MongoDB Lab

Introduction.

MongoDB is NoSQL databases for high volume data and modern applications. It features JSON document like structure and has a flexible schema unlike SQL. This means that each set of documents (similar to rows in SQL) does not need to have same set of fields and data type for a field can vary across documents in a collection. This gives us advantage when there is frequently changing schema based on the requirements. In addition, it also has powerful query language as we will see later.

In this tutorial we will go through various commands of MongoDB to get acquainted with the language and to better understand the features it offers

Problem statement:

We will attempt to answer questions related to airline flights performance. In doing so, we will be cleaning up the data, gather flight statistics and look at how the data is distributed.

Dataset:

The dataset used for this tutorial is on time performance of domestic flights operated by different carriers in the United States in the year 2015. The data was collected from Bureau of Transportation Statistics of US department of Transportation. It is approximately ~0.6 GB with 5.8 million rows. The data consists of 3 files namely:

- Flights.csv *(containing all flight times, departure time, arrival time, delay, reason for delay)*
- Airlines.csv *(Containing airlines and their respective code)*
- Airports.csv *(Containing airports names and location data)*

This dataset consists of the following columns

Flights.csv	
Column	Description
YEAR	Year of the Flight Trip
MONTH	Month of the Flight Trip
DAY	Day of the Flight Trip
DAY_OF_WEEK	Day of week of the Flight Trip
AIRLINE	Airline Identifier
FLIGHT_NUMBER	Flight Identifier
TAIL_NUMBER	Aircraft Identifier
ORIGIN_AIRPORT	Starting Airport
DESTINATION_AIRPORT	Destination Airport
SCHEDULED_DEPARTURE	Planned Departure Time
DEPARTURE_TIME	WHEEL_OFF - TAXI_OUT
DEPARTURE_DELAY	Total Delay on Departure
TAXI_OUT	The time duration elapsed between departure from the origin airport gate and wheels off
WHEELS_OFF	The time point that the aircraft's wheels leave the ground
SCHEDULED_TIME	Planned time amount needed for the flight trip
ELAPSED_TIME	AIR_TIME+TAXI_IN+TAXI_OUT
AIR_TIME	The time duration between wheels_off and wheels_on time
DISTANCE	Distance between two airports
WHEELS_ON	The time point that the aircraft's wheels touch on the ground
TAXI_IN	The time duration elapsed between wheels-on and gate arrival at the destination airport
SCHEDULED_ARRIVAL	Planned arrival time
ARRIVAL_TIME	WHEELS_ON+TAXI_IN
ARRIVAL_DELAY	ARRIVAL_TIME-SCHEDULED_ARRIVAL
DIVERTED	Aircraft landed on airport that out of schedule
CANCELLED	Flight Cancelled (1 = cancelled)

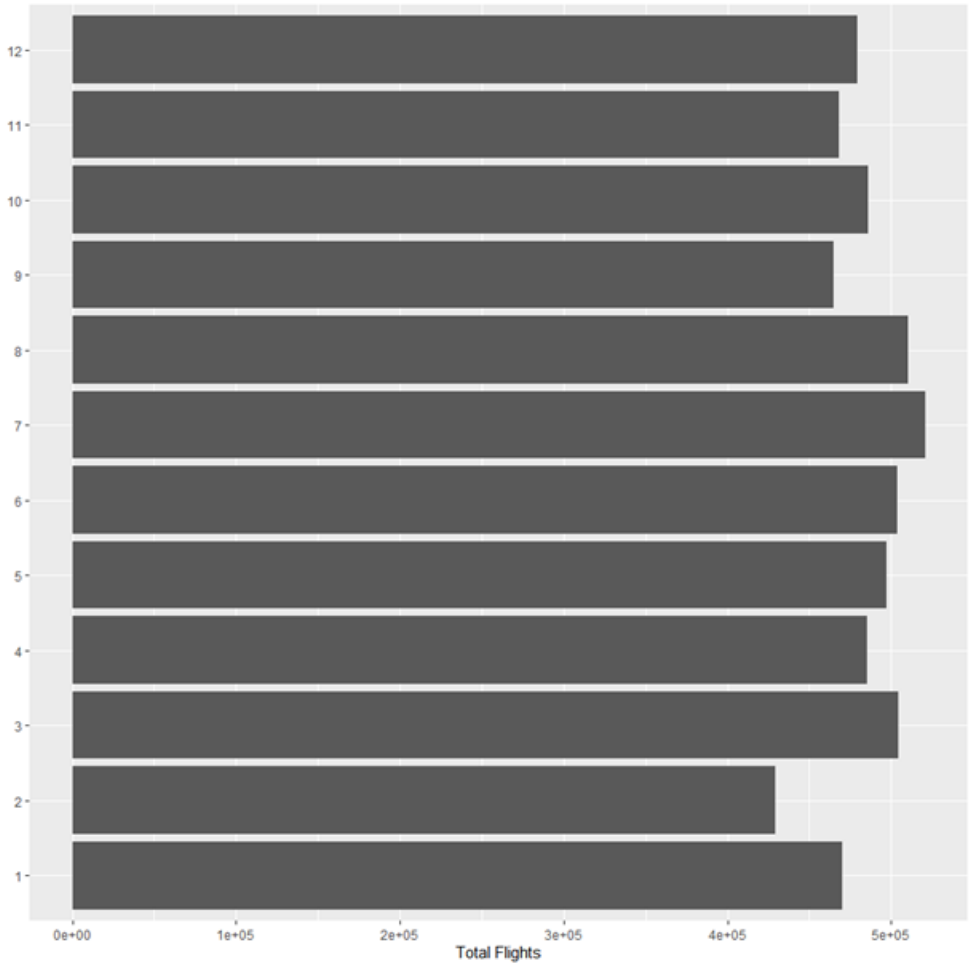
CANCELLATION_REASON	Reason for Cancellation of flight: A - Airline/Carrier; B - Weather; C - National Air System; D - Security
AIR_SYSTEM_DELAY	Delay caused by air system
SECURITY_DELAY	Delay caused by security
AIRLINE_DELAY	Delay caused by the airline
LATE_AIRCRAFT_DELAY	Delay caused by aircraft
WEATHER_DELAY	Delay caused by weather

Airlines.csv	
Column	Description
IATA_CODE	Airline Identifier
AIRLINE	Airport's Name
Airports.csv	
Column	Description
IATA_CODE	Location Identifier
AIRPORT	Airport's Name
CITY	
STATE	
COUNTRY	Country Name of the Airport
LATITUDE	Latitude of the Airport
LONGITUDE	Longitude of the Airport

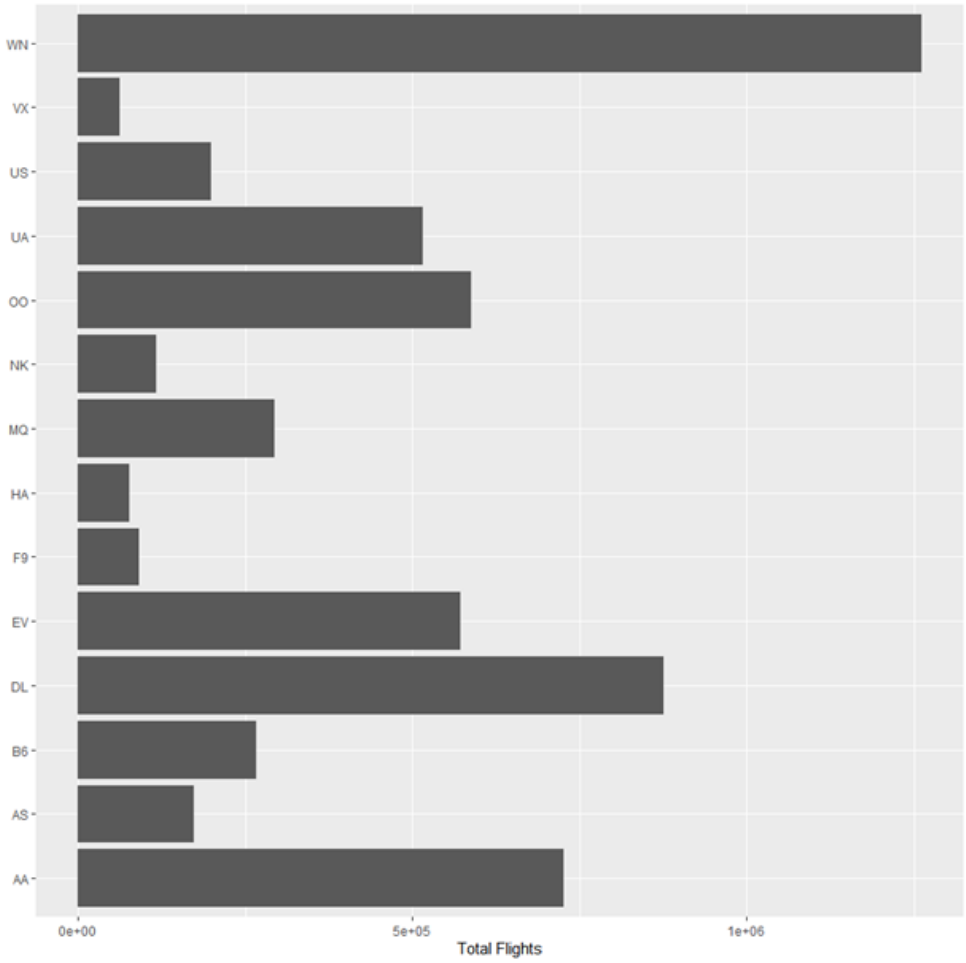
Basic EDA

A basic exploratory data analysis of the data is shown below:

The below plot shows total number of flights in each month (y-axis). We see that Most flights were operated in the month of July.

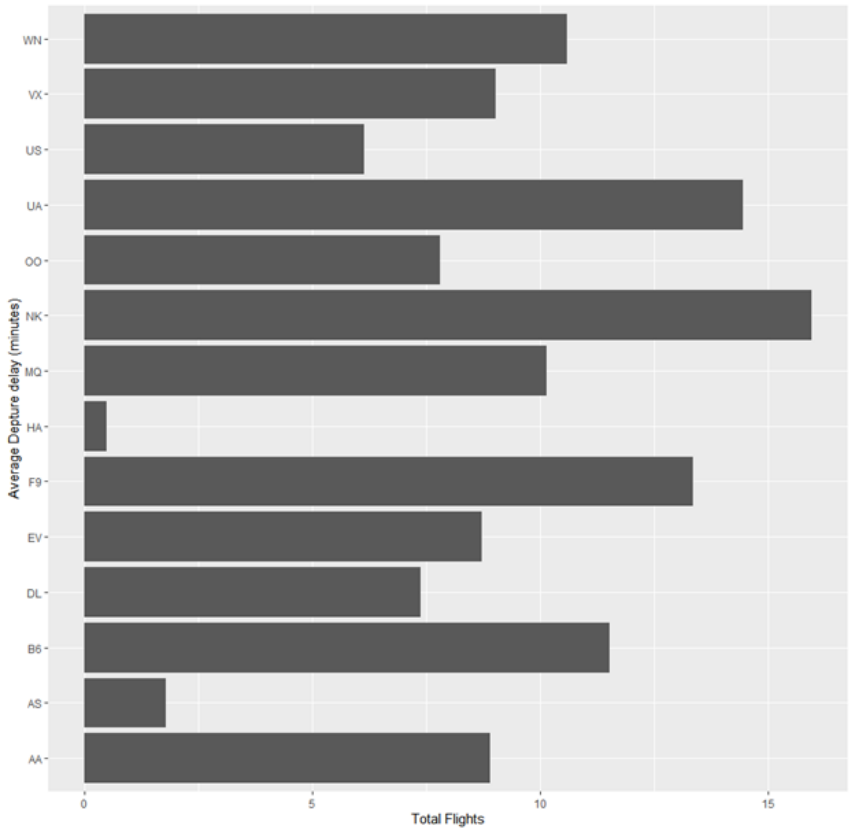


The below figure shows the total number of flights by airlines and we observed here that WN which is Southwest Airlines Co. Had the most flights in 2015



On a different note the below chart shows average departure delay in minutes for some airlines and it is noted that NK which is Spirit Air Lines had the most delayed departure in the year 2015

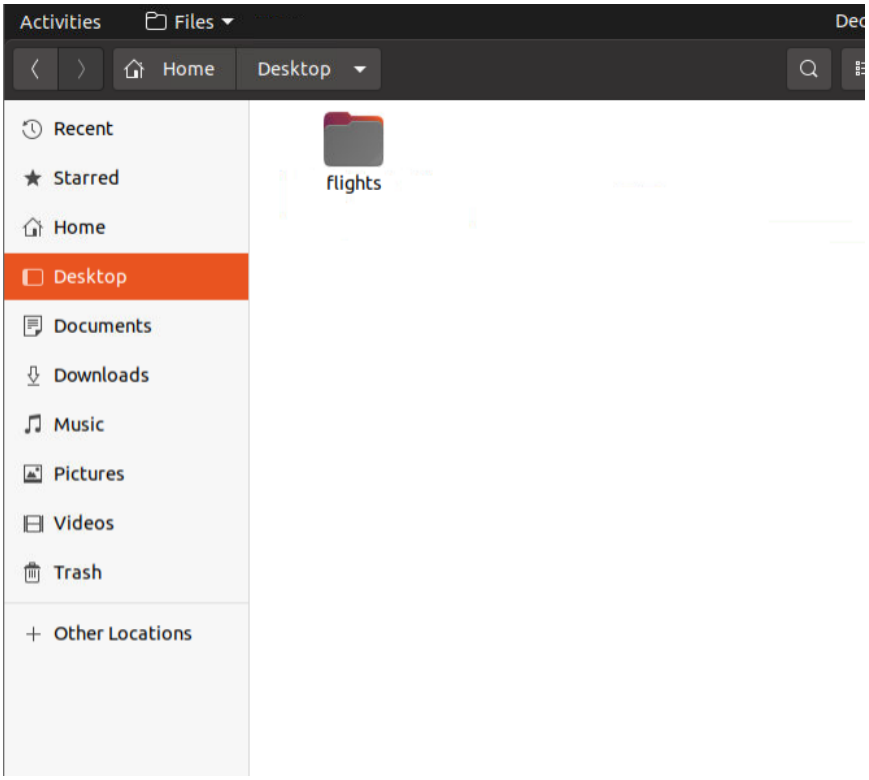
.

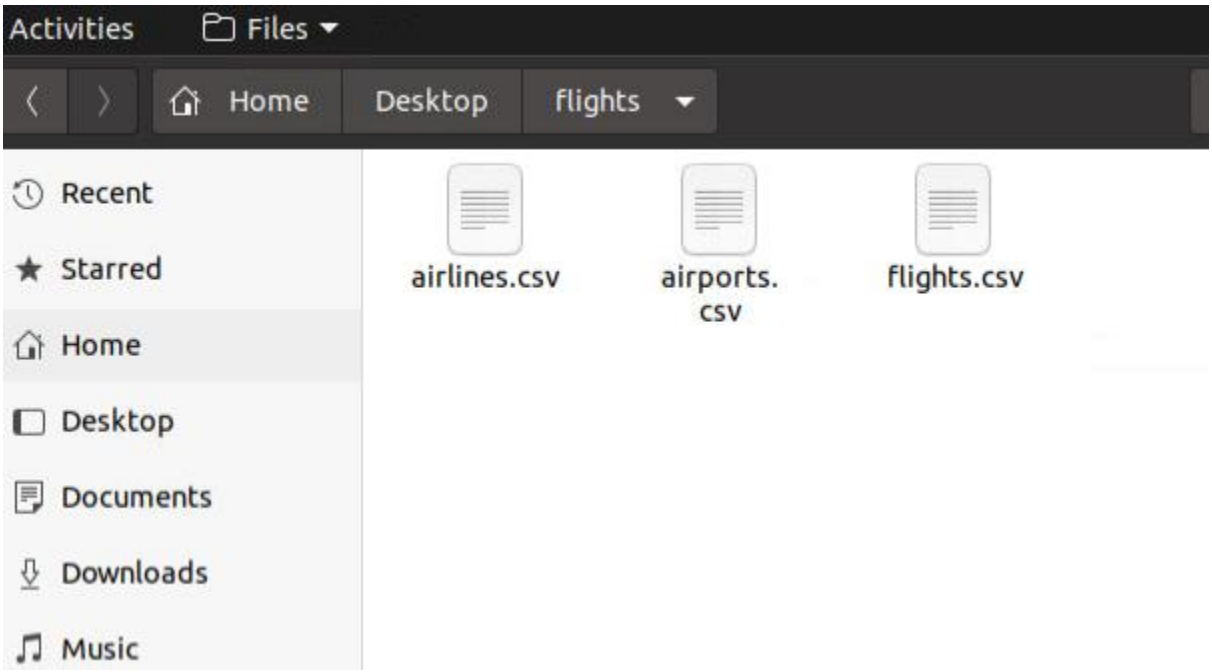


We will attempt to run queries to see output stated above as well as more statistics through different MongoDB commands

MongoDB Tutorial

Before starting MongoDB, first create the folder on desktop called flights and save all 3 csv files there.





.

Next, we will use docker image of Mongo for this tutorial

Running MongoDB on Docker

The first step is to download the official MongoDB image using the terminal. This will pull the latest image of Mongo from docker Hub

```
sudo docker pull mongo
```

```
bbak@bbak:~$ sudo docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
f22ccc0b8772: Pull complete
3cf8fb62ba5f: Pull complete
e80c964ece6a: Pull complete
329e632c35b3: Pull complete
3e1bd1325a3d: Pull complete
4aa6e3d64a4a: Pull complete
035bca87b778: Pull complete
874e4e43cb00: Pull complete
08cb97662b8b: Pull complete
f623ce2ba1e1: Pull complete
f100ac278196: Pull complete
6f5539f9b3ee: Pull complete
Digest: sha256:02e9941ddcb949424fa4eb01f9d235da91a5b7b64feb5887eab77e1ef84a3bad
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
bbak@bbak:~$
```

Next, we will run mongo image and name it mongodb.

```
sudo docker run --name mongodb mongo
```

```
bbak@bbak:~$ sudo docker run --name mongodb mongo
{"t":{"$date":"2020-12-26T17:42:08.694+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2020-12-26T17:42:08.696+00:00"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}
{"t":{"$date":"2020-12-26T17:42:08.696+00:00"},"s":"I",  "c":"NETWORK",  "id":4648601, "ctx":"main","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t":{"$date":"2020-12-26T17:42:08.697+00:00"},"s":"I",  "c":"STORAGE",  "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"ee0704c72e1f"}}
{"t":{"$date":"2020-12-26T17:42:08.697+00:00"},"s":"I",  "c":"CONTROL",  "id":23403,   "ctx":"initandlisten","msg":"Build Info","attr":{"buildInfo":{"version":"4.4.2","gitVersion":"15e73dc5738d2278b688f8929aee605fe4279b0e","openSSLVersion":"OpenSSL 1.1.1  11 Sep 2018","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu1804","distarch":"x86_64","target_arch":"x86_64"}}}}
{"t":{"$date":"2020-12-26T17:42:08.697+00:00"},"s":"I",  "c":"CONTROL",  "id":51765,   "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Ubuntu","version":"18.04"}}}
{"t":{"$date":"2020-12-26T17:42:08.697+00:00"},"s":"I",  "c":"CONTROL",  "id":21951,   "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{"net":{"bindIp":"*"}}}}
{"t":{"$date":"2020-12-26T17:42:08.698+00:00"},"s":"I",  "c":"STORAGE",  "id":22297,   "ctx":"initandlisten","msg":"Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem","tags":["startupWarnings"]}
{"t":{"$date":"2020-12-26T17:42:08.698+00:00"},"s":"I",  "c":"STORAGE",  "id":22315,   "ctx":"initandlisten","msg":"Opening WiredTiger","attr":{"config":"create,cache_size=7437M,session_max=33000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000,close_scan_interval=10,close_handle_minimum=250),statistics_log=(wait=0),verbose=[recovery_progress,checkpoint_progress,compact_progress],"}}
{"t":{"$date":"2020-12-26T17:42:09.477+00:00"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609004529:477858][1:0x7f45a39c6ac0], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Set global recovery timestamp: (0, 0)"}}
{"t":{"$date":"2020-12-26T17:42:09.478+00:00"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1609004529:477993][1:0x7f45a39c6ac0], txn-recover: [WT_VERB_RECOVERY | WT_VERB_RECOVERY_PROGRESS] Set global oldest timestamp: (0, 0)"}}
{"t":{"$date":"2020-12-26T17:42:09.592+00:00"},"s":"I",  "c":"STORAGE",  "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened","attr":{"durationMillis":894}}
{"t":{"$date":"2020-12-26T17:42:09.592+00:00"},"s":"I",  "c":"RECOVERY", "id":23987,   "ctx":"initandlisten","msg":"WiredTiger recoveryTimestamp","attr":{"recoveryTimestamp":{"$timestamp":{"t":0,"i":0}}}}
{"t":{"$date":"2020-12-26T17:42:09.907+00:00"},"s":"I",  "c":"STORAGE",  "id":4366408, "ctx":"initandlisten","msg":"No table logging settings modifica
```

Next, we will connect to this container using an interactive shell to see directories present. Go to new window of terminal and type:

sudo docker exec -it mongodb sh

```
bbak@bbak:~$ sudo docker exec -it mongodb sh
[sudo] password for bbak:
# ls
bin  boot  data  dev  docker-entrypoint-initdb.d  etc  home  js-yaml.js  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr
# cat tmp
cat: tmp: Is a directory
# cd tmp
# ls
mongodb-27017.sock
```

We check all the folders using **ls** command

We need to copy our flights folder located on desktop to tmp folder inside the mongodb container so that we can easily import the data

Write **exit** to exit from this interactive shell

Now run the below command for copying files to mongodb container

sudo docker cp Desktop/flights mongodb:/tmp

We will now try to import flights.csv data into mongodb which will create document like structure.

Execute the below command for this

sudo docker exec mongodb mongoimport -d Flightsdata -c flights --type csv --headerline --file /tmp/flights/flights.csv

```
bbak@bbak:~$ sudo docker exec mongodb mongoimport -d Flightsdata -c flights --type csv --headerline --file /tmp/flights/flights.csv
2020-12-26T18:11:14.455+0000    connected to: mongodb://localhost/
2020-12-26T18:11:17.455+0000    [........................] Flightsdata.flights  10.4MB/565MB (1.8%)
2020-12-26T18:11:20.455+0000    [........................] Flightsdata.flights  21.7MB/565MB (3.8%)
2020-12-26T18:11:23.455+0000    [#.......................] Flightsdata.flights  32.8MB/565MB (5.8%)
2020-12-26T18:11:26.456+0000    [#.......................] Flightsdata.flights  43.8MB/565MB (7.7%)
2020-12-26T18:11:29.455+0000    [##......................] Flightsdata.flights  54.7MB/565MB (9.7%)
2020-12-26T18:11:32.456+0000    [##......................] Flightsdata.flights  65.7MB/565MB (11.6%)
2020-12-26T18:11:35.455+0000    [###.....................] Flightsdata.flights  76.9MB/565MB (13.6%)
2020-12-26T18:11:38.455+0000    [###.....................] Flightsdata.flights  87.7MB/565MB (15.5%)
2020-12-26T18:11:41.455+0000    [####....................] Flightsdata.flights  98.6MB/565MB (17.5%)
2020-12-26T18:11:44.455+0000    [####....................] Flightsdata.flights  110MB/565MB (19.4%)
2020-12-26T18:11:47.455+0000    [#####...................] Flightsdata.flights  119MB/565MB (21.0%)
2020-12-26T18:11:50.455+0000    [#####...................] Flightsdata.flights  130MB/565MB (23.0%)
2020-12-26T18:11:53.455+0000    [#####...................] Flightsdata.flights  141MB/565MB (25.0%)
2020-12-26T18:11:56.455+0000    [######..................] Flightsdata.flights  152MB/565MB (26.9%)
2020-12-26T18:11:59.455+0000    [######..................] Flightsdata.flights  163MB/565MB (28.9%)
2020-12-26T18:12:02.455+0000    [#######.................] Flightsdata.flights  174MB/565MB (30.9%)
2020-12-26T18:12:05.455+0000    [#######.................] Flightsdata.flights  185MB/565MB (32.8%)
2020-12-26T18:12:08.455+0000    [########................] Flightsdata.flights  197MB/565MB (34.8%)
2020-12-26T18:12:11.455+0000    [########................] Flightsdata.flights  208MB/565MB (36.7%)
2020-12-26T18:12:14.455+0000    [########................] Flightsdata.flights  219MB/565MB (38.7%)
2020-12-26T18:12:17.455+0000    [#########...............] Flightsdata.flights  228MB/565MB (40.3%)
2020-12-26T18:12:20.455+0000    [#########...............] Flightsdata.flights  239MB/565MB (42.3%)
2020-12-26T18:12:23.455+0000    [##########..............] Flightsdata.flights  250MB/565MB (44.3%)
2020-12-26T18:12:26.455+0000    [##########..............] Flightsdata.flights  261MB/565MB (46.3%)
2020-12-26T18:12:29.455+0000    [##########..............] Flightsdata.flights  272MB/565MB (48.2%)
2020-12-26T18:12:32.455+0000    [###########.............] Flightsdata.flights  284MB/565MB (50.2%)
2020-12-26T18:12:35.455+0000    [###########.............] Flightsdata.flights  295MB/565MB (52.1%)
2020-12-26T18:12:38.455+0000    [###########.............] Flightsdata.flights  306MB/565MB (54.1%)
2020-12-26T18:12:41.455+0000    [############............] Flightsdata.flights  317MB/565MB (56.1%)
2020-12-26T18:12:44.455+0000    [############............] Flightsdata.flights  328MB/565MB (58.1%)
2020-12-26T18:12:47.455+0000    [############............] Flightsdata.flights  339MB/565MB (60.0%)
```

This process will take few mins as it will try to convert in JSON format and store it inside mongodb database.

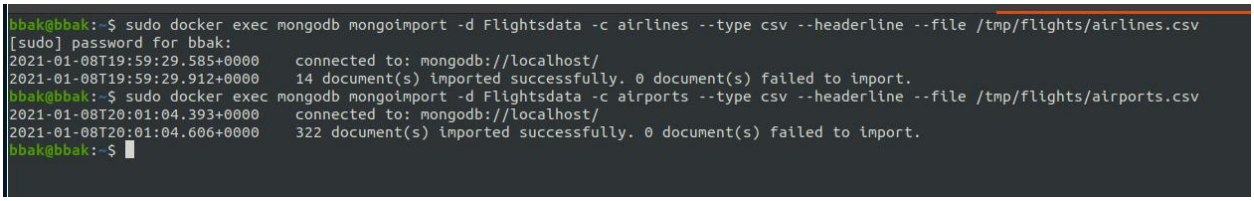
In this command we are telling mongodb to import csv file through "*--type csv*" into a database called Flightsdata in collection flights and also telling that the fields of documents will be the first row of csv file through "*--headerline*"

Similarly import airlines and airports data using below commands

sudo docker exec mongodb mongoimport -d Flightsdata -c airlines --type csv --headerline --file /tmp/flights/airlines.csv

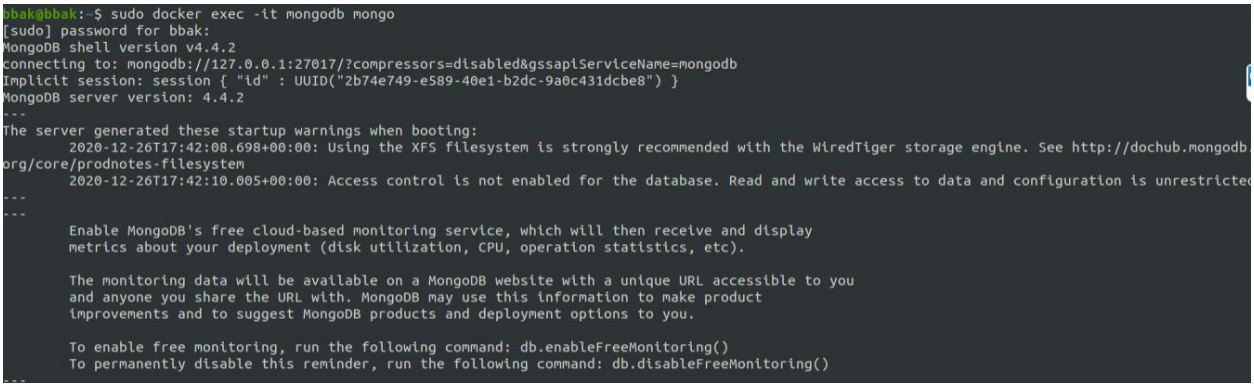
sudo docker exec mongodb mongoimport -d Flightsdata -c airports --type csv --headerline --file /tmp/flights/airports.csv

Below is the output of the execution of above commands



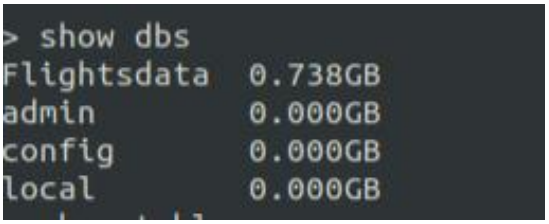
Open another instance of terminal and type the following command to go inside mongo shell

sudo docker exec -it mongodb mongo



We see that now we can interact with our database through this shell

Run command *show dbs* to display all available databases and also to show our newly created database



In order to use our db we need to execute below statement

use Flightsdata

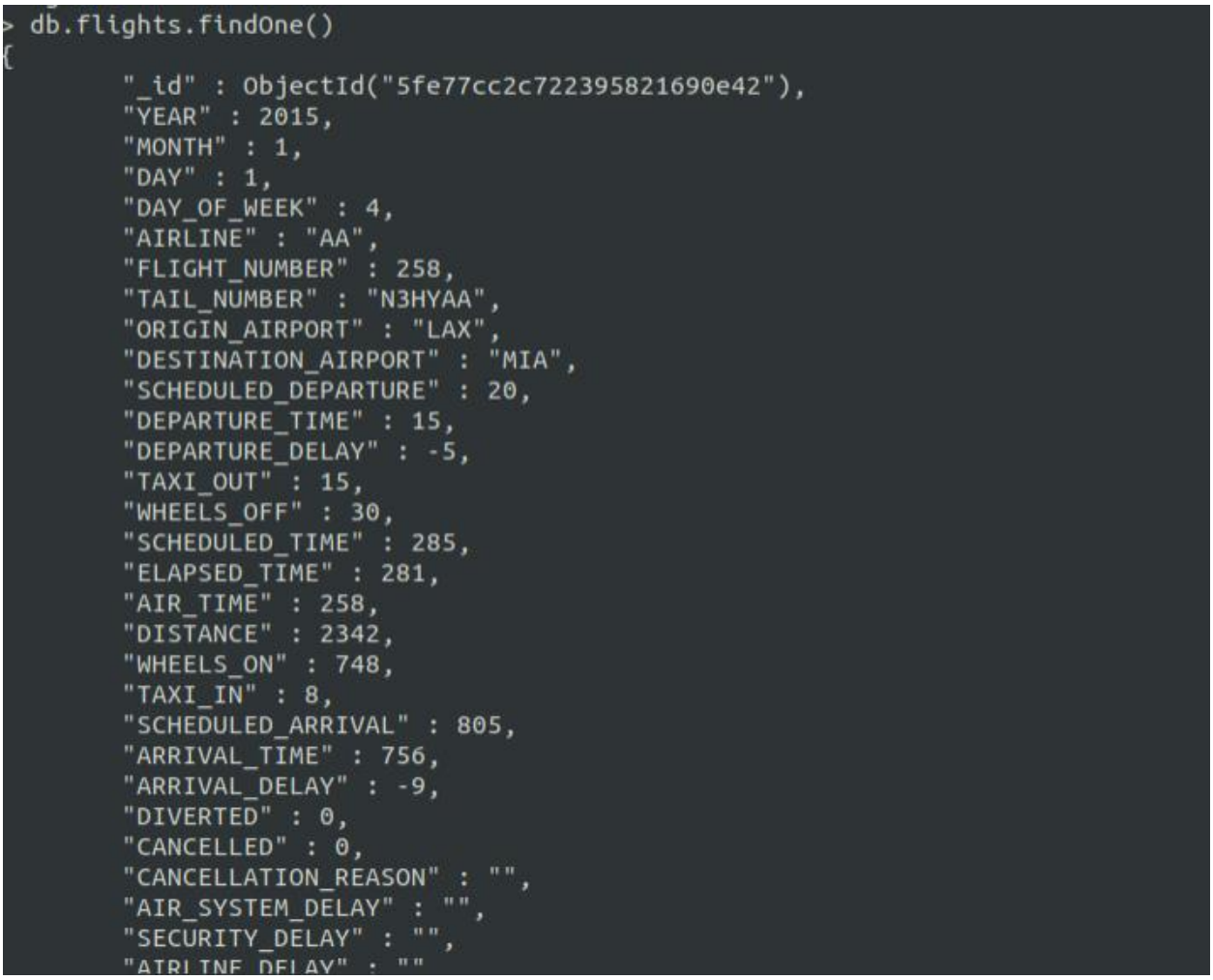
And also to see our collections

show tables

Reading from collection

Now check first document in the flights collection do below:

```
db.flights.findOne()
```



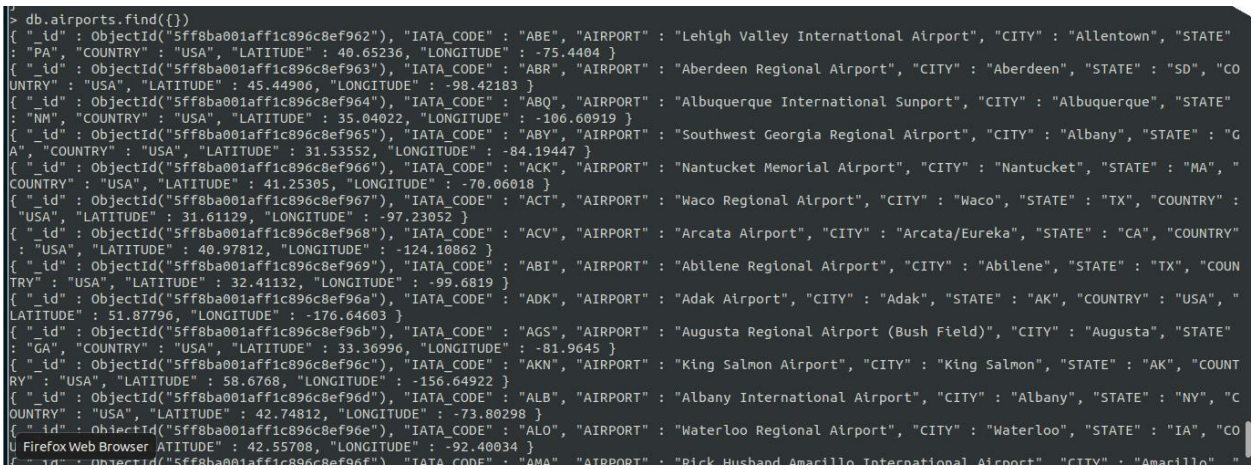
Similarly check the first document in airport collections:

```
db.airports.findOne()
```

```
> db.airports.findOne()
{
        "_id" : ObjectId("5ff8ba001aff1c896c8ef962"),
        "IATA_CODE" : "ABE",
        "AIRPORT" : "Lehigh Valley International Airport",
        "CITY" : "Allentown",
        "STATE" : "PA",
        "COUNTRY" : "USA",
        "LATITUDE" : 40.65236,
        "LONGITUDE" : -75.4404
}
>
```

To see all documents in airports collection:

```
db.airports.find({})
```



The above command will print all the rows (documents) in a format which can be difficult to read and inspect. Therefore a more useful command is the following which will show each columns separately in new line.

```
db.airports.find({}).pretty()
```

```
> db.airports.find({}).pretty()
{
        "_id" : ObjectId("5ff8ba001aff1c896c8ef962"),
        "IATA_CODE" : "ABE",
        "AIRPORT" : "Lehigh Valley International Airport",
        "CITY" : "Allentown",
        "STATE" : "PA",
        "COUNTRY" : "USA",
        "LATITUDE" : 40.65236,
        "LONGITUDE" : -75.4404
}
{
        "_id" : ObjectId("5ff8ba001aff1c896c8ef963"),
        "IATA_CODE" : "ABR",
        "AIRPORT" : "Aberdeen Regional Airport",
        "CITY" : "Aberdeen",
        "STATE" : "SD",
        "COUNTRY" : "USA",
        "LATITUDE" : 45.44906,
        "LONGITUDE" : -98.42183
}
{
        "_id" : ObjectId("5ff8ba001aff1c896c8ef964"),
        "IATA_CODE" : "ABQ",
        "AIRPORT" : "Albuquerque International Sunport",
        "CITY" : "Albuquerque",
        "STATE" : "NM",
        "COUNTRY" : "USA",
        "LATITUDE" : 35.04022,
        "LONGITUDE" : -106.60919
} Firefox Web Browser
```

Let us now check the count of documents in flights collection:

```
db.flights.find().count()
```

```
> db.flights.find().count()
5819079
```

We see that this collection has 5,819,079 documents

Now check airlines data set:

```
db.airlines.find({})
```

```
> db.airlines.find({})
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb40"), "IATA_CODE" : "UA", "AIRLINE" : "United Air Lines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb41"), "IATA_CODE" : "AA", "AIRLINE" : "American Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb42"), "IATA_CODE" : "US", "AIRLINE" : "US Airways Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb43"), "IATA_CODE" : "F9", "AIRLINE" : "Frontier Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb44"), "IATA_CODE" : "B6", "AIRLINE" : "JetBlue Airways" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb45"), "IATA_CODE" : "OO", "AIRLINE" : "Skywest Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb46"), "IATA_CODE" : "AS", "AIRLINE" : "Alaska Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb47"), "IATA_CODE" : "EV", "AIRLINE" : "Atlantic Southeast Airlines" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb48"), "IATA_CODE" : "HA", "AIRLINE" : "Hawaiian Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb49"), "IATA_CODE" : "MQ", "AIRLINE" : "American Eagle Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4a"), "IATA_CODE" : "VX", "AIRLINE" : "Virgin America" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4b"), "IATA_CODE" : "WN", "AIRLINE" : "Southwest Airlines Co." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4c"), "IATA_CODE" : "DL", "AIRLINE" : "Delta Air Lines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4d"), "IATA_CODE" : "NK", "AIRLINE" : "Spirit Air Lines" }
>
```

There are 2 fields in each row and they represent the full form of IATA_Code which is the AIRLINE column in flights collection

Creation of new rows

Let’s try inserting a new airline in airlines collection:

```
db.airlines.insertOne(

{"IATA_CODE" : "PK", "AIRLINE" : "Pakistan International Airline"}

)
```

```
> db.airlines.insertOne(
...
... {"IATA_CODE" : "PK", "AIRLINE" : "Pakistan International Airline"}
...
... )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5ff9ae54784aae87a688feb0")
}
>
```

And now list all the airlines again

```
> db.airlines.find({})
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb40"), "IATA_CODE" : "UA", "AIRLINE" : "United Air Lines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb41"), "IATA_CODE" : "AA", "AIRLINE" : "American Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb42"), "IATA_CODE" : "US", "AIRLINE" : "US Airways Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb43"), "IATA_CODE" : "F9", "AIRLINE" : "Frontier Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb44"), "IATA_CODE" : "B6", "AIRLINE" : "JetBlue Airways" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb45"), "IATA_CODE" : "OO", "AIRLINE" : "Skywest Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb46"), "IATA_CODE" : "AS", "AIRLINE" : "Alaska Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb47"), "IATA_CODE" : "EV", "AIRLINE" : "Atlantic Southeast Airlines" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb48"), "IATA_CODE" : "HA", "AIRLINE" : "Hawaiian Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb49"), "IATA_CODE" : "MQ", "AIRLINE" : "American Eagle Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4a"), "IATA_CODE" : "VX", "AIRLINE" : "Virgin America" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4b"), "IATA_CODE" : "WN", "AIRLINE" : "Southwest Airlines Co." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4c"), "IATA_CODE" : "DL", "AIRLINE" : "Delta Air Lines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4d"), "IATA_CODE" : "NK", "AIRLINE" : "Spirit Air Lines" }
{ "_id" : ObjectId("5ff9ae54784aae87a688feb0"), "IATA_CODE" : "PK", "AIRLINE" : "Pakistan International Airline" }
```

Firefox Web Browser

Now add multiple documents in airlines collection at once using below command

```
db.airlines.insertMany([

{"IATA_CODE" : "ER", "AIRLINE" : "Serene Air"},

{"IATA_CODE" : "PF", "AIRLINE" : "Air Sial"}

])
```

```
> db.airlines.insertMany([
...
... {"IATA_CODE" : "ER", "AIRLINE" : "Serene Air"},
...
... {"IATA_CODE" : "PF", "AIRLINE" : "Air Sial"}
...
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5ff9b010784aae87a688feb1"),
                ObjectId("5ff9b010784aae87a688feb2")
        ]
}
>
```

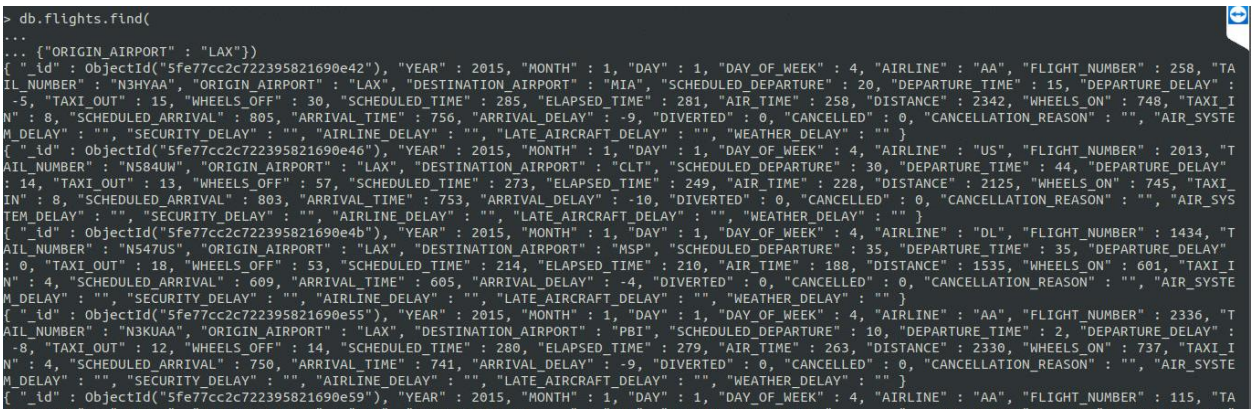
The rows are added at the bottom:

```
> db.airlines.find({})
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb40"), "IATA_CODE" : "UA", "AIRLINE" : "United Air Lines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb41"), "IATA_CODE" : "AA", "AIRLINE" : "American Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb42"), "IATA_CODE" : "US", "AIRLINE" : "US Airways Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb43"), "IATA_CODE" : "F9", "AIRLINE" : "Frontier Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb44"), "IATA_CODE" : "B6", "AIRLINE" : "JetBlue Airways" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb45"), "IATA_CODE" : "OO", "AIRLINE" : "Skywest Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb46"), "IATA_CODE" : "AS", "AIRLINE" : "Alaska Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb47"), "IATA_CODE" : "EV", "AIRLINE" : "Atlantic Southeast Airlines" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb48"), "IATA_CODE" : "HA", "AIRLINE" : "Hawaiian Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb49"), "IATA_CODE" : "MQ", "AIRLINE" : "American Eagle Airlines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4a"), "IATA_CODE" : "VX", "AIRLINE" : "Virgin America" }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4b"), "IATA_CODE" : "WN", "AIRLINE" : "Southwest Airlines Co." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4c"), "IATA_CODE" : "DL", "AIRLINE" : "Delta Air Lines Inc." }
{ "_id" : ObjectId("5ff8b9a1465e3486bc8dbb4d"), "IATA_CODE" : "NK", "AIRLINE" : "Spirit Air Lines" }
{ "_id" : ObjectId("5ff9ae54784aae87a688feb0"), "IATA_CODE" : "PK", "AIRLINE" : "Pakistan International Airline" }
{ "_id" : ObjectId("5ff9b010784aae87a688feb1"), "IATA_CODE" : "ER", "AIRLINE" : "Serene Air" }
{ "_id" : ObjectId("5ff9b010784aae87a688feb2"), "IATA_CODE" : "PF", "AIRLINE" : "Air Sial" }
>
```

Query with conditions

Now let us head back to flights collection. To see documents only with origin airport as LAX

```
db.flights.find(
{"ORIGIN_AIRPORT" : "LAX"})
```



We see that there are too many fields displayed. Let's display only few set of fields for the same query

```
db.flights.find(
{"ORIGIN_AIRPORT" : "LAX"},
{"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1})
```

```
> db.flights.find(
...
... {"ORIGIN_AIRPORT" : "LAX"},
...
... {"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1})
{ "_id" : ObjectId("5fe77cc2c722395821690e42"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690e46"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "US", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690e4b"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "DL", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690e55"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690e59"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690e5b"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690eca"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690ecd"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690ee8"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690f06"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690f22"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690f27"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AS", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690f4b"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "DL", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690fb2"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690fb4"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690ff4"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821690fff"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "B6", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821691009"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "NK", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c72239582169104d"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "AA", "ORIGIN_AIRPORT" : "LAX" }
{ "_id" : ObjectId("5fe77cc2c722395821691069"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "US", "ORIGIN_AIRPORT" : "LAX" }
Type "it" for more
```

The above output looks a lot better.

Let’s find all flights taken by a specific airline (UA) from origin airport “LAX”

```
db.flights.find(

{"ORIGIN_AIRPORT" : "LAX", "AIRLINE" : "UA"},

{"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1}

)
```

Notice we also added DISTANCE column so that it can be displayed. Below is the output

```
> db.flights.find(
...
... {"ORIGIN_AIRPORT" : "LAX", "AIRLINE" : "UA"},
...
... {"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1}
...
... )
{ "_id" : ObjectId("5fe77cc2c722395821690e5b"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c722395821690eca"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821690ee8"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c722395821690f06"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1744 }
{ "_id" : ObjectId("5fe77cc2c722395821690fb2"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2454 }
{ "_id" : ObjectId("5fe77cc2c722395821690fb4"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821691106"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c72239582169124d"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2556 }
{ "_id" : ObjectId("5fe77cc2c722395821691376"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2288 }
{ "_id" : ObjectId("5fe77cc2c722395821691470"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1744 }
{ "_id" : ObjectId("5fe77cc2c72239582169151d"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c72239582169155a"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2475 }
{ "_id" : ObjectId("5fe77cc2c72239582169157a"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2454 }
{ "_id" : ObjectId("5fe77cc2c7223958216915c0"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c7223958216915e1"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c7223958216916ac"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2329 }
{ "_id" : ObjectId("5fe77cc2c7223958216916ed"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2556 }
{ "_id" : ObjectId("5fe77cc2c7223958216917c0"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2504 }
{ "_id" : ObjectId("5fe77cc2c7223958216917c1"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 1744 }
{ "_id" : ObjectId("5fe77cc2c722395821691823"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 2486 }
Type "it" for more
>
```

Now we will find all UA flights from airport LAX but which has distance less than 1000 miles

```
db.flights.find(

{"ORIGIN_AIRPORT" : "LAX", "AIRLINE" : "UA", "DISTANCE" :{ $lt : 1000}},

{"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1}

)
```

```
> db.flights.find(
...
... {"ORIGIN_AIRPORT" : "LAX", "AIRLINE" : "UA", "DISTANCE" :{ $lt : 1000}},
...
... {"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1}
...
... )
{ "_id" : ObjectId("5fe77cc2c722395821690eca"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821690fb4"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c7223958216915c0"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c7223958216915e1"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c7223958216919ca"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821691ef9"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821692451"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c7223958216924e"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c7223958216927b8"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 236 }
{ "_id" : ObjectId("5fe77cc2c722395821692d21"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821692e63"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821692ee7"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821693125"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821693821"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821693b79"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821693e3e"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821693f20"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc3c7223958216943ed"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc3c7223958216943fe"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 236 }
{ "_id" : ObjectId("5fe77cc3c7223958216945ad"), "YEAR" : 2015, "MONTH" : 1, "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DISTANCE" : 862 }
Type "it" for more
>
```

Let’s take the count of these documents

```
db.flights.find(

{"ORIGIN_AIRPORT" : "LAX", "AIRLINE" : "UA", "DISTANCE" :{ $lt : 1000}},

{"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1}

).count()
```

It shows that there are 6990 such flights

```
>
> db.flights.find(
...
... {"ORIGIN_AIRPORT" : "LAX", "AIRLINE" : "UA", "DISTANCE" :{ $lt : 1000}},
...
... {"YEAR":1, "MONTH": 1, "AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1}
...
... ).count()
6990
>
```

Now find all flights by UA from either LAX (Los Angeles) airport or IAH (Houston airport)

This time we do not want Year and month to be displayed but instead we will display departure time along with distance

```
db.flights.find(

    { "AIRLINE" : "UA",

        $or : [{"ORIGIN_AIRPORT" : "LAX"},{"ORIGIN_AIRPORT" : "IAH"}]},

    {"AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1, "DEPARTURE_TIME":1}

)
```

```
> db.flights.find(
...
...     { "AIRLINE" : "UA",
...
...         $or : [{"ORIGIN_AIRPORT" : "LAX"},{"ORIGIN_AIRPORT" : "IAH"}]},
...
...     {"AIRLINE":1, "ORIGIN_AIRPORT":1, "DISTANCE":1, "DEPARTURE_TIME":1}
...
... )
{ "_id" : ObjectId("5fe77cc2c722395821690e5b"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 112, "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c722395821690eca"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 550, "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821690ee8"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 555, "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c722395821690f06"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 607, "DISTANCE" : 1744 }
{ "_id" : ObjectId("5fe77cc2c722395821690fb2"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 603, "DISTANCE" : 2454 }
{ "_id" : ObjectId("5fe77cc2c722395821690fb4"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 601, "DISTANCE" : 337 }
{ "_id" : ObjectId("5fe77cc2c722395821691106"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 636, "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c72239582169124d"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 707, "DISTANCE" : 2556 }
{ "_id" : ObjectId("5fe77cc2c722395821691288"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 716, "DISTANCE" : 925 }
{ "_id" : ObjectId("5fe77cc2c7223958216912b4"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 719, "DISTANCE" : 1235 }
{ "_id" : ObjectId("5fe77cc2c7223958216912bf"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 720, "DISTANCE" : 1635 }
{ "_id" : ObjectId("5fe77cc2c7223958216912e9"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 738, "DISTANCE" : 1190 }
{ "_id" : ObjectId("5fe77cc2c722395821691311"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 729, "DISTANCE" : 1400 }
{ "_id" : ObjectId("5fe77cc2c722395821691312"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 727, "DISTANCE" : 854 }
{ "_id" : ObjectId("5fe77cc2c722395821691317"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 728, "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c722395821691324"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "IAH", "DEPARTURE_TIME" : 727, "DISTANCE" : 862 }
{ "_id" : ObjectId("5fe77cc2c722395821691376"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 726, "DISTANCE" : 2288 }
{ "_id" : ObjectId("5fe77cc2c722395821691470"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 759, "DISTANCE" : 1744 }
{ "_id" : ObjectId("5fe77cc2c72239582169151d"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 810, "DISTANCE" : 1379 }
{ "_id" : ObjectId("5fe77cc2c72239582169155a"), "AIRLINE" : "UA", "ORIGIN_AIRPORT" : "LAX", "DEPARTURE_TIME" : 806, "DISTANCE" : 2475 }
Type "it" for more
>
```

Querying Missing Values

Now find the flight which have no departure time. This is same as querying for missing values.

```
db.flights.find({

"DEPARTURE_TIME" : null

}).count()
```

```
> db.flights.find({
...
... "DEPARTURE_TIME" : null
...
... }).count()
0
>
```

We see there are no missing values for departure. However, when changing the query we get blank departure times

```
db.flights.find({  "DEPARTURE_TIME" : "" }).count()
```

```
> db.flights.find({  "DEPARTURE_TIME" : "" }).count()
86153
>
```

We also notice that such blank departure times also have cancelled field equal to 1 which means these flights were cancelled:

```
db.flights.find({  CANCELLED : 1, "DEPARTURE_TIME" : '' }).count()
```

```
> db.flights.find({  CANCELLED : 1, "DEPARTURE_TIME" : '' }).count()
86153
>
```

Aggregation

Now find the total number of flights taken by each airline

In order to do this we will use aggregation in MongoDB.

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

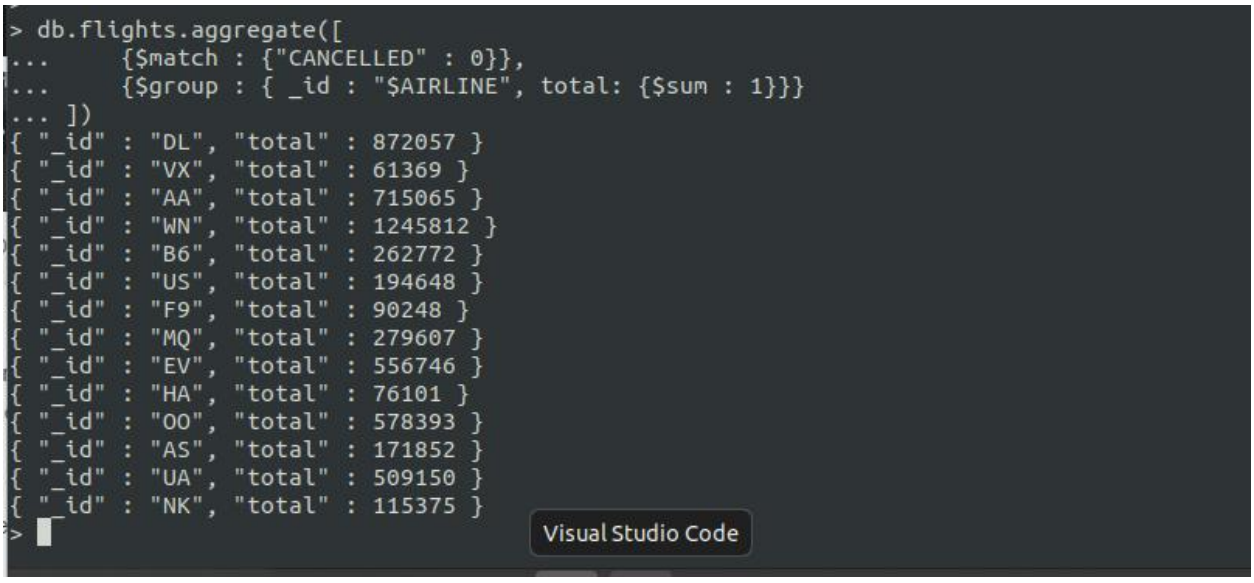
The MongoDB aggregation pipeline consists of stages. Each stage transforms the documents as they pass through the pipeline. Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents.

The below command give the desired result

```
db.flights.aggregate([
    {$match : {"CANCELLED" : 0}},
    {$group : { _id : "$AIRLINE", total: {$sum : 1}}}
])
```

The documents are first filtered with flights having no cancellation and then passed on to the next aggregation stage which is group by Airline.

Below is the output



The above is equivalent to map reduce, however the difference is that here no map reduce function needs to be defined. Most map reduce techniques can also be done via aggregation pipeline.

Now find the top 10 destination airports. We will count the flights for each airport and sort it in descending order and take the first 10 documents.

```
db.flights.aggregate([
    {$match : {"CANCELLED" : 0}},
    {$group : { _id : "$DESTINATION_AIRPORT", total: {$sum : 1}}},
    {$sort : {"total" : -1 }},
    {$limit : 10}
])
```

Here **the $match** operator will filter documents having cancelled = 0,

$group will group by destination airport, assign 1 and then sum. This is summing all ones by airport

$sort will sort the documents in order. Here **-1** means sort in descending order

$limit will show the first 10 documents only

```
> db.flights.aggregate([
...     {$match : {"CANCELLED" : 0}},
...     {$group : { _id : "$DESTINATION_AIRPORT", total: {$sum : 1}}},
...     {$sort : {"total" : -1 }},
...     {$limit : 10}
... ])
{ "_id" : "ATL", "total" : 344189 }
{ "_id" : "ORD", "total" : 276633 }
{ "_id" : "DFW", "total" : 232833 }
{ "_id" : "DEN", "total" : 193700 }
{ "_id" : "LAX", "total" : 192437 }
{ "_id" : "SFO", "total" : 145661 }
{ "_id" : "PHX", "total" : 145602 }
{ "_id" : "IAH", "total" : 144251 }
{ "_id" : "LAS", "total" : 132338 }
{ "_id" : "MSP", "total" : 111270 }
>
```

Next find minimum and maximum arrival delays.

```
db.flights.aggregate([
  {$match : { "ARRIVAL_DELAY" : {$ne : ""}}},
  {$group : { _id : null, min_Arrival : { $min : "$ARRIVAL_DELAY"},
max_Arrival : {$max : "$ARRIVAL_DELAY"}}}
])
```

Again in this command Arrival Delays with blank values are removed.

Grouping by **_id : null** will return a single row

$min & $max are an aggregate operator and

```
>
> db.flights.aggregate([
...     {$match : { "ARRIVAL_DELAY" : {$ne : ""}}},
...     {$group : { _id : null, min_Arrival : { $min : "$ARRIVAL_DELAY"}, max_Arrival : {$max : "$ARRIVAL_DELAY"}}}
... ])
{ "_id" : null, "min_Arrival" : -87, "max_Arrival" : 1971 }
>
```

Exercise: find the airline with the most delay and the most common delay reasons

Next we will try to link origin airport details from airports collection to flights collection

We use the below command. To see output more clearly on command line we have added a limit pipeline along with pretty() option. We will see how our first row appears

```
db.flights.aggregate([
    { $lookup: {
        from: "airports",
        localField: "ORIGIN_AIRPORT",
        foreignField: "IATA_CODE",
        as : "ORIGIN_AIRPORT_DETAILS"
    }},
    {$limit : 1}
 ]).pretty()
```

This query is similar to JOINS in SQL. Below is the output of one of the document

```
> db.flights.aggregate([
...       { $lookup: {
...           from: "airports",
...           localField: "ORIGIN_AIRPORT",
...           foreignField: "IATA_CODE",
...           as : "ORIGIN_AIRPORT_DETAILS"
...       }},
...       {$limit : 1}
...   ]).pretty()
{
        "_id" : ObjectId("5fe77cc2c722395821690e42"),
        "YEAR" : 2015,
        "MONTH" : 1,
        "DAY" : 1,
        "DAY_OF_WEEK" : 4,
        "AIRLINE" : "AA",
        "FLIGHT_NUMBER" : 258,
        "TAIL_NUMBER" : "N3HYAA",
        "ORIGIN_AIRPORT" : "LAX",
        "DESTINATION_AIRPORT" : "MIA",
        "SCHEDULED_DEPARTURE" : 20,
        "DEPARTURE_TIME" : 15,
        "DEPARTURE_DELAY" : -5,
        "TAXI_OUT" : 15,
        "WHEELS_OFF" : 30
```

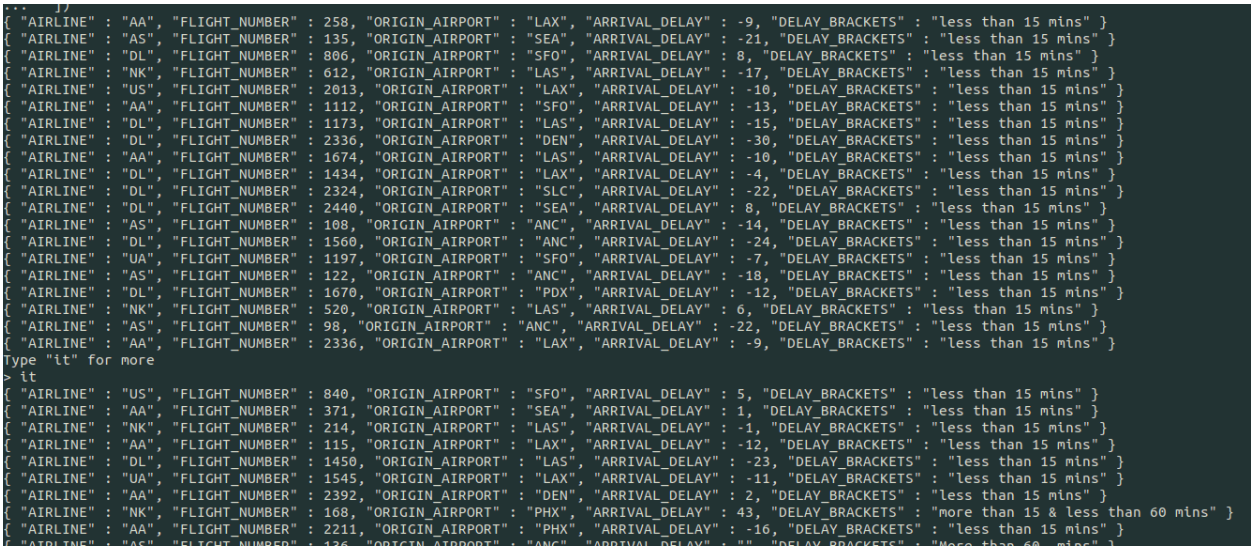
```
"TAXI_OUT" : 15,
"WHEELS_OFF" : 30,
"SCHEDULED_TIME" : 285,
"ELAPSED_TIME" : 281,
"AIR_TIME" : 258,
"DISTANCE" : 2342,
"WHEELS_ON" : 748,
"TAXI_IN" : 8,
"SCHEDULED_ARRIVAL" : 805,
"ARRIVAL_TIME" : 756,
"ARRIVAL_DELAY" : -9,
"DIVERTED" : 0,
"CANCELLED" : 0,
"CANCELLATION_REASON" : "",
"AIR_SYSTEM_DELAY" : "",
"SECURITY_DELAY" : "",
"AIRLINE_DELAY" : "",
"LATE_AIRCRAFT_DELAY" : "",
"WEATHER_DELAY" : "",
"ORIGIN_AIRPORT_DETAILS" : [
        {
                "_id" : ObjectId("5ff8ba001aff1c896c8efa0f"),
                "IATA_CODE" : "LAX",
                "AIRPORT" : "Los Angeles International Airport",
                "CITY" : "Los Angeles",
                "STATE" : "CA",
                "COUNTRY" : "USA",
                "LATITUDE" : 33.94254,
                "LONGITUDE" : -118.40807
        }
]
```


Now add a new field “DELAY_BRACKETS with the following values:

- If ARRIVAL_DELAY < 15 mins: “then less than 15 mins
- If 15 mins <= ARRIVAL_DELAY < 60 mins : more than 15 & less than 60 mins
- Else more than 60 mins

```
db.flights.aggregate([
      {
          $project: {
              _id : 0,
              "AIRLINE" : 1,
              "FLIGHT_NUMBER" : 1,
              "ORIGIN_AIRPORT" : 1,
              "ARRIVAL_DELAY" :1,
              "DELAY_BRACKETS" : {
                  $switch : {
                      branches : [
                          {
                              case: { $lt : ["$ARRIVAL_DELAY" ,15]},
                              then : "less than 15 mins"
                          },
                          {
                              case: { $and : [ { $gte : ["$ARRIVAL_DELAY", 15]},
{$lt : ["$ARRIVAL_DELAY", 60]}]},
                              then : "more than 15 & less than 60 mins"
                          }
                      ],
                      default: "More than 60  mins"
                  }
              }
          }
      }
  ])
```

Below is output



Notice we are displaying few set of fields and excluding the ID field as mentioned in the **$project** operator. The switch statement is what is checking the condition and

applying relevant values to the field DELAY_BRACKETS. The above query will not add a new field in existing collection but instead it will show in output only.

Now find the count of each brackets

We will use the same query as above but now add a new aggregation stage:

```
db.flights.aggregate([
    {
        $project: {
            _id : 0,
            "AIRLINE" : 1,
            "FLIGHT_NUMBER" : 1,
            "ORIGIN_AIRPORT" : 1,
            "ARRIVAL_DELAY" :1,
            "DELAY_BRACKETS" : {
                $switch : {
                    branches : [
                        {
                            case: { $lt : ["$ARRIVAL_DELAY" ,15]},
                            then : "less than 15 mins"
                        },
                        {
                            case: { $and : [ { $gte : ["$ARRIVAL_DELAY", 15]},
{$lt : ["$ARRIVAL_DELAY", 60]}]},
                            then : "more than 15 & less than 60 mins"
                        }
                    ],
                    default: "More than 60  mins"
                }
            }
        }
    },

    {$group : {_id : "$DELAY_BRACKETS", totalcount : {$sum : 1}}}
])
```

Below is the output

```
> db.flights.aggregate([
...     {
...         $project: {
...             _id : 0,
...             "AIRLINE" : 1,
...             "FLIGHT_NUMBER" : 1,
...             "ORIGIN_AIRPORT" : 1,
...             "ARRIVAL_DELAY" :1,
...             "DELAY_BRACKETS" : {
...                 $switch : {
...                     branches : [
...                         {
...                             case: { $lt : ["$ARRIVAL_DELAY" ,15]},
...                             then : "less than 15 mins"
...                         },
...                         {
...                             case: { $and : [ { $gte : ["$ARRIVAL_DELAY", 15]}, {$lt : ["$ARRIVAL_DELAY", 60]}]},
...                             then : "more than 15 & less than 60 mins"
...                         }
...                     ],
...                     default: "More than 60  mins"
...                 }
...             }
...
...         }
...     },
...
...     {$group : {_id : "$DELAY_BRACKETS", totalcount : {$sum : 1}}}
... ])
{ "_id" : "more than 15 & less than 60 mins", "totalcount" : 737848 }
{ "_id" : "More than 60  mins", "totalcount" : 430662 }
{ "_id" : "less than 15 mins", "totalcount" : 4650569 }
```

Now find the average arrival delay for each US State. Observe that US state is in airports collection which will require our query to fetch from there

One way to do is to create a new collection along with airports data:

Run below command for this.

```
db.flights.aggregate([
    { $lookup: {
        from: "airports",
        localField: "DESTINATION_AIRPORT",
        foreignField: "IATA_CODE",
        as : "DEST_AIRPORT_DETAILS"
    }},
    { $out : {db : "Flightsdata", coll: "flightdest"}}
])
```

A new collection named flightdest is created due to **$out** command. This can be verified by using “show tables” command

```
> show tables
airlines
airports
flightdest
flights
>
```

To query a state from this collection run the below command.

Since state is inside an embedded document we use “.” Notation as shown below

This will fetch the first row that has State = “CA”

```
db.flightdest.findOne({"DEST_AIRPORT_DETAILS.STATE" : "CA"})
```

Now to find the average delay by states first filter out those documents that have delays less than 0, then group by each state and take the average. MongoDB's aggregation pipeline will help us in this

```
db.flightdest.aggregate([
    {$match : { "ARRIVAL_DELAY" : {$gt : 0}}},
    {$group : {_id : "$DEST_AIRPORT_DETAILS.STATE",
                avgarrdelay : {$avg : "$ARRIVAL_DELAY"}}}
 ])
```

```
> db.flightdest.aggregate([
...     {$match : { "ARRIVAL_DELAY" : {$gt : 0}}},
...     {$group : {_id : "$DEST_AIRPORT_DETAILS.STATE",
...                 avgarrdelay : {$avg : "$ARRIVAL_DELAY"}}}
...  ]
...  )
{ "_id" : [ "AK" ], "avgarrdelay" : 24.284297520661156 }
{ "_id" : [ "MO" ], "avgarrdelay" : 31.228604151753757 }
{ "_id" : [ "NC" ], "avgarrdelay" : 28.261304078297297 }
{ "_id" : [ "KS" ], "avgarrdelay" : 31.515007146260125 }
{ "_id" : [ "AR" ], "avgarrdelay" : 32.53363567649282 }
{ "_id" : [ "TN" ], "avgarrdelay" : 31.479368625896427 }
{ "_id" : [ "WV" ], "avgarrdelay" : 32.6420395421436 }
{ "_id" : [ "DE" ], "avgarrdelay" : 45.94444444444444 }
{ "_id" : [ "UT" ], "avgarrdelay" : 28.16747203283246 }
{ "_id" : [ "WI" ], "avgarrdelay" : 32.622480355312604 }
{ "_id" : [ "GU" ], "avgarrdelay" : 36.13636363636363 }
{ "_id" : [ "AS" ], "avgarrdelay" : 33.1219512195122 }
{ "_id" : [ "VT" ], "avgarrdelay" : 39.59171597633136 }
{ "_id" : [ ], "avgarrdelay" : 26.959794690265486 }
{ "_id" : [ "SC" ], "avgarrdelay" : 31.70413927215483 }
{ "_id" : [ "WY" ], "avgarrdelay" : 33.47276264591439 }
{ "_id" : [ "NV" ], "avgarrdelay" : 30.442786533619323 }
{ "_id" : [ "NM" ], "avgarrdelay" : 28.2296146953405 }
{ "_id" : [ "PA" ], "avgarrdelay" : 33.965542403113346 }
{ "_id" : [ "VI" ], "avgarrdelay" : 34.53846153846154 }
```

As an exercise try to find the same result but without creating a separate collection.

Indexing and text search.

Indexes provides efficient execution of queries. Without indexes MongoDB checks each documents one by one ad select those that matches the query. With indexes, MongoDB will know number of documents it must check.

Create a text index which will help in finding certain words in a document.

We will create a text index on AIRPORT which is in an embedded document of DES_AIRPORT_DETAILS.

Run below command

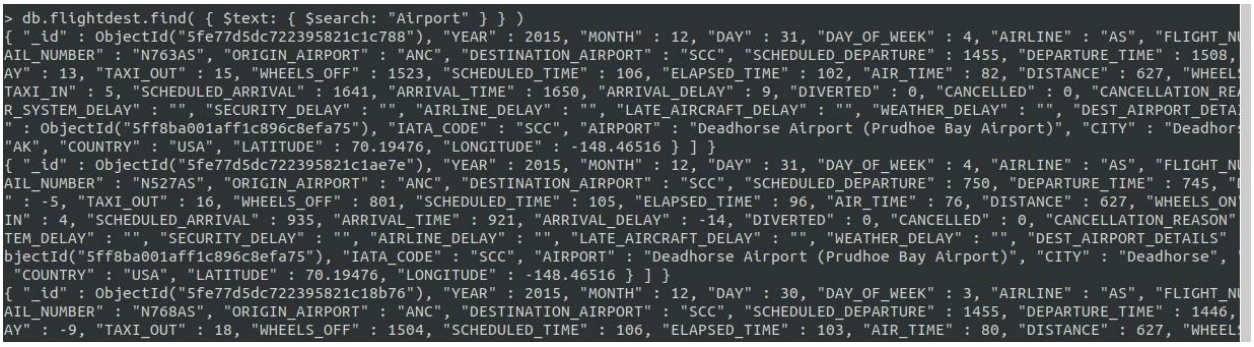
```
db.flightdest.createIndex( {"DEST_AIRPORT_DETAILS.AIRPORT" : "text" } )
```

After creating index system will output below message

```
>
>  db.flightdest.createIndex( {"DEST_AIRPORT_DETAILS.AIRPORT" : "text" } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Now search for documents containing the word “Airport”

```
db.flightdest.find( { $text: { $search: "Airport" } } )
```



Now take the count of these

```
> db.flightdest.find( { $text: { $search: "Airport" } } ).count()
5228776
>
```

Now check the documents that have the word Regional in them.

```
db.flightdest.find( { $text: { $search: "Regional" } },
                          { "DEST_AIRPORT_DETAILS.AIRPORT" : 1})
```

Update document

Count documents which have departure time but have status as cancelled.

```
db.flights.find({"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1}).count()
```

Use below query to update one document by making departure time delay field blank for cases where it is cancelled

```
db.flights.updateOne(

    {"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1},
    { $set : {"DEPARTURE_TIME" : "" , "DEPARTURE_DELAY" : ""} }
)
```

```
> db.flights.updateOne(
...     {"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1},
...     { $set : {"DEPARTURE_TIME" : "" , "DEPARTURE_DELAY" : ""} }
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Check count again

```
db.flights.find({"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1}).count()
```

```
> db.flights.find({"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1}).count()
3730
```

To update many such documents use below

```
db.flights.updateMany(
    {"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1},
    { $set : {"DEPARTURE_TIME" : "" , "DEPARTURE_DELAY" : ""} }
)
```

```
> db.flights.updateMany(
...     {"DEPARTURE_TIME" : { $ne : ""}, "CANCELLED" : 1},
...     { $set : {"DEPARTURE_TIME" : "" , "DEPARTURE_DELAY" : ""} }
... )
{ "acknowledged" : true, "matchedCount" : 3730, "modifiedCount" : 3730 }
>
```

More complicated aggregation

Find out the percentage of flights taken by UA airline in the month of December.

The following aggregation involves many stages.

```
db.flights.aggregate([
    { $match : { "CANCELLED" : 0, "MONTH" : 12}},
    { $group : {_id : "$AIRLINE", airlinecount : {$sum : 1 }}},
    {$group : {_id : 0, totalcount : {$sum : "$airlinecount"},
       airarray :{ $push :{_id: "$_id", aircount:"$airlinecount"}}}},

    {$unwind : "$airarray" },
    {$match : { "airarray._id" : "UA"}},
    {$project : {_id : "UA",  Percentage : {$divide : ["$airarray.aircount",
"$totalcount"]}}}
])
```

```
> db.flights.aggregate([
...     { $match : { "CANCELLED" : 0, "MONTH" : 12}},
...     { $group : {_id : "$AIRLINE", airlinecount : {$sum : 1 }}},
...     {$group : {_id : 0, totalcount : {$sum : "$airlinecount"},
...        airarray :{ $push :{_id: "$_id", aircount:"$airlinecount"}}}},
...
...     {$unwind : "$airarray" },
...     {$match : { "airarray._id" : "UA"}},
...     {$project : {_id : "UA",  Percentage : {$divide : ["$airarray.aircount", "$totalcount"]}}}
... ])
{ "_id" : "UA", "Percentage" : 0.09053477853924405 }
>
```

The first stage is **$match** which filters documents where flights are in december and which are not cancelled

The 2nd stage **$group** counts by airline

The 3rd stage **$group** sums up all the counts by airlines returning the total counts as well also returning an array of dictionaries containing counts by airline. The output till here looks like below. It is a single document returned

```
> db.flights.aggregate([
...     { $match : { "CANCELLED" : 0, "MONTH" : 12}},
...     { $group : {_id : "$AIRLINE", airlinecount : {$sum : 1 }}},
...     {$group : {_id : 0, totalcount : {$sum : "$airlinecount"},
...        airarray :{ $push :{_id: "$_id", aircount:"$airlinecount"}}}},])
{ "_id" : 0, "totalcount" : 471167, "airarray" : [ { "_id" : "AS", "aircount" : 14366 }, { "_id" : "B6", "aircount" : 23277 }, { "
nt" : 46195 }, { "_id" : "AA", "aircount" : 75284 }, { "_id" : "WN", "aircount" : 105854 }, { "_id" : "HA", "aircount" : 6255 }, {
ount" : 70454 }, { "_id" : "MQ", "aircount" : 20188 }, { "_id" : "EV", "aircount" : 42623 }, { "_id" : "VX", "aircount" : 5490 },
count" : 10489 }, { "_id" : "UA", "aircount" : 42657 }, { "_id" : "F9", "aircount" : 8035 } ] }
>
```

The 4th stage **$unwind** unpacks the array of dictionary **as shown below**

```
> db.flights.aggregate([
...     { $match : { "CANCELLED" : 0, "MONTH" : 12}},
...     { $group : {_id : "$AIRLINE", airlinecount : {$sum : 1 }}},
...     {$group : {_id : 0, totalcount : {$sum : "$airlinecount"},
...        airarray :{ $push :{_id: "$_id", aircount:"$airlinecount"}}}},
...
...     {$unwind : "$airarray" },])
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "OO", "aircount" : 46195 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "B6", "aircount" : 23277 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "UA", "aircount" : 42657 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "NK", "aircount" : 10489 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "F9", "aircount" : 8035 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "AS", "aircount" : 14366 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "WN", "aircount" : 105854 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "DL", "aircount" : 70454 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "MQ", "aircount" : 20188 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "HA", "aircount" : 6255 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "EV", "aircount" : 42623 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "VX", "aircount" : 5490 } }
{ "_id" : 0, "totalcount" : 471167, "airarray" : { "_id" : "AA", "aircount" : 75284 } }
>
```

The 5th stage **$match** selects document containing UA airline.

The last stage **$project** outputs _id as UA airline and percentage in decimal.

Such complex aggregation involves many stages and requires some logic to come to the required answer.

Delete documents

Delete one documents in where flight is cancelled:

```
db.flights.deleteOne({"CANCELLED" : 1})
```

```
>
> db.flights.deleteOne({"CANCELLED" : 1})
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

Now delete all documents where flights are cancelled:

```
db.flights.deleteMany({"CANCELLED" : 1})
```

```
>
> db.flights.deleteMany({"CANCELLED" : 1})
{ "acknowledged" : true, "deletedCount" : 89883 }
>
```

Use flowing command to drop database

```
db.dropDatabase()
```

Type **exit** to exit interactive shell

Stopping Docker Container

Go to command line type

```
docker ps –a
```

```
bbak@bbak:~$ sudo docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS
      PORTS               NAMES
de46bd1adaae        mongo                  "docker-entrypoint.s…"   11 minutes ago      Up 11 minutes
      27017/tcp           mongodb
161ce6a0edea        cloudera/quickstart    "/usr/bin/docker-qui…"   5 weeks ago         Exited (137) 5 weeks
ago                       peaceful_dijkstra
0585c730657c        cloudera/quickstart    "/usr/bin/docker-qui…"   6 weeks ago         Exited (137) 6 weeks
ago                       eloquent_clarke
f8401936a970        cloudera/quickstart    "/usr/bin/docker-qui…"   6 weeks ago         Exited (137) 6 weeks
ago                       crazy_pare
a61a04fb6828        cloudera/quickstart    "/usr/bin/docker-qui…"   6 weeks ago         Exited (127) 6 weeks
ago                       hopeful_poincare
bbak@bbak:~$
```

Copy the container ID associated with mongo Image and type

sudo docker stop <your container ID>

and then type

sudo docker rm <your container ID>

Exercise:

Note: (Make sure your none of the rows are deleted in collections)

1. Find the day of the week which has less no. of flights delayed.
2. Find the State with the largest number of flights where destination airport and origin airports are within that state.
3. find the airline with the most cancellation and the most common cancellation reasons
4. Find average departure delay for each airline

Answer:

1)

```
db.flights.aggregate([
    { $match : { "CANCELLED" : 0}},
    {$group : { _id : "$DAY_OF_WEEK", totalcount : {$sum : 1}}},
    {$sort : {"totalcount" : -1}}
])
```

```
> db.flights.aggregate([
...     { $match : { "CANCELLED" : 0}},
...     {$group : { _id : "$DAY_OF_WEEK", totalcount : {$sum : 1}}},
...     {$sort : {"totalcount" : -1}}
... ])
{ "_id" : 4, "totalcount" : 860230 }
{ "_id" : 5, "totalcount" : 853404 }
{ "_id" : 3, "totalcount" : 845168 }
{ "_id" : 1, "totalcount" : 844470 }
{ "_id" : 2, "totalcount" : 829528 }
{ "_id" : 7, "totalcount" : 804599 }
{ "_id" : 6, "totalcount" : 691796 }
```

2)

```
db.flights.aggregate([
    { $lookup: {
        from: "airports",
        localField: "ORIGIN_AIRPORT",
        foreignField: "IATA_CODE",
        as : "ORIGIN_AIRPORT_DETAILS"
    }},

    { $lookup: {
        from: "airports",
        localField: "DESTINATION_AIRPORT",
        foreignField: "IATA_CODE",
        as : "DEST_AIRPORT_DETAILS"
    }},
    {$match : { $expr : {$eq : ["$ORIGIN_AIRPORT_DETAILS.STATE",
"$DEST_AIRPORT_DETAILS.STATE" ]}}},
    {$group : { _id : "$ORIGIN_AIRPORT_DETAILS.STATE", total : {$sum :1 } }},
    {$sort : { "total": -1}}
])
```

The above code will take a very long time to output as MongoDB is merging collection 2 times. Better to create a new collection first and then run query to find the answer

```
{ "_id" : [ ], "total" : 486165 }
{ "_id" : [ "CA" ], "total" : 208859 }
{ "_id" : [ "TX" ], "total" : 175361 }
{ "_id" : [ "HI" ], "total" : 56280 }
{ "_id" : [ "FL" ], "total" : 21488 }
{ "_id" : [ "GA" ], "total" : 18515 }
{ "_id" : [ "AK" ], "total" : 18024 }
{ "_id" : [ "IL" ], "total" : 15063 }
{ "_id" : [ "MI" ], "total" : 11843 }
{ "_id" : [ "CO" ], "total" : 10592 }
{ "_id" : [ "NY" ], "total" : 10421 }
{ "_id" : [ "AZ" ], "total" : 9385 }
{ "_id" : [ "WA" ], "total" : 7206 }
{ "_id" : [ "MN" ], "total" : 6841 }
{ "_id" : [ "NV" ], "total" : 6434 }
{ "_id" : [ "NC" ], "total" : 4863 }
{ "_id" : [ "UT" ], "total" : 4352 }
{ "_id" : [ "MO" ], "total" : 2401 }
{ "_id" : [ "PA" ], "total" : 1812 }
{ "_id" : [ "MS" ], "total" : 907 }
Type "it" for more
>
```

3)

```
db.flights.aggregate([
    { $match : { "CANCELLED" : 1}},
    {$group : { _id : { AIRLINE: "$AIRLINE", REASON :
"$CANCELLATION_REASON"}, total : {$sum :1 } }},
    {$sort : { "total" :-1}},
    {$limit : 1}
])
```

```
> db.flights.aggregate([
...     { $match : { "CANCELLED" : 1}},
...     {$group : { _id : { AIRLINE: "$AIRLINE", REASON : "$CANCELLATION_REASON"}, total : {$sum :1 } }},
...     {$sort : { "total" :-1}},
...     {$limit : 1}
... ])
{ "_id" : { "AIRLINE" : "MQ", "REASON" : "B" }, "total" : 9164 }
>
```

4)

```
db.flights.aggregate([
    {$match : { "ARRIVAL_DELAY" : {$gt : 0}}},
    {$group : {_id : "$AIRLINE",
                avgarrdelay : {$avg : "$ARRIVAL_DELAY"}}}
 ])
```

```
> db.flights.aggregate([
...     {$match : { "ARRIVAL_DELAY" : {$gt : 0}}},
...     {$group : {_id : "$AIRLINE",
...                 avgarrdelay : {$avg : "$ARRIVAL_DELAY"}}}
...  ])
{ "_id" : "HA", "avgarrdelay" : 15.37976738791875 }
{ "_id" : "EV", "avgarrdelay" : 35.19804236997988 }
{ "_id" : "MQ", "avgarrdelay" : 39.50920245398773 }
{ "_id" : "F9", "avgarrdelay" : 41.19043461389212 }
{ "_id" : "WN", "avgarrdelay" : 29.418495773917883 }
{ "_id" : "B6", "avgarrdelay" : 38.13280652561815 }
{ "_id" : "US", "avgarrdelay" : 27.41992528019925 }
{ "_id" : "AA", "avgarrdelay" : 34.148363740181054 }
{ "_id" : "DL", "avgarrdelay" : 32.077423855844366 }
{ "_id" : "VX", "avgarrdelay" : 30.72522746071133 }
{ "_id" : "AS", "avgarrdelay" : 22.562411110916017 }
{ "_id" : "UA", "avgarrdelay" : 39.20931443883003 }
{ "_id" : "NK", "avgarrdelay" : 40.659851987273015 }
{ "_id" : "OO", "avgarrdelay" : 32.43727830602199 }
>
```