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",
  },
}
```

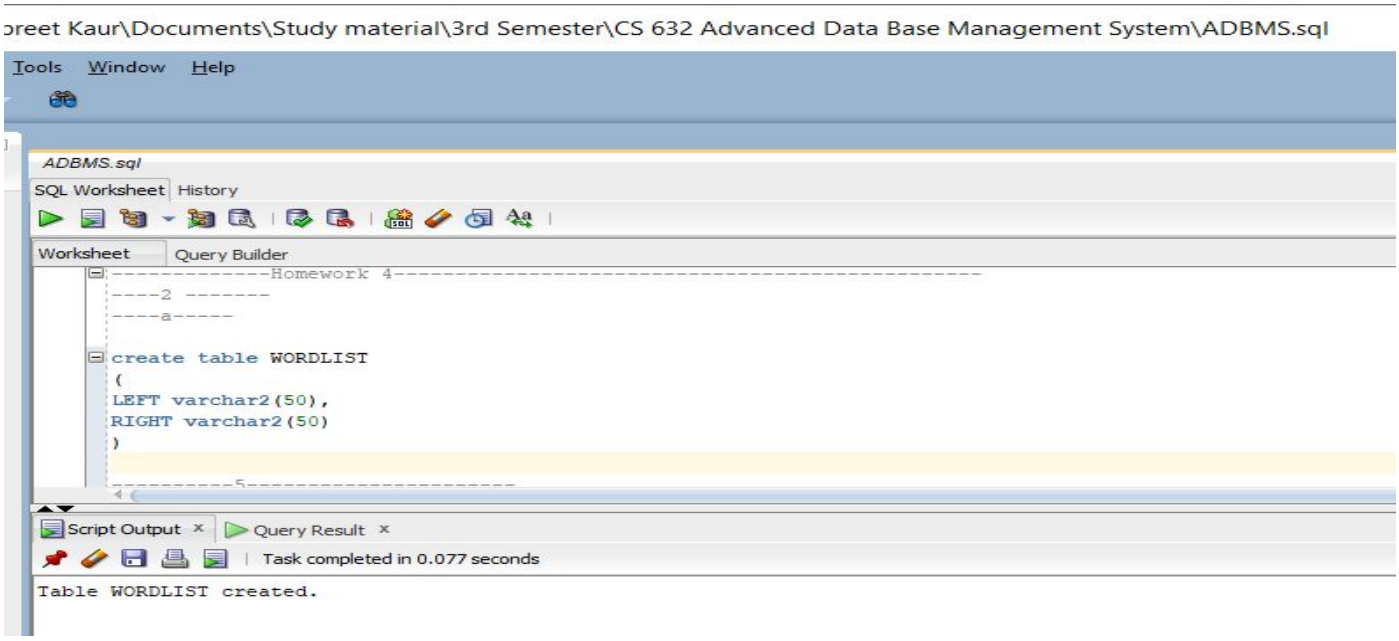

Answer 1)

JSON structure is in the another page

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

Answer 2a)

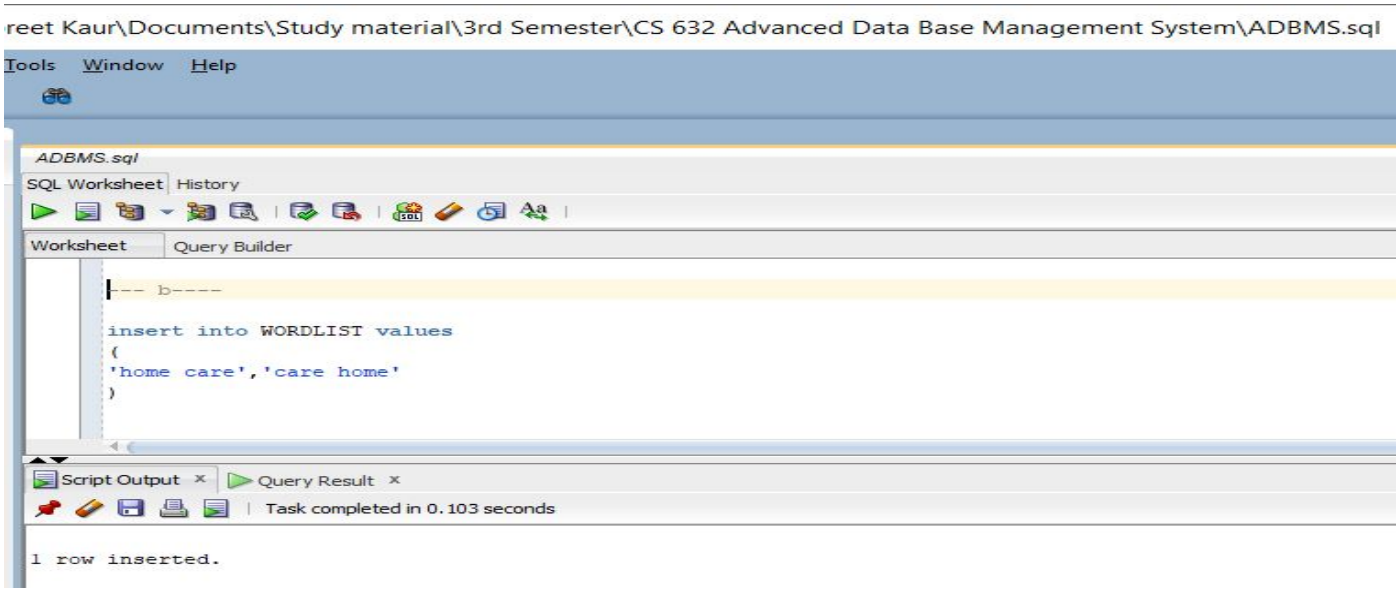
create table WORDLIST
(
LEFT varchar2(50),
RIGHT varchar2(50)
)



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)

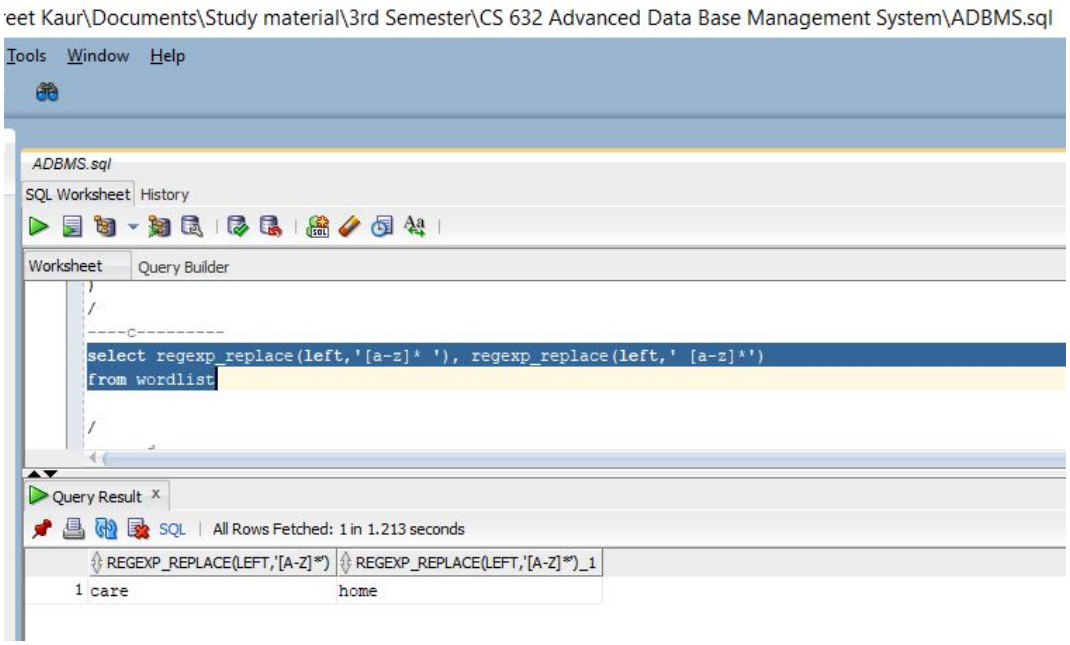
Insert Into Wordlist Values(
'home care','care home'
)



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)

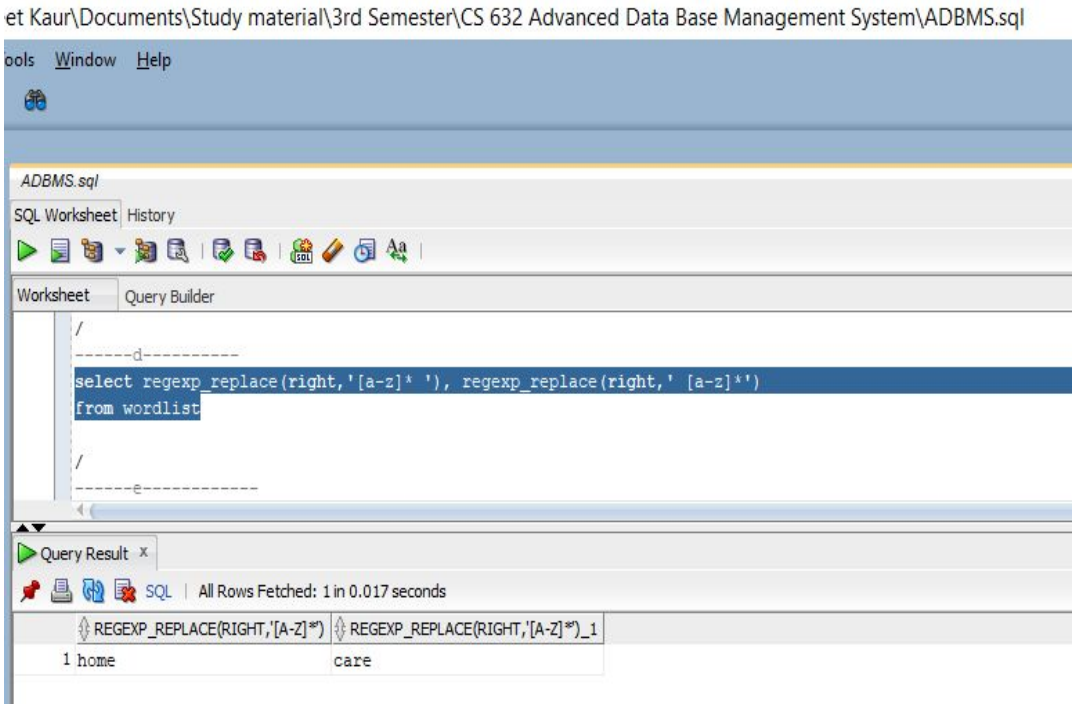
select regexp_replace(left,'[a-z]* '), regexp_replace(left,' [a-z]*')
from wordlist



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

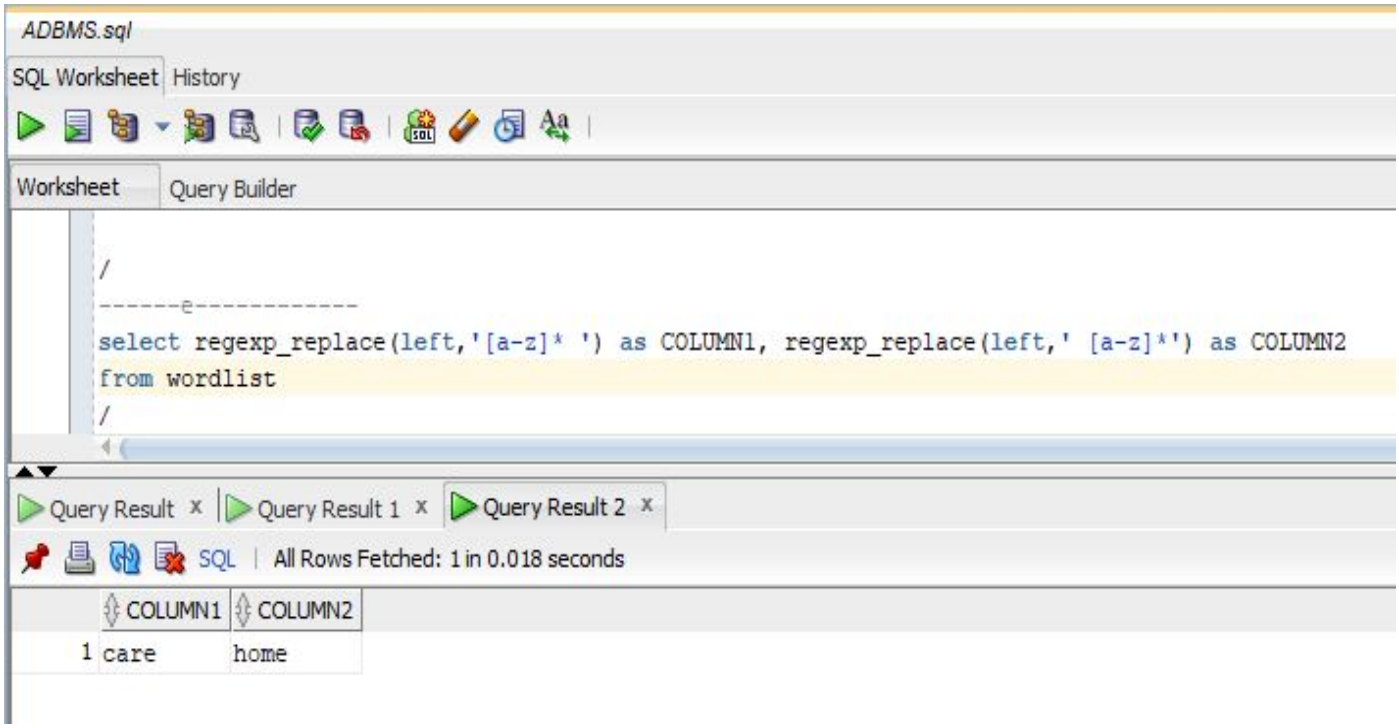


e) Assign column aliases to all four regular expressions. (Use: Column1, Column2, Column3, Column4). I showed this already above. Redo c) and d) to show this.

Answer e)

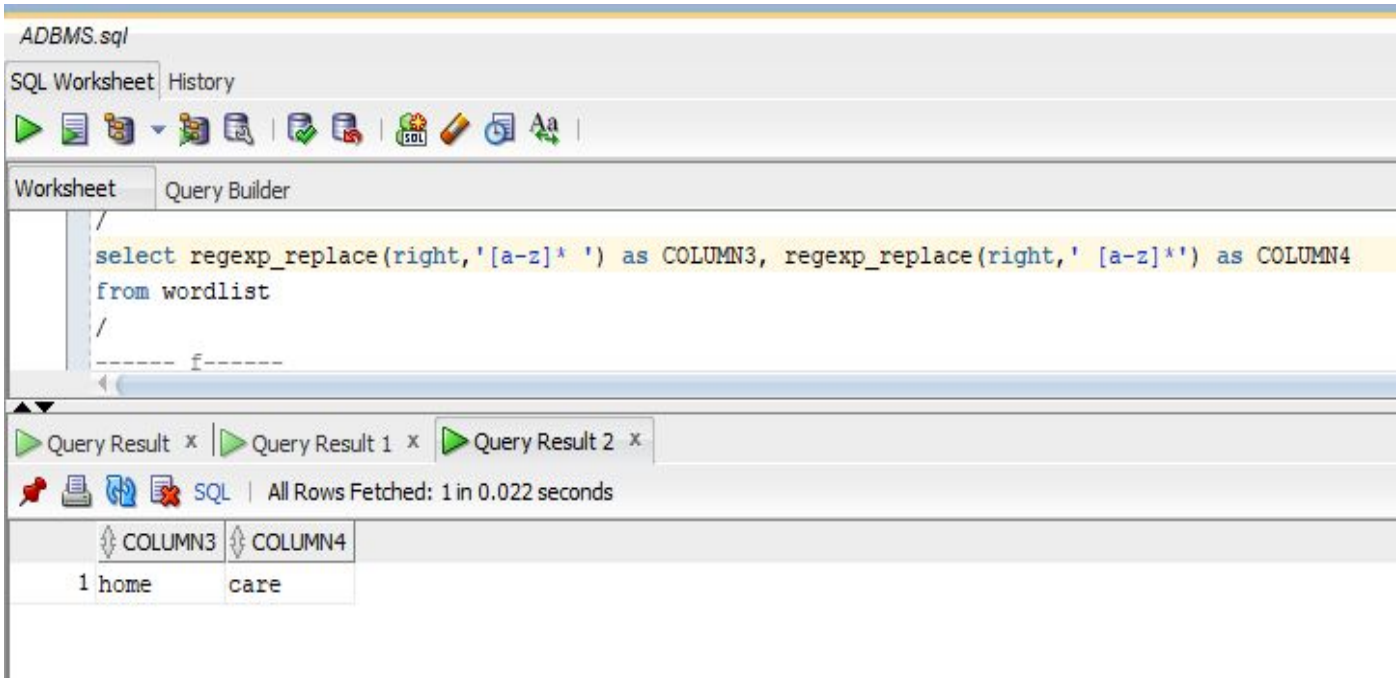
select regexp_replace(left,'[a-z]* ') as COLUMN1, regexp_replace(left,' [a-z]*') as COLUMN2 from wordlist

/



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

/

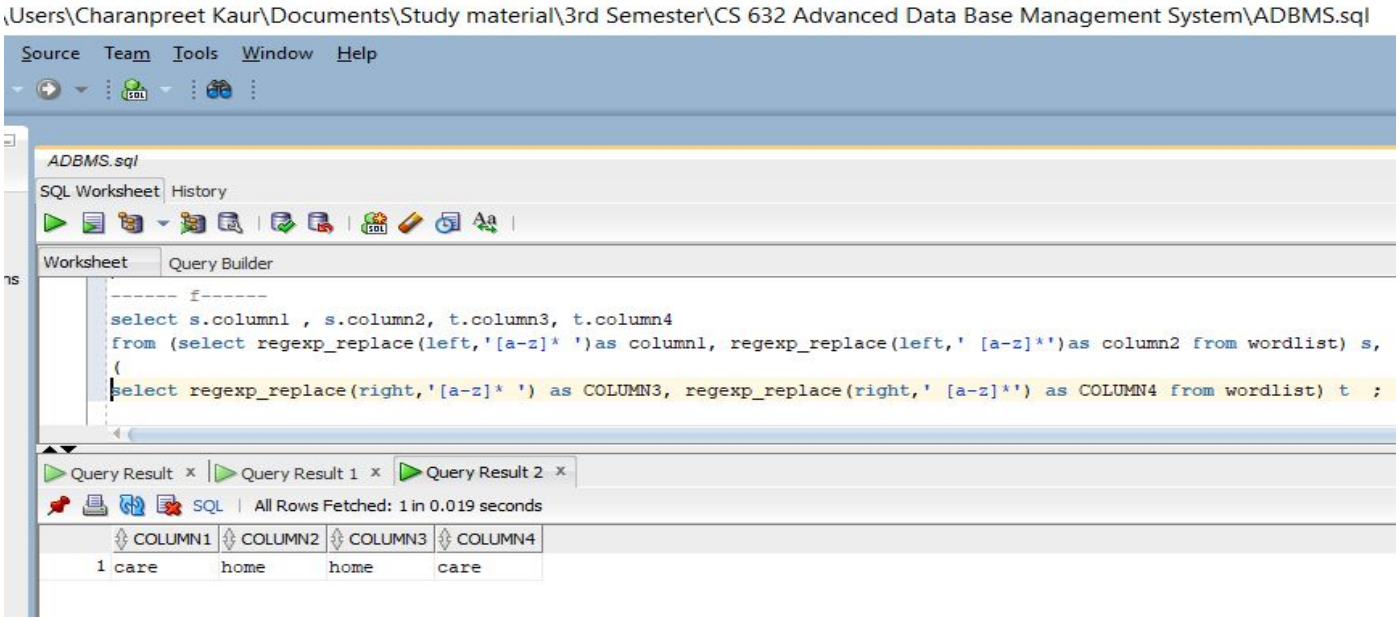


f) Now write a join operation (reminder: don't use the word join) between the two previous select statements, by NESTING them inside of another SELECT with no WHERE clause.
This should give you something like this:

Column1	Column2	Column3	Column4
care	home	home	care

Answer f)

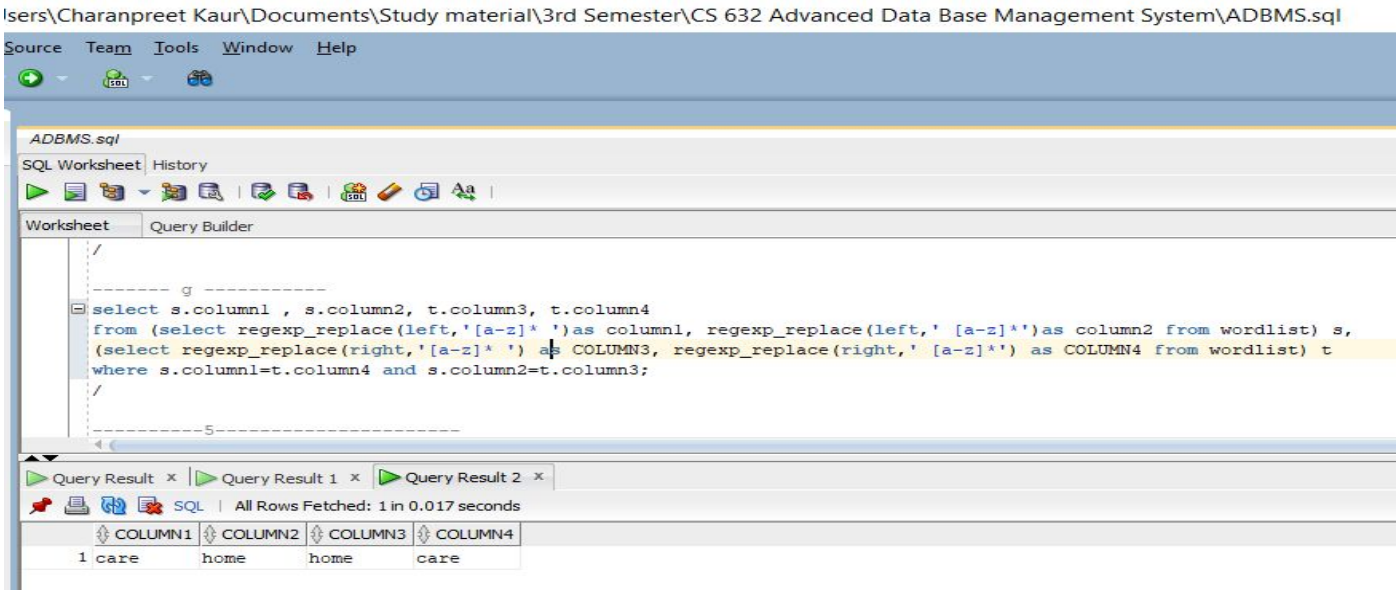
select s.column1 , s.column2, t.column3, t.column4
from (select regexp_replace(left,'[a-z]* ')as column1, regexp_replace(left,' [a-z]*')as column2 from wordlist) s,
(select regexp_replace(right,'[a-z]* ') as COLUMN3, regexp_replace(right,' [a-z]*') as COLUMN4 from wordlist) t ;



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;



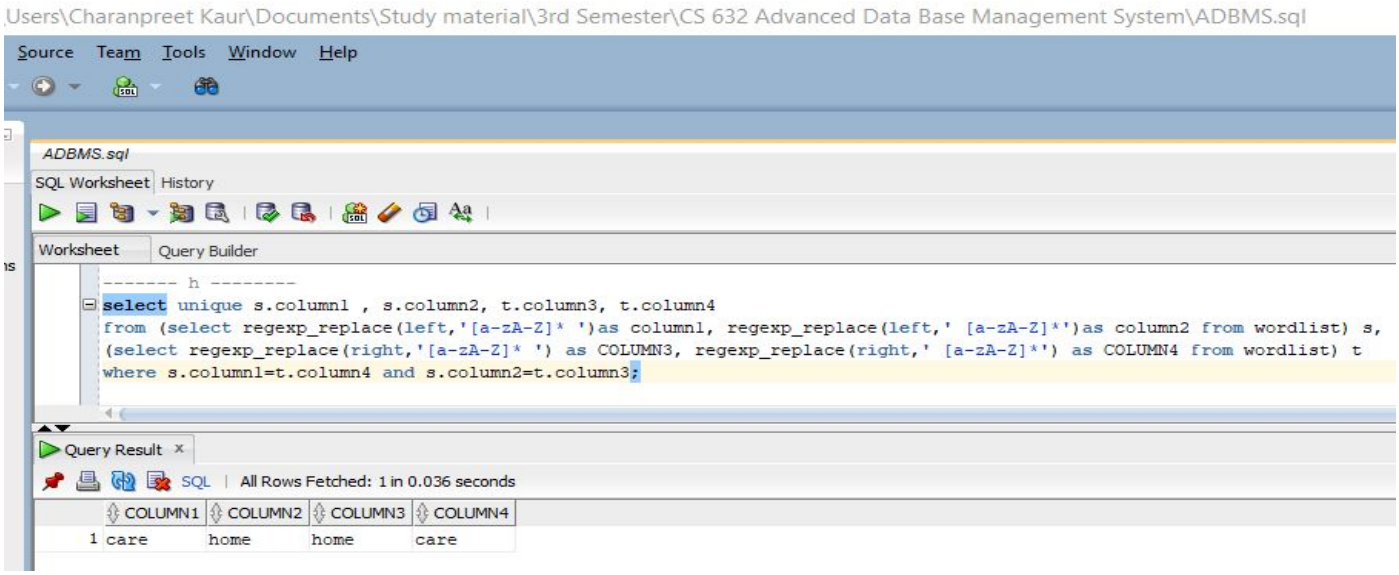
h) Add the keyword *unique* to the outer SELECT so that you get only ONE row that looks like this.

Column1	Column2	Column3	Column4
care	home	home	care

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

Answer h)

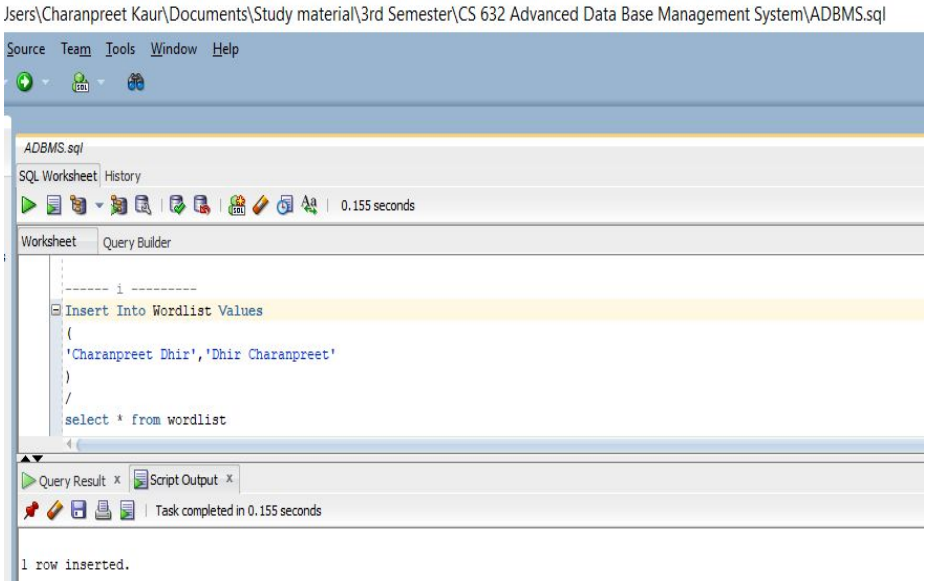
select unique s.column1 , s.column2, t.column3, t.column4
from (select regexp_replace(left,'[a-zA-Z]* ')as column1, regexp_replace(left,' [a-zA-Z]*')as column2 from wordlist) s,
(select regexp_replace(right,'[a-zA-Z]* ') as COLUMN3, regexp_replace(right,' [a-zA-Z]*') as COLUMN4 from wordlist) t
where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



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'
)

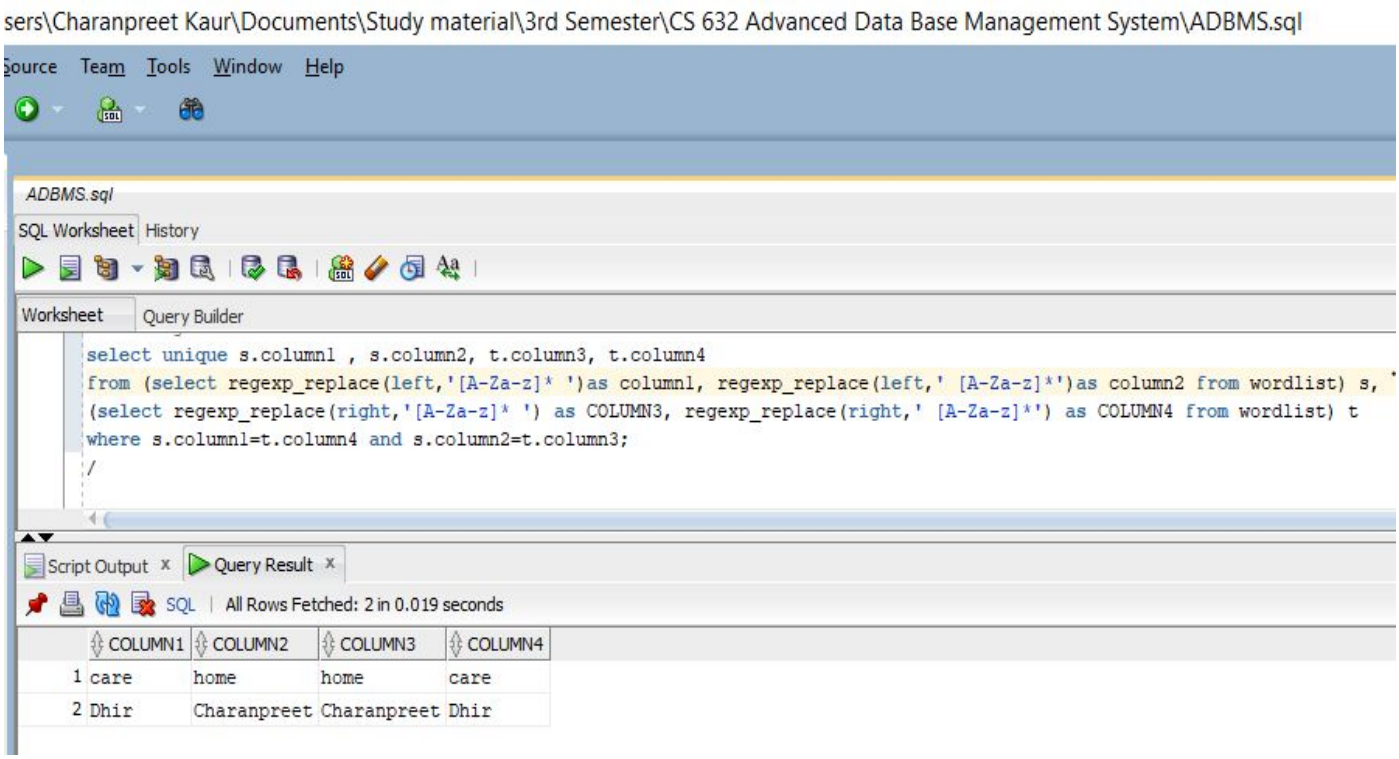


j) Now run the select statement from question h) again and show the output.
It should look like this:

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

<u>Answer j)</u>

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;



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’

<u>Answer 3 a)</u>

insert into wordlist values(
'John 3brown', 'Brown John'
)
/
insert into wordlist values(
'Tom! Smith', 'Smith Tom'
)

ADBMS.sql

SQL Worksheet | History

Worksheet | Query Builder

```
insert into wordlist values(
'John 3brown', 'Brown John'
)
/
insert into wordlist values(
'Tom! Smith', 'Smith Tom'
)
/
```

Script Output | Query Result

Task completed in 0.106 seconds

```
1 row inserted.


1 row inserted.
```

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

Source Team Tools Window Help

ADBMS.sql

SQL Worksheet | History

Worksheet | Query Builder

```
----- a -------
insert into wordlist values(
'John 3brown', 'Brown John'
)
/
insert into wordlist values(
'Tom! Smith', 'Smith Tom'
)

select * from wordlist
/
```

Script Output | Query Result

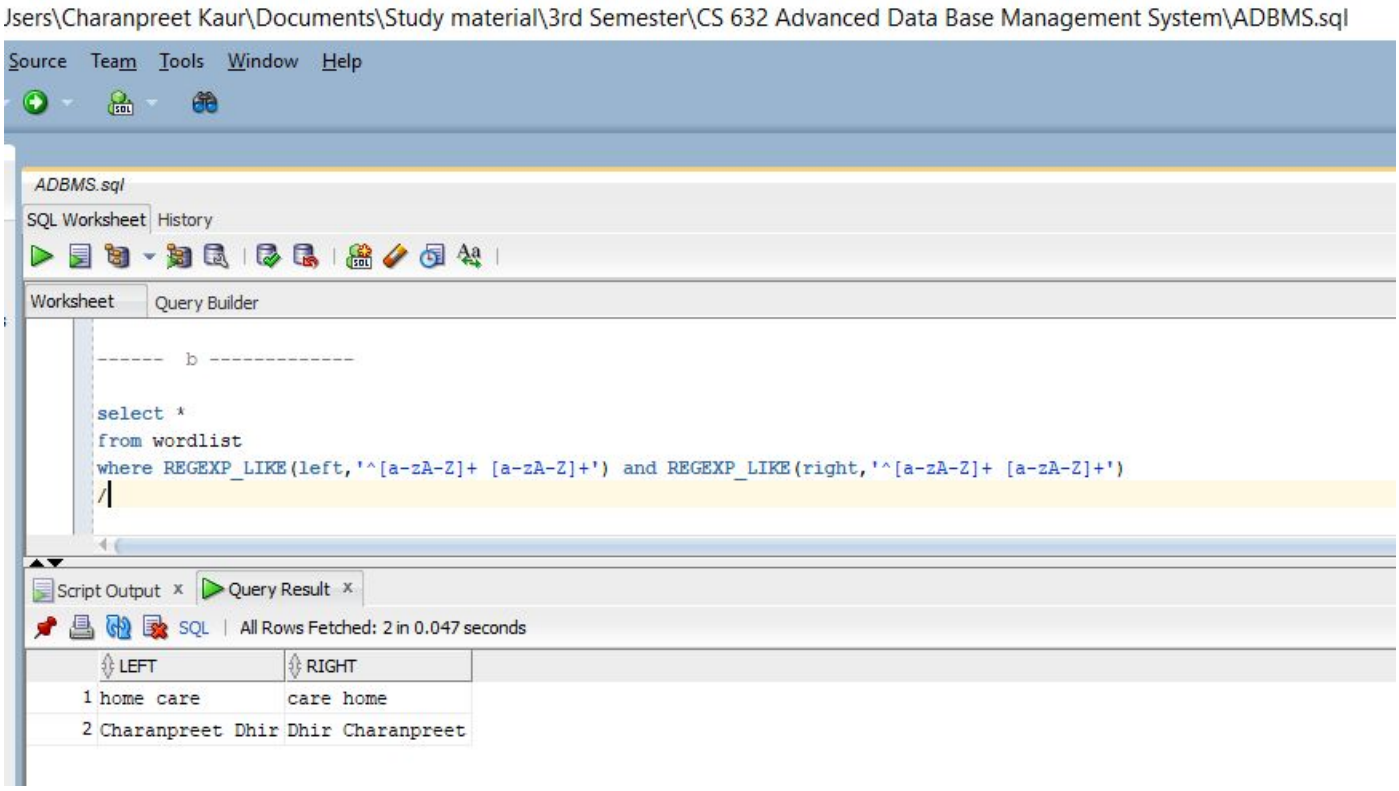
SQL | All Rows Fetched: 4 in 0.015 seconds

	LEFT	RIGHT
1	home care	care home
2	Tom! Smith	Smith Tom
3	Charanpreet Dhir	Dhir Charanpreet
4	John 3brown	Brown John

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 !!!!!

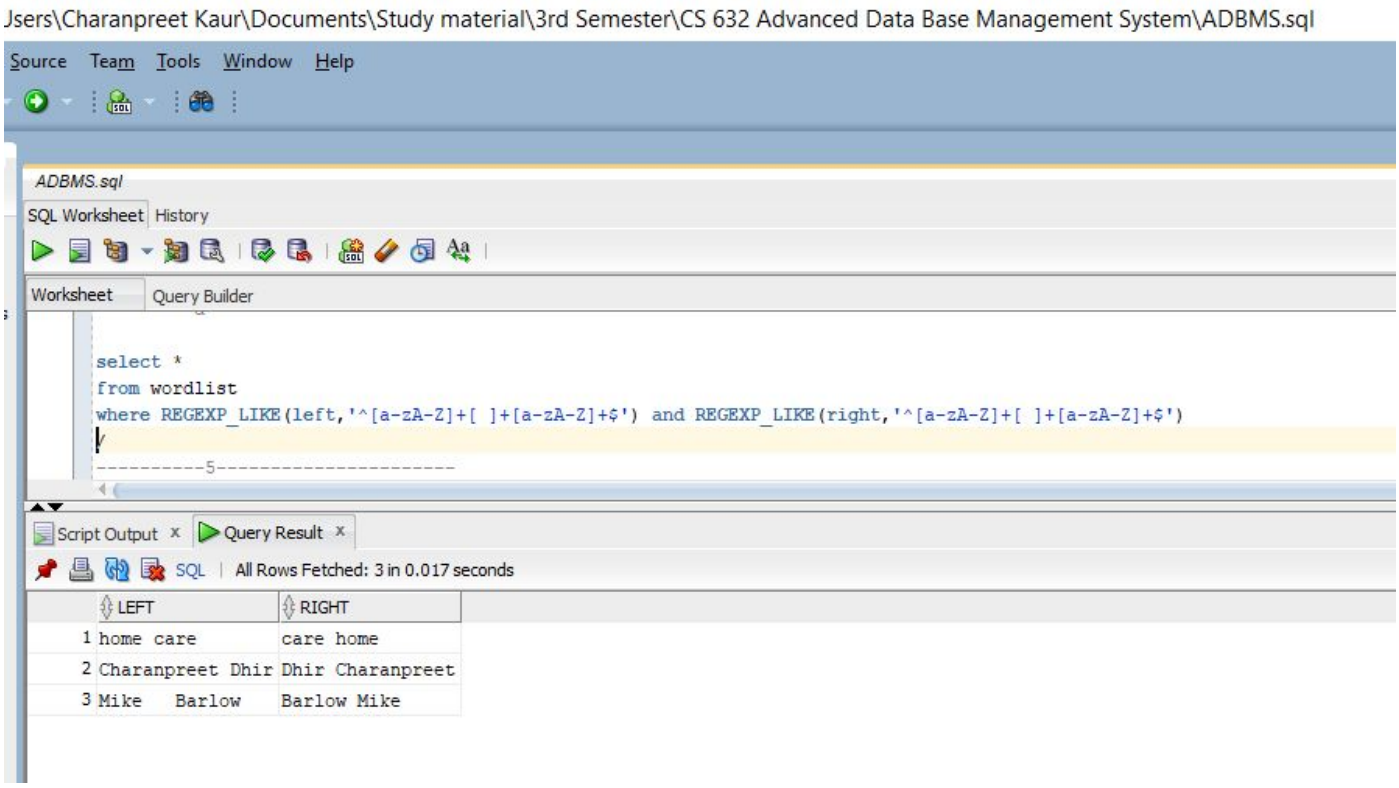
Answer c)

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.

<u>Answer d)</u>

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'

<u>Answer 4) a)</u>

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.

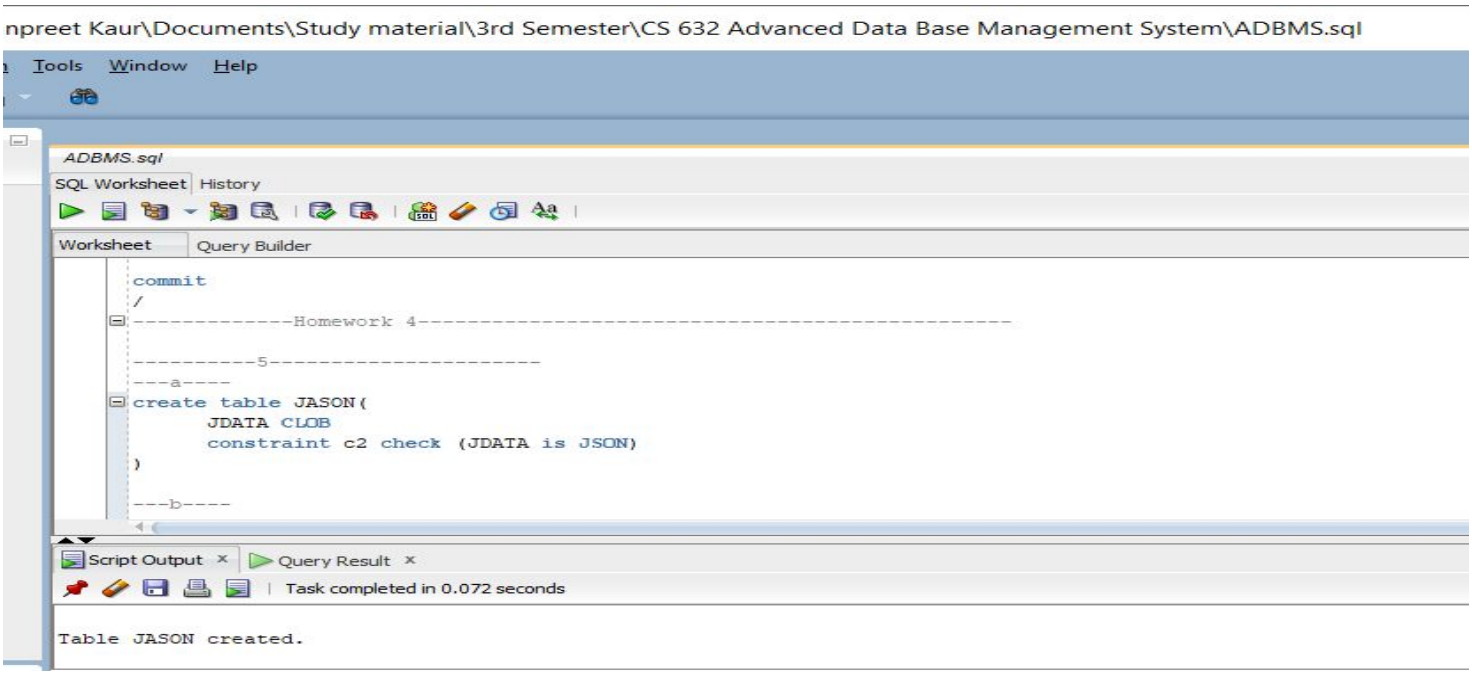
<u>Answer b)</u>

```
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)
)
```



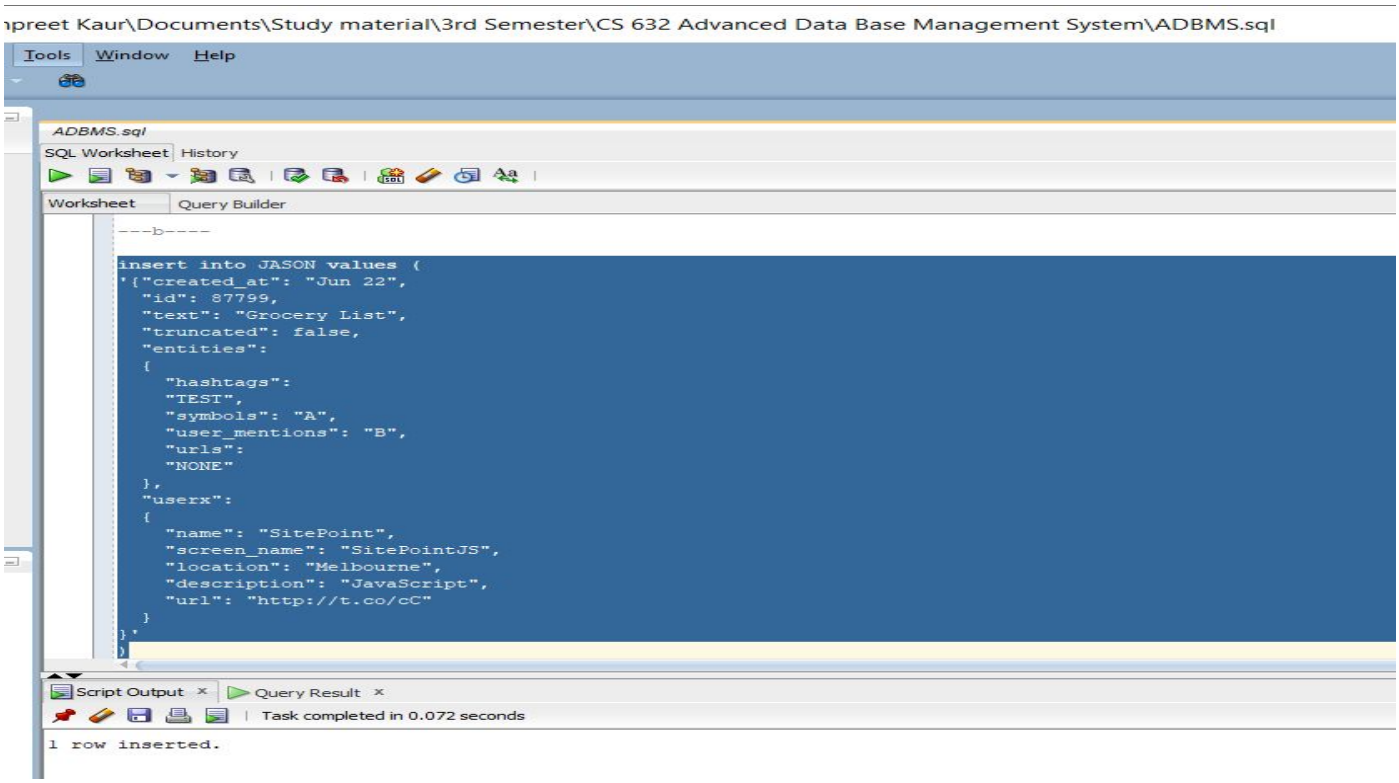
b) Insert the following into the table JASON.


```
{"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



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

Tools Window Help

ADBMS.sql

SQL Worksheet History

Worksheet Query Builder

```
SELECT * FROM JASON
```

Script Output x Query Result x

SQL | All Rows Fetched: 1 in 0.015 seconds

JDATA

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

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

Tools Window Help

ADBMS.sql

SQL Worksheet History

Worksheet Query Builder

SELECT * FROM JASON

Script Output x Query Result x

SQL | All Rows Fetched: 1 in 0.015 seconds

JDATA

View Value

Line Terminator: Platform Default Change...

Value:

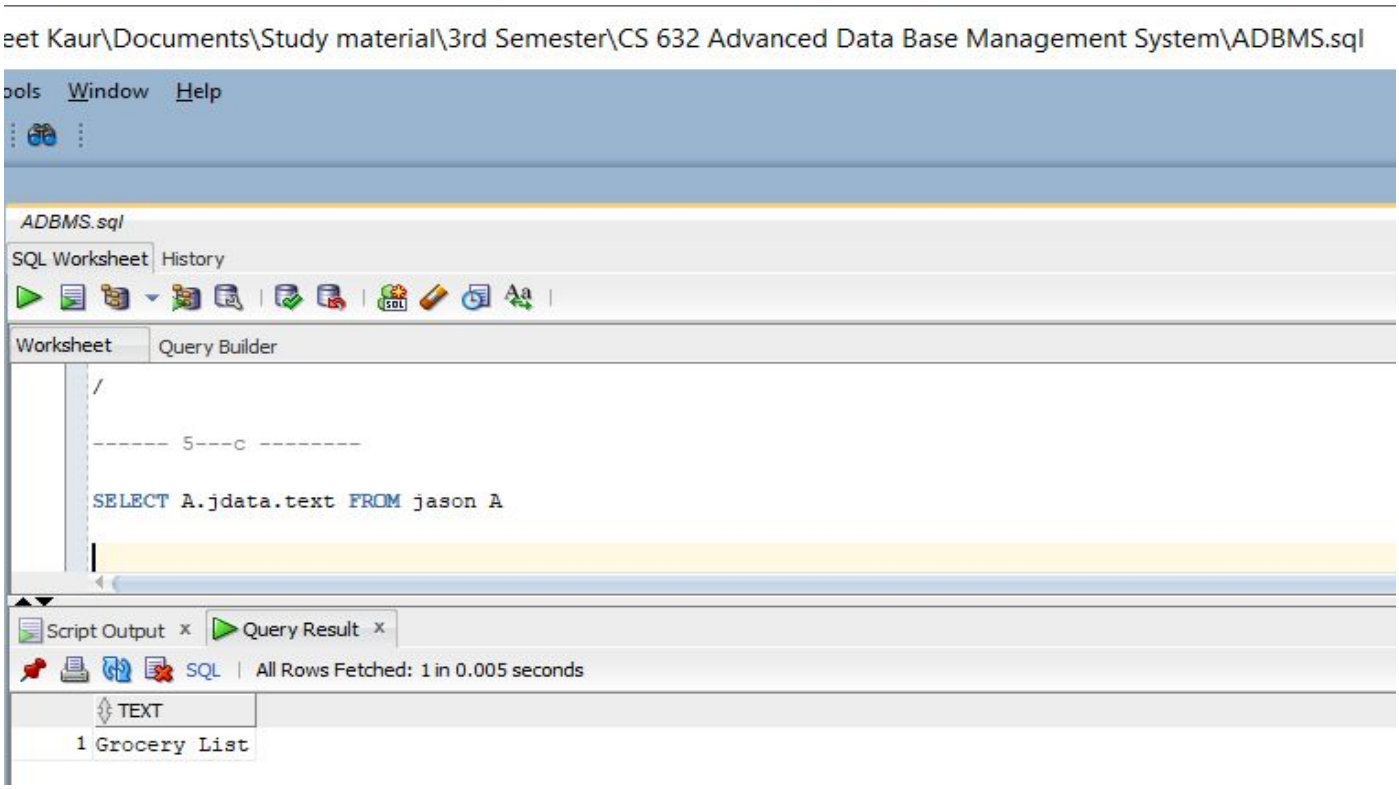

```
{"created_at": "Jun 22",
  "id": 87799,
  "text": "Grocery List",
  "truncated": false,
  "entities":
  {
    "hashtags":
    "TEST",
    "symbols": "A",
    "user_mentions": "B",
    "urls":
    "NONE"
  },
  "userx":
  {
    "name": "SitePoint",
    "screen_name": "SitePointJS",
    "location":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


Help OK Cancel

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

<u>Answer c)</u>

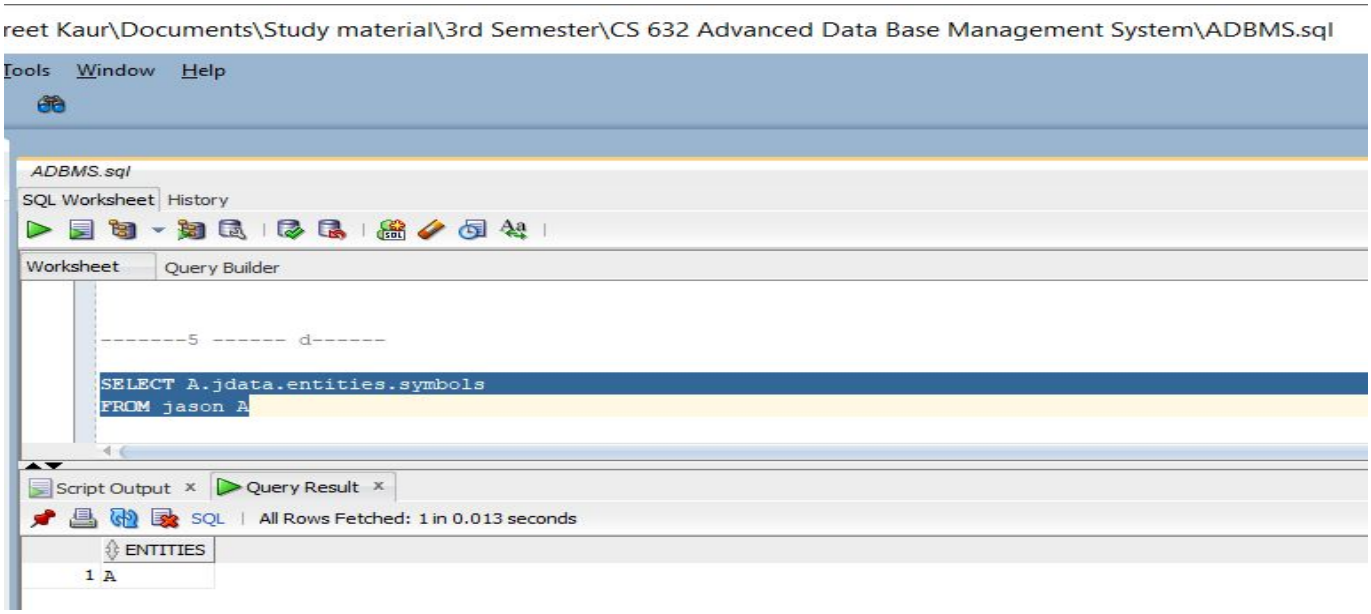
SELECT A.jdata.text
FROM jason A



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

<u>Answer d)</u>

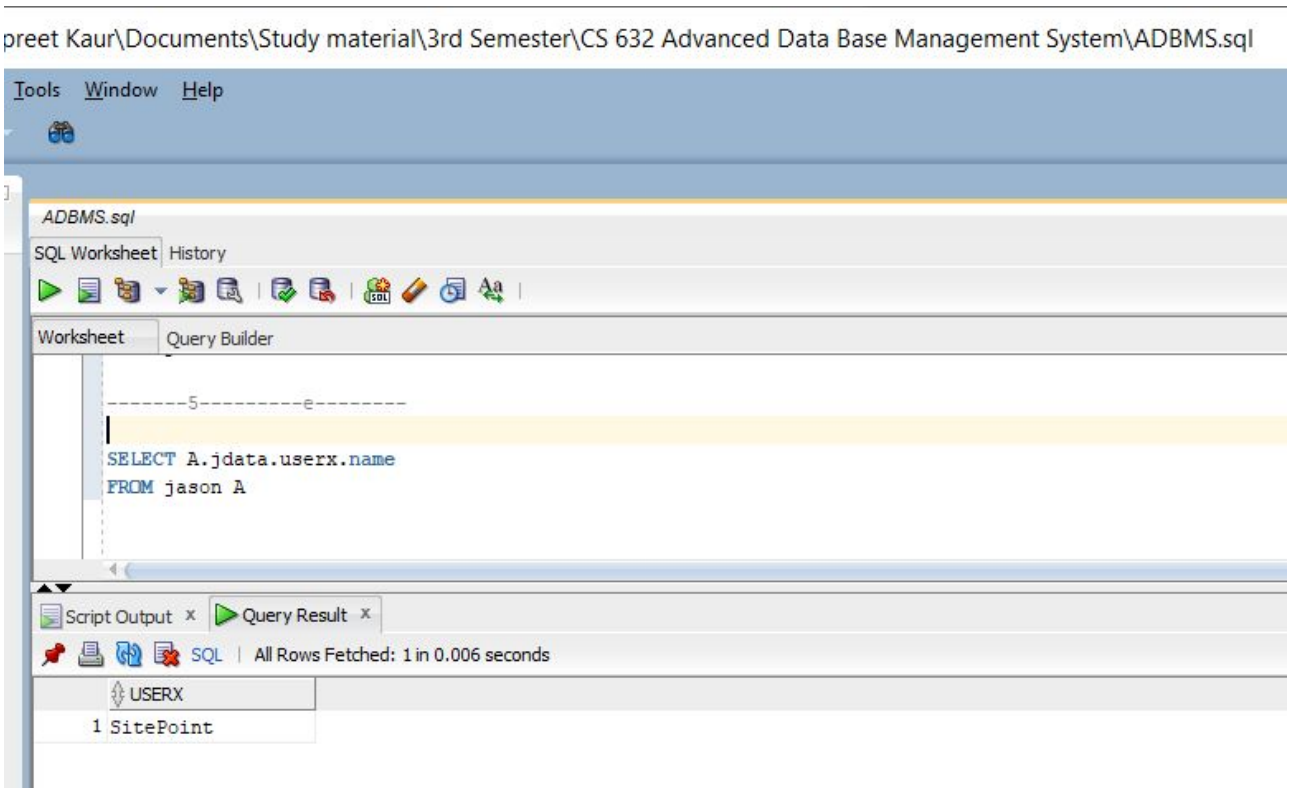
SELECT A.jdata.entities.symbols
FROM jason A



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

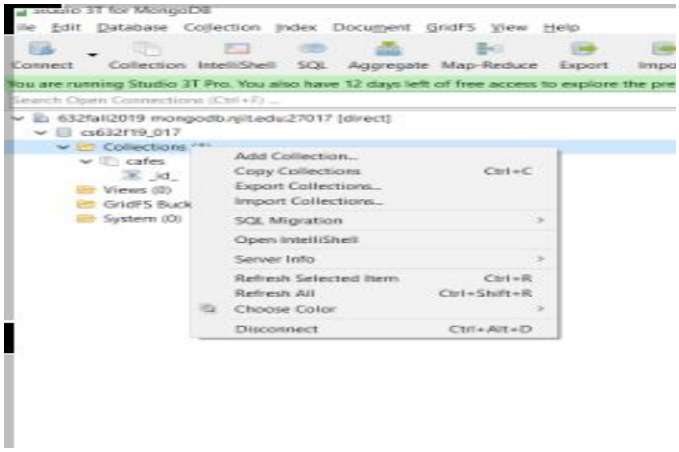


6) SWITCH TO MONGODB NOW.

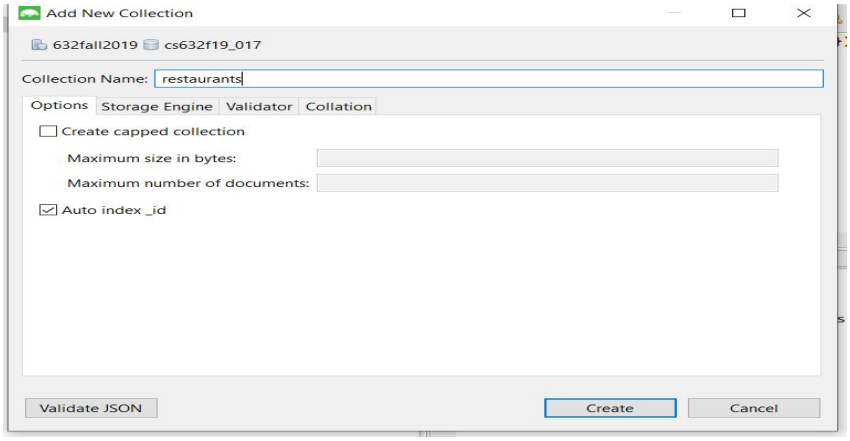
a) Create a new collection called “restaurants”

Answer 6 a)

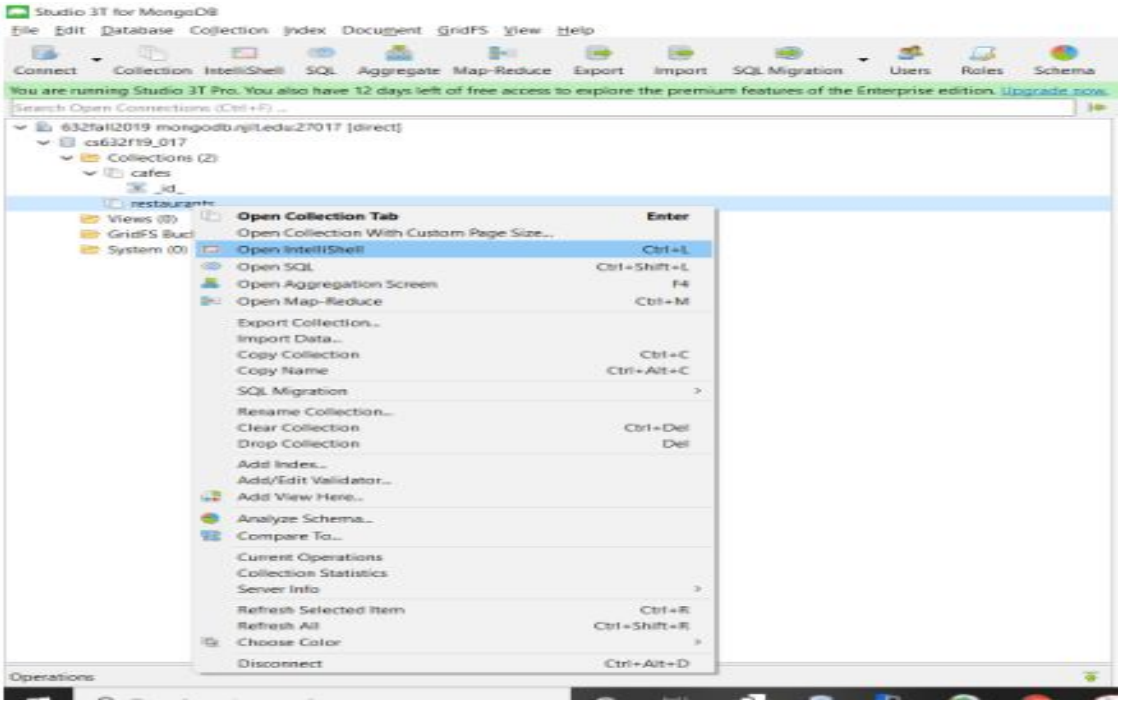
1) **Right click on Collections.**
2) **Click on Add Collection Type in a name - restaurants**



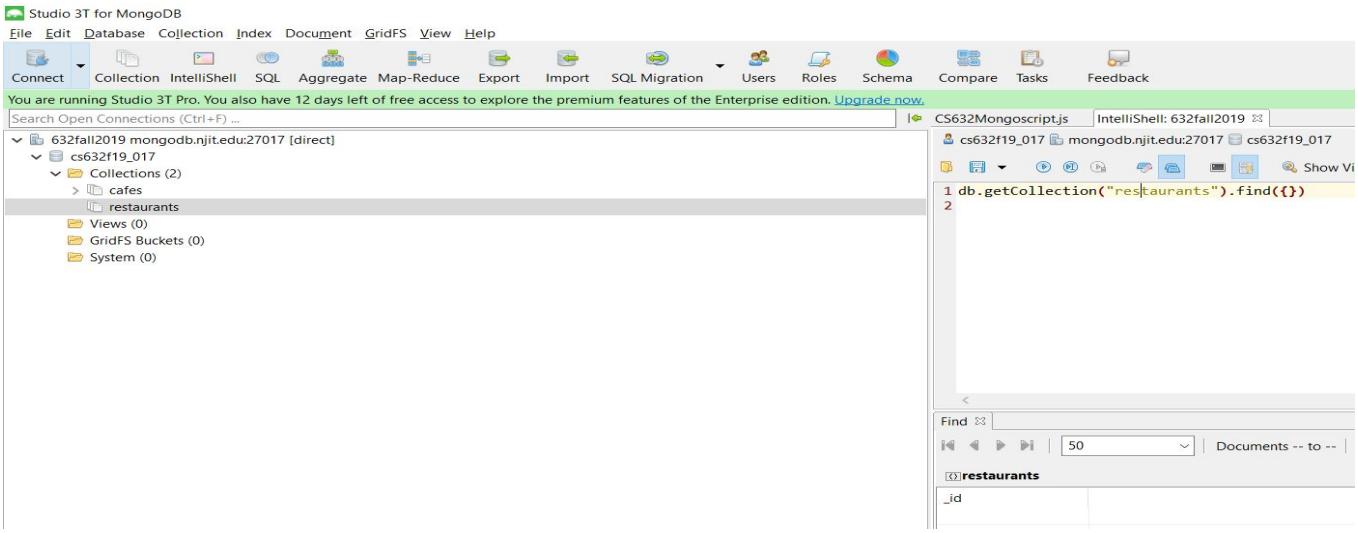
3) **Click on Create. Right click on restaurants.**



4) **Click on Open in Intellishell**



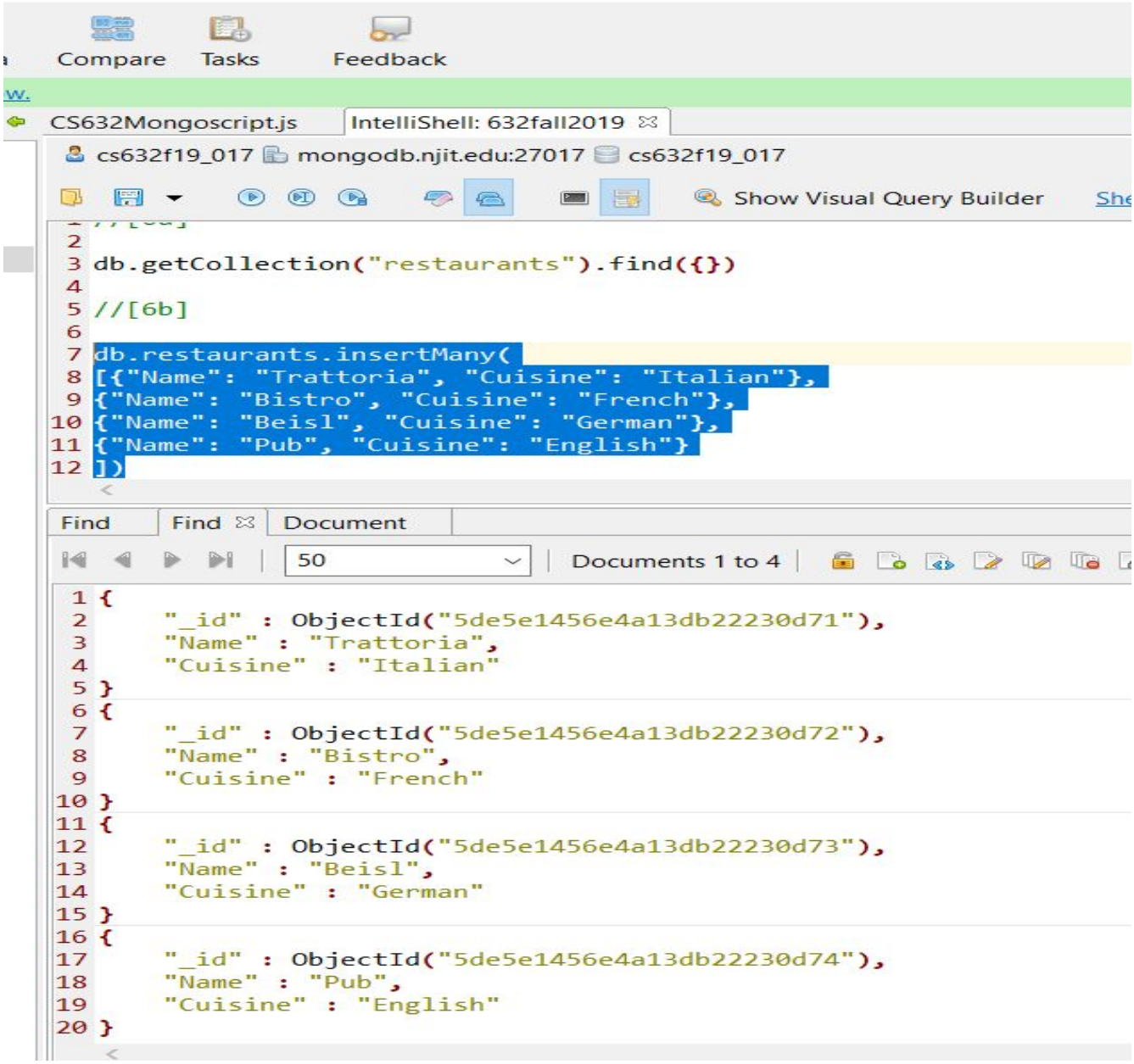
5) **After opening in intellishell. Screenshot attached is attached below.**



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

<u>Answer b)</u>

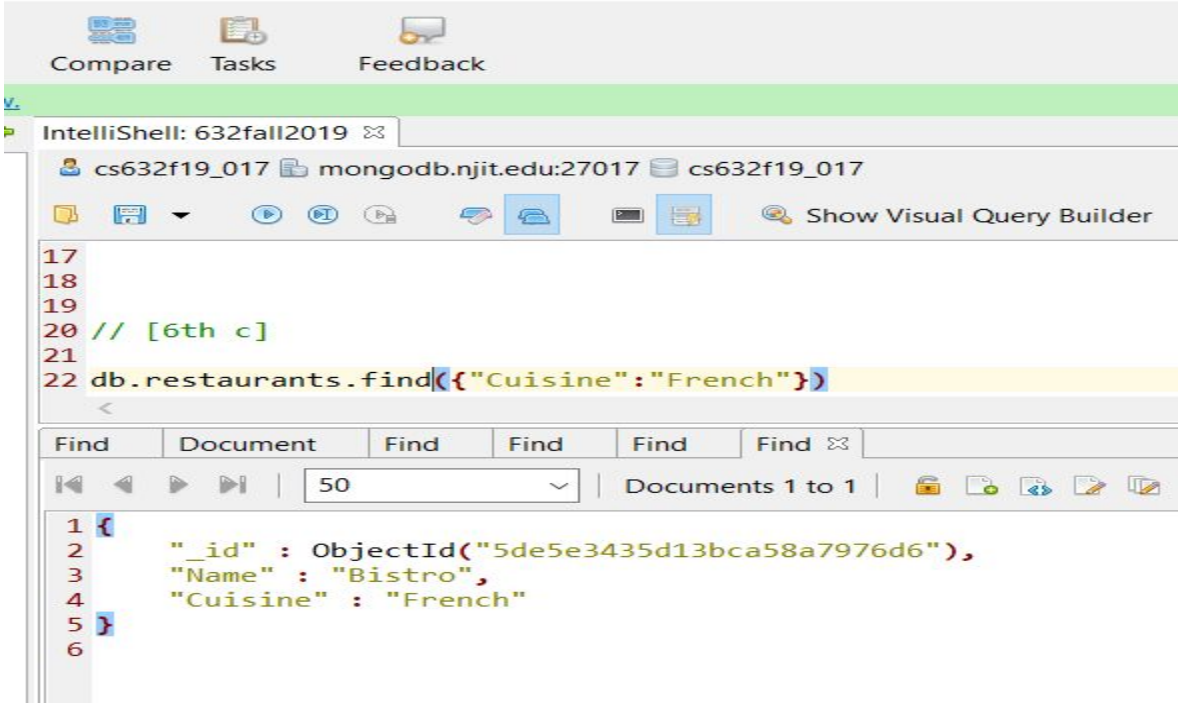
db.restaurants.insertMany(
[{"Name": "Trattoria", "Cuisine": "Italian"},
{"Name": "Bistro", "Cuisine": "French"},
{"Name": "Beisl", "Cuisine": "German"},
{"Name": "Pub", "Cuisine": "English"}
])



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

Answer c)

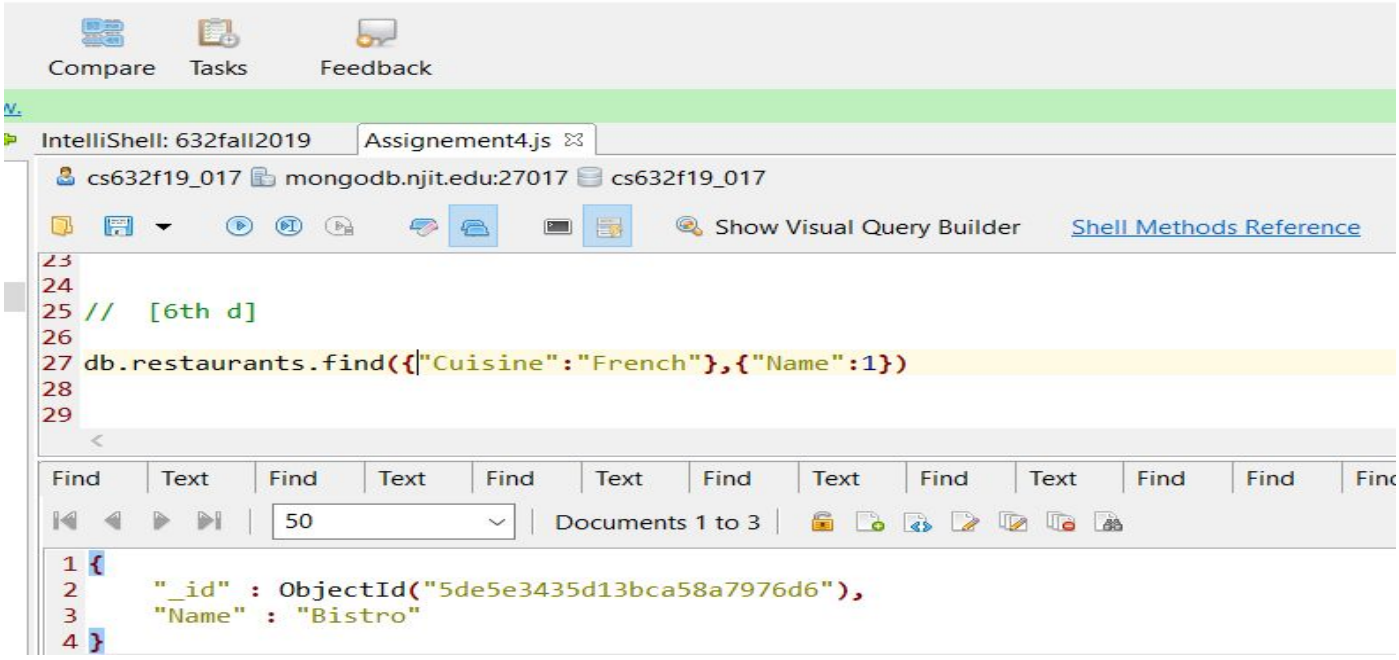
db.restaurants.find({"Cuisine":"French"})



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

Answer d)

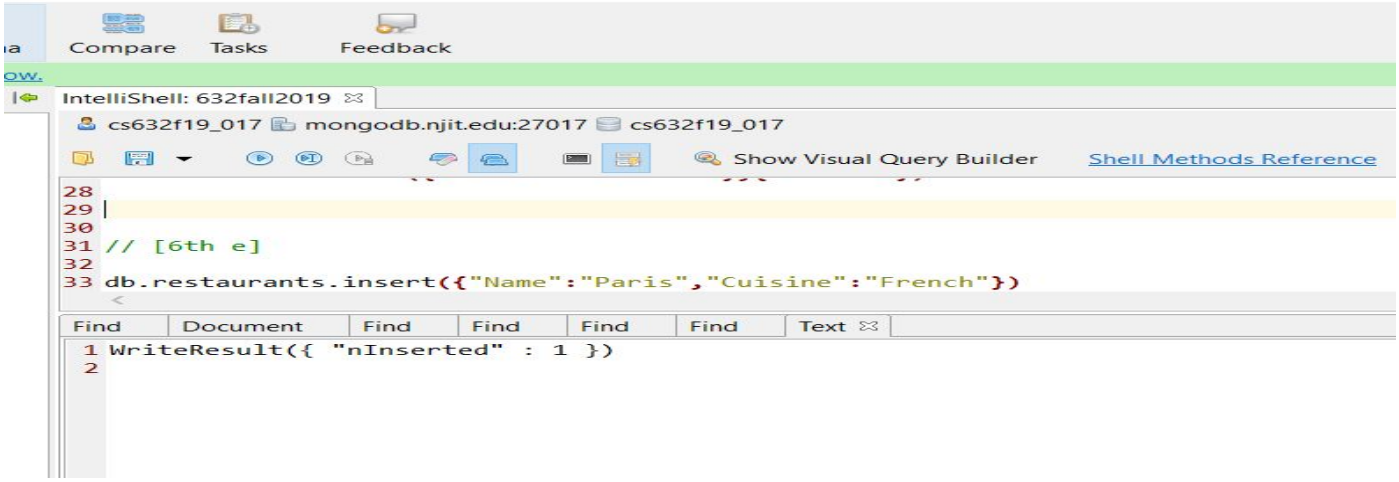
db.restaurants.find({"Cuisine":"French"},{"Name":1})

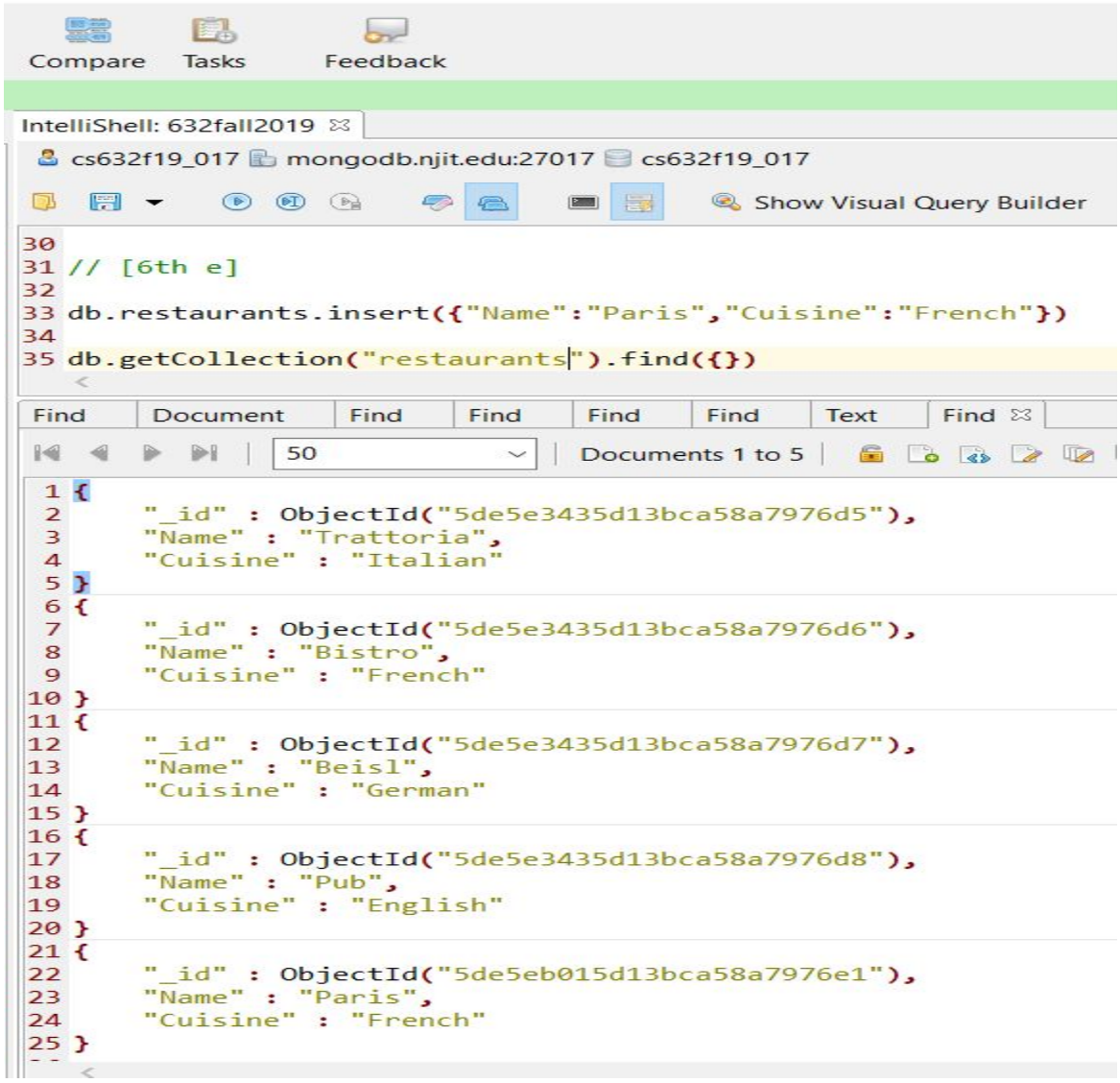


e) Insert an additional French cuisine restaurant called Paris.

Answer e)

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





f) Redo question c)

Answer f)

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

```
IntelliShell: 632fall2019
cs632f19_017 mongodb.njit.edu:27017 cs632f19_017
Show Visual Query Bu

// [6th f]

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

Find | Document | Find | Find | Find | Find | Text | Find
50 | Documents 1 to 2

{
    "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
    "Name" : "Bistro",
    "Cuisine" : "French"
}
{
    "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
    "Name" : "Paris",
    "Cuisine" : "French"
}
```

g) Redo question d)

Answer g)

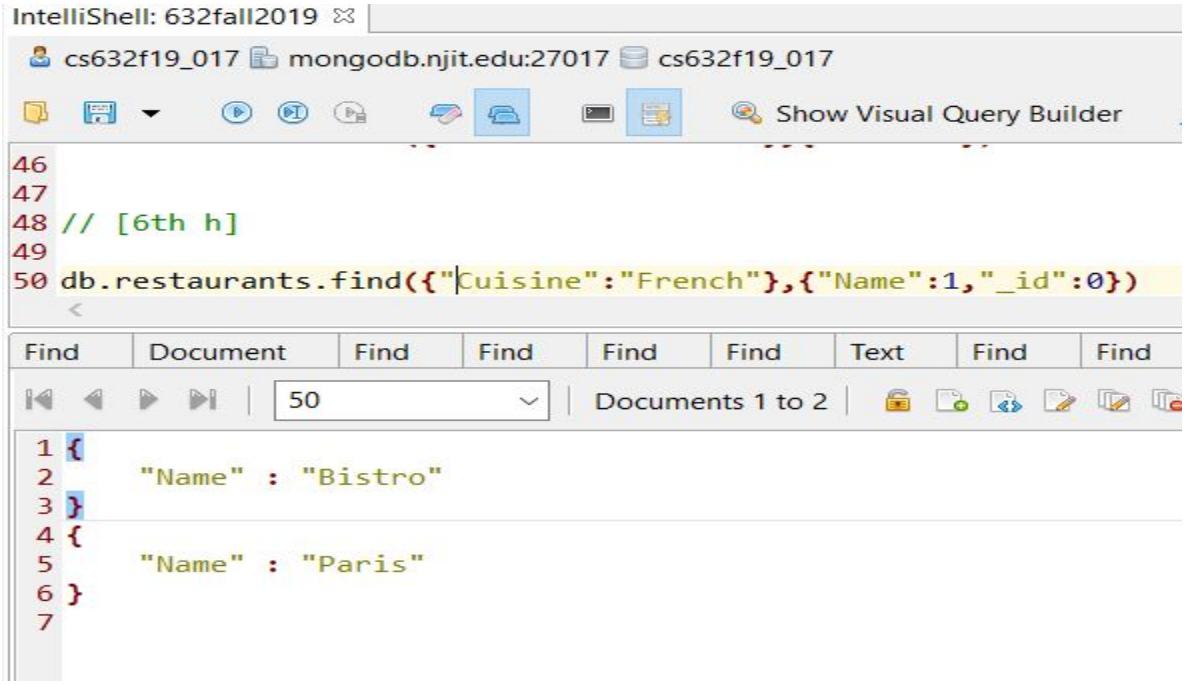
db.restaurants.find({"Cuisine":"French"},{"Name":1})

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

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

// [6th g]

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

Find | Document | Find | Find | Find | Find | Text | Find | Fi
50 | Documents 1 to 2

{
    "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
    "Name" : "Bistro"
}
{
    "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
    "Name" : "Paris"
}
```

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

Answer h)

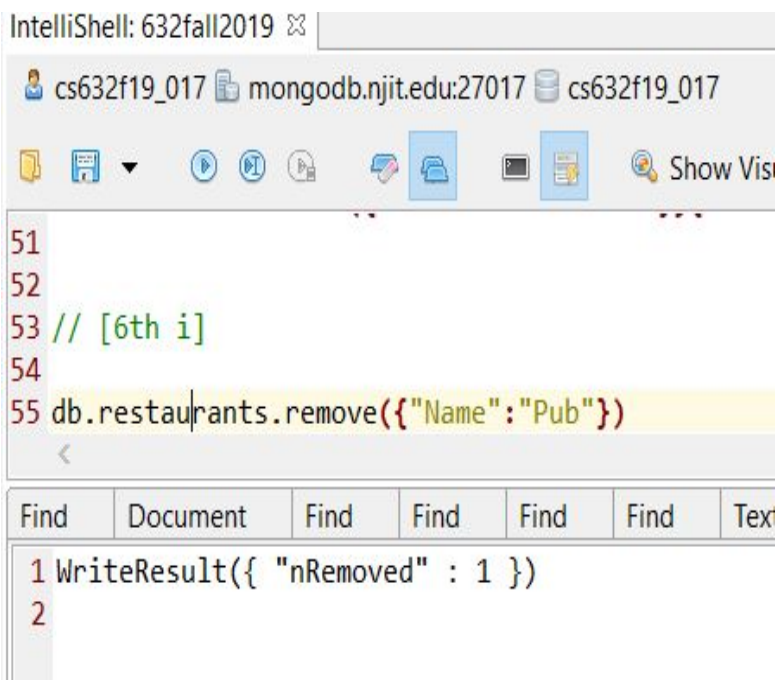
db.restaurants.find({"Cuisine":"French"},{"Name":1,"_id":0})



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

Answer i)

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



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 :

Compare Tasks Feedback

(don't sh

IntelliShell: 632fall2019

cs632f19_017 mongodb.njit.edu:27017 cs632f19_017

Show Visual Query Builder Shell Methods Reference

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

Document | Find | Find | Find | Find | Text | Find | Find | Find | Find | Find | Find | Text

50 Documents 1 to 4 JSON

```
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
```

4 documents

k) Update “Paris” to be named LaFrance. Then show all restaurants.

Answer k)

db.restaurants.update({"Name":"Paris"},{$set:{"Name":"LaFrance"}})

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

// [6th k]

db.restaurants.update({"Name":"Paris"},{$set:{"Name":"LaFrance"}})
db.getCollection("restaurants").find({})

Find | Find | Find | Text | Find | Find | Find | Find | Find | Find | Text

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

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

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

// [6th k]

db.restaurants.update({"Name":"Paris"},{$set:{"Name":"LaFrance"}})
db.getCollection("restaurants").find({})

Find | Find | Text | Find | Find | Find | Find | Find | Find | Text | Fi

50   Documents 1 to 4

{
    "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
    "Name" : "Trattoria",
    "Cuisine" : "Italian"
}
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
    "Name" : "Bistro",
    "Cuisine" : "French"
}
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
    "Name" : "Beisl",
    "Cuisine" : "German"
}
{
    "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
    "Name" : "LaFrance",
    "Cuisine" : "French"
}
```

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

<u>Answer l)</u>

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

// [6th L]

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

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

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

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

// [6th L]

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

Find   Text   Find
50   Documents 1 to 4
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
    "Name" : "Trattoria",
    "Cuisine" : "Italian"
}
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
    "Name" : "Bistro",
    "Cuisine" : "French"
}
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
    "Name" : "Beisl",
    "Cuisine" : "German"
}
{
    "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
    "Name" : "LaFrance",
    "Cuisine" : "French and Italian"
}
```

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

// [6th m ]

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

Find | Text | Find | Text

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

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

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

// [6th m ]

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

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

{
    "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
    "Name" : "Trattoria",
    "Cuisine" : "Italian",
    "Drinks" : [
        "Wine",
        "Coffee",
        "Water"
    ]
}
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d6"),
    "Name" : "Bistro",
    "Cuisine" : "French"
}
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d7"),
    "Name" : "Beisl",
    "Cuisine" : "German"
}
{
    "_id" : ObjectId("5de5eb015d13bca58a7976e1"),
    "Name" : "LaFrance",
    "Cuisine" : "French and Italian"
}
```

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

Answer n)

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

```
// [6th n]

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

```
4
```

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})

```
//[6th O]

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

```
{
    "_id" : ObjectId("5de5e3435d13bca58a7976d5"),
    "Drinks" : [
        "Wine",
        "Coffee",
        "Water"
    ]
}
```