[redacted]

MongoDB & Cassandra

[redacted]

Table of content

Table of content 2

Cover sheet for group assignments 3

Contribution declaration form 4

C.1 MongoDB 5

C.2 Cassandra 17

C.3 Reflections 21

C.4 Connecting to Drivers 27

Appendix 29

C.1 MongoDB

C.1.1. Using the MongoDB Compass, create a database called FIT5137MASL and create the collection called ufo.

Explanation: Open MongoDB Compass, then load local database connection. after connecting successfully, create a database called FIT5137MASL and collection ufo, finished.

C.1.2. Using the MongoDB Compass, import ufo.json into the ufo collection. All fields should be assigned appropriate data types (i.e. whole number data should be integers, numbers with decimal points should be double etc.)

Explanation: Stay in ufo collection in FIT5137MASL database, import data and choose ufo.json file, check number data with integer type and double type, waiting for data input successfully.

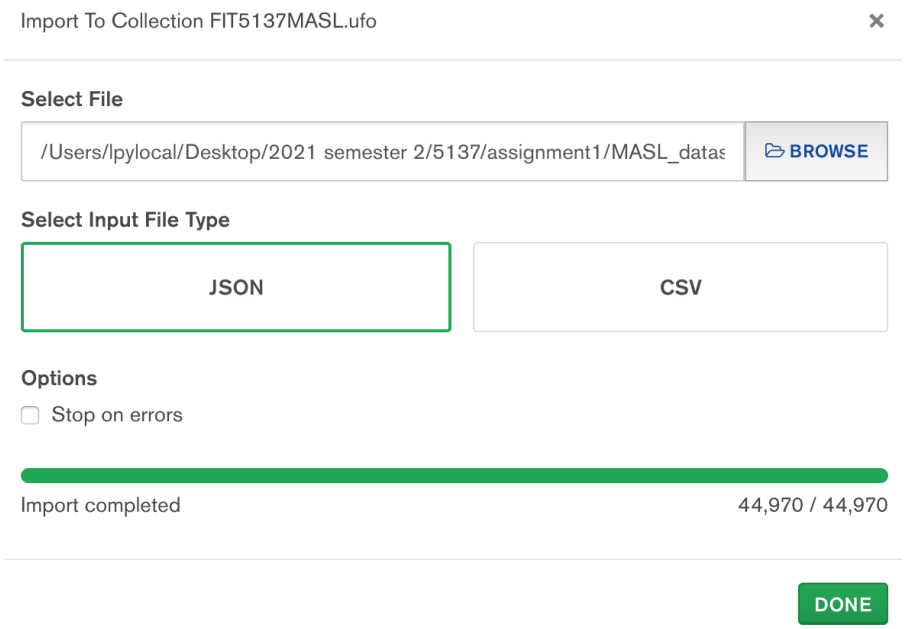


Figure 1: Screenshot of C.1.2

C.1.3. Using the ufo collection with one MongoDB Shell Command:

(i) combine the date data (from the fields for year, month, day and hour) to proper MongoDB date data types.

Explanation: Combine fields year, month, day, hour as one date type field. Use dateFromParts function to extract corresponding values from year, month, day and hour fields.
Reference: https://docs.mongodb.com/manual/reference/operator/aggregation/dateFromParts/
Code:

```
db.ufo.aggregate([
{$project:{"dateTime":{$dateFromParts:{"year":"$year","month":"$month","day":"$day","hour":"
$hour"}},_id:0}}]).pretty();
```

```
> db.ufo.aggregate([
... {$project:{"dateTime":{$dateFromParts:{"year":"$year","month":"$month","day":"$day","hour":"$hour"}},_id:0}}
... ]).pretty();
{ "dateTime" : ISODate("1997-11-14T21:00:00Z") }
{ "dateTime" : ISODate("1997-11-14T21:00:00Z") }
{ "dateTime" : ISODate("1997-11-24T02:00:00Z") }
{ "dateTime" : ISODate("1997-11-24T02:00:00Z") }
{ "dateTime" : ISODate("1997-12-31T17:00:00Z") }
{ "dateTime" : ISODate("1998-03-07T18:00:00Z") }
{ "dateTime" : ISODate("1998-07-04T23:00:00Z") }
{ "dateTime" : ISODate("1998-07-28T18:00:00Z") }
{ "dateTime" : ISODate("1998-08-03T17:00:00Z") }
{ "dateTime" : ISODate("1998-08-07T00:00:00Z") }
{ "dateTime" : ISODate("1998-08-19T21:00:00Z") }
{ "dateTime" : ISODate("1998-09-23T21:00:00Z") }
{ "dateTime" : ISODate("1998-10-04T21:00:00Z") }
{ "dateTime" : ISODate("1998-10-04T21:00:00Z") }
{ "dateTime" : ISODate("1998-10-11T20:00:00Z") }
{ "dateTime" : ISODate("1998-10-12T19:00:00Z") }
{ "dateTime" : ISODate("1998-12-03T23:00:00Z") }
{ "dateTime" : ISODate("1999-02-27T14:00:00Z") }
{ "dateTime" : ISODate("1999-03-12T07:00:00Z") }
{ "dateTime" : ISODate("1999-04-22T22:00:00Z") }
Type "it" for more
```

Figure 2: Screenshot of combine day,month,year,hour

(ii) store the combined date data in a new field called dateTime.

Explanation: Use addFields aggregation to add a new field called dateTime into ufo collection with extra aggregation manipulation. Use the merge function to overwrite new data into the original collection.
Reference: https://docs.mongodb.com/manual/reference/operator/aggregation/addFields/
Code:

```
db.ufo.aggregate([
{$addFields:{"dateTime":
{$dateFromParts:{"year":"$year","month":"$month","day":"$day","hour":"$hour"}}}},
{$merge:{into:"ufo"}}
]);
```

Services

Output | Result 154

1-20 of 21+ JSON

	_id	city	countyName	dateTime	day	hour	month	shape	sighting(
1	60f6b847a5e4e6647265ae6c	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	14	21	11	formation	[{}]
2	60f6b847a5e4e6647265ae6d	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	14	21	11	fireball	[{}]
3	60f6b847a5e4e6647265ae6e	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	24	2	11	oval	[{}]
4	60f6b847a5e4e6647265ae6f	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	24	2	11	oval	[{}]
5	60f6b847a5e4e6647265ae70	GULFPORT	HARRISON	1997-12-31T17:00:00.000Z	31	17	12	triangle	[{}]
6	60f6b847a5e4e6647265ae71	ORLANDO	ORANGE	1998-03-07T18:00:00.000Z	7	18	3	light	[{}]
7	60f6b847a5e4e6647265ae72	BOSTON	SUFFOLK	1998-07-04T23:00:00.000Z	4	23	7	sphere	[{}]
8	60f6b847a5e4e6647265ae73	BRISTOL	BRISTOL	1998-07-28T18:00:00.000Z	28	18	7	disk	[{}]
9	60f6b847a5e4e6647265ae74	EL PASO	EL PASO	1998-08-03T17:00:00.000Z	3	17	8	fireball	[{}]
10	60f6b847a5e4e6647265ae75	CASA GRANDE	PINAL	1998-08-07T00:00:00.000Z	7	0	8	triangle	[{}]
11	60f6b847a5e4e6647265ae76	RENO	WASHOE	1998-08-19T21:00:00.000Z	19	21	8	light	[{}]
12	60f6b847a5e4e6647265ae77	HOUSTON	HARRIS	1998-09-23T21:00:00.000Z	23	21	9	fireball	[{}]
13	60f6b847a5e4e6647265ae78	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	4	21	10	fireball	[{}]
14	60f6b847a5e4e6647265ae79	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	4	21	10	flash	[{}]
15	60f6b847a5e4e6647265ae7a	HIGHLAND	LAKE	1998-10-11T20:00:00.000Z	11	20	10	sphere	[{}]
16	60f6b847a5e4e6647265ae7b	ASHEVILLE	BUNCOMBE	1998-10-12T19:00:00.000Z	12	19	10	light	[{}]
17	60f6b847a5e4e6647265ae7c	KIRKLAND	KING	1998-12-03T23:00:00.000Z	3	23	12	triangle	[{}]
18	60f6b847a5e4e6647265ae7d	LAUGHLIN	CLARK	1999-02-27T14:00:00.000Z	27	14	2	formation	[{}]
19	60f6b847a5e4e6647265ae7e	GRAND RAPIDS	KENT	1999-03-12T07:00:00.000Z	12	7	3	triangle	[{}]
20	60f6b847a5e4e6647265ae7f	ALBUQUERQUE	BERNALILLO	1999-04-22T22:00:00.000Z	22	22	4	fireball	[{}]

Figure 3: Screenshot of add new dateTime field

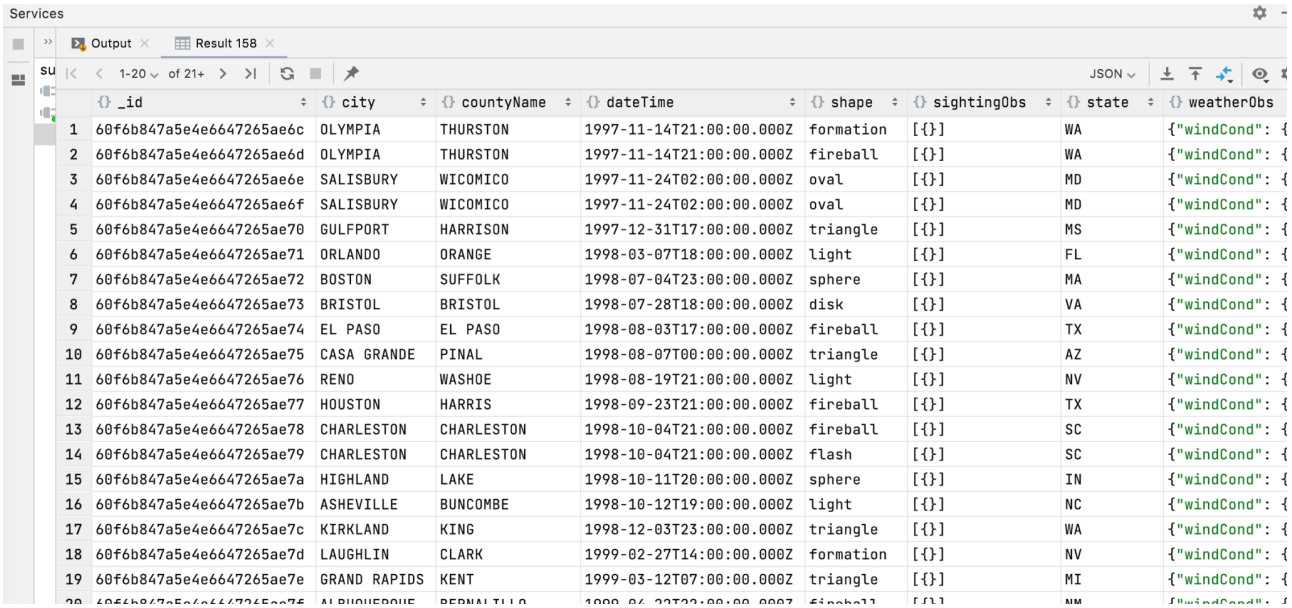
(iii) remove the fields for year, month, day and hour.

Explanation: Group members use the unset function to Remove year, month, day and hour filed from ufo collection.

Reference: /

Code:

```
db.ufo.updateMany({},{$unset:{"year":1,"month":1,"day":1,"hour":1}});
```



Services

Output | Result 158

1-20 of 21+ | JSON

	_id	city	countyName	dateTime	shape	sightingObs	state	weatherObs
1	60f6b847a5e4e6647265ae6c	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	formation	[{}]	WA	{"windCond": {
2	60f6b847a5e4e6647265ae6d	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	fireball	[{}]	WA	{"windCond": {
3	60f6b847a5e4e6647265ae6e	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	oval	[{}]	MD	{"windCond": {
4	60f6b847a5e4e6647265ae6f	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	oval	[{}]	MD	{"windCond": {
5	60f6b847a5e4e6647265ae70	GULFPORT	HARRISON	1997-12-31T17:00:00.000Z	triangle	[{}]	MS	{"windCond": {
6	60f6b847a5e4e6647265ae71	ORLANDO	ORANGE	1998-03-07T18:00:00.000Z	light	[{}]	FL	{"windCond": {
7	60f6b847a5e4e6647265ae72	BOSTON	SUFFOLK	1998-07-04T23:00:00.000Z	sphere	[{}]	MA	{"windCond": {
8	60f6b847a5e4e6647265ae73	BRISTOL	BRISTOL	1998-07-28T18:00:00.000Z	disk	[{}]	VA	{"windCond": {
9	60f6b847a5e4e6647265ae74	EL PASO	EL PASO	1998-08-03T17:00:00.000Z	fireball	[{}]	TX	{"windCond": {
10	60f6b847a5e4e6647265ae75	CASA GRANDE	PINAL	1998-08-07T00:00:00.000Z	triangle	[{}]	AZ	{"windCond": {
11	60f6b847a5e4e6647265ae76	RENO	WASHOE	1998-08-19T21:00:00.000Z	light	[{}]	NV	{"windCond": {
12	60f6b847a5e4e6647265ae77	HOUSTON	HARRIS	1998-09-23T21:00:00.000Z	fireball	[{}]	TX	{"windCond": {
13	60f6b847a5e4e6647265ae78	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	fireball	[{}]	SC	{"windCond": {
14	60f6b847a5e4e6647265ae79	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	flash	[{}]	SC	{"windCond": {
15	60f6b847a5e4e6647265ae7a	HIGHLAND	LAKE	1998-10-11T20:00:00.000Z	sphere	[{}]	IN	{"windCond": {
16	60f6b847a5e4e6647265ae7b	ASHEVILLE	BUNCOMBE	1998-10-12T19:00:00.000Z	light	[{}]	NC	{"windCond": {
17	60f6b847a5e4e6647265ae7c	KIRKLAND	KING	1998-12-03T23:00:00.000Z	triangle	[{}]	WA	{"windCond": {
18	60f6b847a5e4e6647265ae7d	LAUGHLIN	CLARK	1999-02-27T14:00:00.000Z	formation	[{}]	NV	{"windCond": {
19	60f6b847a5e4e6647265ae7e	GRAND RAPIDS	KENT	1999-03-12T07:00:00.000Z	triangle	[{}]	MI	{"windCond": {

Figure 4: Screenshot of remove year,month,day,hour

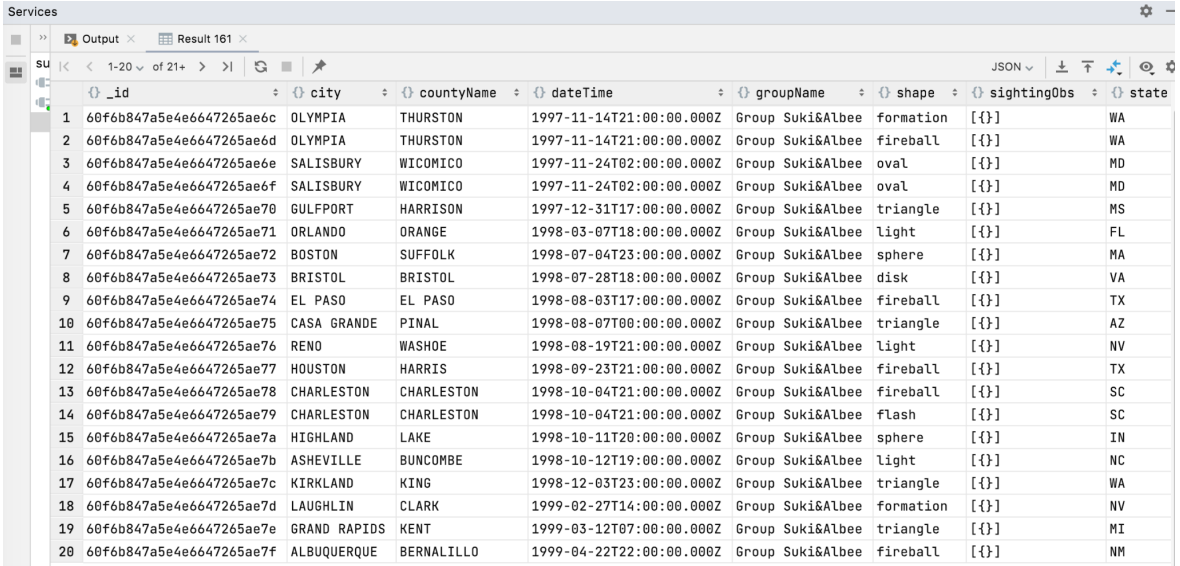
(iv) add another field with your group name (e.g. groupName: "Group FIT5137" if your group name is FIT5137)

Explanation: Using the set function to add group name as a new field into ufo collection. Update all rows, so use updateMany. The format is "groupName": "Group Suki&Albee".

Reference:/

Code:

```
db.ufo.updateMany({},{$set:{"groupName":"Group Suki&Albee"}});
```



Services

Output | Result 161

1-20 of 21+ | JSON

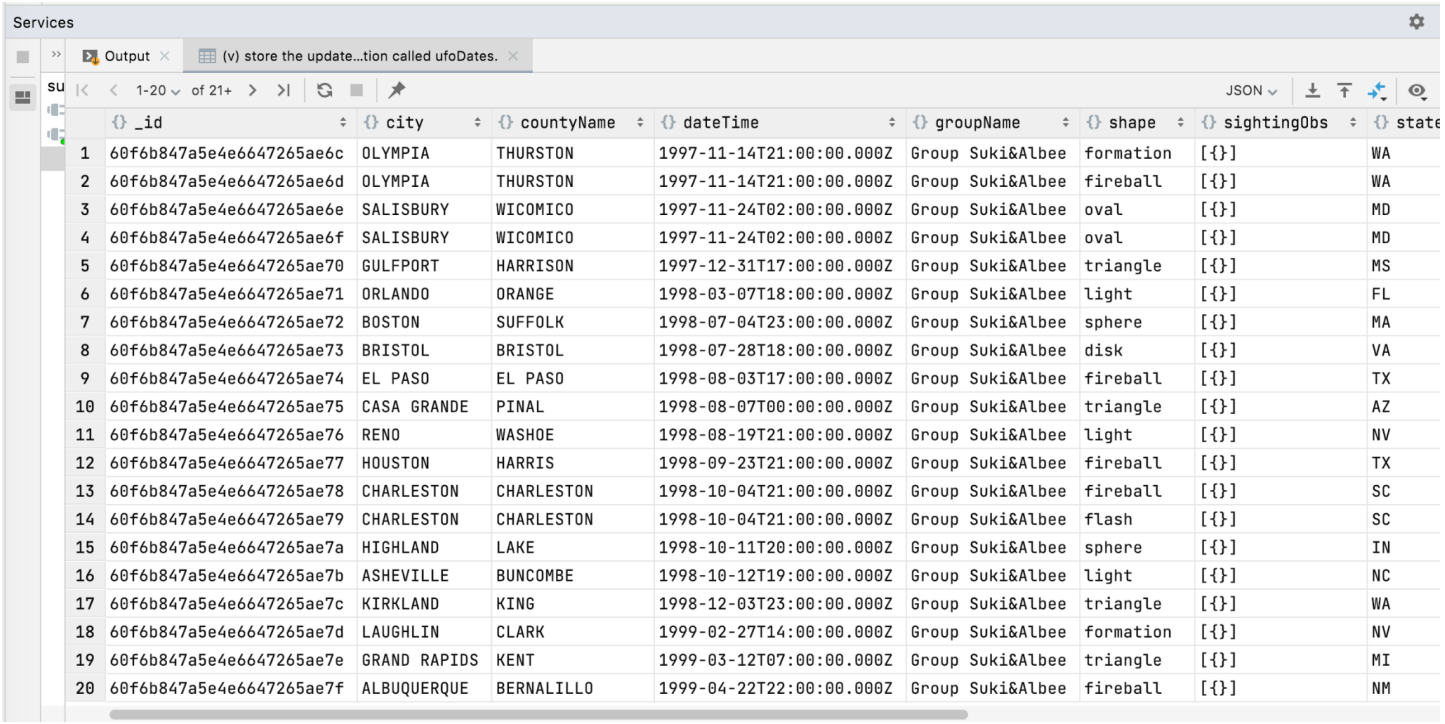
	_id	city	countyName	dateTime	groupName	shape	sightingObs	state
1	60f6b847a5e4e6647265ae6c	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	Group Suki&Albee	formation	[{}]	WA
2	60f6b847a5e4e6647265ae6d	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	Group Suki&Albee	fireball	[{}]	WA
3	60f6b847a5e4e6647265ae6e	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	Group Suki&Albee	oval	[{}]	MD
4	60f6b847a5e4e6647265ae6f	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	Group Suki&Albee	oval	[{}]	MD
5	60f6b847a5e4e6647265ae70	GULFPORT	HARRISON	1997-12-31T17:00:00.000Z	Group Suki&Albee	triangle	[{}]	MS
6	60f6b847a5e4e6647265ae71	ORLANDO	ORANGE	1998-03-07T18:00:00.000Z	Group Suki&Albee	light	[{}]	FL
7	60f6b847a5e4e6647265ae72	BOSTON	SUFFOLK	1998-07-04T23:00:00.000Z	Group Suki&Albee	sphere	[{}]	MA
8	60f6b847a5e4e6647265ae73	BRISTOL	BRISTOL	1998-07-28T18:00:00.000Z	Group Suki&Albee	disk	[{}]	VA
9	60f6b847a5e4e6647265ae74	EL PASO	EL PASO	1998-08-03T17:00:00.000Z	Group Suki&Albee	fireball	[{}]	TX
10	60f6b847a5e4e6647265ae75	CASA GRANDE	PINAL	1998-08-07T00:00:00.000Z	Group Suki&Albee	triangle	[{}]	AZ
11	60f6b847a5e4e6647265ae76	RENO	WASHOE	1998-08-19T21:00:00.000Z	Group Suki&Albee	light	[{}]	NV
12	60f6b847a5e4e6647265ae77	HOUSTON	HARRIS	1998-09-23T21:00:00.000Z	Group Suki&Albee	fireball	[{}]	TX
13	60f6b847a5e4e6647265ae78	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	Group Suki&Albee	fireball	[{}]	SC
14	60f6b847a5e4e6647265ae79	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	Group Suki&Albee	flash	[{}]	SC
15	60f6b847a5e4e6647265ae7a	HIGHLAND	LAKE	1998-10-11T20:00:00.000Z	Group Suki&Albee	sphere	[{}]	IN
16	60f6b847a5e4e6647265ae7b	ASHEVILLE	BUNCOMBE	1998-10-12T19:00:00.000Z	Group Suki&Albee	light	[{}]	NC
17	60f6b847a5e4e6647265ae7c	KIRKLAND	KING	1998-12-03T23:00:00.000Z	Group Suki&Albee	triangle	[{}]	WA
18	60f6b847a5e4e6647265ae7d	LAUGHLIN	CLARK	1999-02-27T14:00:00.000Z	Group Suki&Albee	formation	[{}]	NV
19	60f6b847a5e4e6647265ae7e	GRAND RAPIDS	KENT	1999-03-12T07:00:00.000Z	Group Suki&Albee	triangle	[{}]	MI
20	60f6b847a5e4e6647265ae7f	ALBUQUERQUE	BERNALILLO	1999-04-22T22:00:00.000Z	Group Suki&Albee	fireball	[{}]	NM

Figure 5: Screenshot of add group name field

(v) store the updated collection in a new collection called ufoDates.

Explanation: Combine all date fields as one date field, remove separate date fields, add group name field into ufo collection after all actions use $out to export changed data as a new collection called ufoDates.
Reference:/
Code:

```
db.ufo.aggregate([
{$match:{}},
{$out:"ufoDates"}
]);
```



	_id	city	countyName	dateTime	groupName	shape	sightingObs	state
1	60f6b847a5e4e6647265ae6c	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	Group Suki&Albee	formation	[{}]	WA
2	60f6b847a5e4e6647265ae6d	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	Group Suki&Albee	fireball	[{}]	WA
3	60f6b847a5e4e6647265ae6e	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	Group Suki&Albee	oval	[{}]	MD
4	60f6b847a5e4e6647265ae6f	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	Group Suki&Albee	oval	[{}]	MD
5	60f6b847a5e4e6647265ae70	GULFPORT	HARRISON	1997-12-31T17:00:00.000Z	Group Suki&Albee	triangle	[{}]	MS
6	60f6b847a5e4e6647265ae71	ORLANDO	ORANGE	1998-03-07T18:00:00.000Z	Group Suki&Albee	light	[{}]	FL
7	60f6b847a5e4e6647265ae72	BOSTON	SUFFOLK	1998-07-04T23:00:00.000Z	Group Suki&Albee	sphere	[{}]	MA
8	60f6b847a5e4e6647265ae73	BRISTOL	BRISTOL	1998-07-28T18:00:00.000Z	Group Suki&Albee	disk	[{}]	VA
9	60f6b847a5e4e6647265ae74	EL PASO	EL PASO	1998-08-03T17:00:00.000Z	Group Suki&Albee	fireball	[{}]	TX
10	60f6b847a5e4e6647265ae75	CASA GRANDE	PINAL	1998-08-07T00:00:00.000Z	Group Suki&Albee	triangle	[{}]	AZ
11	60f6b847a5e4e6647265ae76	RENO	WASHOE	1998-08-19T21:00:00.000Z	Group Suki&Albee	light	[{}]	NV
12	60f6b847a5e4e6647265ae77	HOUSTON	HARRIS	1998-09-23T21:00:00.000Z	Group Suki&Albee	fireball	[{}]	TX
13	60f6b847a5e4e6647265ae78	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	Group Suki&Albee	fireball	[{}]	SC
14	60f6b847a5e4e6647265ae79	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	Group Suki&Albee	flash	[{}]	SC
15	60f6b847a5e4e6647265ae7a	HIGHLAND	LAKE	1998-10-11T20:00:00.000Z	Group Suki&Albee	sphere	[{}]	IN
16	60f6b847a5e4e6647265ae7b	ASHEVILLE	BUNCOMBE	1998-10-12T19:00:00.000Z	Group Suki&Albee	light	[{}]	NC
17	60f6b847a5e4e6647265ae7c	KIRKLAND	KING	1998-12-03T23:00:00.000Z	Group Suki&Albee	triangle	[{}]	WA
18	60f6b847a5e4e6647265ae7d	LAUGHLIN	CLARK	1999-02-27T14:00:00.000Z	Group Suki&Albee	formation	[{}]	NV
19	60f6b847a5e4e6647265ae7e	GRAND RAPIDS	KENT	1999-03-12T07:00:00.000Z	Group Suki&Albee	triangle	[{}]	MI
20	60f6b847a5e4e6647265ae7f	ALBUQUERQUE	BERNALILLO	1999-04-22T22:00:00.000Z	Group Suki&Albee	fireball	[{}]	NM

Figure 6: Screenshot of store changed into new collection

C.1.4. Add the sighting with the following data to the ufoDates collection using MongoDB

Explanation: Add given data into ufoDates collection. only one-row data, so use insertOne function and match these values with related fields. dateTime should use new Data() format, sightingObs is a list object contains fields duration,text,summary.
Reference:/
Code:

```
db.ufoDates.insertOne(
{state:"MA",
city:"BOSTON",
countryName:"SUFFOLK",
dateTime:new Date("1998-07-14T23:00:00.000+00:00"),
shape:"sphere",
sightingObs:
[{duration:"40 min",
text:"I was going to my work on my night shift at the St Albin’s hospital and saw an unearthly ray of shooting lights which could be none other than a UFO!",
```

```
summary:"Unearthly ray of shooting lights"}]
});
```

2 rows JSON

	_id	city	countyName	dateTime	groupName	shape	sightingObs
1	60f6b87ba5e4e664726631b1	FRANKLIN FURNACE	SCIOTO	2007-09-04T21:00:00.000Z	Group Suki&Albee	formation	[{"duration": "40 min",
2	613d9f1641b2bb0fd2dbbf01	BOSTON	*unset*	1998-07-14T23:00:00.000Z	*unset*	sphere	[{"duration": "40 min",

Figure 7: Screenshot of add new data

C.1.5. Keeping the location, weather observations and dateTime information, find and remove the sighting observation record which has a duration of "2 1/2 minutes". To find the duration information please look closely at all of the fields of the documents in the ufoDates collection.

Explanation: Find out which row contains information the duration of sighting observation is "2 1/2 minutes" and remove sightingObs field but do not change other fields. Before getting the result, the number of rows is uncertain, so use the updateMany function. After filter data that matching condition, then uses $unset function to remove sightingObs field.
Reference:/
Code:

```
db.ufoDates.updateMany({"sightingObs.duration":"2 1/2 minutes"},{$unset:{"sightingObs":""}});
```

```
FIT5137MASL> db.ufoDates.updateMany({"sightingObs.duration":"2 1/2 minutes"},{$unset:{"sightingObs":""}});
[2021-09-12 16:36:18] completed in 241 ms
```

Figure 8: Screenshot of remove sightingObs

C.1.6. Use the aggregation pipeline to answer the following queries:

(i) What was the total number of sightings observed from the year 1990 to 2000 in the city of 'SAN FRANCISCO', in the state of 'CA'. Your output can be in any format as long as it displays the required information.

Explanation: These questions' conditions are years between 1990 and 2000. The city is SAN FRANCISCO. The state is CA. It needs to calculate the total number of ufo records, so after filtering data, use the group by state and count numbers.
Reference:/
Code:

```
db.ufoDates.aggregate([
{$match:{"city":"SAN FRANCISCO","state":"CA","dateTime":{$gte:ISODate("1990-01-01"),$lt:ISODate("2001-01-01")}}},
{$group:{_id:"$state", totalNumOfSObs:{$sum:1}}}
]);
```

Output (i)

1 row

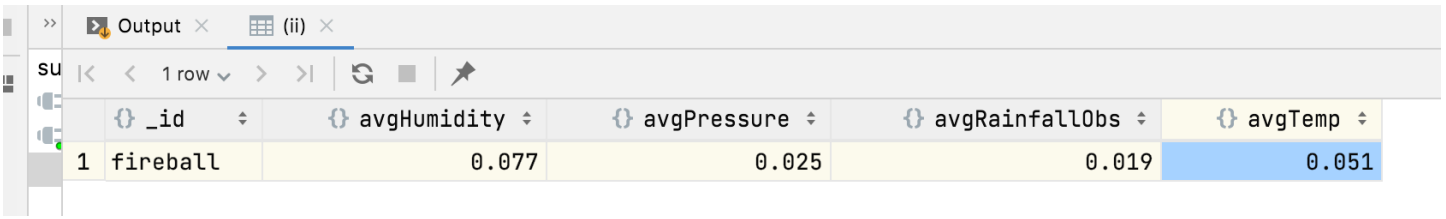
	_id	totalNumOfSObs
1	CA	16

Figure 9: Screenshot of count result

(ii) Using one MongoDB Shell query, find the average temperature, humidity, pressure and rainfall observed for all fireball shaped UFO sightings. The output should also display the average values rounded to 3 decimal places.

Explanation: Condition is the fireball shape ufo that record in the sightingObs field. Group by fireball ufo and calculate average temperature, humidity, pressure and rainfall using $avg, then use $round to leave Three decimal places.
Reference: https://docs.mongodb.com/manual/reference/operator/aggregation/round/
Code:

```
db.ufoDates.aggregate([
{$match:{"shape":"fireball"}},
{$group:{_id:"fireball",
avgTemp:{$avg:"$weatherObs.temp"},
avgHumidity:{$avg:"$weatherObs.hum"},
avgPressure:{$avg:"$weatherObs.pressure"},
avgRainfallObs:{$avg:"$weatherObs.rain"}}},
{$project:{_id:"fireball",
avgTemp:{$round:["$avgTemp",3]},
avgHumidity:{$round:["$avgHumidity",3]},
avgPressure:{$round:["$avgPressure",3]},
avgRainfallObs:{$round:["$avgRainfallObs",3]}}}
]);
```



	_id	avgHumidity	avgPressure	avgRainfallObs	avgTemp
1	fireball	0.077	0.025	0.019	0.051

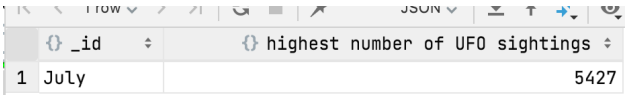
Figure 10: Screenshot of average results

(iii) What was the month with the highest UFO sightings?

Explanation: Use $switch to convert integer type month into month name. group by each month and calculate records' numbers in each month, sort by descending order pick up the first one row.
Reference: https://docs.mongodb.com/manual/reference/operator/aggregation/switch/
Code:

```
db.ufoDates.aggregate([
{$project:{"monthName":{$switch:{branches:[
{case:{$eq:[{$month:"$dateTime"},1]},then:"January"},
{case:{$eq:[{$month:"$dateTime"},2]},then:"February"},
{case:{$eq:[{$month:"$dateTime"},3]},then:"March"},
{case:{$eq:[{$month:"$dateTime"},4]},then:"April"},
{case:{$eq:[{$month:"$dateTime"},5]},then:"May"},
{case:{$eq:[{$month:"$dateTime"},6]},then:"June"},
{case:{$eq:[{$month:"$dateTime"},7]},then:"July"},
{case:{$eq:[{$month:"$dateTime"},8]},then:"August"},
{case:{$eq:[{$month:"$dateTime"},9]},then:"September"},
{case:{$eq:[{$month:"$dateTime"},10]},then:"October"},
{case:{$eq:[{$month:"$dateTime"},11]},then:"November"},
```

```
{case:{$eq:[{$month:"$dateTime"},12]},then:"December"}],
default:"..."}}}},
{$group:{_id:"$monthName", "highest number of UFO sightings":{$sum:1}}},
{$sort:{"highest number of UFO sightings":-1}},
{$limit:1}
]);
```



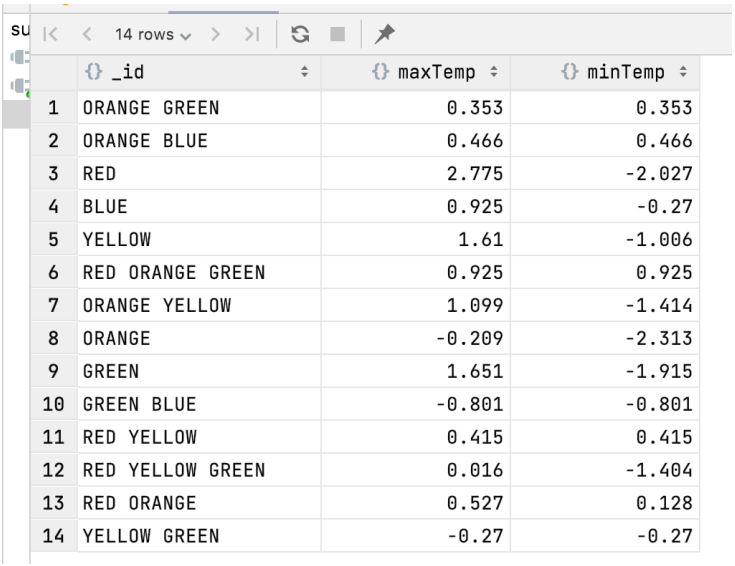
	_id	highest number of UFO sightings
1	July	5427

Figure 11: Screenshot of month result

(iv) What was the maximum and minimum temperatures observed during each colour of UFO observations. Your output should include the name of the colour in uppercase,the temperatures rounded to 3 decimal places and ordered by ascending alphabetical order of the colour names.

Explanation: use $toUpper to change colour to upper case, meanwhile filter null value in the colour field. $max and $min can get minimum and maximum value. Then $round the value with three decimal places.
Reference: https://docs.mongodb.com/manual/reference/operator/aggregation/toUpper/
Code:

```
db.ufoDates.aggregate([
{$group:{_id:{$toUpper:"$colour"},maxTemp:{$max:"$weatherObs.temp"},
minTemp:{$min:"$weatherObs.temp"}}},
{$match:{_id:{$ne:""}}},
{$project:{_id:1,maxTemp:{$round:["$maxTemp",3]},minTemp:{$round:["$minTemp",3]}}}
]);
```



14 rows

	_id	maxTemp	minTemp
1	ORANGE GREEN	0.353	0.353
2	ORANGE BLUE	0.466	0.466
3	RED	2.775	-2.027
4	BLUE	0.925	-0.27
5	YELLOW	1.61	-1.006
6	RED ORANGE GREEN	0.925	0.925
7	ORANGE YELLOW	1.099	-1.414
8	ORANGE	-0.209	-2.313
9	GREEN	1.651	-1.915
10	GREEN BLUE	-0.801	-0.801
11	RED YELLOW	0.415	0.415
12	RED YELLOW GREEN	0.016	-1.404
13	RED ORANGE	0.527	0.128
14	YELLOW GREEN	-0.27	-0.27

Figure 12: Screenshot of maximum and minimum results of each colour

(v) Which wind direction was the most observed during oval-shaped UFO sightings. Your output can be in any format as long as it displays the required information.

Explanation: Condition is oval shape ufo, group by direction in weather's field, and count numbers. Sort the result by descending order and pick up the first row.
Reference:/
Code:

```
db.ufoDates.aggregate([
{$match:{shape:"oval"}},
{$group:{_id:{Direction:"$weatherObs.windCond.wdire"},ObsRecord:{$sum:1}}},
{$sort:{ObsRecord:-1}},
{$limit:1}
]);
```

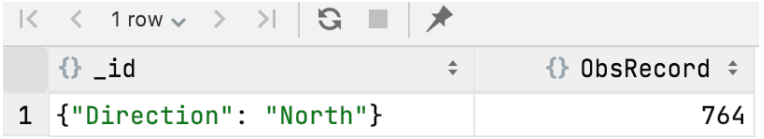


Figure 13: Screenshot of oval result

(vi) With the help of an index search for and then count how many UFO sighting observations contain the word 'light' (ignoring the letter casing), anywhere in the description's text or summary.

Explanation: Create an index for text and summary in the sightingObs field. Then do the following action using $text and $search function to search texts in text and summary which contains light or LIGHT words. for ignoring case sensitive so set false. After the match, the above conditions use count to calculate total numbers.
Reference: https://docs.mongodb.com/manual/reference/operator/query/text/
Code:

```
db.ufoDates.createIndex({"sightingObs.text":"text","sightingObs.summary":"text"});
db.ufoDates.aggregate([
{$match:{$text:{$search:"light", $caseSensitive: false}}},
{$count:"sumOfLight"}
]);
```

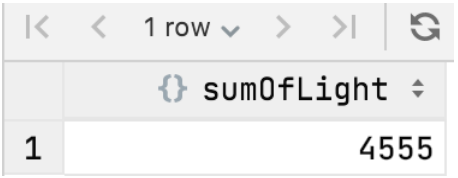


Figure 14: Screenshot of light result

C.1.7.Using MongoDB Compass, create a new collection called states and import the state.csv file. While importing, appropriate data types must be used (i.e. whole number data should be integers, numbers with decimal points should be doubles etc.). Using MongoDB Shell Commands, combine the ufoDates and states collection into a new collection named ufoStates which should include the lat and lng data for each city.

Explanation: Use $lookup function to combine the collection, use $let function to generate a new field with related original data because different states might have the same county name. Hence, I use the city, county name and state as identities to match two collections. Use a pipeline to do multi-condition work. After matching two collections, use $project to filter unneeded fields and save longitude and latitude as one field named

geoDocs. I want to divide latitude and longitude as two separate fields, so I use $unwind to split geoDocs. After that, use the $addField function to add two new location fields into the combined collection, remove geoDocs and generate a new collection to store changed data called ufoStates.
Reference: https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/
Code:

```
db.ufoDates.aggregate([
{$lookup:{from: "states",
let:{"post_city":"$city","post_countyName":"$countyName","post_state":"$state"},
pipeline:[
{$match:
{$expr:{$and:[
{$eq:["$$post_city","$city"]},
{$eq:["$$post_countyName","$countyName"]},
{$eq:["$$post_state","$state"]}]}},
}},
{$project:{_id:0,countyName:0,city:0,state:0}}],
as:"geoDocs"}},
{$unwind:"$geoDocs"},
{$addFields:{"geoLat":"$geoDocs.lat","geoLng":"$geoDocs.lng"}},
{$project:{"geoDocs":0}},
{$out:"ufoStates"}
]);
```

Output | Result 207

1-20 of 21+

	_id	city	countyName	dateTime	geoLat	geoLng	groupName	shape
1	60f6b847a5e4e6647265ae6c	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	47.0417	-122.8959	Group Suki&Albee	formation
2	60f6b847a5e4e6647265ae6d	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	47.0417	-122.8959	Group Suki&Albee	fireball
3	60f6b847a5e4e6647265ae6e	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	38.3756	-75.5867	Group Suki&Albee	oval
4	60f6b847a5e4e6647265ae6f	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	38.3756	-75.5867	Group Suki&Albee	oval
5	60f6b847a5e4e6647265ae70	GULFPORT	HARRISON	1997-12-31T17:00:00.000Z	30.4271	-89.0703	Group Suki&Albee	triangle
6	60f6b847a5e4e6647265ae71	ORLANDO	ORANGE	1998-03-07T18:00:00.000Z	28.4772	-81.3369	Group Suki&Albee	light
7	60f6b847a5e4e6647265ae72	BOSTON	SUFFOLK	1998-07-04T23:00:00.000Z	42.3188	-71.0846	Group Suki&Albee	sphere
8	60f6b847a5e4e6647265ae73	BRISTOL	BRISTOL	1998-07-28T18:00:00.000Z	36.618	-82.1606	Group Suki&Albee	disk
9	60f6b847a5e4e6647265ae74	EL PASO	EL PASO	1998-08-03T17:00:00.000Z	31.8479	-106.4309	Group Suki&Albee	fireball
10	60f6b847a5e4e6647265ae75	CASA GRANDE	PINAL	1998-08-07T00:00:00.000Z	32.9068	-111.7624	Group Suki&Albee	triangle
11	60f6b847a5e4e6647265ae76	RENO	WASHOE	1998-08-19T21:00:00.000Z	39.5497	-119.8483	Group Suki&Albee	light
12	60f6b847a5e4e6647265ae77	HOUSTON	HARRIS	1998-09-23T21:00:00.000Z	29.7863	-95.3889	Group Suki&Albee	fireball
13	60f6b847a5e4e6647265ae78	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	32.8153	-79.9628	Group Suki&Albee	fireball
14	60f6b847a5e4e6647265ae79	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	32.8153	-79.9628	Group Suki&Albee	flash
15	60f6b847a5e4e6647265ae7a	HIGHLAND	LAKE	1998-10-11T20:00:00.000Z	41.5485	-87.4587	Group Suki&Albee	sphere
16	60f6b847a5e4e6647265ae7b	ASHEVILLE	BUNCOMBE	1998-10-12T19:00:00.000Z	35.5704	-82.5536	Group Suki&Albee	light
17	60f6b847a5e4e6647265ae7c	KIRKLAND	KING	1998-12-03T23:00:00.000Z	47.6974	-122.2054	Group Suki&Albee	triangle
18	60f6b847a5e4e6647265ae7d	LAUGHLIN	CLARK	1999-02-27T14:00:00.000Z	35.1316	-114.689	Group Suki&Albee	formation
19	60f6b847a5e4e6647265ae7e	GRAND RAPIDS	KENT	1999-03-12T07:00:00.000Z	42.962	-85.6562	Group Suki&Albee	triangle
20	60f6b847a5e4e6647265ae7f	ALBUQUERQUE	BERNALILLO	1999-04-22T22:00:00.000Z	35.1053	-106.6464	Group Suki&Albee	fireball

Figure 15: Screenshot of lookup two collections

C.1.8. Using Mongo Shell commands, convert the latitude and longitude for each observation in ufoStates to MongoDB GeoJSON objects with location type as a point in a field called location and store the modified data into a new collection called ufoStatesGeojson. Use MongoDB Compass's schema tab to display the location on a map visualisation

Explanation: Add a new field called location into the collection. Use the geospatial function to set location type is the point and set latitude and longitude as geojson object. Create a new collection called ufoStateGeojson to store changed data.
Reference: https://docs.mongodb.com/manual/geospatial-queries/
Code:

```
db.ufoStates.aggregate([
{$addFields:{location:{type:"Point", coordinates:["$geoLng","$geoLat"]}}},
{$project:{"geoLat":0,"geoLng":0}},
{$out:"ufoStatesGeojson"}
]);
```

Output | geo location

1-20 of 21+

	city	countyName	dateTime	groupName	location
1	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-122.8959, 47.0417]}
2	OLYMPIA	THURSTON	1997-11-14T21:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-122.8959, 47.0417]}
3	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-75.5867, 38.3756]}
4	SALISBURY	WICOMICO	1997-11-24T02:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-75.5867, 38.3756]}
5	GULFPORT	HARRISON	1997-12-31T17:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-89.0703, 30.4271]}
6	ORLANDO	ORANGE	1998-03-07T18:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-81.3369, 28.4772]}
7	BOSTON	SUFFOLK	1998-07-04T23:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-71.0846, 42.3188]}
8	BRISTOL	BRISTOL	1998-07-28T18:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-82.1606, 36.618]}
9	EL PASO	EL PASO	1998-08-03T17:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-106.4309, 31.8479]}
10	CASA GRANDE	PINAL	1998-08-07T00:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-111.7624, 32.9068]}
11	RENO	WASHOE	1998-08-19T21:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-119.8483, 39.5497]}
12	HOUSTON	HARRIS	1998-09-23T21:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-95.3889, 29.7863]}
13	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-79.9628, 32.8153]}
14	CHARLESTON	CHARLESTON	1998-10-04T21:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-79.9628, 32.8153]}
15	HIGHLAND	LAKE	1998-10-11T20:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-87.4587, 41.5485]}
16	ASHEVILLE	BUNCOMBE	1998-10-12T19:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-82.5536, 35.5704]}
17	KIRKLAND	KING	1998-12-03T23:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-122.2054, 47.6974]}
18	LAUGHLIN	CLARK	1999-02-27T14:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-114.689, 35.1316]}
19	GRAND RAPIDS	KENT	1999-03-12T07:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-85.6562, 42.962]}
20	ALBUQUERQUE	BERNALILLO	1999-04-22T22:00:00.000Z	Group Suki&Albee	{"type": "Point", "coordinates": [-106.6464, 35.1053]}

Figure 16: Screenshot of geojson result with point type

Map visualization in Mongodb Compass schema.

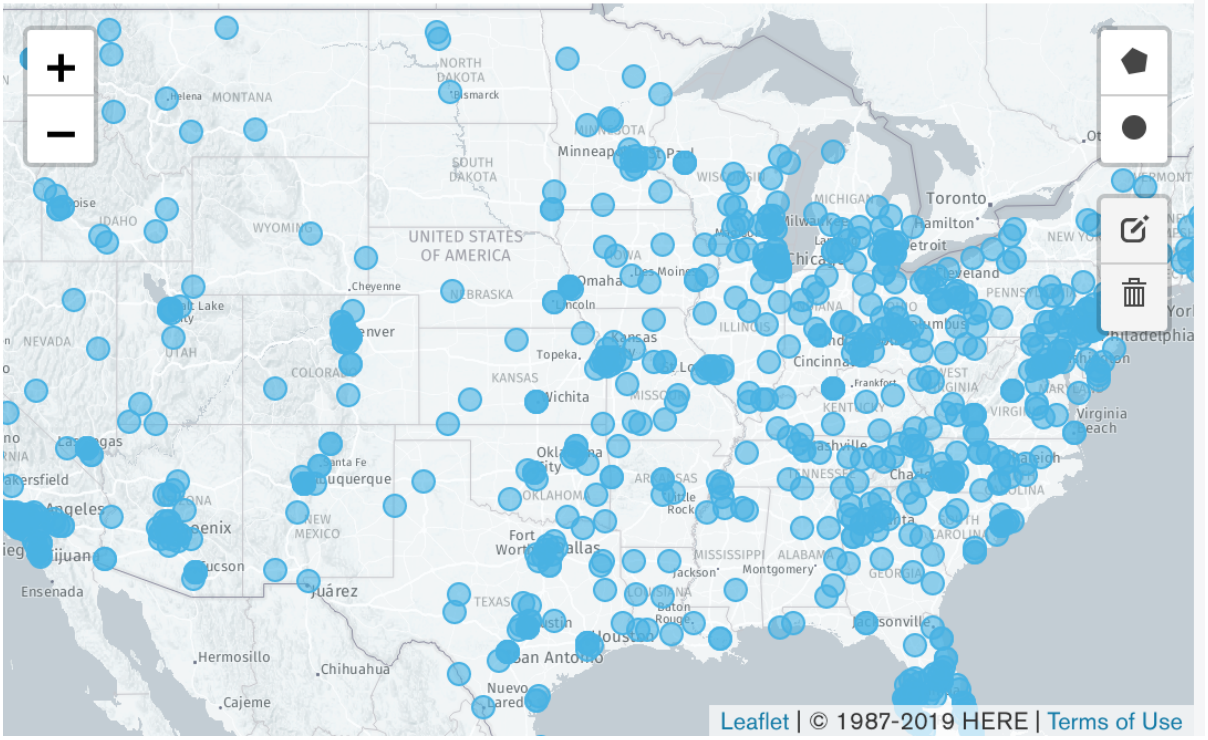


Figure 17: Map visualization in MongoDB Compass

C.1.9. Using MongoDB's Geospatial Query Operators and MongoDB shell commands, find the list of cities within 10km and 100km of the location with the most UFO sightings. The list should not show duplicate city names. If required you can split this into more than one query.

Explanation: Create an index for Coordinates in order to search coordinates in the collection. Use geospatial query function $geoNear to search points near our set point and range within 10000 to 100000m. One thing that needs to be considered is that different states might have the same city name, so use group by with city and state as identities to avoid repeated city names in different states.
Reference: https://docs.mongodb.com/manual/geospatial-queries/
Code:

```
db.ufoStatesGeojson.createIndex({"location.coordinates":"2dsphere"});
db.ufoStatesGeojson.aggregate([
{$geoNear:{near: { type: "Point", coordinates: [-112.0891, 33.5722] },
        spherical:true,
        key:"location.coordinates",
        distanceField:"distance",
        minDistance:10000,
        maxDistance:100000}},
        {$group:{_id:{"city":"$city","countyName":"$countyName","state":"$state"}}}]);
```

Output | Result 246

1-20 of 41

	{} _id
1	{"city": "MARICOPA", "countyName": "PINAL", "state": "AZ"}
2	{"city": "COOLIDGE", "countyName": "PINAL", "state": "AZ"}
3	{"city": "SUPERIOR", "countyName": "PINAL", "state": "AZ"}
4	{"city": "TONOPAH", "countyName": "MARICOPA", "state": "AZ"}
5	{"city": "GLENDALE", "countyName": "MARICOPA", "state": "AZ"}
6	{"city": "PEORIA", "countyName": "MARICOPA", "state": "AZ"}
7	{"city": "SUN CITY WEST", "countyName": "MARICOPA", "state": "AZ"}
8	{"city": "MAYER", "countyName": "YAVAPAI", "state": "AZ"}
9	{"city": "BUCKEYE", "countyName": "MARICOPA", "state": "AZ"}
10	{"city": "GILA BEND", "countyName": "MARICOPA", "state": "AZ"}
11	{"city": "MORRISTOWN", "countyName": "MARICOPA", "state": "AZ"}
12	{"city": "WICKENBURG", "countyName": "MARICOPA", "state": "AZ"}
13	{"city": "NEW RIVER", "countyName": "MARICOPA", "state": "AZ"}
14	{"city": "FOUNTAIN HILLS", "countyName": "MARICOPA", "state": "AZ"}
15	{"city": "SCOTTSDALE", "countyName": "MARICOPA", "state": "AZ"}
16	{"city": "RIO VERDE", "countyName": "MARICOPA", "state": "AZ"}
17	{"city": "SAN TAN VALLEY", "countyName": "PINAL", "state": "AZ"}
18	{"city": "GOODYEAR", "countyName": "MARICOPA", "state": "AZ"}
19	{"city": "CHANDLER", "countyName": "MARICOPA", "state": "AZ"}
20	{"city": "SUN CITY", "countyName": "MARICOPA", "state": "AZ"}

Figure 18: Screenshot of Cities in minimum and maximum radius distances

C.1.10.Using MongoDB Compass, please export the ufo collection as a CSV file named ufo.csv to use in Task
Explanation: Export ufo collection to ufo.csv.
Reference:/
Code: /

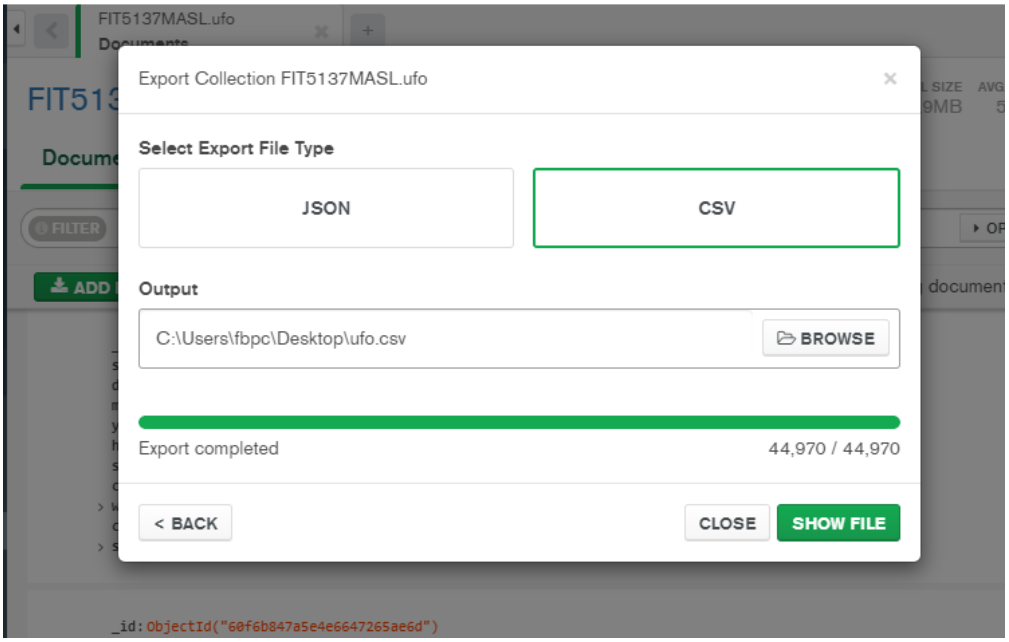


Figure 19: Screenshot of export ufo.csv

C.2 Cassandra

C.2.1. Create a keyspace called FIT5137_MASL for the Cassandra database, with SimpleStrategy and replication factor of 1.
Explanation: create keyspace
Reference:/
Code:

```
CREATE KEYSPACE FIT5137_MASL WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};
```

```
cqlsh:fit5137_masl> CREATE KEYSPACE FIT5137_MASL WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};
cqlsh:fit5137_masl> describe Keyspaces;

books_keyspace     my_keyspace  system_auth         system_traces
fit5137_masl       suki         system_distributed  system_views
my_first_keyspace  system       system_schema       system_virtual_schema
```

Figure 20: Screenshot of create keyspace

C.2.2. Using the Cassandra COPY command, import the ufo.csv data into the ufos table using custom data types for weatherObs and each element of sightingObs.
Explanation: create udt sightingObs, then create table. Group cleaned original ufo.csv, combine each columns but split each columns' names and values by symbol |.So when copy csv file to cassandra table, it can use delimiter = '|'. create table ufos, all columns are matching with my cleaned ufo.csv. Use UDT for sightingObs. Use state and month as partition keys, because one question need to group data by state and month. There are many different years, days and hours, country names, cities, so I add these columns into composite key as a part of the primary key.
Reference: https://docs.datastax.com/en/dse/5.1/cql/cql/cql_using/useInsertCopyCSV.html
Code:

```
CREATE TYPE sightingObs_type
(
duration text,
text text,
summary text,
);

CREATE TABLE ufos
(
   id text,
   state text,
   day int,
   month int,
   year int,
   hour int,
   shape text,
   city text,
   weatherObs_windchill double,
```

```
    weatherObs_wdire text,
    weatherObs_wspd double,
    weatherObs_pressure double,
    weatherObs_temp double,
    weatherObs_hail int,
    weatherObs_rain int,
    weatherObs_vis double,
    weatherObs_dewpt double,
    weatherObs_thunder int,
    weatherObs_fog int,
    weatherObs_tornado int,
    weatherObs_hum double,
    weatherObs_snow int,
    weatherObs_conds text,
    countyName text,
    sightingObs list<frozen<sightingObs_type>>,
    PRIMARY KEY((month,state),day,city,countyName,hour,year)
);
COPY FIT5137_MASL.ufos(id,state,day,month,year,hour,shape,city,weatherObs_windchill,
weatherObs_wdire,weatherObs_wspd,weatherObs_pressure,weatherObs_temp,weatherObs_hail,weatherObs_rai
n,weatherObs_vis,weatherObs_dewpt,weatherObs_thunder,weatherObs_fog,weatherObs_tornado,weatherObs_h
um,weatherObs_snow,weatherObs_conds,countyName,sightingObs)
FROM '/Users/lpylocal/Desktop/2021 semester 2/5137/assignment1/output/newufo.csv' WITH HEADER =
TRUE AND DELIMITER='|';
```

```
cqlsh:fit5137_masl> drop keyspace fit5137_masl;
cqlsh:fit5137_masl> CREATE KEYSPACE FIT5137_MASL WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};
cqlsh:fit5137_masl> CREATE TYPE sightingObs_type
                ... (
                ... duration text,
                ... text text,
                ... summary text,
                ... );
cqlsh:fit5137_masl> CREATE TABLE ufos
                ... (
                ...     id text,
                ...     state text,
                ...     day int,
                ...     month int,
                ...     year int,
                ...     hour int,
                ...     shape text,
                ...     city text,
                ...     weatherObs_windchill double,
                ...     weatherObs_wdire text,
                ...     weatherObs_wspd double,
                ...     weatherObs_pressure double,
                ...     weatherObs_temp double,
                ...     weatherObs_hail int,
                ...     weatherObs_rain int,
                ...     weatherObs_vis double,
                ...     weatherObs_dewpt double,
                ...     weatherObs_thunder int,
                ...     weatherObs_fog int,
                ...     weatherObs_tornado int,
                ...     weatherObs_hum double,
                ...     weatherObs_snow int,
                ...     weatherObs_conds text,
                ...     countyName text,
                ...     sightingObs list<frozen<sightingObs_type>>,
                ...     PRIMARY KEY((month,state),day,city,countyName,hour,year)
                ... );
cqlsh:fit5137_masl> COPY FIT5137_MASL.ufos(id,state,day,month,year,hour,shape,city,weatherObs_windchill,weatherObs_wdire,weatherObs_wspd,weatherObs_pressure,weathe
rObs_temp,weatherObs_hail,weatherObs_rain,weatherObs_vis,weatherObs_dewpt,weatherObs_thunder,weatherObs_fog,weatherObs_tornado,weatherObs_hum,weatherObs_snow,weath
erObs_conds,countyName,sightingObs)
                ... FROM '/Users/lpylocal/Desktop/2021 semester 2/5137/assignment1/output/newufo.csv' WITH HEADER = TRUE AND DELIMITER='|';
Using 3 child processes

Starting copy of fit5137_masl.ufos with columns [id, state, day, month, year, hour, shape, city, weatherobs_windchill, weatherobs_wdire, weatherobs_wspd, weatherob
s_pressure, weatherobs_temp, weatherobs_hail, weatherobs_rain, weatherobs_vis, weatherobs_dewpt, weatherobs_thunder, weatherobs_fog, weatherobs_tornado, weatherobs
_hum, weatherobs_snow, weatherobs_conds, countyname, sightingobs].
Processed: 44970 rows; Rate:    5345 rows/s; Avg. rate:    5066 rows/s
44970 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 8.877 seconds (0 skipped).
```

Figure 21: Screenshot of create table

C.2.3. Use Cassandra shell to answer the following queries:

(i) How many UFO sightings were recorded in the database for each month and each state?
Explanation: There are two partition keys, month and state, in my table. So for this question, it can be deal with GROUP BY month and state. Select all UFO records, use COUNT to calculate numbers, and save the count result as a new name.
Reference: https://stackoverflow.com/questions/53453405/cassandra-equivalent-of-group-by
Code:

```
SELECT COUNT(*) AS numOfUfo,month,state FROM ufos GROUP BY month,state;
```

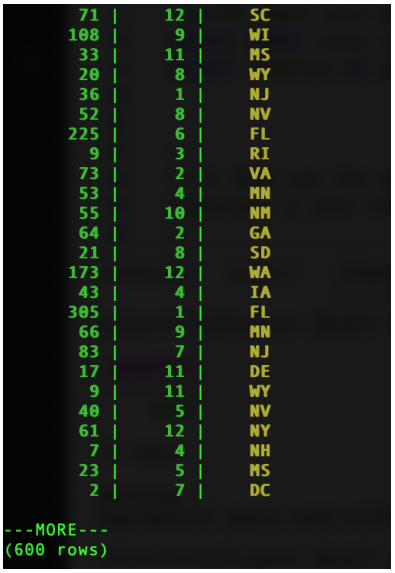


Figure 22: Screenshot of total number of UFO sightings for each month and each state

(ii) How many UFO sightings were recorded during 'Overcast' weather conditions?
Explanation: Condition is Overcast cond, select all records related to this condition and summarise numbers.
Reference: https://stackoverflow.com/questions/37114455/reading-error-in-cassandra
Code:

```
CREATE INDEX conds_index on FIT5137_MASL.ufos(weatherObs_conds);

SELECT COUNT(*) AS numOfUfo,weatherObs_conds  FROM ufos WHERE weatherObs_conds = 'Overcast' ALLOW FILTERING;
```

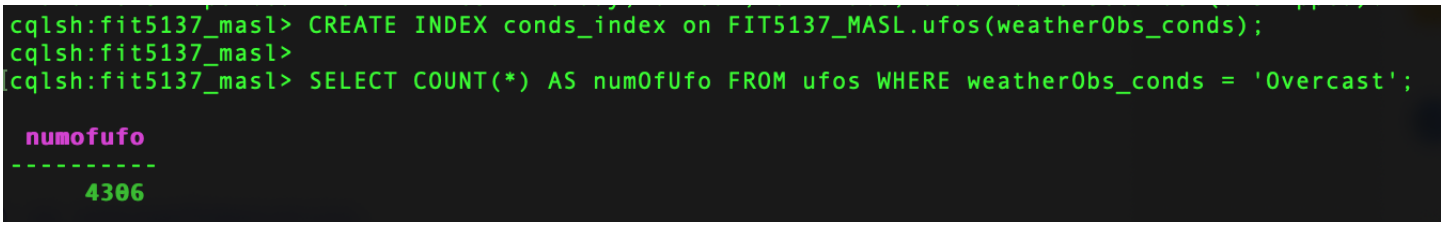


Figure 23: Screenshot of Overcast result

(iii) What was the average temperature, pressure and humidity during the year 2000?
Explanation: Condition is year =2000. Use avg() to calculate average temperature, pressure and humidity.
Reference: https://stackoverflow.com/questions/37114455/reading-error-in-cassandra
Code:

```
CREATE INDEX year_index on FIT5137_MASL.ufos(year);
```

```
SELECT year,AVG(weatherObs_temp) as averageTemperature,
AVG(weatherObs_pressure) as averagePressure,
AVG(weatherObs_hum) as averageHumidity
FROM ufos where year = 2000 ALLOW FILTERING;
```

```
cqlsh:fit5137_masl> CREATE INDEX year_index on FIT5137_MASL.ufos(year);
cqlsh:fit5137_masl>
cqlsh:fit5137_masl>
cqlsh:fit5137_masl> SELECT year,AVG(weatherObs_temp) as averageTemperature,
                ... AVG(weatherObs_pressure) as averagePressure,
                ... AVG(weatherObs_hum) as averageHumidity
                ... FROM ufos where year = 2000;

 year | averagetemperature | averagepressure | averagehumidity
------+--------------------+-----------------+-----------------
 2000 |           -0.11292 |       -0.115885 |        0.054391

(1 rows)
```

Figure 24: Screenshot of query result

C.2.4. Add the following UFO sightings to the database.

Explanation: Select information about details of weather record about that observed during 11th October 1998 10PM, after getting necessary weather data, input new data into ufos.

Reference:/

Code:

```
SELECT
weatherObs_windchill,weatherObs_wdire,weatherObs_wspd,weatherObs_pressure,weatherObs_temp,weatherObs_hail,
weatherObs_rain,weatherObs_vis,weatherObs_dewpt,weatherObs_thunder,weatherObs_fog,weatherObs_tornado,weatherObs_hum,
weatherObs_snow,weatherObs_conds FROM ufos WHERE day = 11 and month =10 and year = 1998 and hour = 22 ALLOW FILTERING;

INSERT INTO ufos (sightingObs,day,month,year,hour,city,countyName,state,weatherObs_wdire,
weatherObs_wspd,weatherObs_pressure,weatherObs_temp,weatherObs_hail,weatherObs_rain,weatherObs_vis,
weatherObs_dewpt,weatherObs_thunder,weatherObs_fog,weatherObs_tornado,weatherObs_hum,weatherObs_snow,weatherObs_conds)
VALUES([{duration:'25 minutes',text:'Awesome lights were seen in the sky',summary:'Awesome lights'}],14,1,2021,23,
'HIGHLAND','LAKE','IN','North',-0.911239,-0.372984,-0.20865,0,0,0.64272,-0.090929,0,0,0,0.02922,0,'Clear');
```

```
cqlsh:fit5137_masl> SELECT weatherObs_windchill,weatherObs_wdire,weatherObs_wspd,weatherObs_pressure,weatherObs_temp,weatherObs_hail,
                ... weatherObs_rain,weatherObs_vis,weatherObs_dewpt,weatherObs_thunder,weatherObs_fog,weatherObs_tornado,weatherObs_hum,
                ... weatherObs_snow,weatherObs_conds FROM ufos WHERE day = 11 and month =10 and year = 1998 and hour = 22 ALLOW FILTERING;

 weatherobs_windchill | weatherobs_wdire | weatherobs_wspd | weatherobs_pressure | weatherobs_temp | weatherobs_hail | weatherobs_rain | weatherobs_
vis | weatherobs_dewpt | weatherobs_thunder | weatherobs_fog | weatherobs_tornado | weatherobs_hum | weatherobs_snow | weatherobs_conds
----------------------+------------------+-----------------+---------------------+-----------------+-----------------+-----------------+------------
----+------------------+--------------------+----------------+--------------------+----------------+-----------------+-----------------

                 null |            North |       -0.911239 |           -0.372984 |        -0.20865 |               0 |               0 |       0.064
272 |        -0.090929 |                0 |               0 |                   0 |         0.02922 |               0 |           Clear

(1 rows)
cqlsh:fit5137_masl>
cqlsh:fit5137_masl> INSERT INTO ufos (sightingObs,day,month,year,hour,city,countyName,state,weatherObs_wdire,
                ... weatherObs_wspd,weatherObs_pressure,weatherObs_temp,weatherObs_hail,weatherObs_rain,weatherObs_vis,
                ... weatherObs_dewpt,weatherObs_thunder,weatherObs_fog,weatherObs_tornado,weatherObs_hum,weatherObs_snow,weatherObs_conds)
                ... VALUES([{duration:'25 minutes',text:'Awesome lights were seen in the sky',summary:'Awesome lights'}],14,1,2021,23,
                ... 'HIGHLAND','LAKE','IN','North',-0.911239,-0.372984,-0.20865,0,0,0.64272,-0.090929,0,0,0,0.02922,0,'Clear');
```

Figure 25: Screenshot of query and insert

C.3 Reflections

C.3.1. Select three real-world examples where each database is used and analyse them by providing the following: Relational Database, Document-Oriented Database and Column-Oriented Database.

(i) in your own words provide an explanation of how each type of database works using your selected examples.

Tabular 1: Three real-word examples

	Relational database	Document-oriented database	Column-oriented database
Real-world example	Facebook(MySQL)	Ebay (Mongodb)	Uber (Cassandra)

Tabular 2: Working mechanism of examples

Working mechanism	
Facebook (MySQL)	MySQL is a relational database software that is great for storing data that requires ACID characteristics, for example a financial transaction. The core programming language sql is widely used since it is great at accessing and organizing data, and more importantly it is easy to use. MySQL is amenable to automation including backup, schema changes and failover, making it easy for a small team to manage thousands of MySQL servers while providing high-quality service(Matsunobu, 2021). Facebook is utilising MySQL for storing social data. The data involved with users' profiles are more than just a list of attributes, it is actually a social graph considering not only basic information like age and name are involved, but much more information such as friends' behaviour, group activity and shares are involved too. And these social graphs are powered by MySQL. The MySQL storage engine provides high reliability and performance. Together with InnoDB, it provides the optimal environment for online transaction processing, with high efficiency and concurrency.
Ebay (Mongodb)	As an e-commerce brand with a wide range of users, eBay has many user operation information and product information every day. eBay includes a large number of classified data of merchant shops and

	products. Users use the search function to find their favourite products and businesses. When the user enters the keyword, eBay has already returned the suggested product name. Users can choose the product they are going to search from among the products listed on eBay or continue to enter the product's name. eBay must store a large amount of product data and return the data to customers in a short time when users inquire. So eBay uses MongoDB files to store and return data. MongoDB uses a replica set. When the primary data centre fails, the database cluster can establish connections between the remaining data centres and obtain data. Storing the data in memory improves the speed of the query. Furthermore, the data in the shopping cart is in JSON format. MongoDB is very good at processing JSON format.
Uber (Cassandra)	Uber's primary business is to provide software taxi services for customers and provide location data for drivers and passengers. When drivers and customers use the Uber App, Uber will constantly refresh the location data about once every 30 seconds(Todd, 2016). Furthermore, this data will be stored in the Uber server. In addition to location data, Uber also needs to process real-time data messages, such as text messages between drivers and passengers. So in the face of massive real-time data, Uber uses Cassandra as the core of the database. Uber must preserve the accuracy of data requests, so Uber's database system is built based on multiple data centres(Abhishek, 2016), and Cassandra meets the requirements for processing massive amounts of data across data centres. Uber combines the Mesos data centre operating system and Cassandra, turning the data centre into a single resource pool with multiple Cassandra machines(Todd, 2016). Set up clusters in various countries to realize cross-border data sharing. Use Cassandra cluster to store driver and passenger location information and refresh it in real-time. If there is a problem with one of the Cassandra nodes, the other node can provide the task(Lakshman & Malik, 2010). When adding a new node, Cassandra can also use the gossip protocol to roll out automatically. Repairing the Cassandra node will not affect the work of other nodes.

Reference (APA7)

Abhishek, V. (2016, September). *Cassandra on Mesos Across Multiple Datacenters at Uber (Abhishek Verma) | C* Summit 2016*[Video]. Youtube. https://youtu.be/4Ap-1VT2ChU

Facebook Database [Updated] – A Thorough Insight Into The Databases Used @Facebook - scaleyourapp.com. scale your app.com. (2021). Retrieved 13 September 2021, from https://www.scaleyourapp.com/what-database-does-facebook-use-a-1000-feet-deep-dive/.

Lakshman, A. & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, *44*(2), 35–40. https://doi.org/10.1145/1773912.1773922

Matsunobu, Y. (2021). *MyRocks: A space- and write-optimized MySQL database*. Facebook Engineering. Retrieved 13 September 2021, from https://engineering.fb.com/2016/08/31/core-data/myrocks-a-space-and-write-optimized-mysql-database/.

Todd, H. (2016,September). *How Uber Manages a Million Writes Per Second Using Mesos and Cassandra Across Multiple Datacenters*: http://highscalability.com/blog/2016/9/28/how-uber-manages-a-million-writes-per-second-using-mesos-and.html

Hows, D., Membrey, P. & Plugge, E. (2014). *MongoDB Basics*. 19–36. https://doi.org/10.1007/978-1-4842-0895-3_2

Harkins, S. S. & Reid, M. W. P. (2002). *SQL: Access to SQL Server*. 397–440. https://doi.org/10.1007/978-1-4302-1573-8_15

Venkatesh, R. (2017, January). *How eBay's Shopping Cart used compression techniques to solve network I/O bottlenecks?* https://tech.ebayinc.com/engineering/how-ebays-shopping-cart-used-compression-techniques-to-solve-network-io-bottlenecks/

Mat, K. (2017 September). *eBay: Building Mission-Critical Multi-Data Center Applications with MongoDB.* https://www.mongodb.com/blog/post/ebay-building-mission-critical-multi-data-center-applications-with-mongodb

(ii) with the help of your selected examples provide a comparison in a tabular format with details on the main strengths and weaknesses of each database.

Tabular 3: Weakness and strengths of each database

	Relational database	Document-oriented database	Column-oriented database
Disadvantages	• Since complex searching can be performed, deadlock sometimes occurs • There is a high number of concurrent instructions, so it will bring pressure to I/O. • Cannot store complex data • Unstructured data like videos or photos are not ideal for sql system	• Does not have a uniformed format • References between documents will cause pressure on the system since the relations are complex	• Complex way to do join table • Since data are stored on disc, the write speed will be decreased • Searching for multiple columns might be slow • Not that mature compared to sql
Advantages	• Stored in table format, it is a uniformed format, data consistency is maintained • Easy to understand • Easy to sort and use • It can be used for searching in a complex way	• Based on key-value pair, so it utilizes data coupling • More flexible comparing to relational database which requires the field to exist even when no information is present • Easier to integrate new information • Can hold huge databases	• Aggregation query is performed faster • Scale horizontally rather than vertically, so it is easier and more economical to upgrade the system

Reference (APA7)

Document Database {Definition, Features, Use Cases}. Knowledge Base by phoenixNAP. (2021). Retrieved 13 September 2021, from https://phoenixnap.com/kb/document-database.

Drake, M. (2021). *A Comparison of NoSQL Database Management Systems and Models | DigitalOcean*. DigitalOcean. Retrieved 13 September 2021, from https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models.

Relational Versus Document Model: advantages and disadvantages. Social net: Databases Oracle, MySQL, programming SQL, Java, APEX, administration. (2021). Retrieved 13 September 2021, from https://oracle-patches.com/en/databases/relational-model-versus-document-model.

Relational vs. Non-Relational Database: Pros & Cons | The Aloa Blog. Aloa.co. (2021). Retrieved 13 September 2021, from https://aloa.co/blog/relational-vs-non-relational-database-pros-cons.

Weber, A. (2021). *What's Unique About a Columnar Database?*. Flydata.com. Retrieved 13 September 2021, from https://www.flydata.com/blog/whats-unique-about-a-columnar-database/.

C.3.2. Decide which database to use (either MongoDB or Cassandra) and provide a non-generic explanation specifically valid for MASL on why you have selected one database over the other.

By observing the data file, it is found that MongoDB's BSON data format is more suitable for storage and querying.

1. One of the most important requirements is to do aggregations and output to new files on the data. This can be quickly done with MongoDB with its built-in features of aggregations. However, Apache Cassandra does not have any built-in aggregations framework, so it will be hard to deal with when it comes to complex aggregation. Thus it will be better for us to utilize MongoDB for the aggregation function.
2. Moreover, MongoDB supports multiple file formats such as JSON CSV, while Cassandra only supports CSV files.
3. Another reason for choosing MongoDB is that it is flexible in its data formats. For example, the MASL requires us to add fields for some documents. MongoDB can provide a flexible and agile form of data structure by adding different fields for different documents. The data structure of MongoDB is designed for easy modifications. While for Cassandra, it is relatively more complex to add new fields or tables since the data format is structured and data types have to be defined first.
4. Furthermore, importing data into MongoDB is easier than Cassandra. MongoDB imports data by directly selecting from software with UI and clear instructions and no coding is required. While for Cassandra, coding is required when importing data.

Reference

[Infographic] Comparing Cassandra vs. MongoDB. ScaleGrid Blog - Fully Managed Cloud Databases Tips. (2021). Retrieved 13 September 2021, from https://scalegrid.io/blog/infographic-comparing-cassandra-vs-mongodb/

Chhabra, M., Tiwari, R., & Amber, S. (2021). *Apache Cassandra vs MongoDB: A Comprehensive Analysis*. Learn | Hevo. Retrieved 13 September 2021, from https://hevodata.com/learn/apache-cassandra-vs-mongodb/

MongoDB vs. Cassandra 2021: Choose Your Winner. Fulcrum Blog. (2021). Retrieved 13 September 2021, from https://fulcrum.rocks/blog/mongodb-vs-cassandra/

C.4 Connecting to Drivers

(i) List of steps to take in order to connect MongoDB and Cassandra drivers.

Explanation: connect to mongodb
Reference:
https://www.patricia-anong.com/blog/pycharm,https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb
Code:

1. Install pymongo in terminal

   ```
   python -m pip install pymongo
   ```

2. Open up python

   ```
   Python
   ```

3. Import pymongo

   ```
   Import pymongo
   ```

4. Connect to local host

   ```
   client = pymongo.MongoClient('localhost', 27017)
   ```

5. Connect to database

   ```
   database = client['FIT5137MASL']
   ```

6. Connect to collection

   ```
   ufo = database['ufo']
   ```

EXplanation: connect to cassandra
Reference: https://stackoverflow.com/questions/59366216/import-csv-file-in-cassandra-using-python-script
Code:

1. Download cassandra-driver

   ```
   python -m pip install cassandra-driver
   ```

2. Use python in terminal and Import Cluster

   ```
   from cassandra.cluster import Cluster
   ```

3. Use local Cluster

   ```
   cluster = Cluster(['localhost'])
   ```

4. Connect a new session

   ```
   session = cluster.connect()
   ```

5. Set keyspace to connect to local keyspace

   ```
   session.set_keyspace('FIT5137_MASL')
   ```

6. Display data

   ```
   rows = session.execute("SELECT * FROM ufos")
   ```

```
rows.current_rows
```

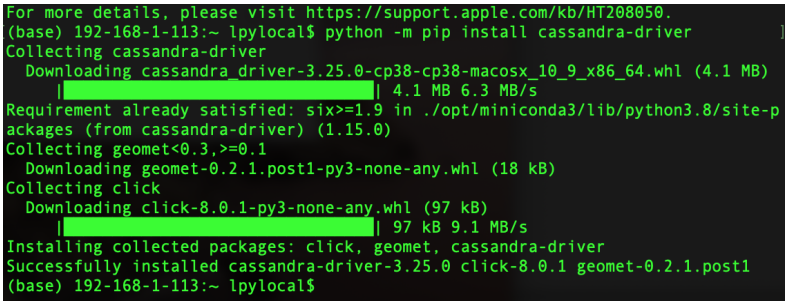


Figure 26: python install cassandra driver

(ii) Two properly commented driver script files i.e. one for MongoDB (named e.g. C4_MongoDB.py) and another for Cassandra (named e.g. C4_Cassandra.py).

```
Files in zip :)
```