

Homework 4

1) Draw the following JSON structure using the method shown in class. You can use any tool you like, but no hand-drawing.

```
{
  "created_at": "Jun 22",
  "id": 87799,
  "text": "Grocery List",
  "truncated": false,
  "entities": {
    "hashtags": [
      {
        "text": "Angular",
        "indices": [103, 111]
      }
    ],
    "symbols": [A, B],
    "user_mentions": [3, 4],
    "urls": [
      {
        "url": "https://t.co/xF",
        "indices": [79, 102]
      }
    ]
  },
  "user": {
    "name": "SitePoint",
    "screen_name": "SitePointJS",
    "location": "Melbourne",
    "description": "JavaScript",
    "url": "http://t.co/cC",
  },
}
```

Answer 1)

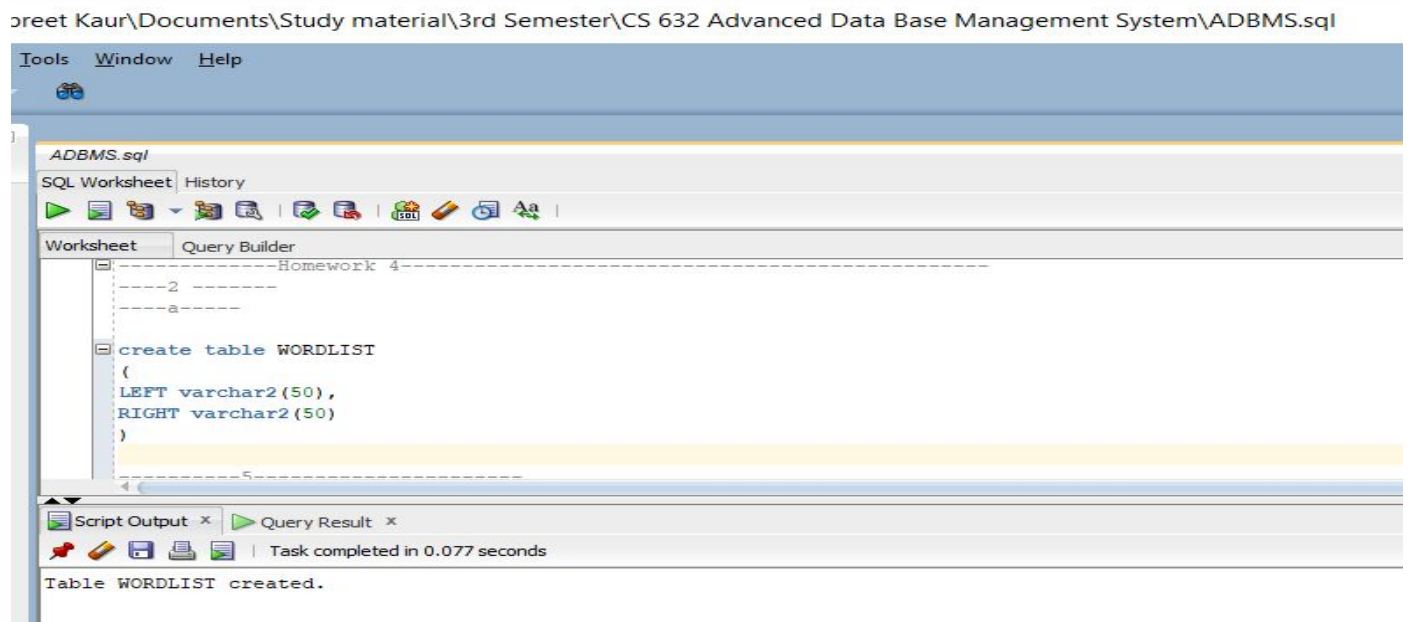
JSON structure is in the another page

2) a) Create a table WORDLIST that contains two columns LEFT and RIGHT, both varchar of 40 or more characters.

Answer 2a)

create table WORDLIST

```
(  
LEFT varchar2(50),  
RIGHT varchar2(50)  
)
```

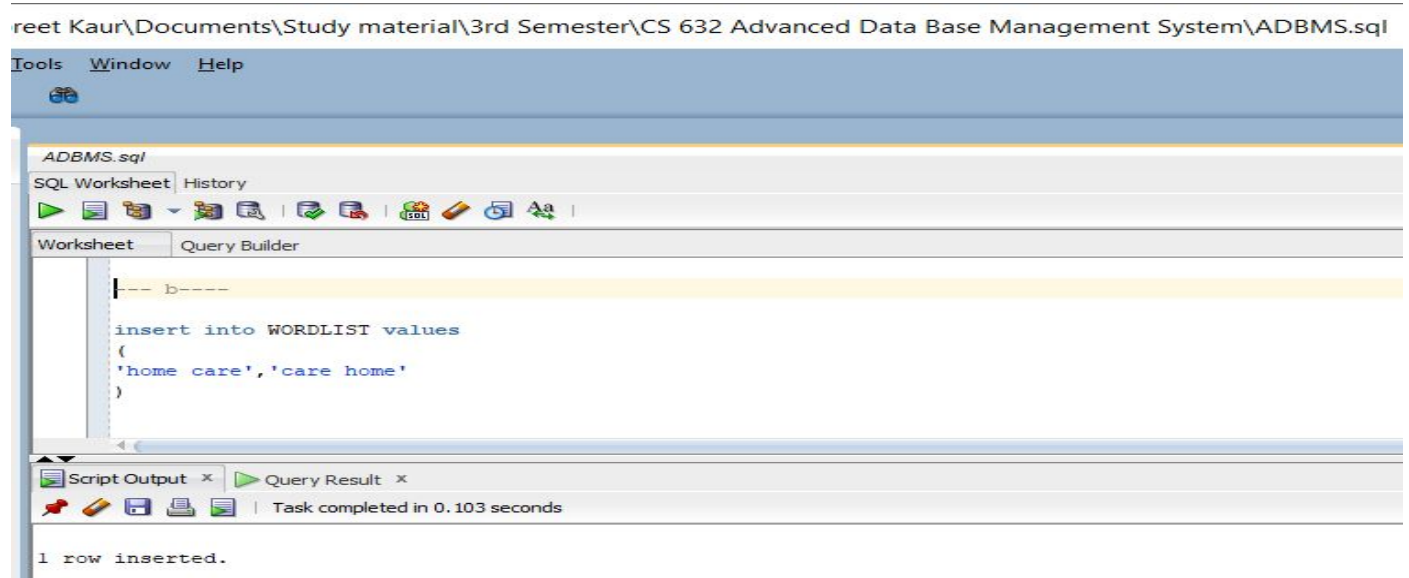


b) Insert one row into the table as follows:

The column LEFT should contain 'home care' and the column right should contain 'care home'
'home care', 'care home'

Answer b)

Insert Into Wordlist Values(
'home care','care home'
)

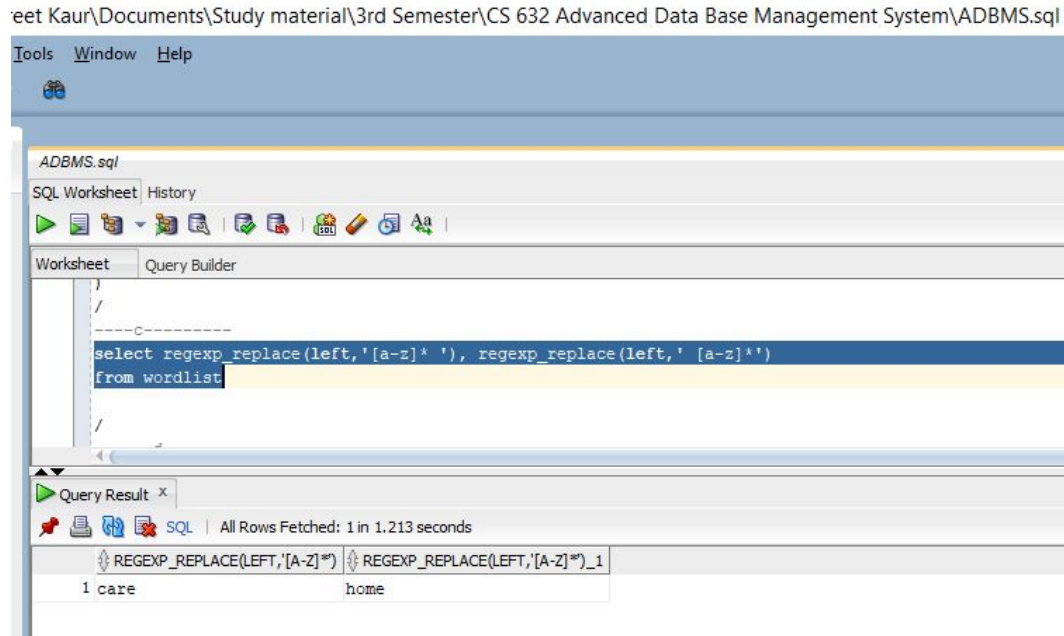


c) Our goal is to find any rows that contain such pairs, but only such pairs.

Write an SQL select statement using regular expressions that will display the two words in LEFT in reverse order, in two separate output columns.

Answer c)

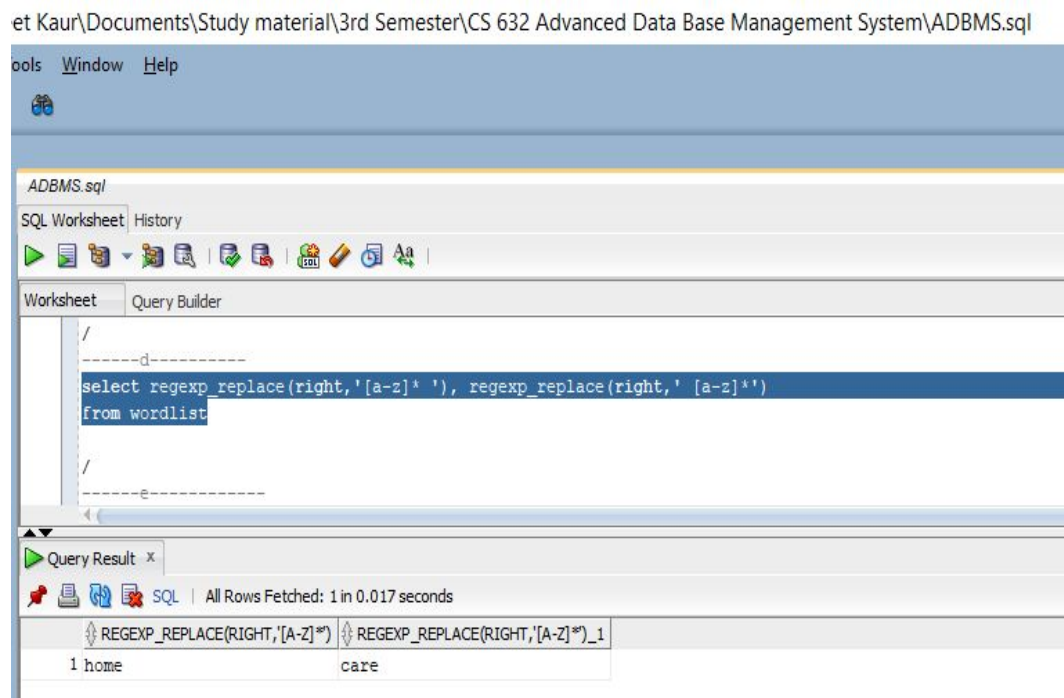
```
select regexp_replace(left,'[a-z]* '), regexp_replace(left,' [a-z]*')
from wordlist
```



d) Do the same thing in a separate SELECT statement for the right column.

Answer d)

```
select regexp_replace(right,'[a-z]* '), regexp_replace(right,' [a-z]*')
from wordlist
```

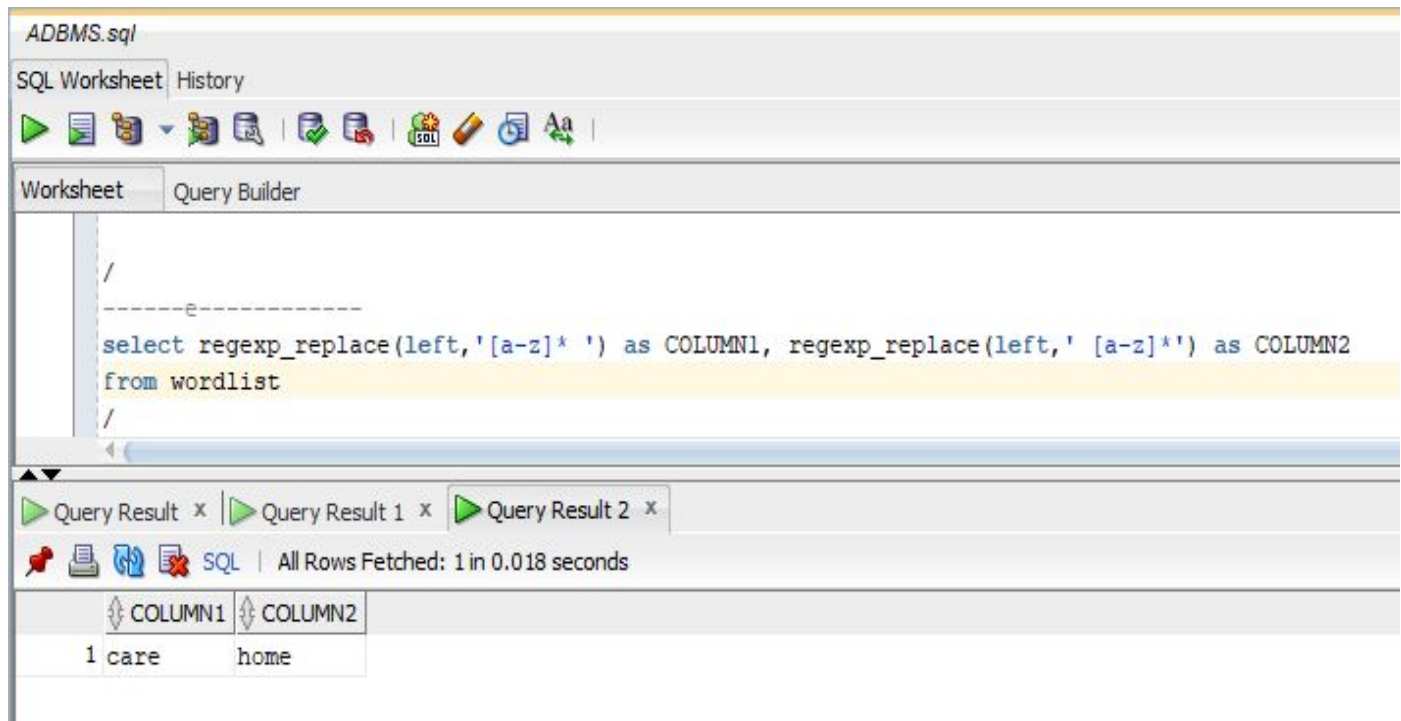


e) Assign column aliases to all four regular expressions. (Use: Column1, Column2, Column3, Column4). I showed this already above. Redo c) and d) to show this.

Answer e)

**select regexp_replace(left,'[a-z]* ') as COLUMN1, regexp_replace(left,' [a-z]*') as COLUMN2
from wordlist**

/



The screenshot shows the ADBMS SQL Worksheet interface. The SQL query is entered in the Worksheet tab:

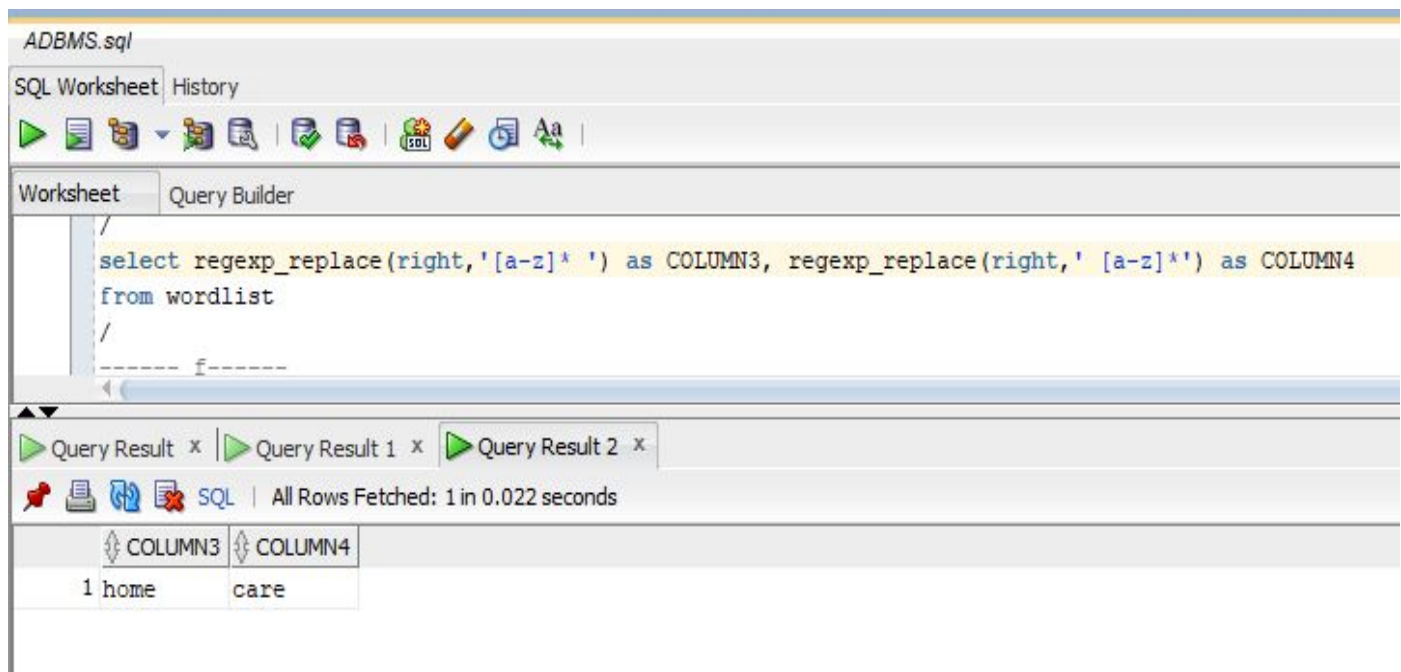
```
/
-----e-----
select regexp_replace(left,'[a-z]* ') as COLUMN1, regexp_replace(left,' [a-z]*') as COLUMN2
from wordlist
/
```

The Query Results pane shows the results of the query:

	COLUMN1	COLUMN2
1	care	home

**select regexp_replace(right,'[a-z]* ') as COLUMN3, regexp_replace(right,' [a-z]*') as COLUMN4
from wordlist**

/



The screenshot shows the ADBMS SQL Worksheet interface. The SQL query is entered in the Worksheet tab:

```
/
select regexp_replace(right,'[a-z]* ') as COLUMN3, regexp_replace(right,' [a-z]*') as COLUMN4
from wordlist
/
-----f-----
```

The Query Results pane shows the results of the query:

	COLUMN3	COLUMN4
1	home	care

f) Now write a join operation (reminder: don't use the word join) between the two previous select statements, by NESTING them inside of another SELECT with no WHERE clause.

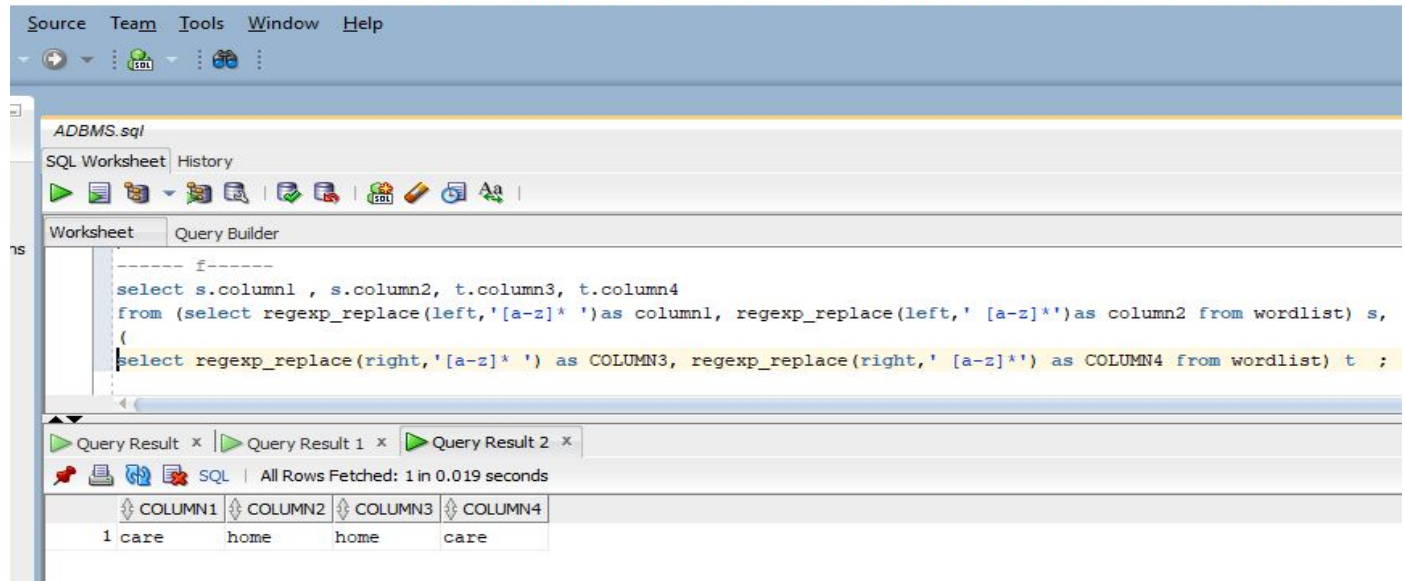
This should give you something like this:

Column1	Column2	Column3	Column4
care	home	home	care

Answer f)

```
select s.column1 , s.column2, t.column3, t.column4
from (select regexp_replace(left,'[a-z]* ')as column1, regexp_replace(left,' [a-z]*')as column2 from
wordlist) s,
(select regexp_replace(right,'[a-z]* ') as COLUMN3, regexp_replace(right,' [a-z]*') as COLUMN4 from
wordlist) t ;
```

Users\Charanpreet Kaur\Documents\Study material\3rd Semester\CS 632 Advanced Data Base Management System\ADBMS.sql

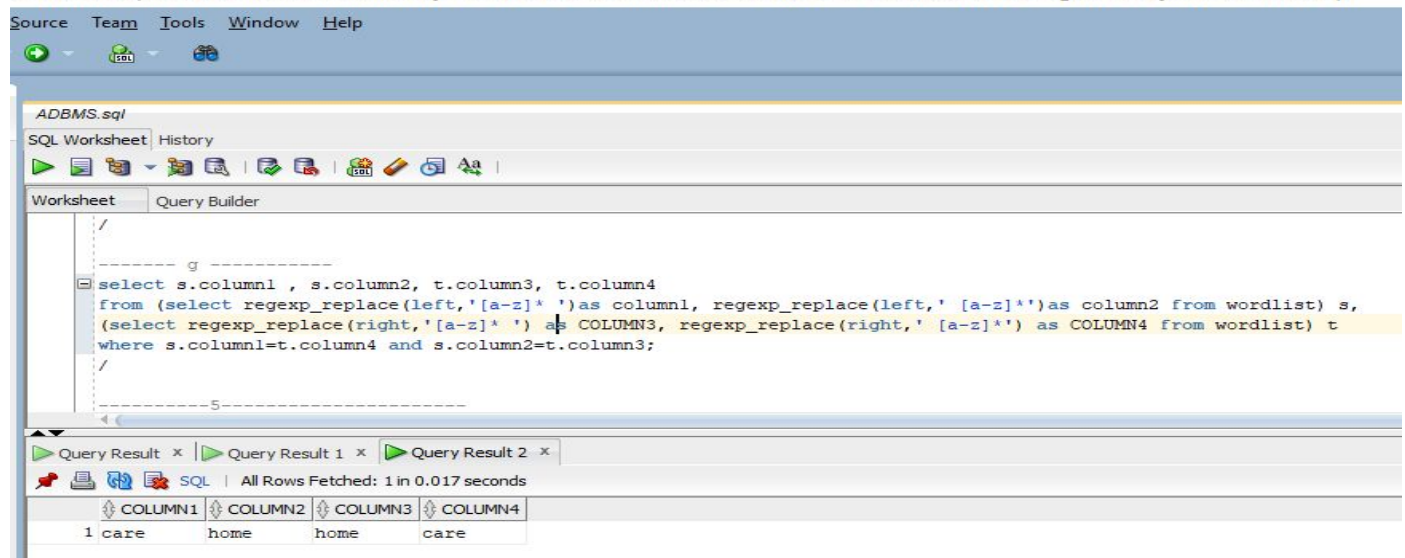


g) Add a WHERE clause to the outer SELECT so that Column1 = Column4 and Column2 = Column3.

Answer g)

```
select s.column1 , s.column2, t.column3, t.column4
from (select regexp_replace(left,'[a-z]* ')as column1, regexp_replace(left,' [a-z]*')as column2 from
wordlist) s,
(select regexp_replace(right,'[a-z]* ') as column3, regexp_replace(right,' [a-z]*') as column4 from
wordlist) t
where s.column1=t.column4 and s.column2=t.column3;
```

Users\Charanpreet Kaur\Documents\Study material\3rd Semester\CS 632 Advanced Data Base Management System\ADBMS.sql



h) Add the keyword ***unique*** to the outer SELECT so that you get only ONE row that looks like this.

Column1	Column2	Column3	Column4
---------	---------	---------	---------

care	home	home	care
------	------	------	------

SHOW the complete nested SELECT and show that it correctly produces the above result.

Answer h)

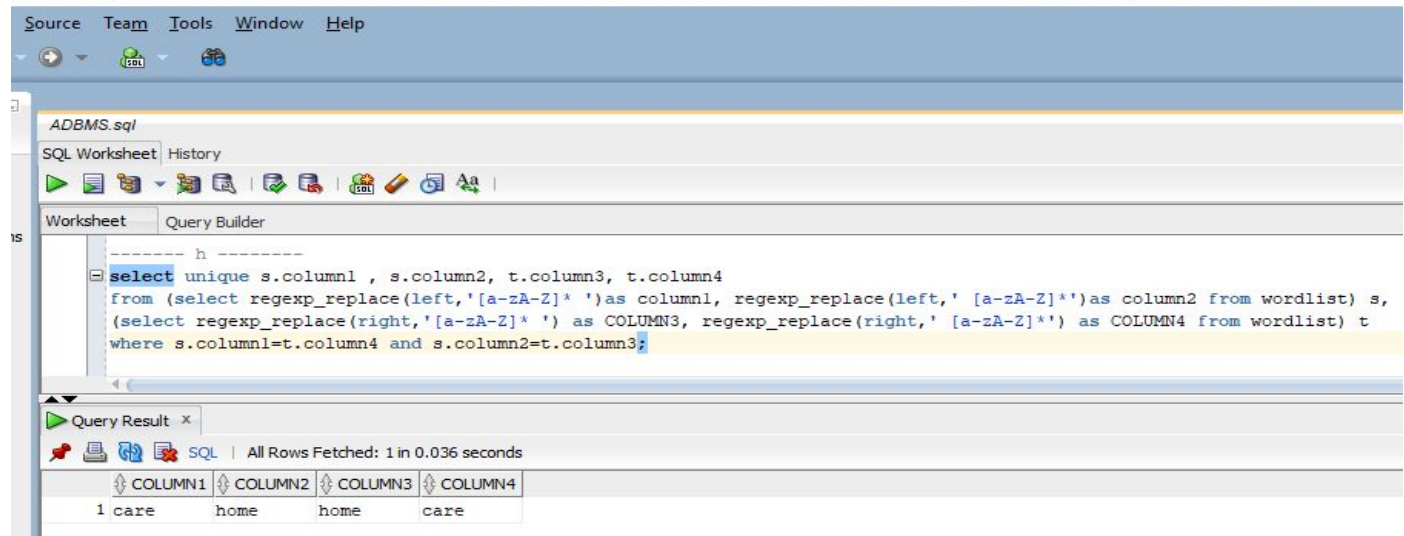
```
select unique s.column1 , s.column2, t.column3, t.column4
```

```
from (select regexp_replace(left,'[a-zA-Z]* ')as column1, regexp_replace(left,' [a-zA-Z]*')as column2  
from wordlist) s,
```

```
(select regexp_replace(right,'[a-zA-Z]* ') as COLUMN3, regexp_replace(right,' [a-zA-Z]*') as  
COLUMN4 from wordlist) t
```

```
where s.column1=t.column4 and s.column2=t.column3;
```

Users\Charanpreet Kaur\Documents\Study material\3rd Semester\CS 632 Advanced Data Base Management System\ADBMS.sql



i) Now insert into the column LEFT your own first name and last name as a string.

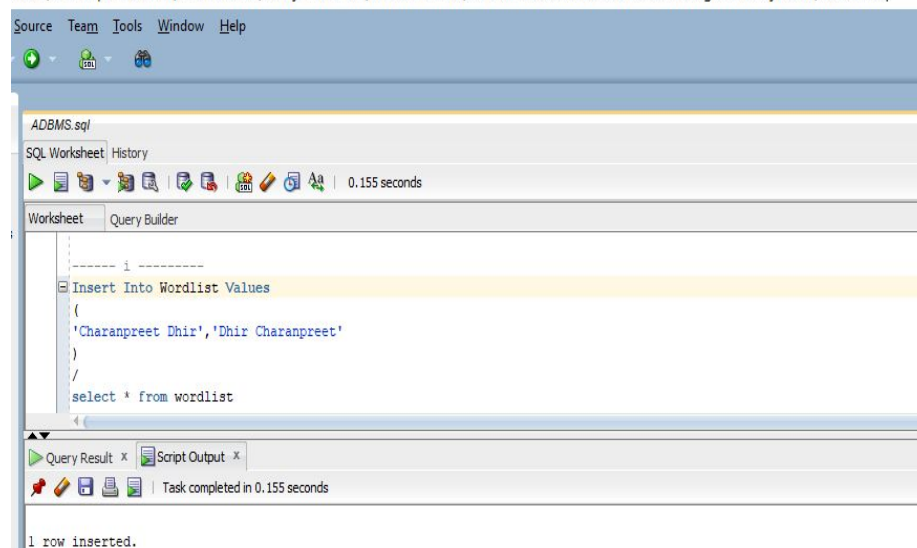
Insert into the column RIGHT your own last name and first name as a string.

Answer i)

Insert Into Wordlist Values

```
(  
'Charanpreet Dhir','Dhir Charanpreet'  
)
```

Users\Charanpreet Kaur\Documents\Study material\3rd Semester\CS 632 Advanced Data Base Management System\ADBMS.sql



j) Now run the select statement from question h) again and show the output.

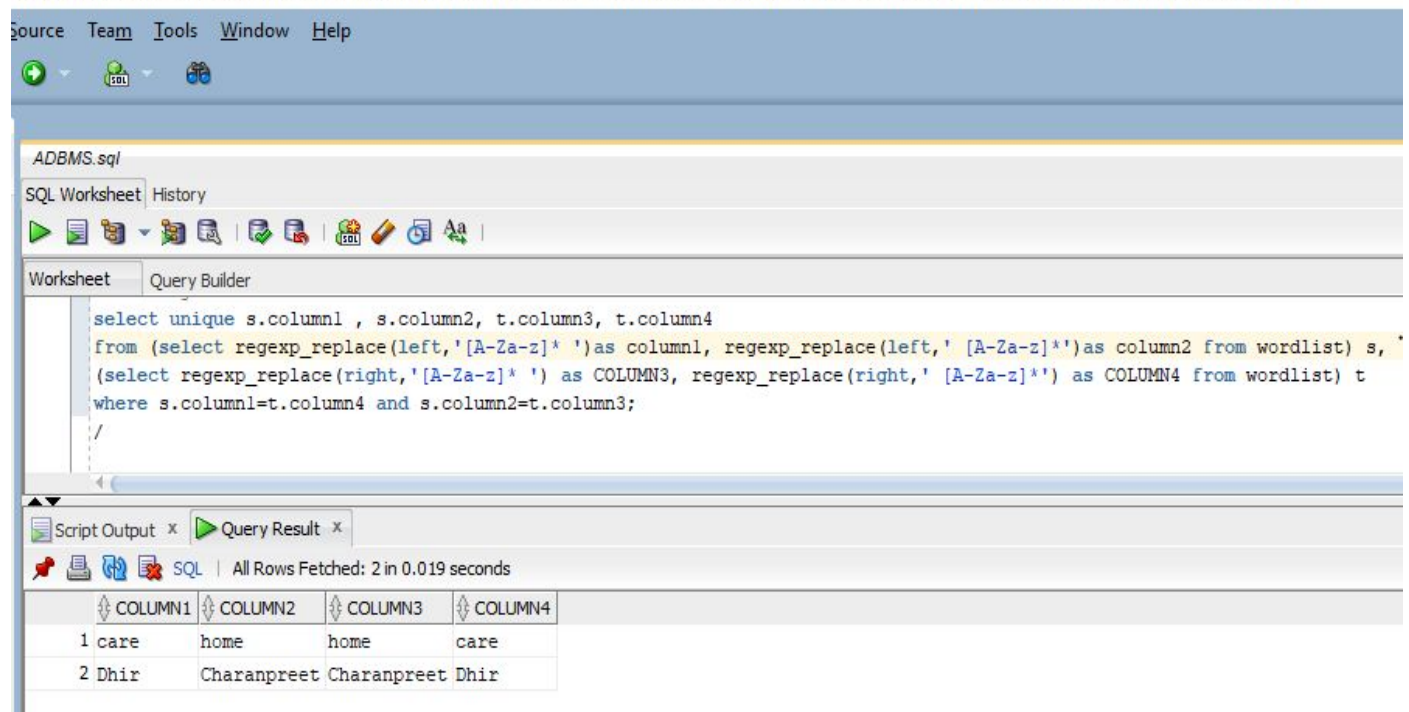
It should look like this:

Column1	Column2	Column3	Column4
care	home	home	care
Smith	John	John	Smith

Answer j)

```
select unique s.column1 , s.column2, t.column3, t.column4
from (select regexp_replace(left,'[a-z]* ')as column1, regexp_replace(left,' [a-z]*')as column2 from wordlist) s,
(select regexp_replace(right,'[a-z]* ') as COLUMN3, regexp_replace(right,' [a-z]*') as COLUMN4 from wordlist) t
where s.column1=t.column4 and s.column2=t.column3;
```

ers\Charanpreet Kaur\Documents\Study material\3rd Semester\CS 632 Advanced Data Base Management System\ADBMS.sql



3) We have to be afraid that there is “garbage” in the table WORDLIST. Like numbers.

a) Insert the following two rows into the table.

‘John 3brown’, ‘Brown John’

‘Tom! Smith’, ‘Smith Tom’

Answer 3 a)

insert into wordlist values(

‘John 3brown’, ‘Brown John’

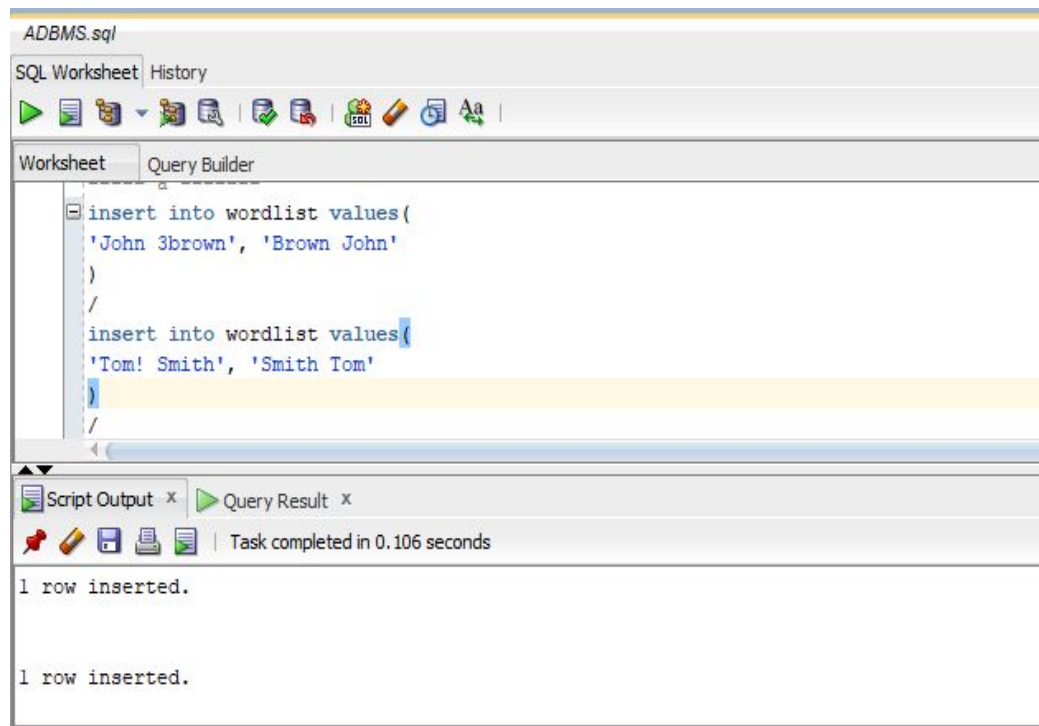
)

/

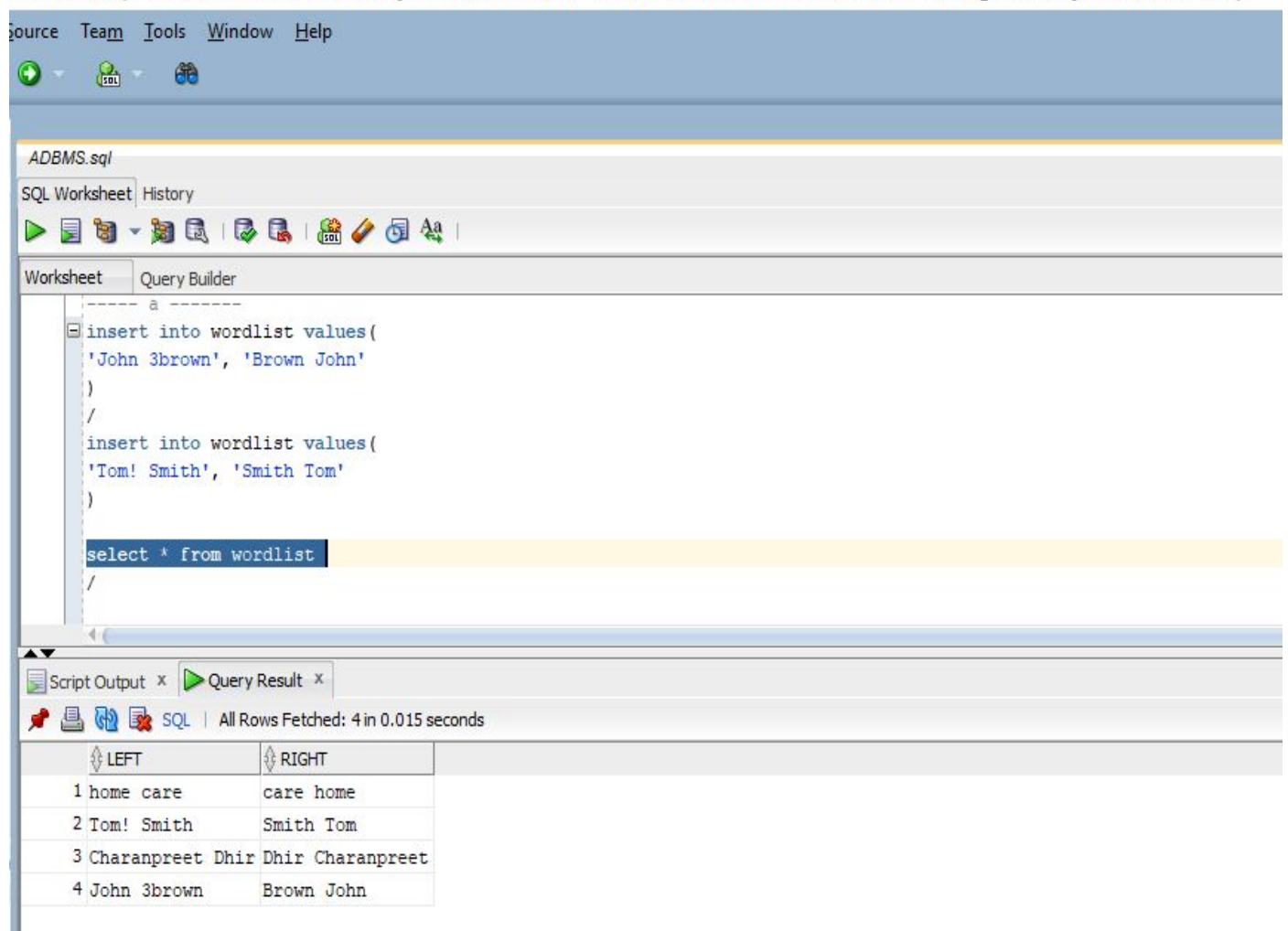
insert into wordlist values(

‘Tom! Smith’, ‘Smith Tom’

)



sers\Charanpreet Kaur\Documents\Study material\3rd Semester\CS 632 Advanced Data Base Management System\ADBMS.sql



b) Write a SELECT statement that will show only rows that consist of names made from pure letters. In other words, the above two rows will not show up.

Use regexp_like.

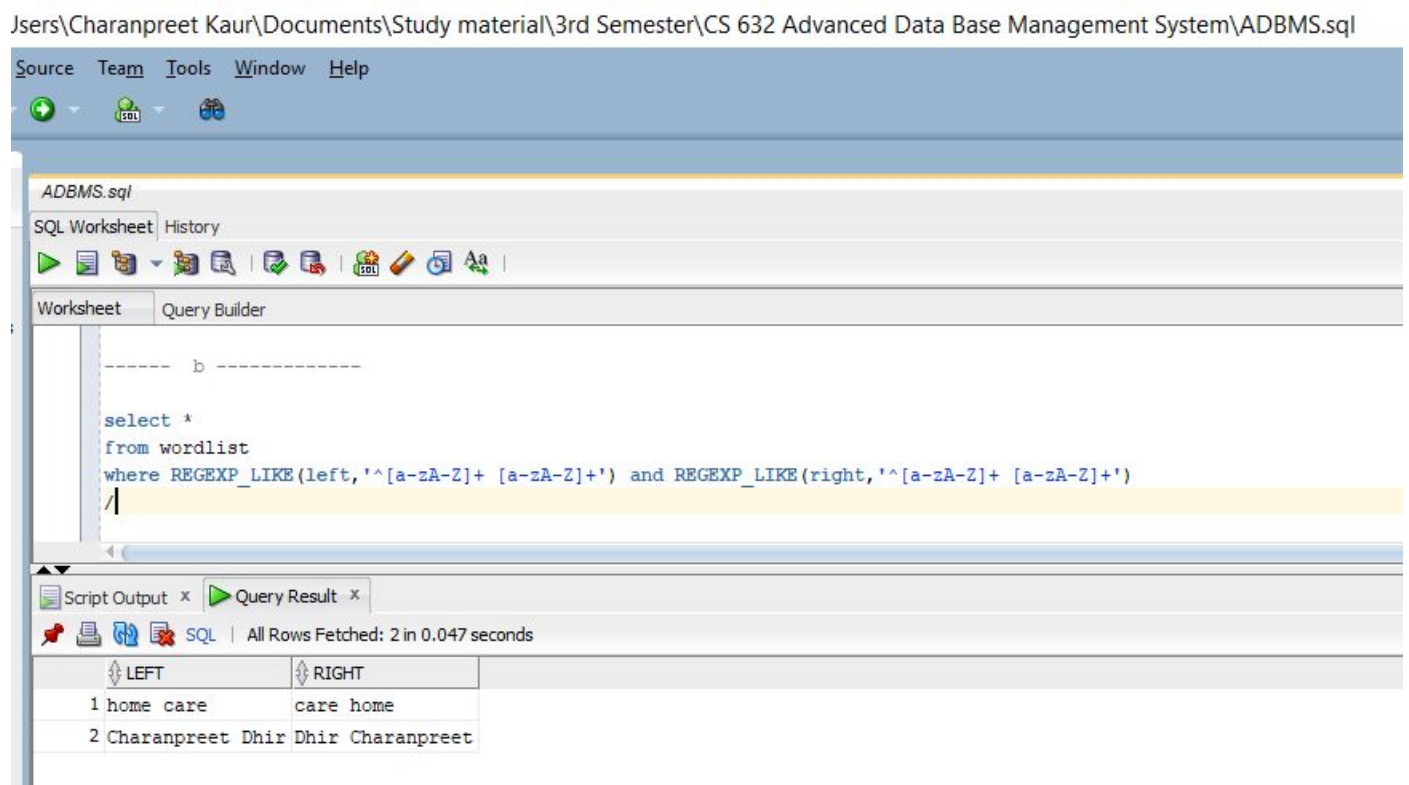
HINT: This can be very confusing. I got it wrong the first time.

regexp_like just verifies that your column contains a certain string. If you want to force it to contain ONLY this kind of string you have to use the special markers ^ and \$. Look it up in the lecture what those do.

Show your select statement and the output table.

Answer b)

```
select *
from wordlist
where REGEXP_LIKE(left,'^[a-zA-Z]+ [a-zA-Z]+') and REGEXP_LIKE(right,'^[a-zA-Z]+ [a-zA-Z]+')
```



c) Insert the following rows into the table:

'John Barry Brown', 'Brown John'

'Mike Barlow', 'Barlow Mike'

'Tom', 'Franklin Tom'

Note that there must be three blanks after the first Mike !!!!!

Answer c)

```
insert into wordlist values('John Barry Brown','Brown John')
```

```
/
```

```
insert into wordlist values('Mike Barlow','Barlow Mike')
```

```
/
```

```
insert into wordlist values('Tom','Franklin Tom')
```

```
/
```

```
Select * from wordlist
```

d) Write a SELECT statement that will NOT show ANY row where there are more than two words in either column. It should also NOT show any row where there is just one word in a column. HOWEVER, it should show rows where there are two words separated by several blanks. Thus your new select statement should NOT show

'John Barry Brown', 'Brown John'

'Tom', 'Franklin Tom'

But it SHOULD show

'Mike Barlow', 'Barlow Mike'

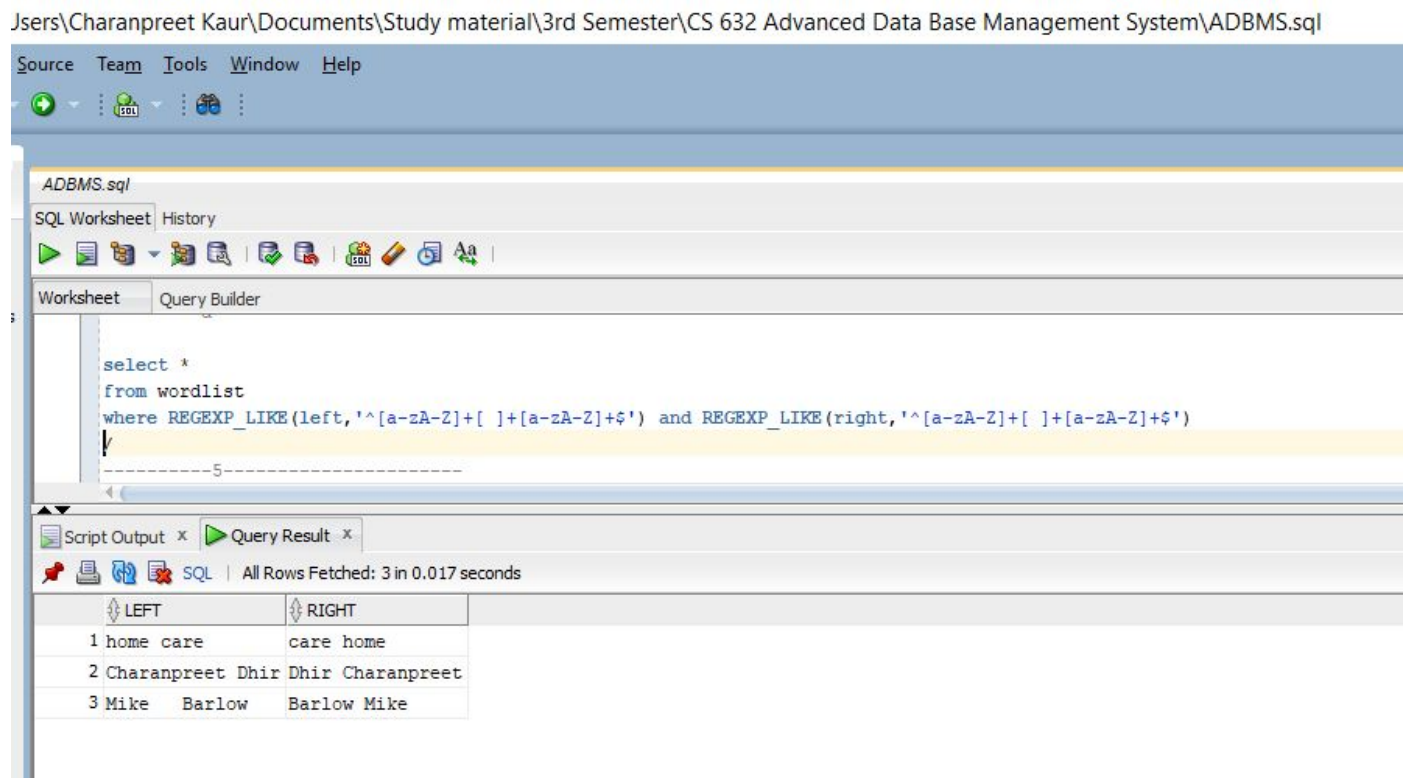
Also WORDS means again only combinations of letters. There shouldn't be any digits or special characters in the words.

Thus this SELECT statement should return only two rows, if you did everything right.

Again, all the work must be done with regular expressions.

Answer d)

```
select * from wordlist where REGEXP_LIKE(left,'^[a-zA-Z]+[ ]+[a-zA-Z]+$') and  
REGEXP_LIKE(right,'^[a-zA-Z]+[ ]+[a-zA-Z]+$')
```



4) a) Insert into WORDLIST this data row.

'Barbara Ball', 'Anita4 House'

Answer 4) a)

```
insert into wordlist values(  
'Barbara Ball','Anita4 House'  
)
```

b) Write a complete PL/SQL program using a cursor that will send to the screen... now this is complicated...

We want to send to the screen a single column.

This should contain all the column entries that contain+----- one digit.

But it does not matter if the digit is in the left column or in the right column.

So, for the table as it is now, the output should be:

John 3brown

Anita4 House

Note that John 3brown comes from the left column and Anita4 House comes from the right column. No header required.

Also notice that there should be NO BLANK lines, not before, not after and not between these two lines.

HINT: Review the use of "is null" and "is not null" in Oracle.

Show the program and its output.

Answer b)

begin

for expression in (select * from wordlist)

loop

if REGEXP_LIKE(expression.LEFT,'[0-9]')

then

DBMS_OUTPUT.PUT_LINE(expression.left);

elsif REGEXP_LIKE(expression.right,'[0-9]')

then

DBMS_OUTPUT.PUT_LINE(expression.right);

end if;

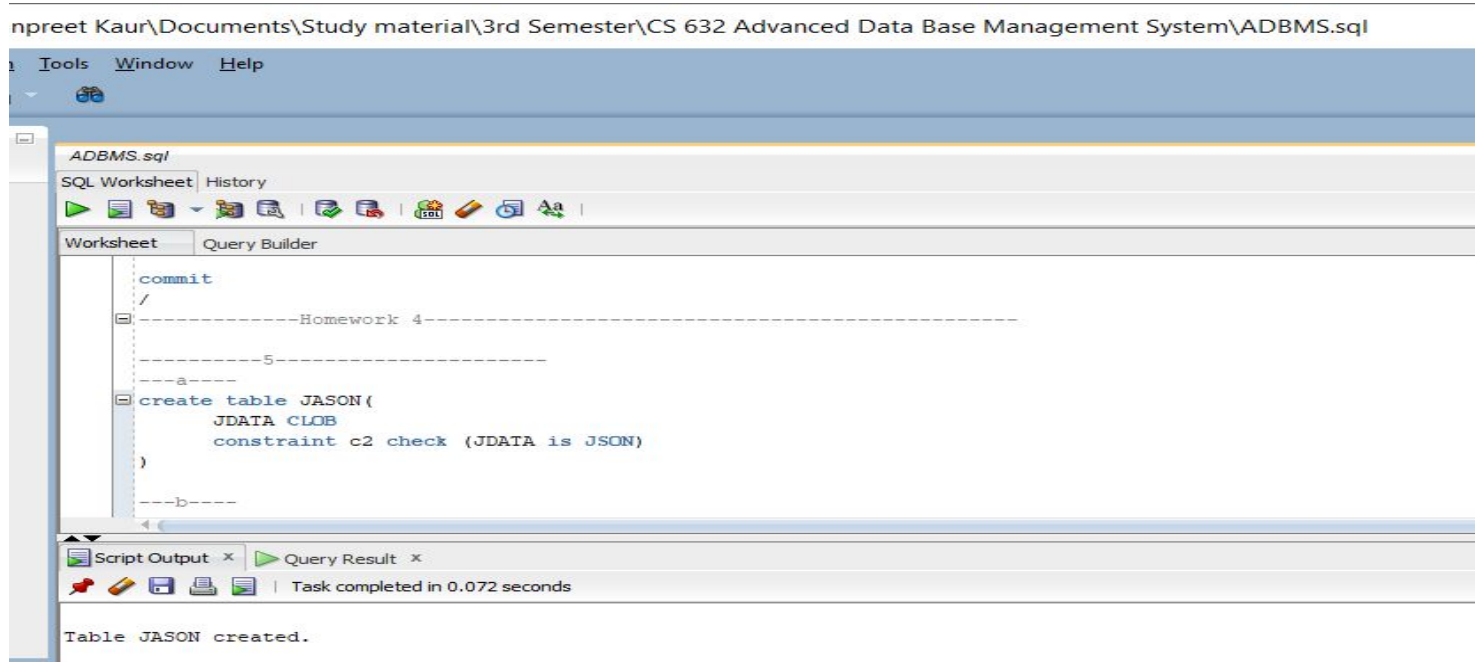
end loop;

end;

5) a) In ORACLE: Create a table JASON with a single column called JDATA of type JSON.

Answer 5 a)

```
create table JASON(  
    JDATA CLOB  
    constraint c2 check (JDATA is JSON)  
)
```



b) Insert the following into the table JASON.

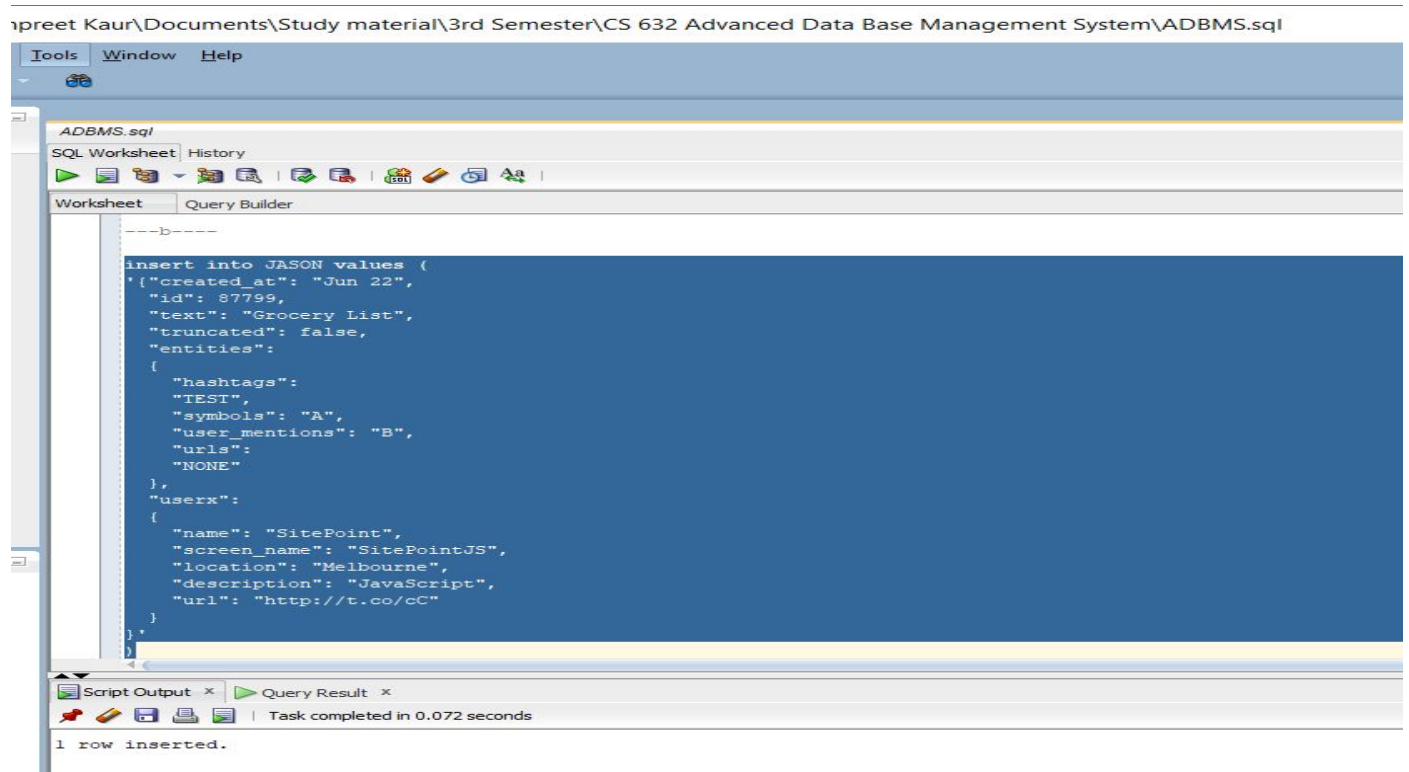
```
{  
  "created_at": "Jun 22",  
  "id": 87799,  
  "text": "Grocery List",  
  "truncated": false,  
  "entities":  
  {  
    "hashtags":  
    "TEST",  
    "symbols": "A",  
    "user_mentions": "B",  
    "urls":  
    "NONE"  
  },  
  "userx":  
  {  
    "name": "SitePoint",  
    "screen_name": "SitePointJS",  
    "location": "Melbourne",  
    "description": "JavaScript",  
    "url": "http://t.co/cC"  
  }  
}
```

Answer 5 b)

insert into JASON values (

```
{ "created_at": "Jun 22",  
  "id": 87799,  
  "text": "Grocery List",  
  "truncated": false,  
  "entities":  
  {  
    "hashtags":  
    "TEST",  
    "symbols": "A",  
    "user_mentions": "B",  
    "urls":  
    "NONE"  
  },  
  "userx":  
  {  
    "name": "SitePoint",  
    "screen_name": "SitePointJS",  
    "location": "Melbourne",  
    "description": "JavaScript",  
    "url": "http://t.co/cC"  
  }  
}
```

Select * from JASON



The screenshot shows the ADBMS SQL Worksheet interface. The menu bar includes Tools, Window, and Help. The toolbar contains icons for running queries, saving, and other functions. The main workspace displays the SQL query `SELECT * FROM JASON`. Below the query, the 'Query Result' tab is active, showing a table with one row of data. The status bar indicates 'All Rows Fetched: 1 in 0.015 seconds'.

IDATA
1 [{"created_at": "Jun 22", "id": 87799, "text": "Grocery List", "truncated": false, "entities": { "hashtags": "TEST", ...

This screenshot shows the same ADBMS SQL Worksheet interface as the first image, but with a 'View Value' dialog box open on the right side. The dialog box displays the JSON data from the query result in a formatted, readable view. The background interface remains the same, showing the query `SELECT * FROM JASON` and the 'Query Result' tab.

View Value

Line Terminator: Platform Default [Change...](#)

Value:

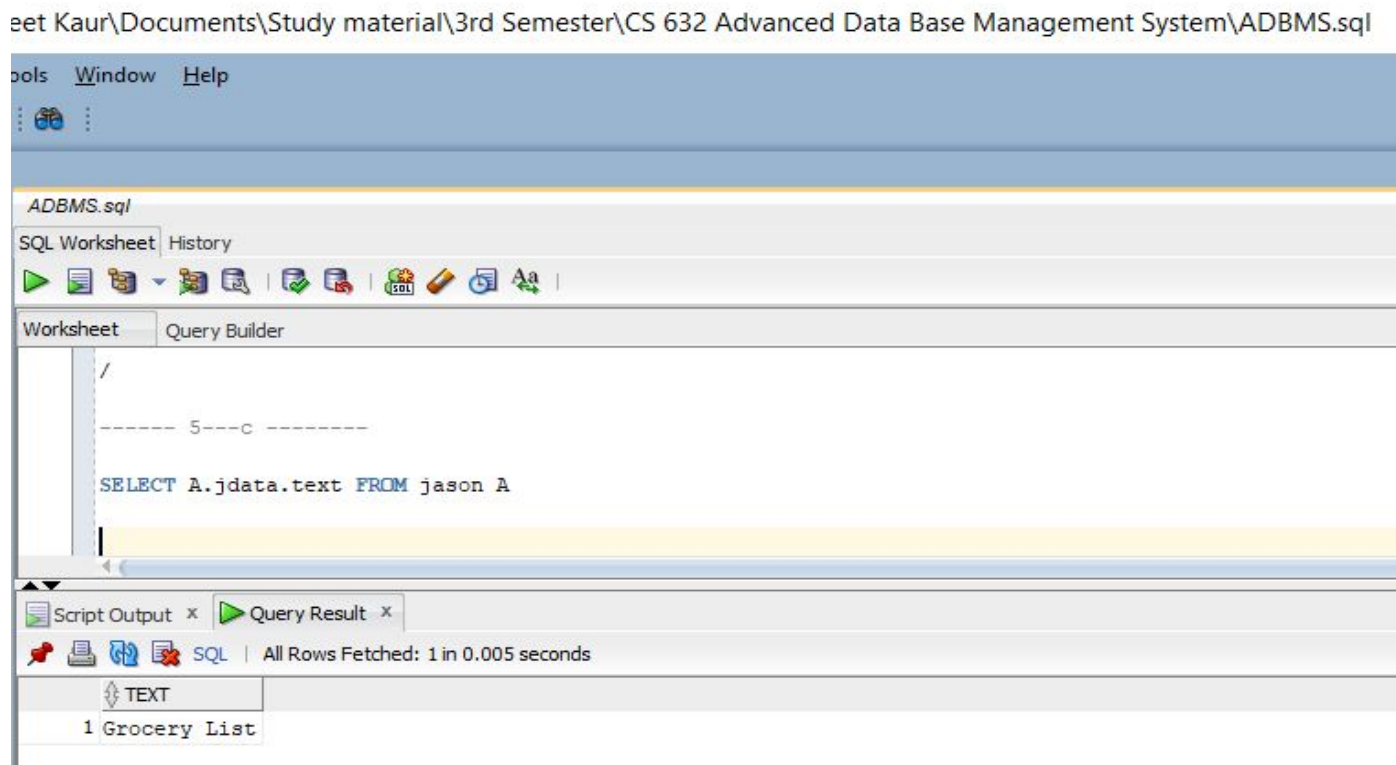
```
{
  "created_at": "Jun 22",
  "id": 87799,
  "text": "Grocery List",
  "truncated": false,
  "entities": {
    "hashtags": "TEST",
    "symbols": "A",
    "user_mentions": "B",
    "urls": "NONE"
  },
  "userx": {
    "name": "SitePoint",
    "screen_name": "SitePointJS",
    "location": "Melbourne",
    "description": "JavaScript",
    "url": "http://t.co/cC"
  }
}
```

Buttons: Help, OK, Cancel

c) Write a select statement that will return ONLY the value of the key “text”.

Answer c)

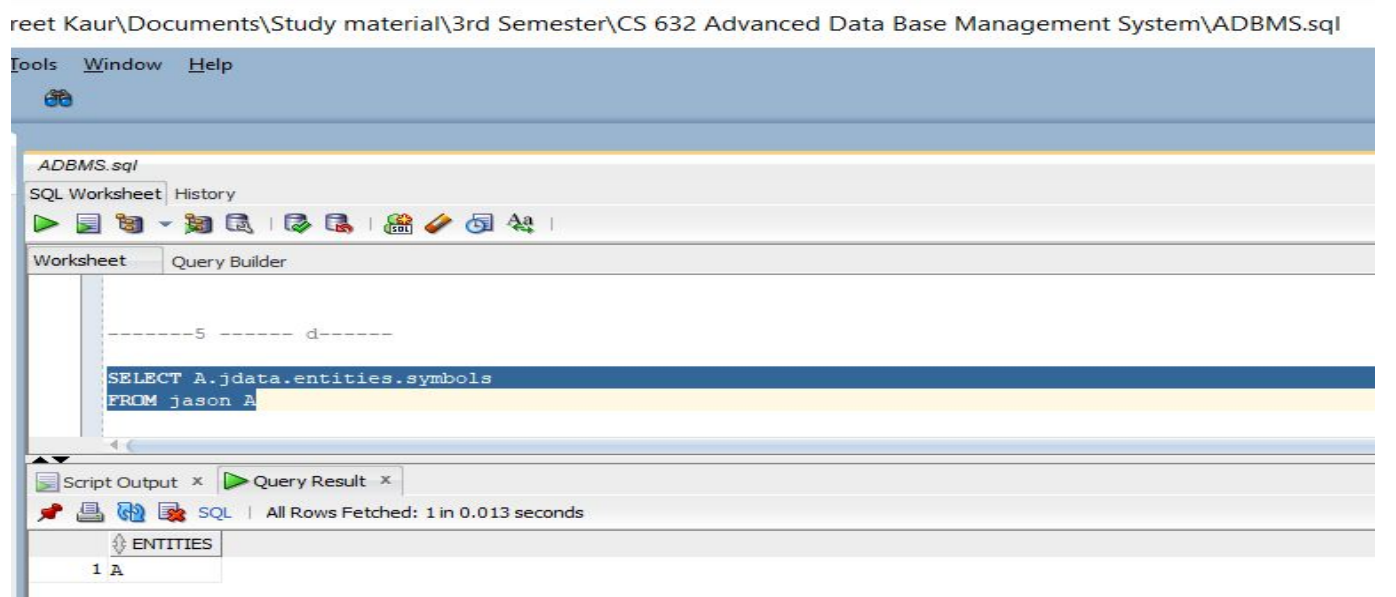
**SELECT A.jdata.text
FROM jason A**



d) Write a select statement that will return ONLY the value of the key “symbols”.

Answer d)

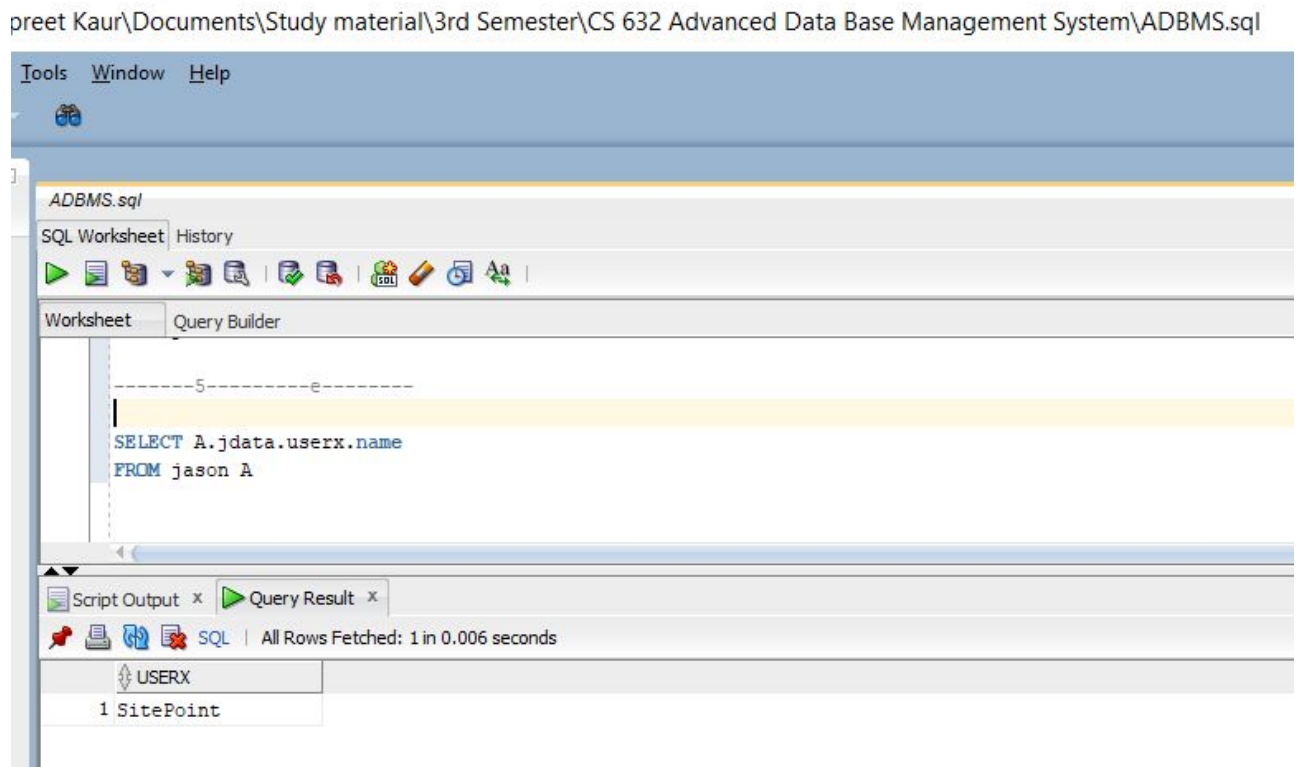
**SELECT A.jdata.entities.symbols
FROM jason A**



e) Write a SELECT statement that will return ONLY the value of the “name” of the “userx”.

Answer e)

**SELECT A.jdata.userx.name
FROM jason A**

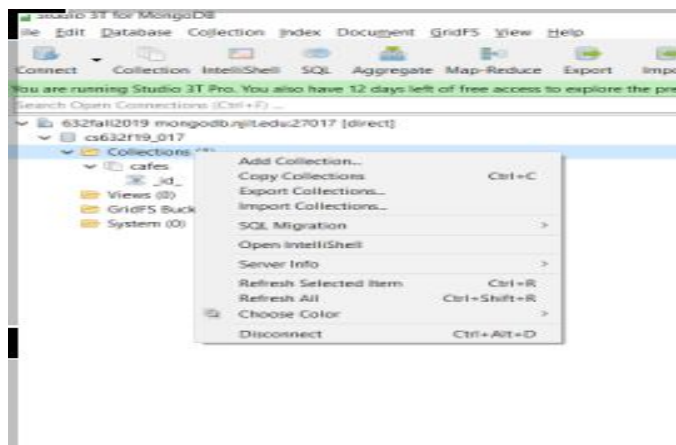


6) SWITCH TO MONGODB NOW.

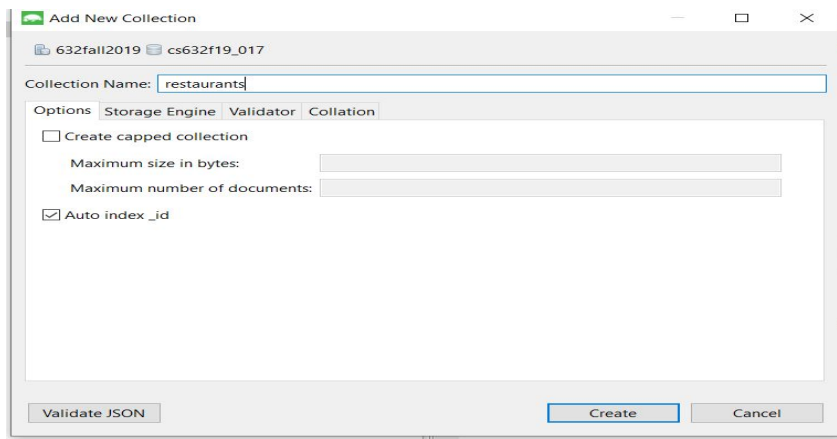
a) Create a new collection called “restaurants”

Answer 6 a)

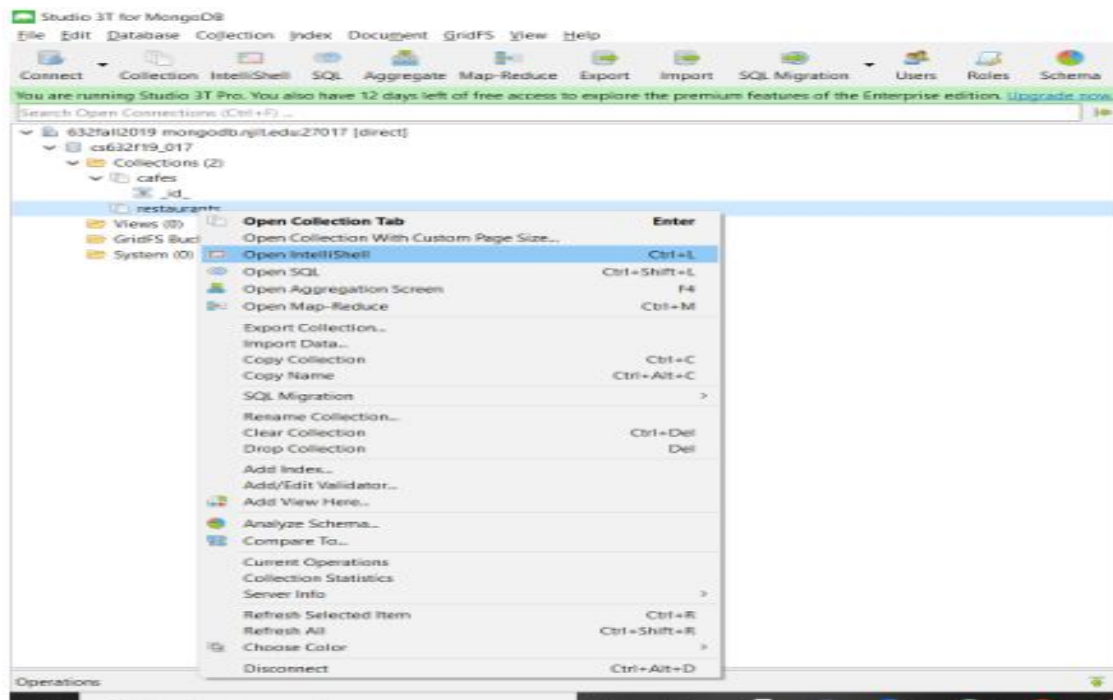
- 1) Right click on Collections.
- 2) Click on Add Collection Type in a name - restaurants



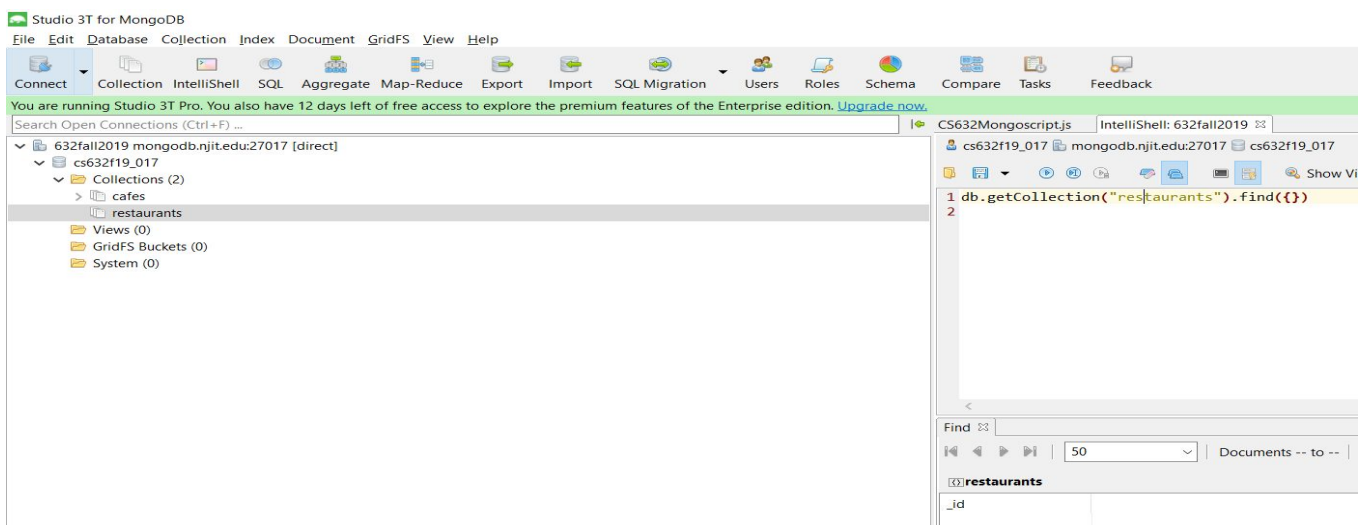
- 3) Click on Create. Right click on restaurants.



4) Click on Open in Intellishell



5) After opening in intellishell. Screenshot attached is attached below.



b) Insert into restaurants four restaurants in a single statement.

We have the name and the cuisine for each:

Name: Trattoria Cuisine: Italian

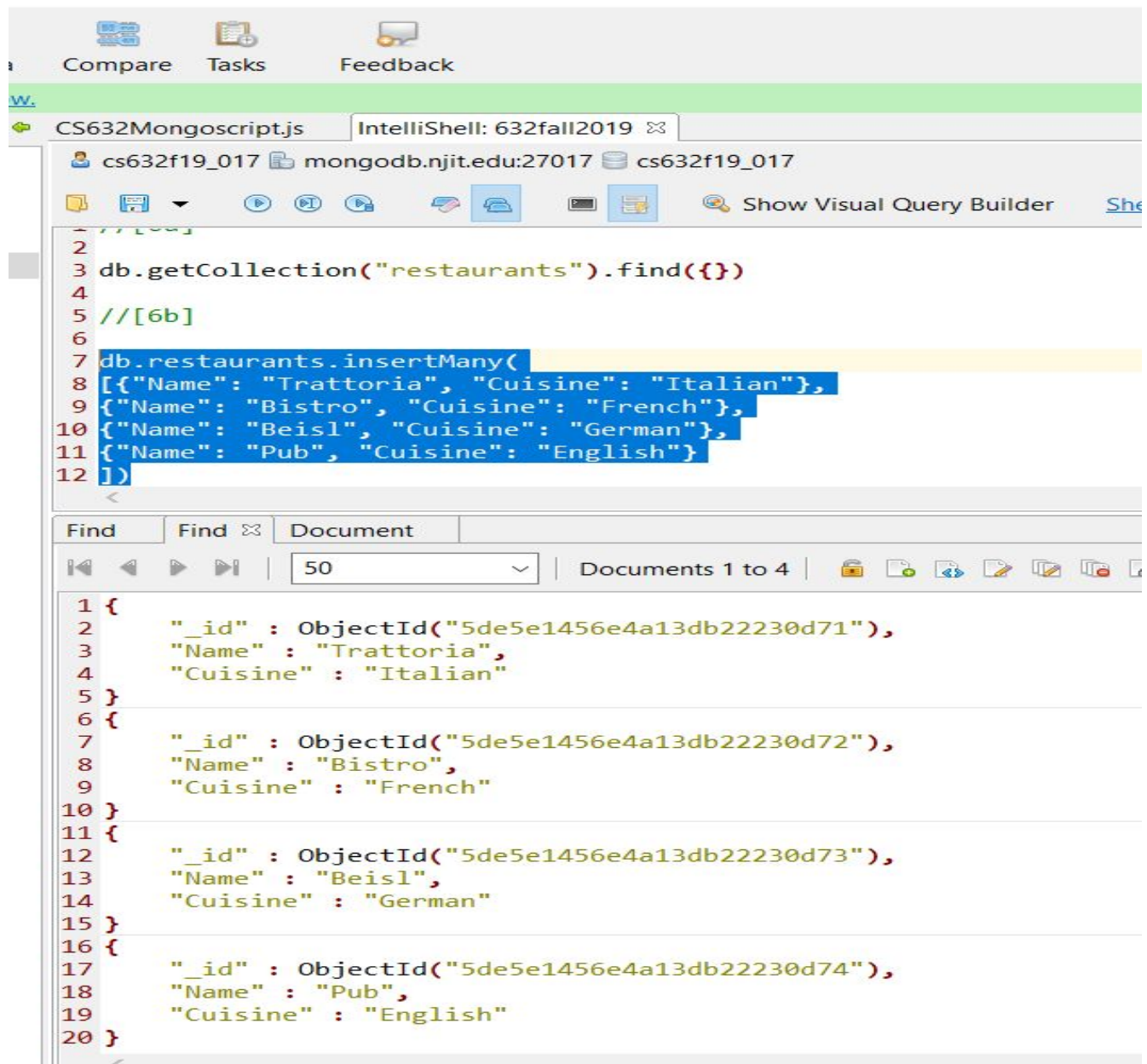
Name: Bistro Cuisine: French

Name: Beisl Cuisine: German

Name: Pub Cuisine: English

Answer b)

```
db.restaurants.insertMany(  
  [{"Name": "Trattoria", "Cuisine": "Italian"},  
  {"Name": "Bistro", "Cuisine": "French"},  
  {"Name": "Beisl", "Cuisine": "German"},  
  {"Name": "Pub", "Cuisine": "English"}  
])
```



The screenshot shows the MongoDB Shell (IntelliShell) interface. The top bar includes 'Compare', 'Tasks', and 'Feedback' buttons. Below the bar, the connection string 'cs632f19_017' and the database 'mongodb.njit.edu:27017' are displayed. The main editor shows the following JavaScript code:

```
1 // [6b]  
2  
3 db.getCollection("restaurants").find({})  
4  
5 // [6b]  
6  
7 db.restaurants.insertMany(  
8   [{"Name": "Trattoria", "Cuisine": "Italian"},  
9   {"Name": "Bistro", "Cuisine": "French"},  
10  {"Name": "Beisl", "Cuisine": "German"},  
11  {"Name": "Pub", "Cuisine": "English"}  
12 ])
```

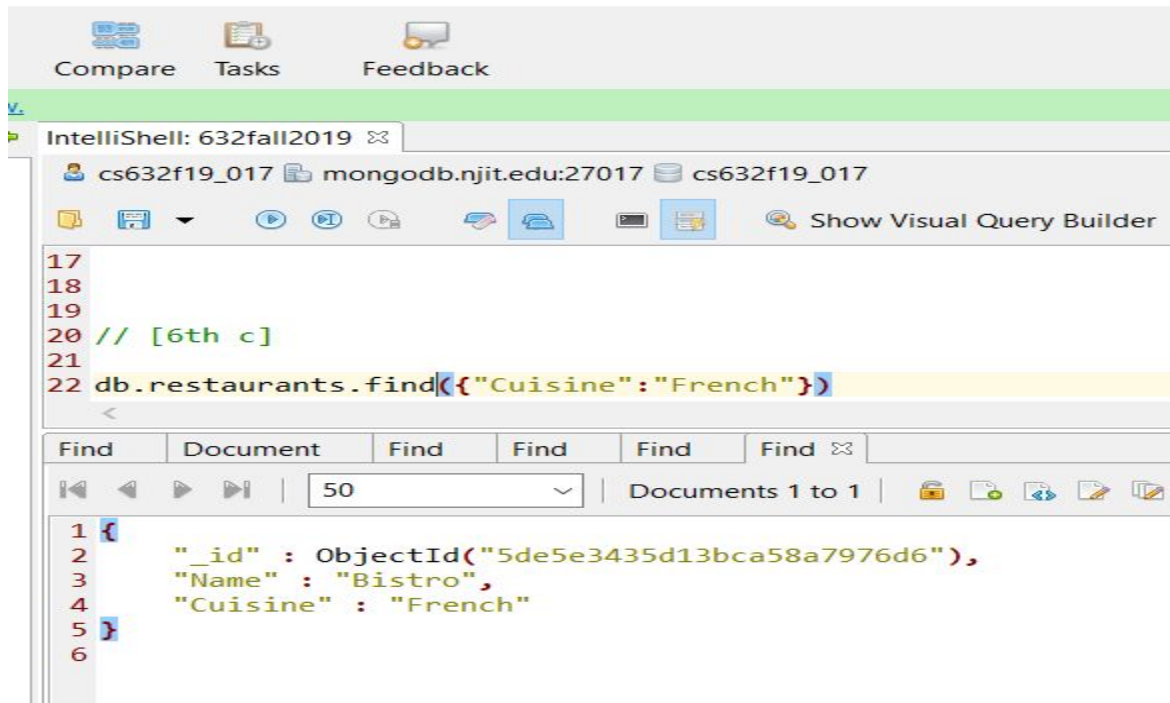
The bottom panel shows the results of the query, displaying four documents in the 'restaurants' collection:

```
1 {  
2   "_id" : ObjectId("5de5e1456e4a13db22230d71"),  
3   "Name" : "Trattoria",  
4   "Cuisine" : "Italian"  
5 }  
6 {  
7   "_id" : ObjectId("5de5e1456e4a13db22230d72"),  
8   "Name" : "Bistro",  
9   "Cuisine" : "French"  
10 }  
11 {  
12   "_id" : ObjectId("5de5e1456e4a13db22230d73"),  
13   "Name" : "Beisl",  
14   "Cuisine" : "German"  
15 }  
16 {  
17   "_id" : ObjectId("5de5e1456e4a13db22230d74"),  
18   "Name" : "Pub",  
19   "Cuisine" : "English"  
20 }
```


c) Write a Mongo Query that finds (all) restaurants that offer French Cuisine.

Answer c)

```
db.restaurants.find({"Cuisine":"French"})
```



The screenshot shows the MongoDB IntelliShell interface. The top bar includes 'Compare', 'Tasks', and 'Feedback' buttons. Below the toolbar, the connection details are 'cs632f19_017' and 'mongodb.njit.edu:27017'. The main editor area contains the following code:

```
17
18
19
20 // [6th c]
21
22 db.restaurants.find({"Cuisine":"French"})
```

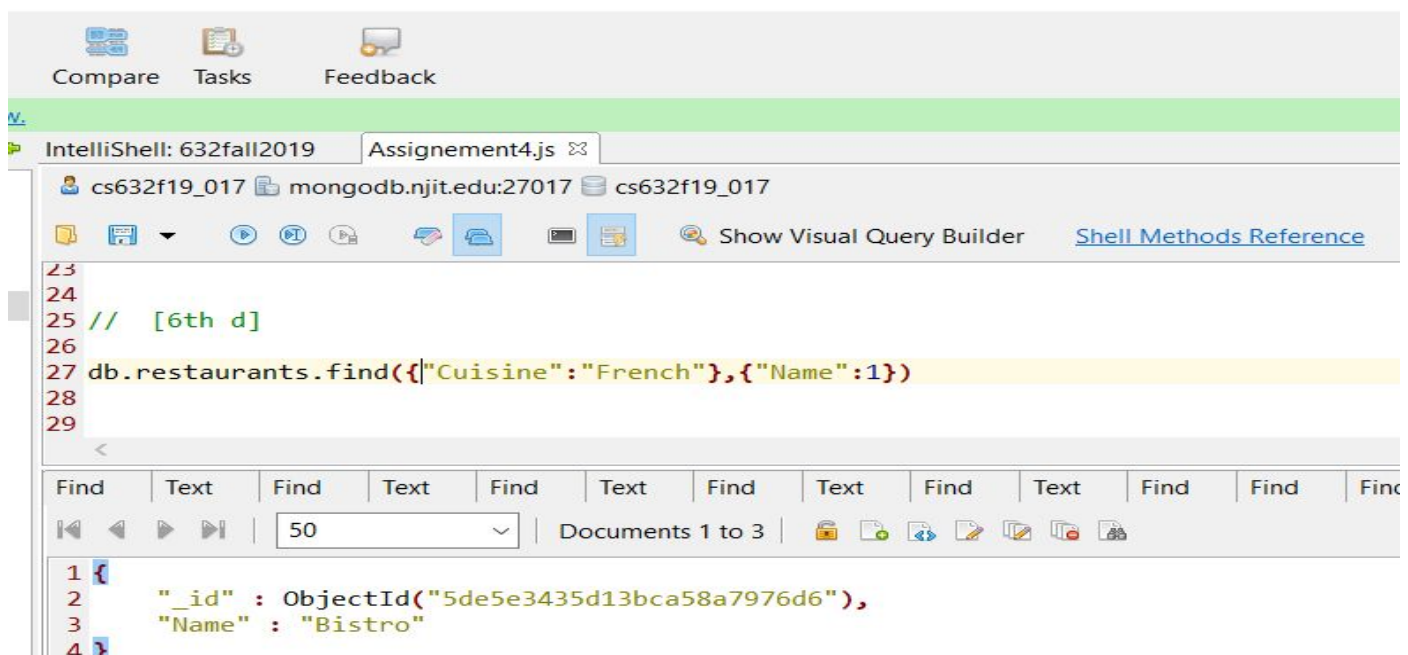
The bottom panel shows the results of the query, displaying a single document:

```
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
3   "Name" : "Bistro",
4   "Cuisine" : "French"
5 }
6
```

d) Redo question c) so that it does NOT show the key value pair of Cuisine. Only the Name and _id.

Answer d)

```
db.restaurants.find({"Cuisine":"French"},{"Name":1})
```



The screenshot shows the MongoDB IntelliShell interface. The top bar includes 'Compare', 'Tasks', and 'Feedback' buttons. Below the toolbar, the connection details are 'cs632f19_017' and 'mongodb.njit.edu:27017'. The main editor area contains the following code:

```
23
24
25 // [6th d]
26
27 db.restaurants.find({"Cuisine":"French"},{"Name":1})
28
29
```

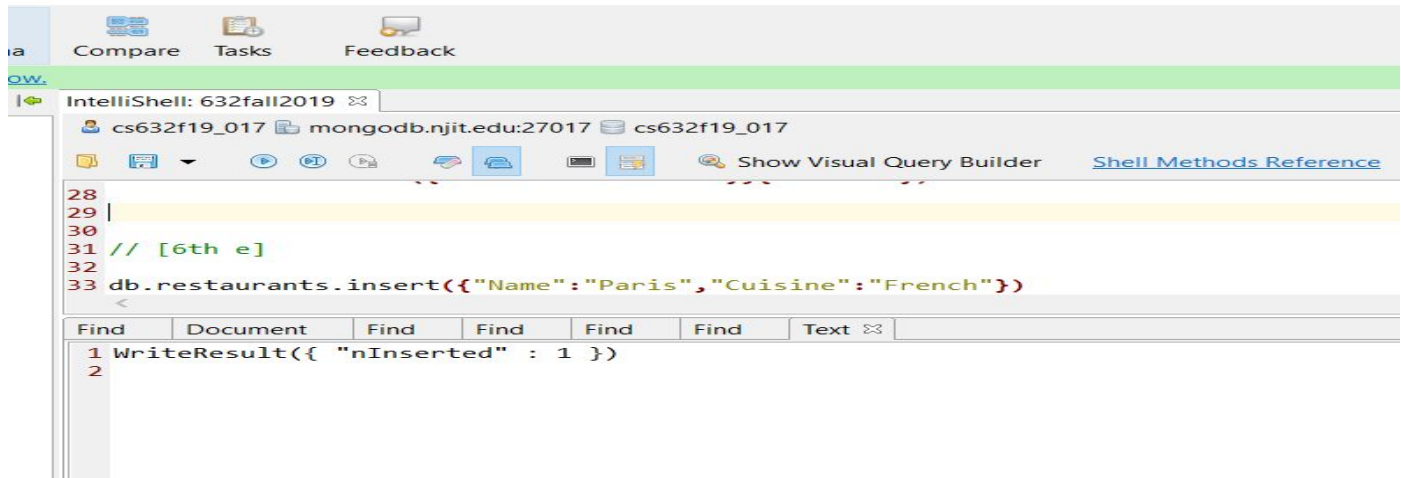
The bottom panel shows the results of the query, displaying a single document:

```
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
3   "Name" : "Bistro"
4 }
```

e) Insert an additional French cuisine restaurant called Paris.

Answer e)

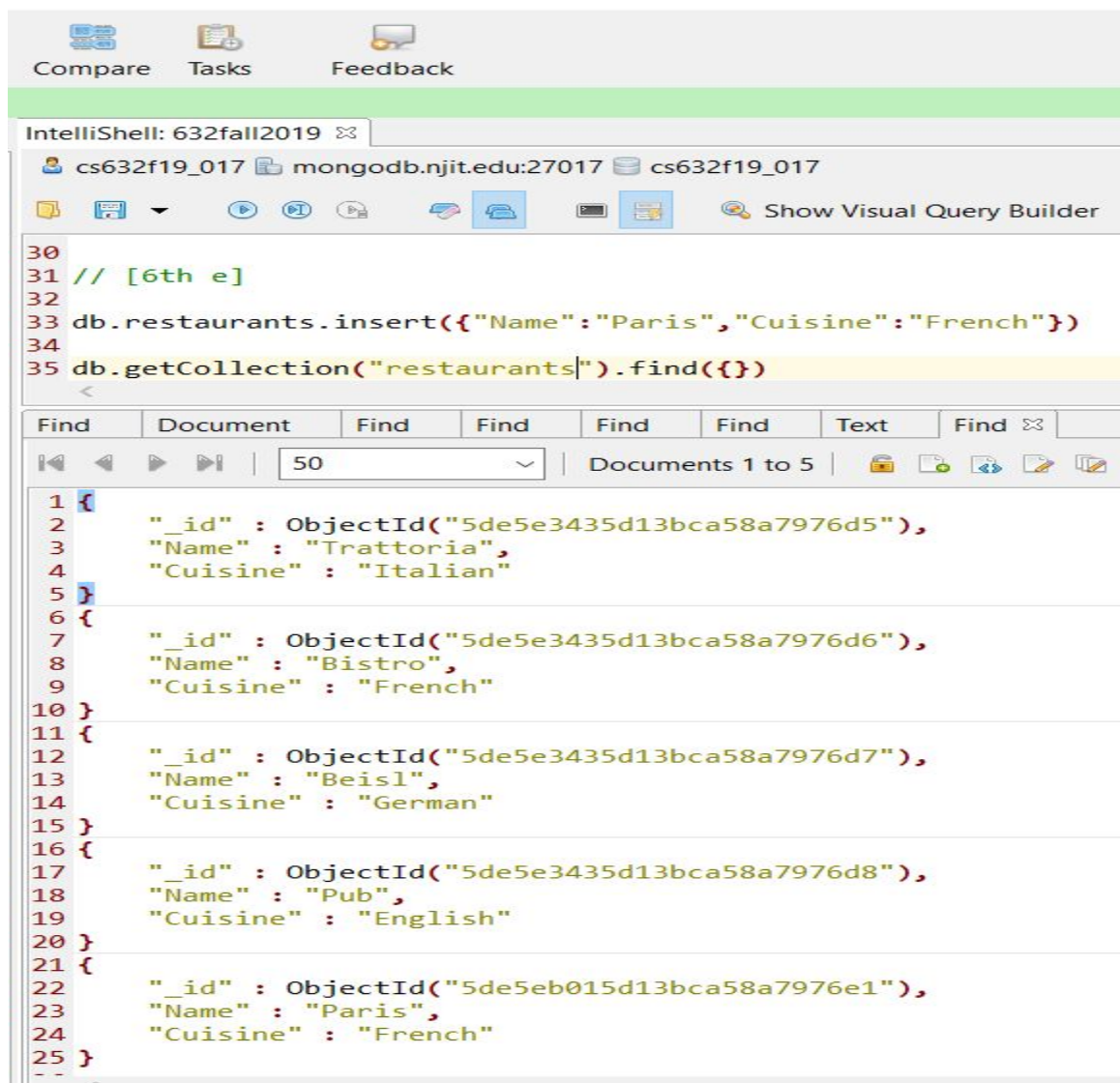
db.restaurants.insert({"Name":"Paris","Cuisine":"French"})



The screenshot shows the MongoDB Shell interface. The command `db.restaurants.insert({"Name":"Paris","Cuisine":"French"})` has been executed. The output shows a `WriteResult` object with `"nInserted" : 1`, indicating that one document was successfully inserted into the `restaurants` collection.

```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017

28
29
30
31 // [6th e]
32
33 db.restaurants.insert({"Name":"Paris","Cuisine":"French"})
34
35
Find Document Find Find Find Find Text
1 WriteResult({ "nInserted" : 1 })
2
```



The screenshot shows the MongoDB Shell interface with the `find` command executed on the `restaurants` collection. The output displays five documents, including the newly inserted one. The documents are:

- `{ "_id" : ObjectId("5de5e3435d13bca58a7976d5"), "Name" : "Trattoria", "Cuisine" : "Italian" }`
- `{ "_id" : ObjectId("5de5e3435d13bca58a7976d6"), "Name" : "Bistro", "Cuisine" : "French" }`
- `{ "_id" : ObjectId("5de5e3435d13bca58a7976d7"), "Name" : "Beisl", "Cuisine" : "German" }`
- `{ "_id" : ObjectId("5de5e3435d13bca58a7976d8"), "Name" : "Pub", "Cuisine" : "English" }`
- `{ "_id" : ObjectId("5de5eb015d13bca58a7976e1"), "Name" : "Paris", "Cuisine" : "French" }`

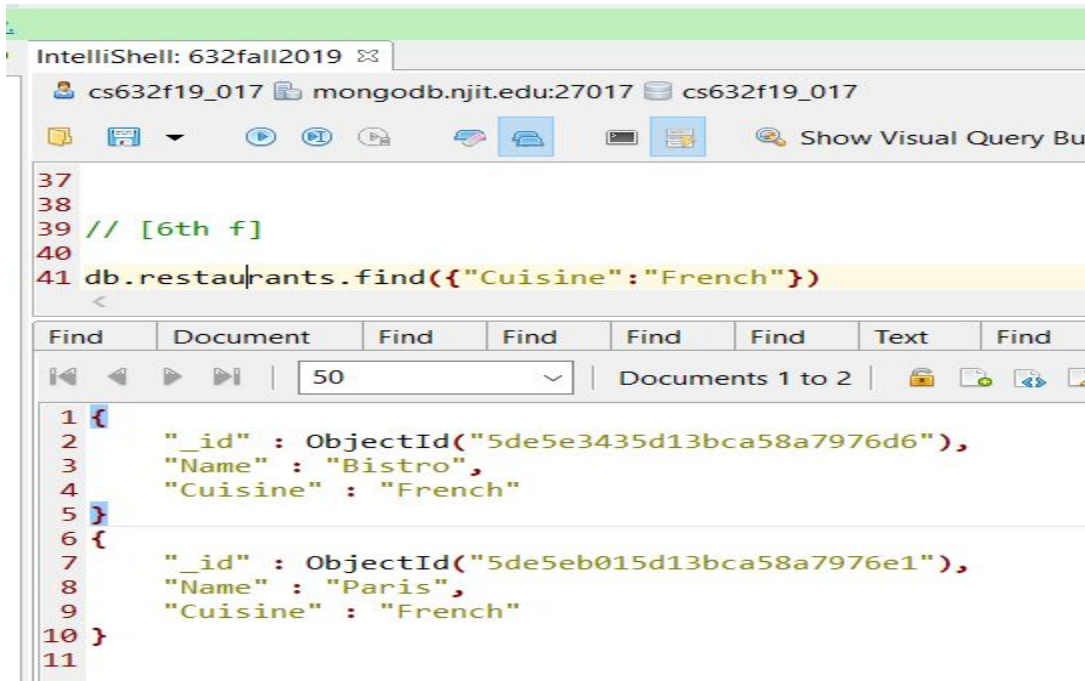
```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017

30
31 // [6th e]
32
33 db.restaurants.insert({"Name":"Paris","Cuisine":"French"})
34
35 db.getCollection("restaurants").find({})
36
Find Document Find Find Find Find Text Find
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
3   "Name" : "Trattoria",
4   "Cuisine" : "Italian"
5 }
6 {
7   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
8   "Name" : "Bistro",
9   "Cuisine" : "French"
10 }
11 {
12   "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
13   "Name" : "Beisl",
14   "Cuisine" : "German"
15 }
16 {
17   "_id" : ObjectId("5de5e3435d13bca58a7976d8"),
18   "Name" : "Pub",
19   "Cuisine" : "English"
20 }
21 {
22   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
23   "Name" : "Paris",
24   "Cuisine" : "French"
25 }
--
```

f) Redo question c)

Answer f)

db.restaurants.find({"Cuisine":"French"})



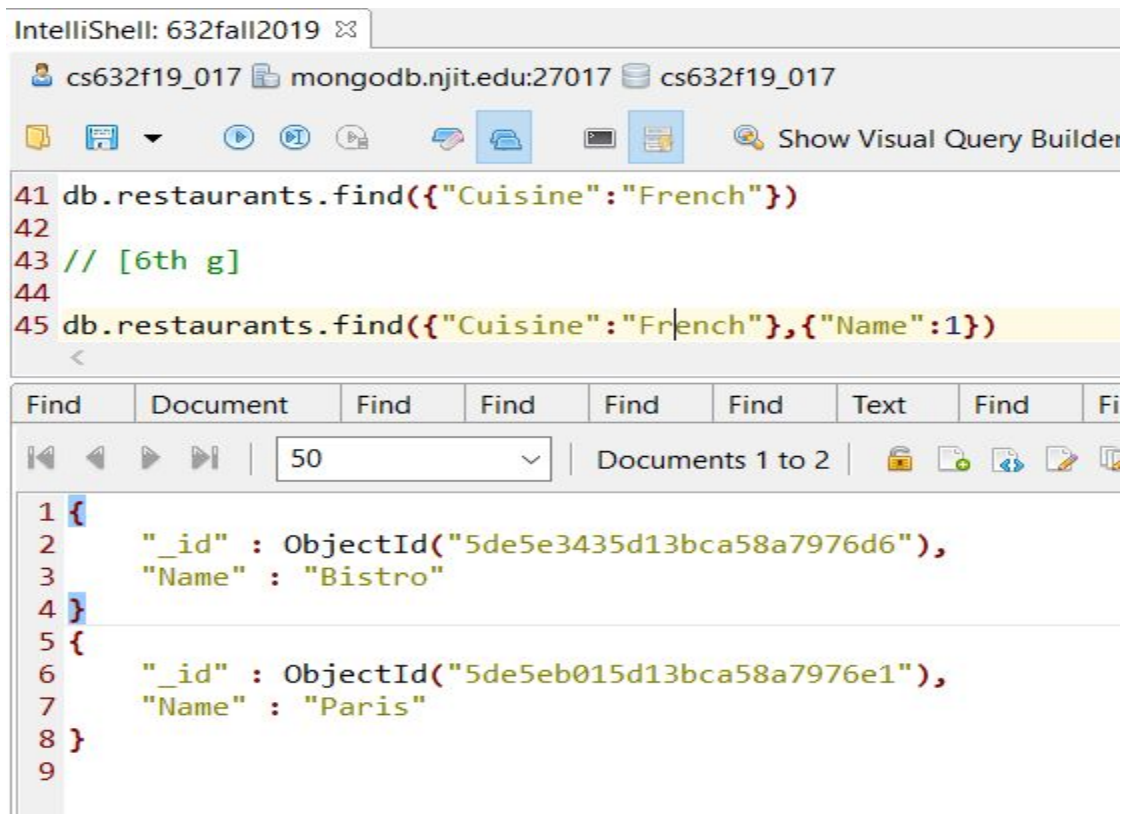
The screenshot shows the IntelliJShell interface for a MongoDB database named 'cs632f19_017'. The query `db.restaurants.find({"Cuisine":"French"})` is entered in the command line. The results pane displays two documents:

```
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
3   "Name" : "Bistro",
4   "Cuisine" : "French"
5 }
6 {
7   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
8   "Name" : "Paris",
9   "Cuisine" : "French"
10 }
11
```

g) Redo question d)

Answer g)

db.restaurants.find({"Cuisine":"French"},"Name":1))



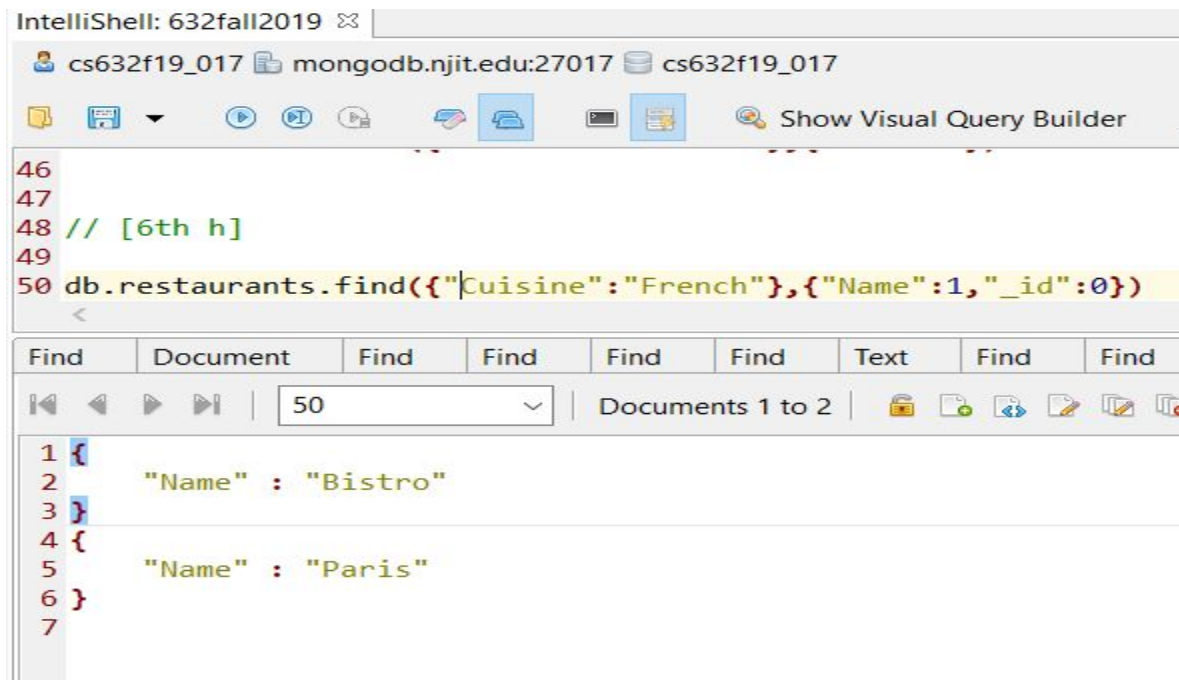
The screenshot shows the IntelliJShell interface for the same MongoDB database. The query `db.restaurants.find({"Cuisine":"French"},"Name":1))` is entered. The results pane displays the same two documents as in the previous screenshot, but only the 'Name' field is highlighted in each document:

```
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
3   "Name" : "Bistro"
4 }
5 {
6   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
7   "Name" : "Paris"
8 }
9
```

h) Redo question d) so that it ALSO does not show the JSON _id.

Answer h)

db.restaurants.find({"Cuisine":"French"}, {"Name":1, "_id":0})



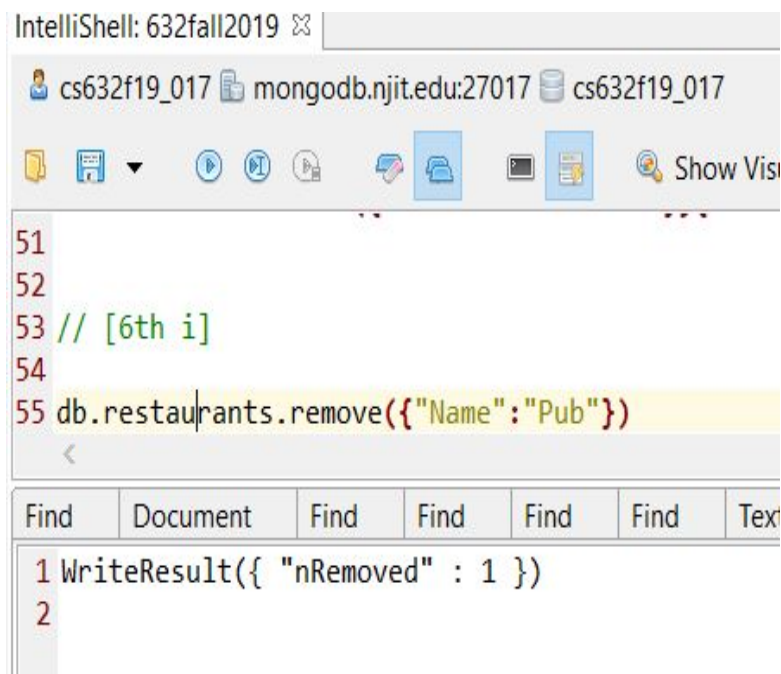
The screenshot shows the IntelliJShell interface for a MongoDB instance. The command prompt shows the connection details: `cs632f19_017`, `mongodb.njit.edu:27017`, and `cs632f19_017`. The command bar contains the query: `db.restaurants.find({"Cuisine":"French"}, {"Name":1, "_id":0})`. The results pane shows two documents:

```
1 {
2   "Name" : "Bistro"
3 }
4 {
5   "Name" : "Paris"
6 }
7
```

i) Remove the pub completely and show all restaurants after you do that.

Answer i)

db.restaurants.remove({"Name":"Pub"})



The screenshot shows the IntelliJShell interface for a MongoDB instance. The command prompt shows the connection details: `cs632f19_017`, `mongodb.njit.edu:27017`, and `cs632f19_017`. The command bar contains the query: `db.restaurants.remove({"Name":"Pub"})`. The results pane shows the result of the removal operation:

```
1 WriteResult({ "nRemoved" : 1 })
2
```

All the restaurants after removing Pub is :

db.getCollection("restaurants").find({})

j) By clicking on Count Documents on Studio3T prove that restaurants now contains only 4 restaurants.

Answer j)

After clicking count Documents :

The screenshot displays the Studio3T interface. At the top, there are tabs for 'Compare', 'Tasks', and 'Feedback'. Below these is a green bar with a '(don't sh)' link. The main window shows the 'IntelliShell: 632fall2019' tab with a connection to 'cs632f19_017' on 'mongodb.njit.edu:27017'. The query bar contains the following commands:

```
54 db.restaurants.remove({"Name":"Pub"})
55 db.getCollection("restaurants").find({})
56
57
58
```

The results pane shows 4 documents in JSON format:

```
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
3   "Name" : "Trattoria",
4   "Cuisine" : "Italian"
5 }
6 {
7   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
8   "Name" : "Bistro",
9   "Cuisine" : "French"
10 }
11 {
12   "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
13   "Name" : "Beisl",
14   "Cuisine" : "German"
15 }
16 {
17   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
18   "Name" : "Paris",
19   "Cuisine" : "French"
20 }
21
```

At the bottom right, a box indicates '4 documents'.


```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017

58 // [6th k]
59
60 db.restaurants.update({"Name":"Paris"},{$set:{"Name":"LaFrance"}})
61 db.getCollection("restaurants").find({})
62

Find Find Text Find Find Find Find Find Find Find Find
50 Documents 1 to 4

1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
3   "Name" : "Trattoria",
4   "Cuisine" : "Italian"
5 }
6 {
7   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
8   "Name" : "Bistro",
9   "Cuisine" : "French"
10 }
11 {
12   "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
13   "Name" : "Beisl",
14   "Cuisine" : "German"
15 }
16 {
17   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
18   "Name" : "LaFrance",
19   "Cuisine" : "French"
20 }
21
```

l) Update the cuisine of LaFrance to be now "French and Italian" (as one single value). Then show all the restaurants.

Answer l)

db.restaurants.update({"Name":"LaFrance"},{\$set:{"Cuisine":"French and Italian"}})

```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017
Show Visual Query Builder Shell Methods Reference

63
64 // [6th L]
65
66 db.restaurants.update({"Name":"LaFrance"},{$set:{"Cuisine":"French and Italian"}})
67 db.getCollection("restaurants").find({})

Find Text
1 WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
2
```

db.getCollection("restaurants").find({})

```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017
Show Visual Query Builder Shell Methods Reference

63
64 // [6th L]
65
66 db.restaurants.update({"Name":"LaFrance"},{$set:{"Cuisine":"French and Italian"}})
67 db.getCollection("restaurants").find({})

Find Text Find
50 Documents 1 to 4

1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
3   "Name" : "Trattoria",
4   "Cuisine" : "Italian"
5 }
6 {
7   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
8   "Name" : "Bistro",
9   "Cuisine" : "French"
10 }
11 {
12   "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
13   "Name" : "Beisl",
14   "Cuisine" : "German"
15 }
16 {
17   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
18   "Name" : "LaFrance",
19   "Cuisine" : "French and Italian"
20 }
```

m) Add a new key/value pair to Trattoria where the key is Drinks and the value is the array that contains wine, coffee and water. Show all restaurants.

Answer m)

db.restaurants.update({"Name":"Trattoria"},{\$set:{"Drinks":["Wine","Coffee","Water"]}})

```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017
Show Visual Query Builder Shell Methods Reference

68
69
70 // [6th m ]
71
72 db.restaurants.update({"Name":"Trattoria"},{$set:{"Drinks":["Wine","Coffee","Water"]}})
<

Find Text Find Text
1 WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
2
```

db.getCollection("restaurants").find({})

```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017
Show Visual Query Builder Shell Methods Reference

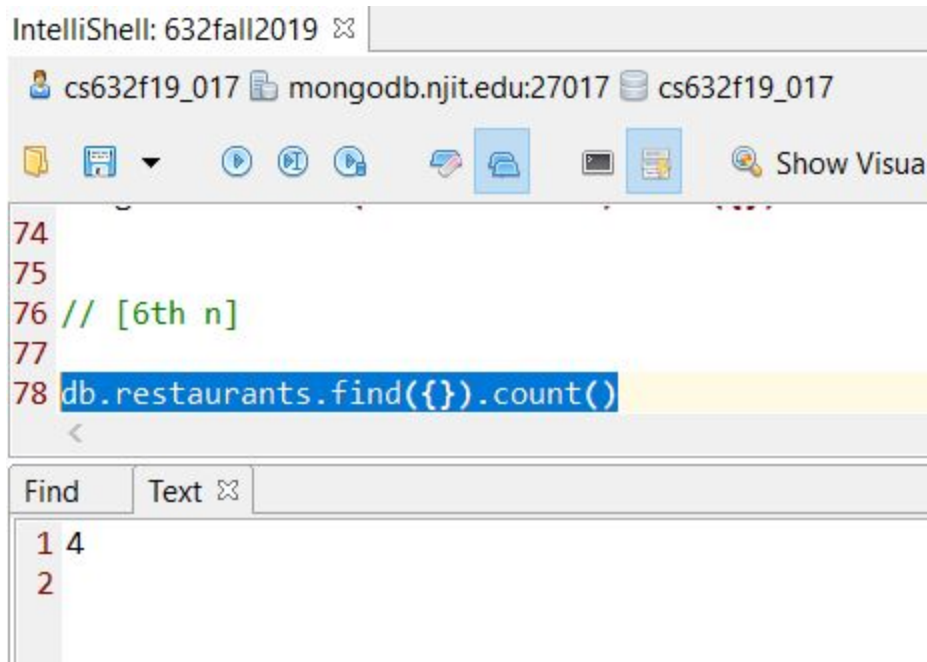
69
70 // [6th m ]
71
72 db.restaurants.update({"Name":"Trattoria"},{$set:{"Drinks":["Wine","Coffee","Water"]}})
73 db.getCollection("restaurants").find({})
<

Find Text Find Text Find
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
3   "Name" : "Trattoria",
4   "Cuisine" : "Italian",
5   "Drinks" : [
6     "Wine",
7     "Coffee",
8     "Water"
9   ]
10 }
11 {
12   "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
13   "Name" : "Bistro",
14   "Cuisine" : "French"
15 }
16 {
17   "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
18   "Name" : "Beisl",
19   "Cuisine" : "German"
20 }
21 {
22   "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
23   "Name" : "LaFrance",
24   "Cuisine" : "French and Italian"
25 }
```


n) Use the count() aggregate function to show that there are still only 4 restaurants.

Answer n)

db.restaurants.find({}).count()



IntelliShell: 632fall2019

cs632f19_017 mongodb.njit.edu:27017 cs632f19_017

```
74
75
76 // [6th n]
77
78 db.restaurants.find({}).count()
```

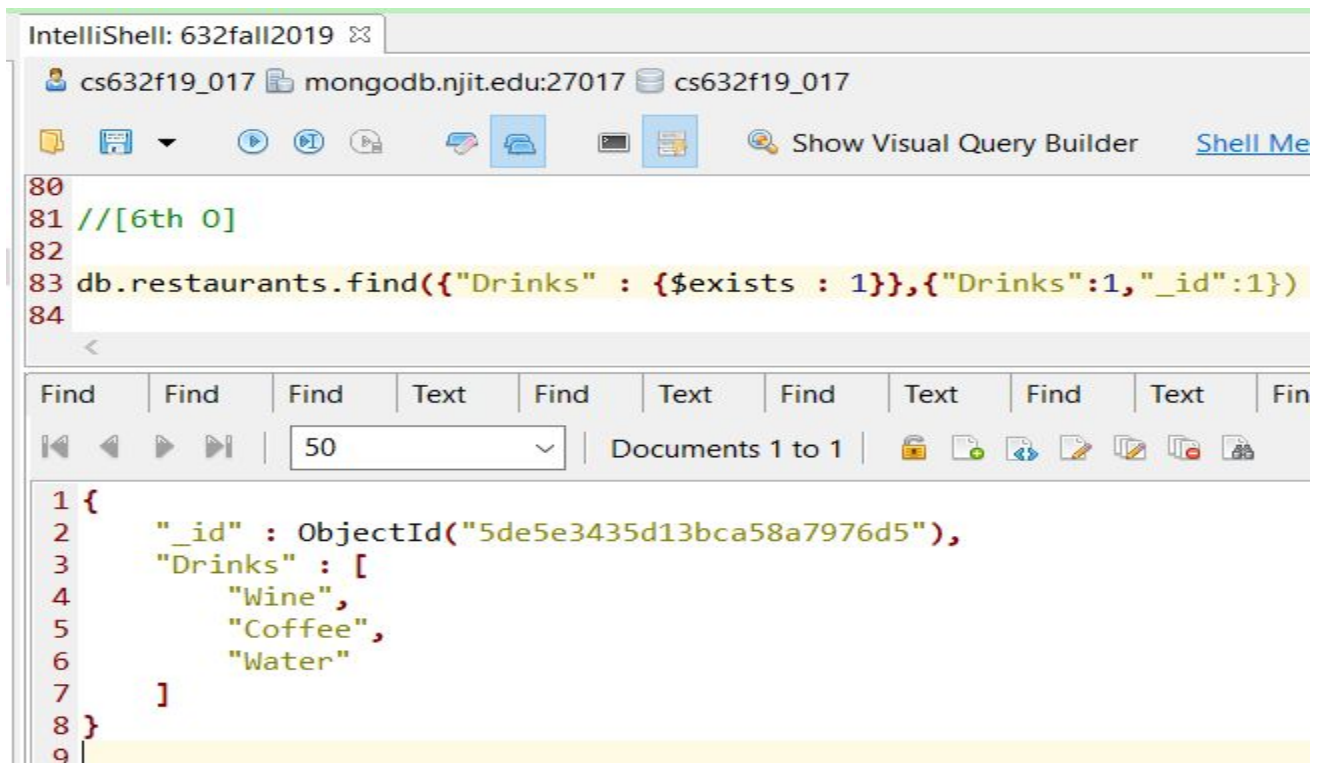
Find Text

```
1 4
2
```

o) For all and only the restaurants for which there are drinks, show only the drinks (and the _id). HINT: Use \$exists.

Answer o)

db.restaurants.find({"Drinks" : {\$exists : 1}}, {"Drinks":1, "_id":1})



IntelliShell: 632fall2019

cs632f19_017 mongodb.njit.edu:27017 cs632f19_017

```
80
81 //[6th o]
82
83 db.restaurants.find({"Drinks" : {$exists : 1}}, {"Drinks":1, "_id":1})
84
```

Find Find Find Text Find Text Find Text Find Text Find

50 Documents 1 to 1

```
1 {
2   "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
3   "Drinks" : [
4     "Wine",
5     "Coffee",
6     "Water"
7   ]
8 }
9
```

