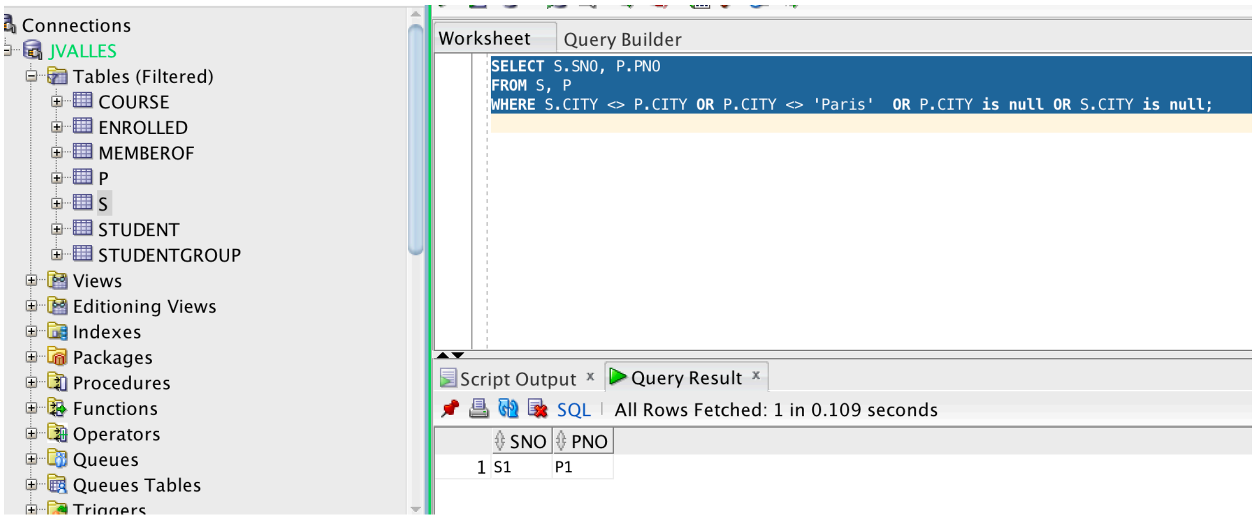
```
All Rows Fetched: 1 in 0.069 seconds

  SNO PNO
1 S1  P1
```



```
  SNO CITY
1 S1  Paris
```

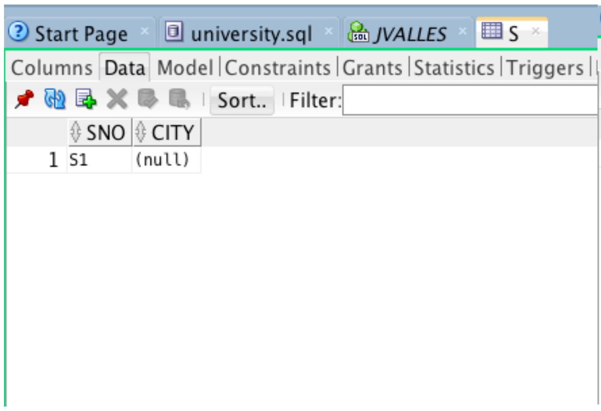
```
  P... CITY
1 P1   London
```

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.

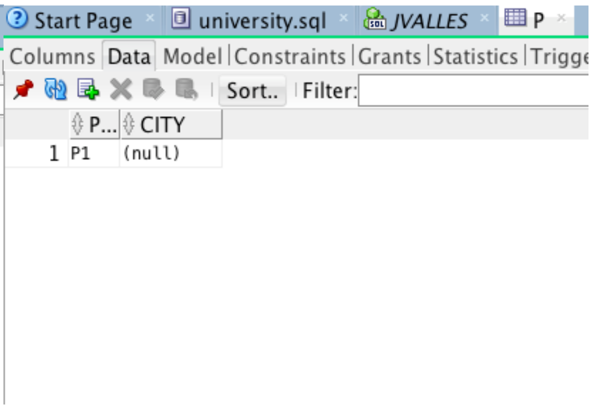


```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris'  OR P.CITY is null OR S.CITY is null;
```

```
All Rows Fetched: 1 in 0.109 seconds

  SNO PNO
1 S1  P1
```



	SNO	CITY
1	S1	(null)

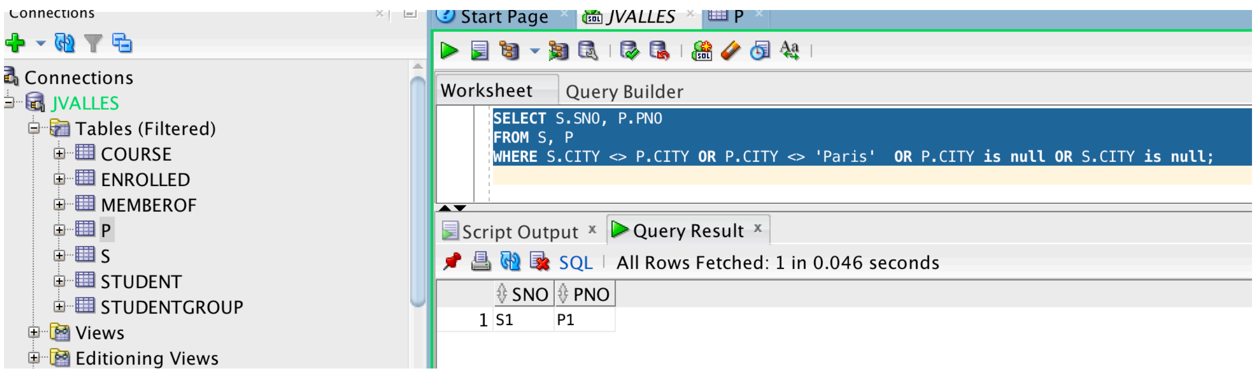


	P...	CITY
1	P1	(null)

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

Unknown or True = True
Unknown or False = Unknown

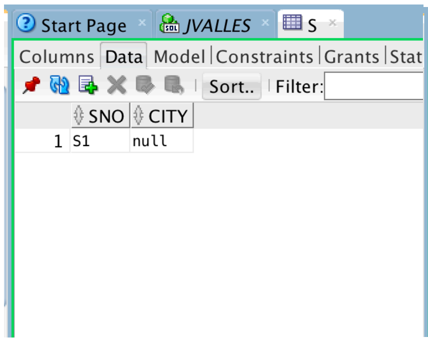
If S.city is null but P.city is not , but evaluates to False, the query returns unknown. In my example, P.city = 'Paris' and S.city is NULL in the real world this would return (S1, P1) since S.City is not the same as 'Paris'. Therefore, we must account for that, so I've added S.City is null.



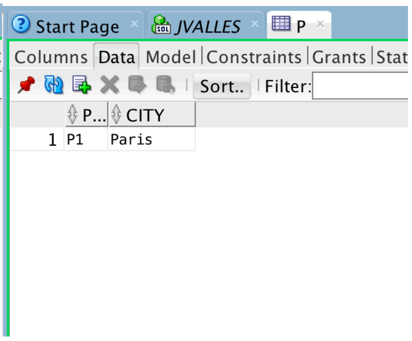
```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris'  OR P.CITY is null OR S.CITY is null;
```

All Rows Fetched: 1 in 0.046 seconds

	SNO	PNO
1	S1	P1



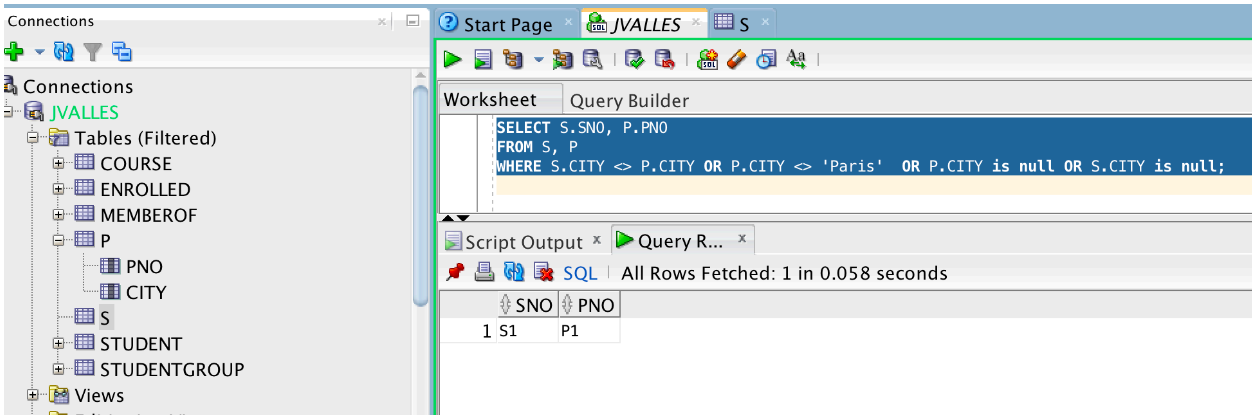
	SNO	CITY
1	S1	null



	P...	CITY
1	P1	Paris

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

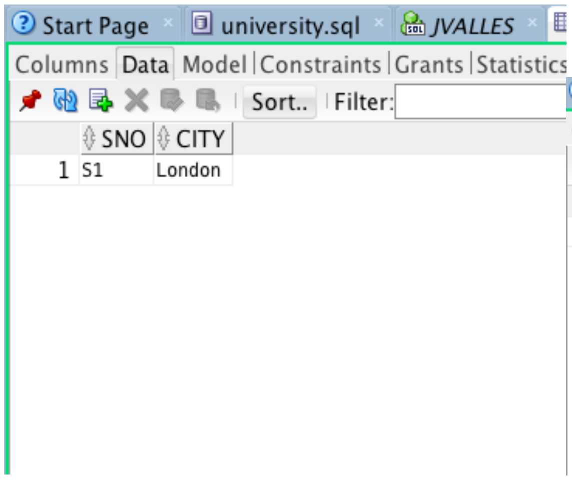
True or Unknown = True
False or Unkown = Unkown
Unkown or Unkown = Unknown
P.city is unknown in the first comparison S.city <> P.city and in the second, P.city <> 'Paris' also evaluates to unknown. When P.city is unknown both expressions result in unknown, thus Date's query fails. That is why P.City 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.



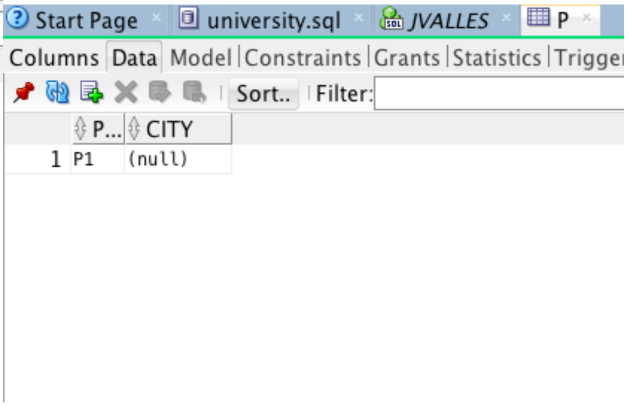
```
SELECT S.SNO, P.PNO
FROM S, P
WHERE S.CITY <> P.CITY OR P.CITY <> 'Paris'  OR P.CITY is null OR S.CITY is null;
```

All Rows Fetched: 1 in 0.058 seconds

	SNO	PNO
1	S1	P1



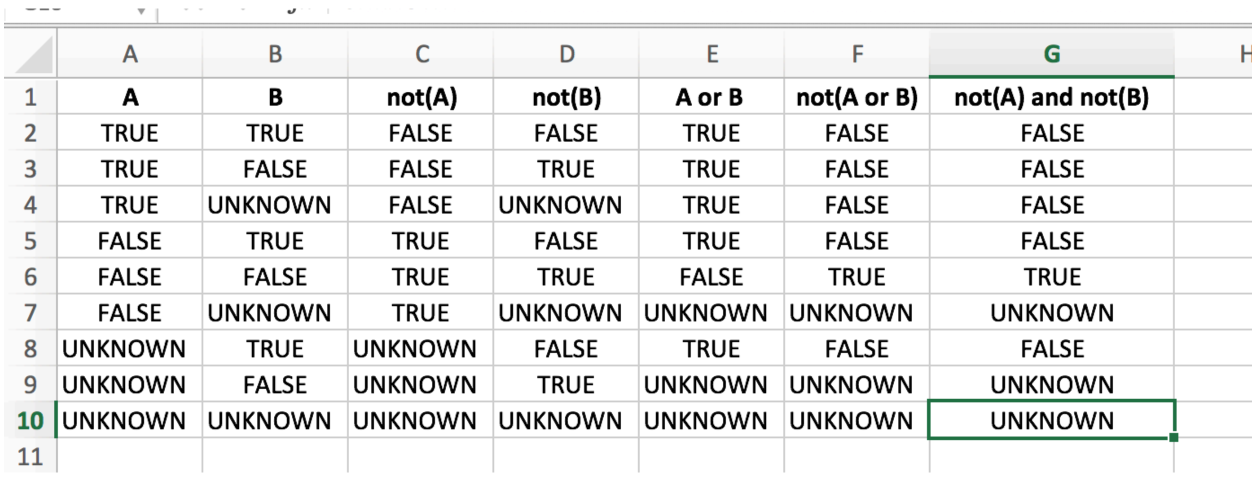
	SNO	CITY
1	S1	London



	P...	CITY
1	P1	(null)

3.

3-value logic truth table.



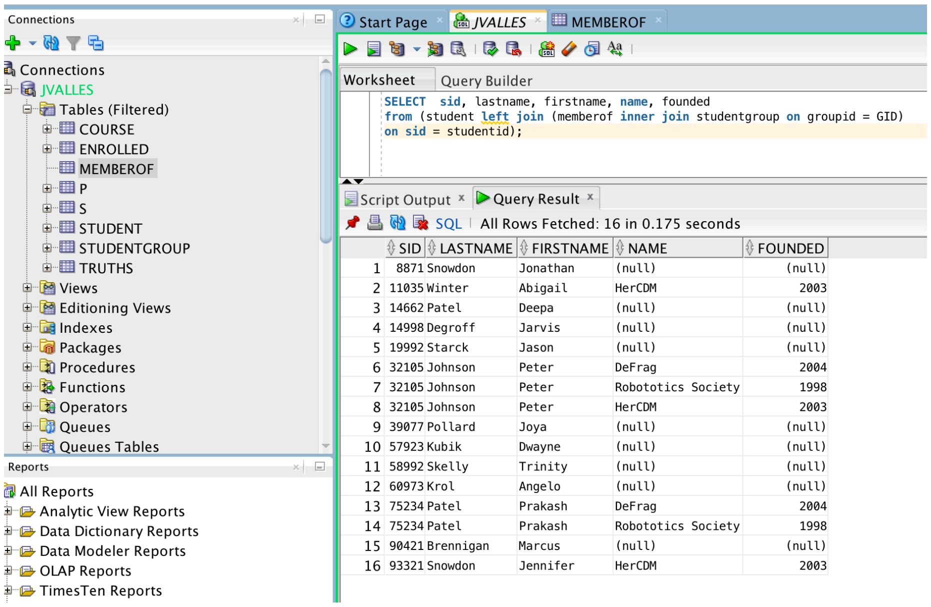
	A	B	C	D	E	F	G
1	**A**	**B**	**not(A)**	**not(B)**	**A or B**	**not(A or B)**	**not(A) and not(B)**
2	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
3	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
4	TRUE	UNKNOWN	FALSE	UNKNOWN	TRUE	FALSE	FALSE
5	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
6	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	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);

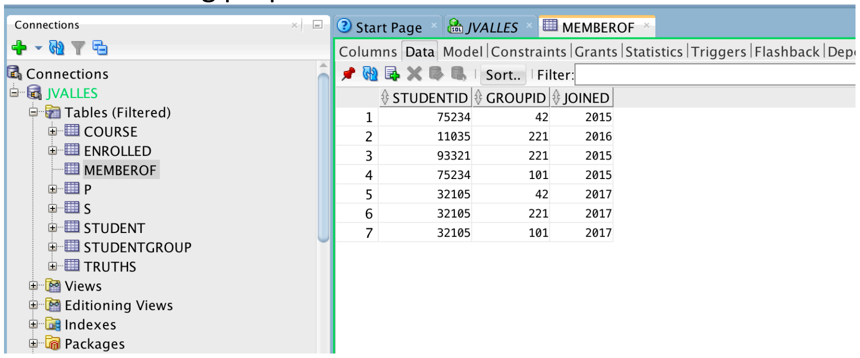


```
SELECT  sid, lastname, firstname, name, founded
from (student left join (memberof inner join studentgroup on groupid = GID)
on sid = studentid);
```

All Rows Fetched: 16 in 0.175 seconds

	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

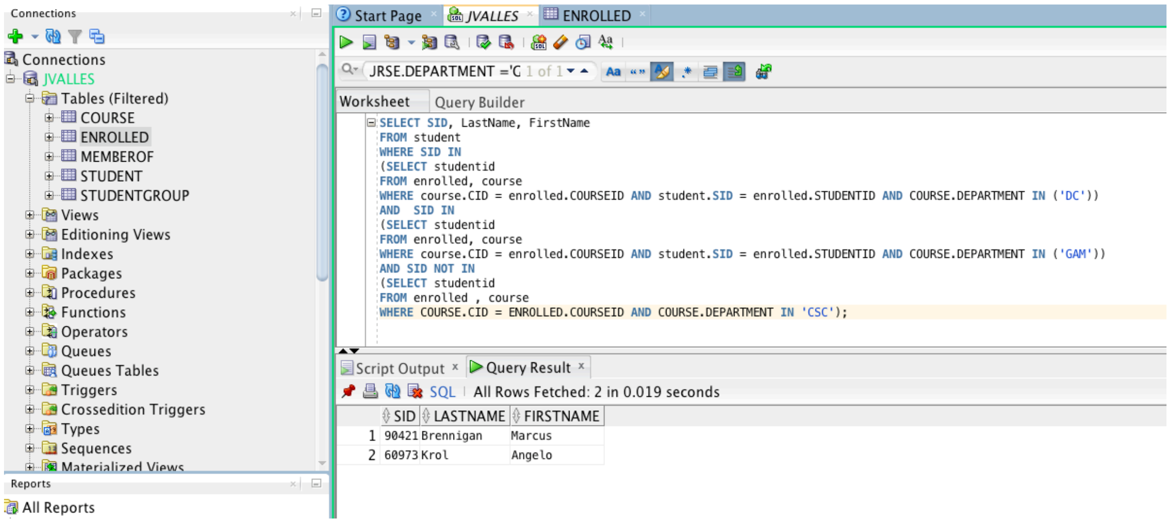
Groups that have no members is Computer Science Society -2 , I removed from Member of Table for testing purposes.



	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');

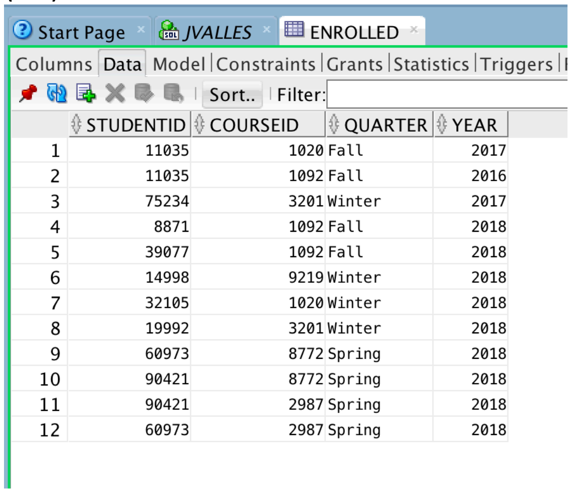


```
SELECT SID, LastName, FirstName
FROM student
WHERE SID IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND student.SID = enrolled.STUDENTID AND COURSE.DEPARTMENT IN ('DC'))
AND  SID IN
(SELECT studentid
FROM enrolled, course
WHERE course.CID = enrolled.COURSEID AND 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');
```

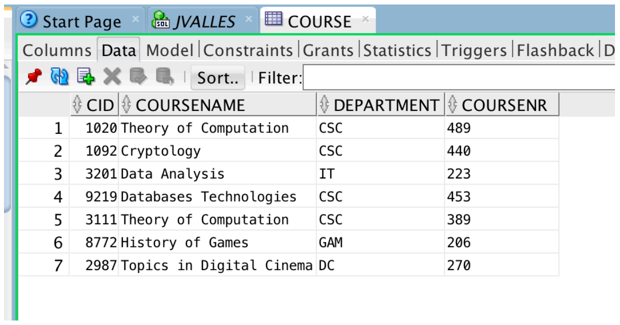
All Rows Fetched: 2 in 0.019 seconds

	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).



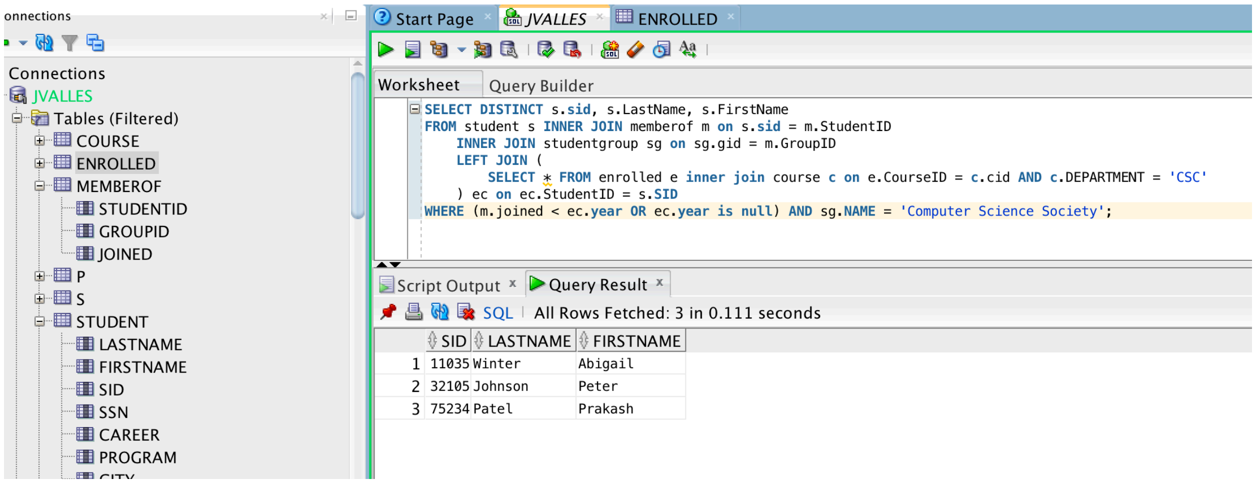
	STUDENTID	COURSEID	QUARTER	YEAR
1	11035	1020	Fall	2017
2	11035	1092	Fall	2016
3	75234	3201	Winter	2017
4	8871	1092	Fall	2018
5	39077	1092	Fall	2018
6	14998	9219	Winter	2018
7	32105	1020	Winter	2018
8	19992	3201	Winter	2018
9	60973	8772	Spring	2018
10	90421	8772	Spring	2018
11	90421	2987	Spring	2018
12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



	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.

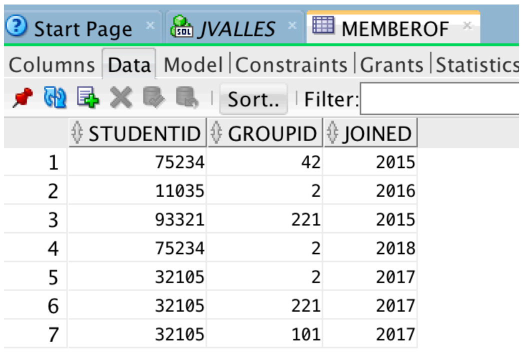


```
SELECT DISTINCT s.sid, s.LastName, s.FirstName
FROM student s INNER JOIN memberof m on s.sid = m.StudentID
    INNER JOIN studentgroup sg on sg.gid = m.GroupID
    LEFT JOIN (
        SELECT * FROM enrolled e inner join course c on e.CourseID = c.cid AND c.DEPARTMENT = 'CSC'
    ) ec on ec.StudentID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

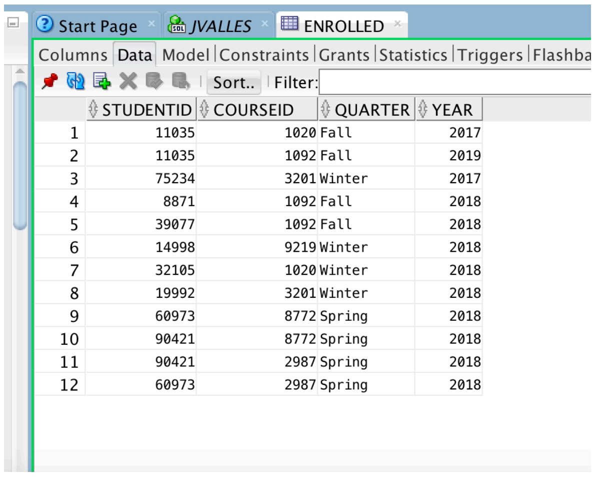
All Rows Fetched: 3 in 0.111 seconds

	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.



	STUDENTID	GROUPID	JOINED
1	75234	42	2015
2	11035	2	2016
3	93321	221	2015
4	75234	2	2018
5	32105	2	2017
6	32105	221	2017
7	32105	101	2017

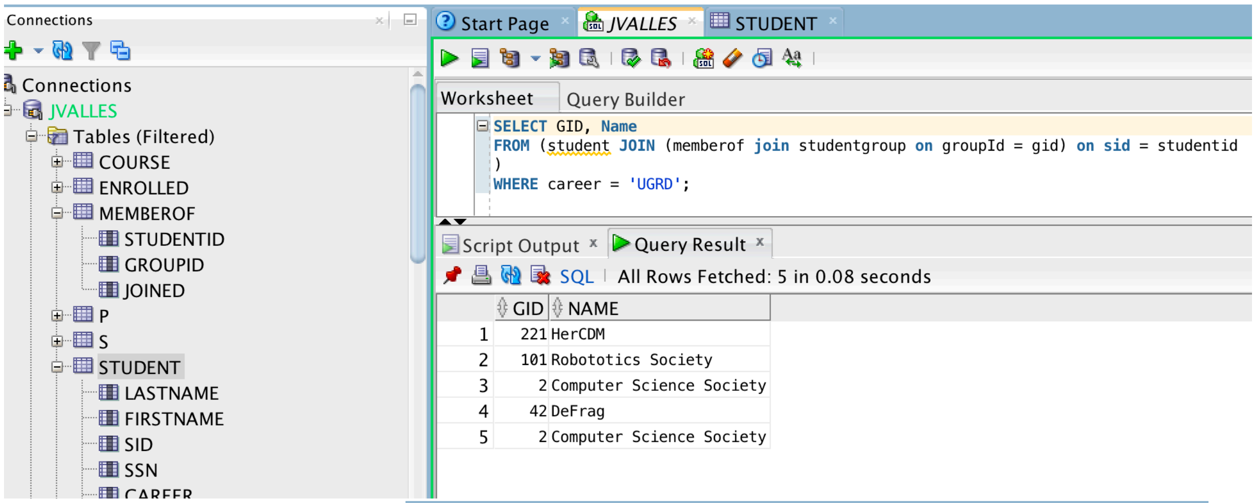


	STUDENTID	COURSEID	QUARTER	YEAR
1	11035	1020	Fall	2017
2	11035	1092	Fall	2019
3	75234	3201	Winter	2017
4	8871	1092	Fall	2018
5	39077	1092	Fall	2018
6	14998	9219	Winter	2018
7	32105	1020	Winter	2018
8	19992	3201	Winter	2018
9	60973	8772	Spring	2018
10	90421	8772	Spring	2018
11	90421	2987	Spring	2018
12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

d.

List groups that only have undergraduate members

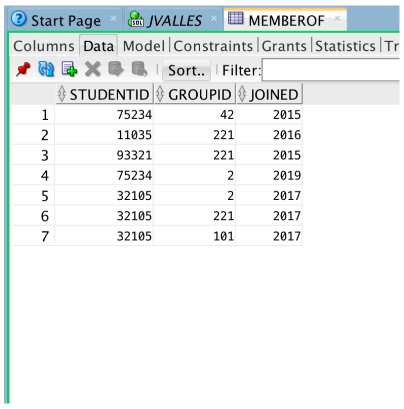
SELECT GID, Name, career, FirstName
FROM (student JOIN (memberof JOIN studentgroup on groupId = gid) on sid = studentid
)
WHERE career = 'UGRD';



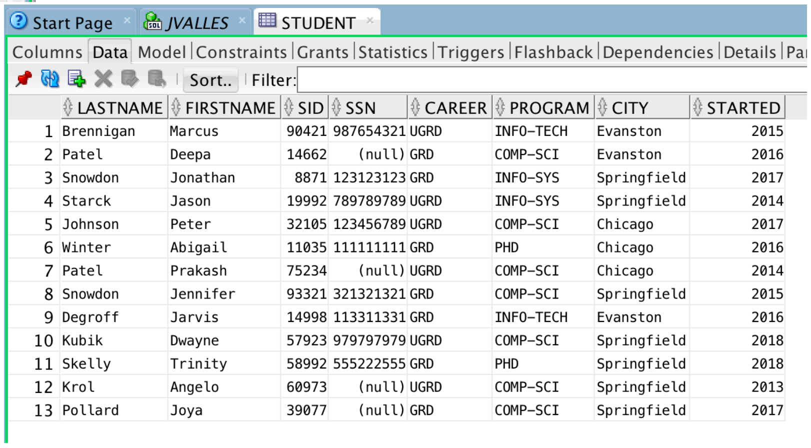
```
SELECT GID, Name
FROM (student JOIN (memberof join studentgroup on groupId = gid) on sid = studentid
)
WHERE career = 'UGRD';
```

All Rows Fetched: 5 in 0.08 seconds

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



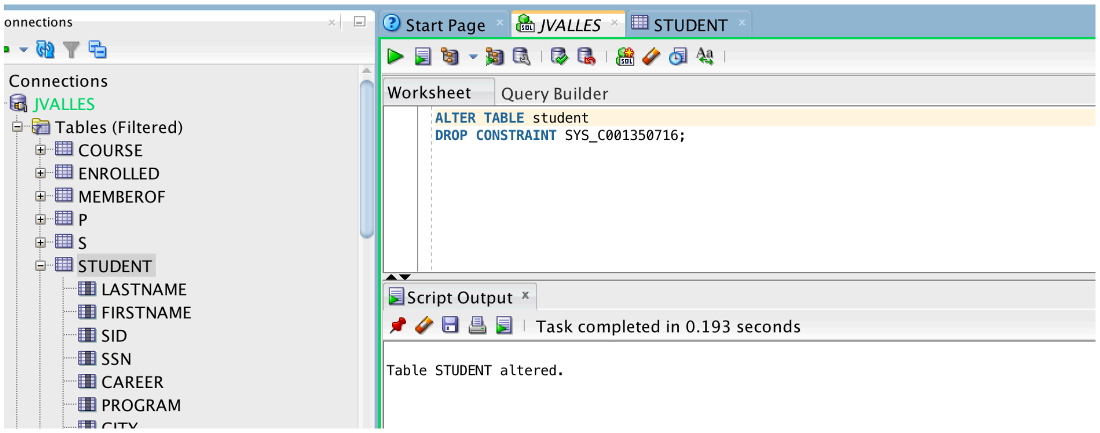
	STUDENTID	GROUPID	JOINED
1	75234	42	2015
2	11035	221	2016
3	93321	221	2015
4	75234	2	2019
5	32105	2	2017
6	32105	221	2017
7	32105	101	2017



	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	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

e.

Removing table constraint (unique)

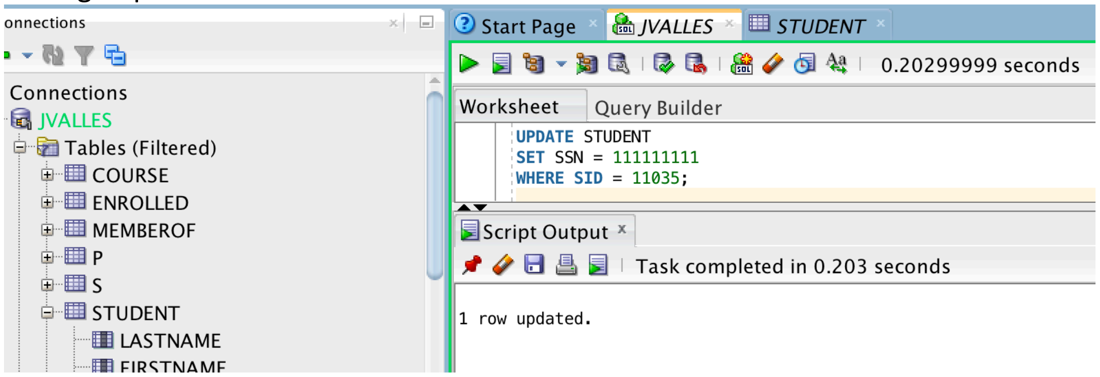


```
ALTER TABLE student
DROP CONSTRAINT SYS_C001350716;
```

Task completed in 0.193 seconds

```
Table STUDENT altered.
```

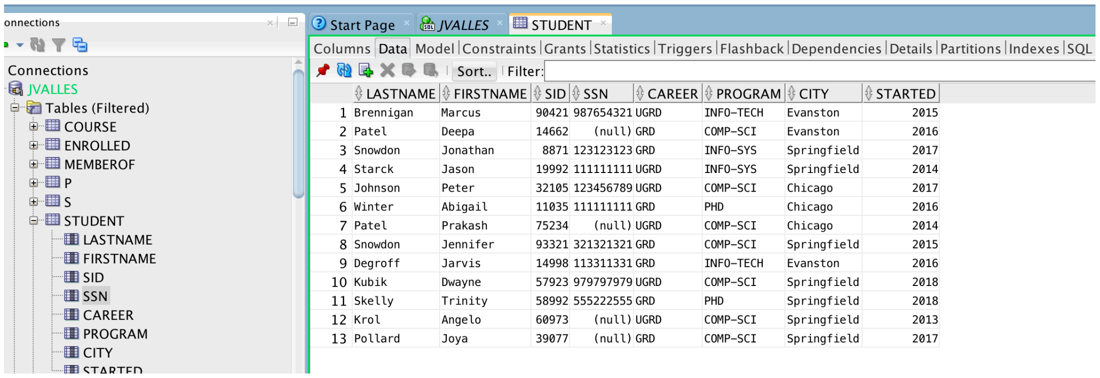
adding duplicate SSN number



```
UPDATE STUDENT
SET SSN = 111111111
WHERE SID = 11035;
```

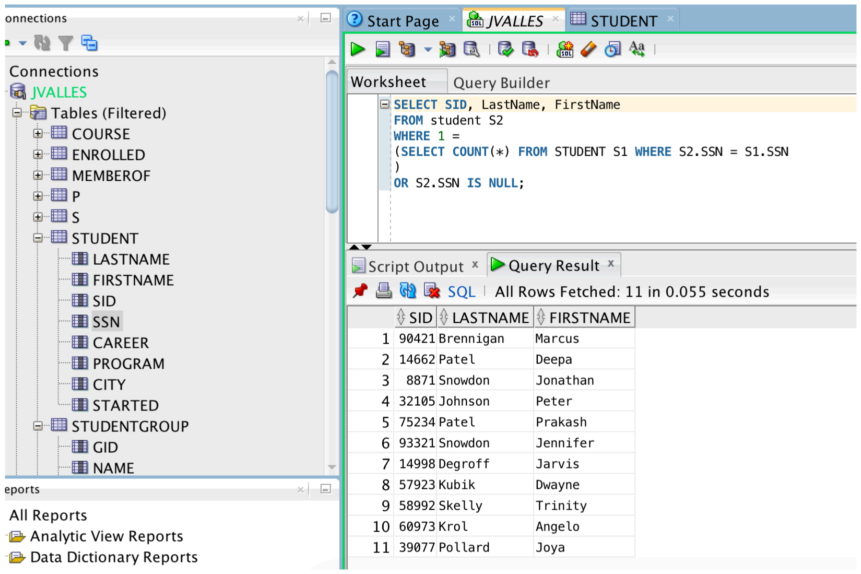
Task completed in 0.203 seconds

```
1 row updated.
```



	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	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;



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

All Rows Fetched: 11 in 0.055 seconds

	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 = ss1.ssn or where s2.ssn is null and return those records.

Abigal 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.