Project Phase II

SOEN 363: Database Systems for Software Engineers

Group 13
Mila Roisin (29575774)
Collin Zhou (40033295)
Michelle Choi (26307647)

Concordia University
Prof. Essam Mansour

April 15, 2020

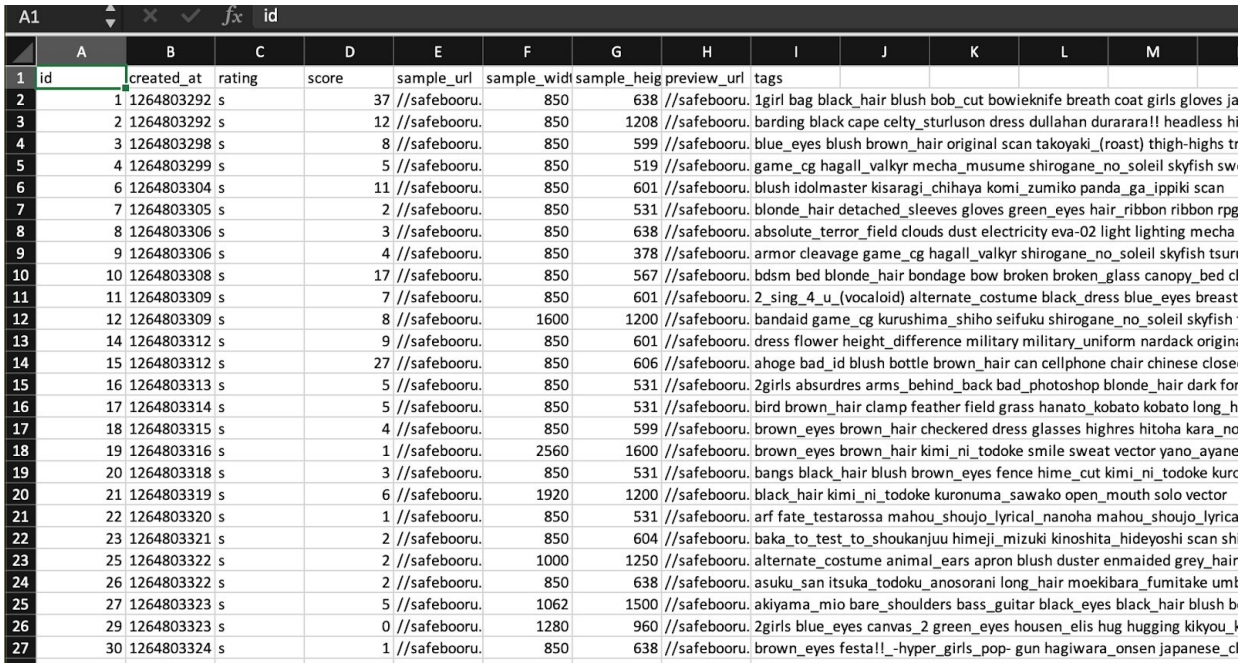
Analyzing Big Data Using NoSQL Systems

(a) *Download a big real dataset; it is recommended to get a dataset of at least 0.5 GBs.*

We selected the following dataset from Kaggle: [SafeBooru - Anime Image Dataset]

Characteristics of chosen dataset
- Metadata: 2,736,037 rows of tag-based anime image metadata - Contains 2736034 unique values - Size: 1.24 GB - 9 Columns - Filetype: .csv (comma separated values) - No. of files: 1

Preview of dataset:



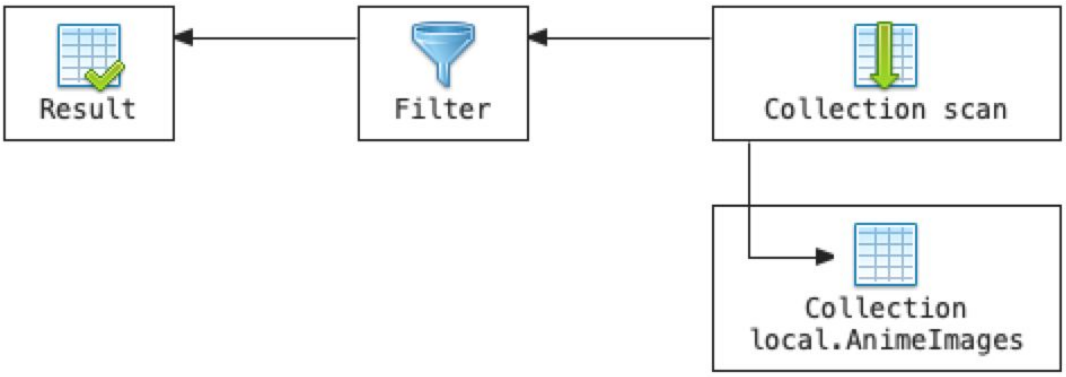
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	id	created_at	rating	score	sample_url	sample_wid	sample_heig	preview_url	tags				
2	1	1264803292	s	37	//safebooru.	850	638	//safebooru.	1girl bag black_hair blush bob_cut bowieknife breath coat girls gloves ja				
3	2	1264803292	s	12	//safebooru.	850	1208	//safebooru.	barding black cape celty_sturluson dress dullahan durarara!! headless hi				
4	3	1264803298	s	8	//safebooru.	850	599	//safebooru.	blue_eyes blush brown_hair original scan takoyaki_(roast) thigh-highs tr				
5	4	1264803299	s	5	//safebooru.	850	519	//safebooru.	game_cg hagall_valkyr mecha_musume shirogane_no_soleil skyfish sw				
6	6	1264803304	s	11	//safebooru.	850	601	//safebooru.	blush idolmaster kisaragi_chihaya komi_zumiko panda_ga_ippiki scan				
7	7	1264803305	s	2	//safebooru.	850	531	//safebooru.	blonde_hair detached_sleeves gloves green_eyes hair_ribbon ribbon rpg				
8	8	1264803306	s	3	//safebooru.	850	638	//safebooru.	absolute_terror_field clouds dust electricity eva-02 light lighting mecha				
9	9	1264803306	s	4	//safebooru.	850	378	//safebooru.	armor cleavage game_cg hagall_valkyr shirogane_no_soleil skyfish tsur				
10	10	1264803308	s	17	//safebooru.	850	567	//safebooru.	bdsm bed blonde_hair bondage bow broken broken_glass canopy_bed cl				
11	11	1264803309	s	7	//safebooru.	850	601	//safebooru.	2_sing_4_u_(vocaloid) alternate_costume black_dress blue_eyes breast				
12	12	1264803309	s	8	//safebooru.	1600	1200	//safebooru.	bandaid game_cg kurushima_shiho seifuku shirogane_no_soleil skyfish				
13	14	1264803312	s	9	//safebooru.	850	601	//safebooru.	dress flower height_difference military military_uniform nardack origin				
14	15	1264803312	s	27	//safebooru.	850	606	//safebooru.	ahoge bad_id blush bottle brown_hair can cellphone chair chinese close				
15	16	1264803313	s	5	//safebooru.	850	531	//safebooru.	2girls absurdres arms_behind_back bad_photoshop blonde_hair dark for				
16	17	1264803314	s	5	//safebooru.	850	531	//safebooru.	bird brown_hair clamp feather field grass hanato_kobato kobato long_h				
17	18	1264803315	s	4	//safebooru.	850	599	//safebooru.	brown_eyes brown_hair checkered dress glasses highres hitoha kara_no				
18	19	1264803316	s	1	//safebooru.	2560	1600	//safebooru.	brown_eyes brown_hair kimi_ni_todoke smile sweat vector yano_ayane				
19	20	1264803318	s	3	//safebooru.	850	531	//safebooru.	bangs black_hair blush brown_eyes fence hime_cut kimi_ni_todoke kurc				
20	21	1264803319	s	6	//safebooru.	1920	1200	//safebooru.	black_hair kimi_ni_todoke kuronuma_sawako open_mouth solo vector				
21	22	1264803320	s	1	//safebooru.	850	531	//safebooru.	arf fate_testarossa mahou_shoujo_lyrical_nanoha mahou_shoujo_lyrica				
22	23	1264803321	s	2	//safebooru.	850	604	//safebooru.	baka_to_test_to_shoukanjuu himeji_mizuki kinoshita_hideyoshi scan sh				
23	25	1264803322	s	2	//safebooru.	1000	1250	//safebooru.	alternate_costume animal_ears apron blush duster enmaided grey_hair				
24	26	1264803322	s	2	//safebooru.	850	638	//safebooru.	asuku_san itsuka_todoku_anosorani long_hair moekibara_fumitake umb				
25	27	1264803323	s	5	//safebooru.	1062	1500	//safebooru.	akiyama_mio bare_shoulders bass_guitar black_eyes black_hair blush b				
26	29	1264803323	s	0	//safebooru.	1280	960	//safebooru.	2girls blue_eyes canvas_2 green_eyes housen_elis hug hugging kikyou_k				
27	30	1264803324	s	1	//safebooru.	850	638	//safebooru.	brown_eyes festa!!_-hyper_girls_pop- gun hagiwara_onsen japanese_cl				

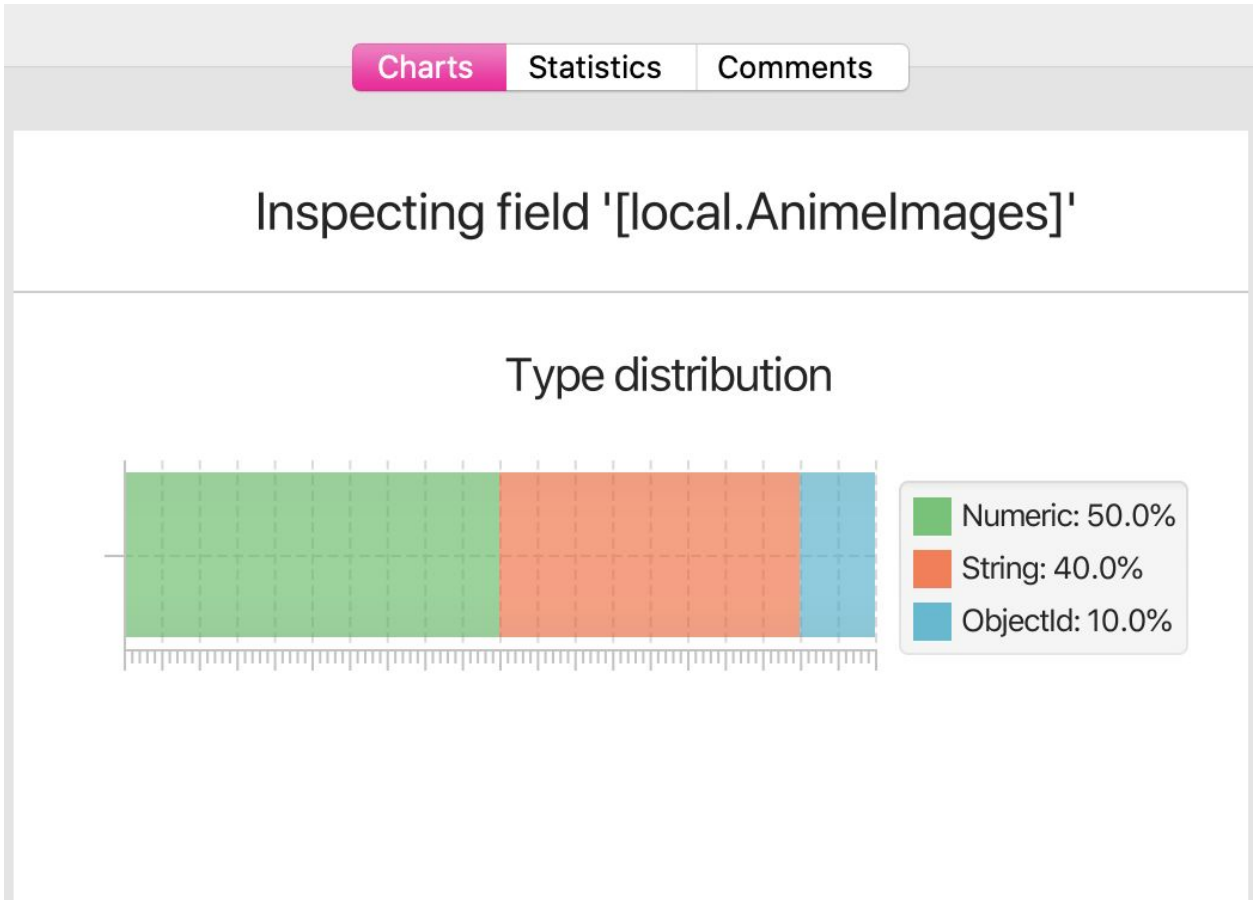
(b) Provide the data model for your datasets (i.e. graph, document, key-value, or column-store).

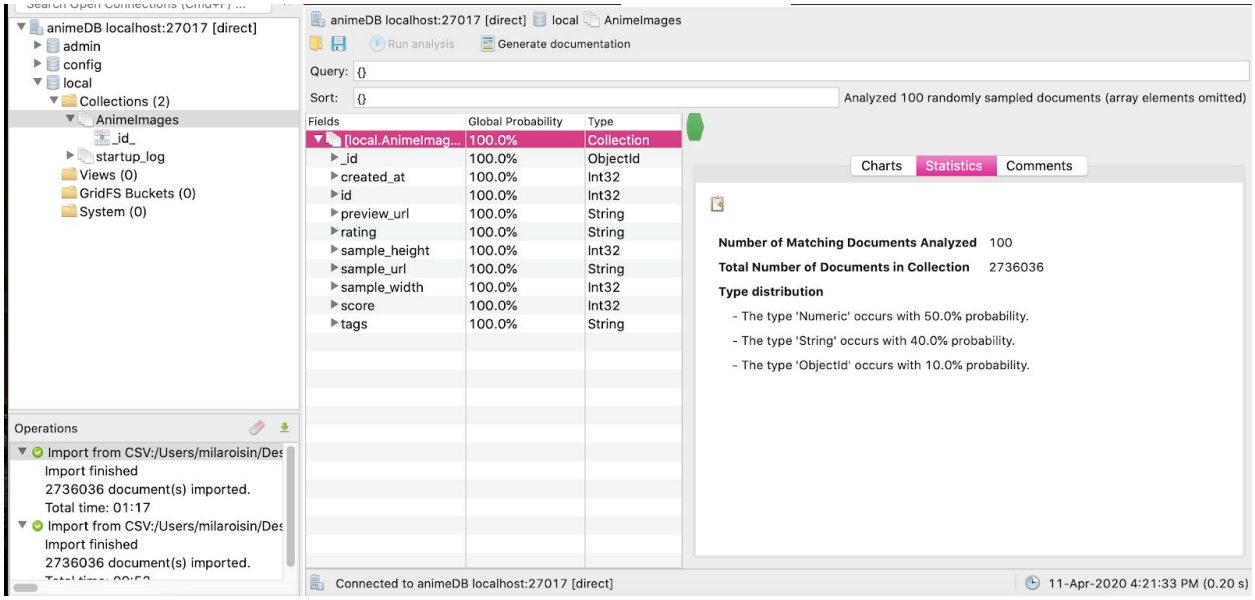
We'll be using MongoDB for our metadata database analysis. MongoDB is a document-oriented database. We used the Studio3T GUI to help streamline our queries with MongoDB. We included the schema report generated by the IDE alongside this document for your perusal.

Here are the columns for our dataset:

id	created_at	rating	score	sample_url	sample_width	sample_height	preview_url	tags
unique id for each image	time in seconds since epoch (defaults to 0 if there was an error)	always 's' for safe, as all images in the booru are non-explicit	net upvote / downvote score for image	url for medium-size version of image	width of medium-size version of image	height of medium-size version of image	url for thumbnail version of image	space-separated list of tags





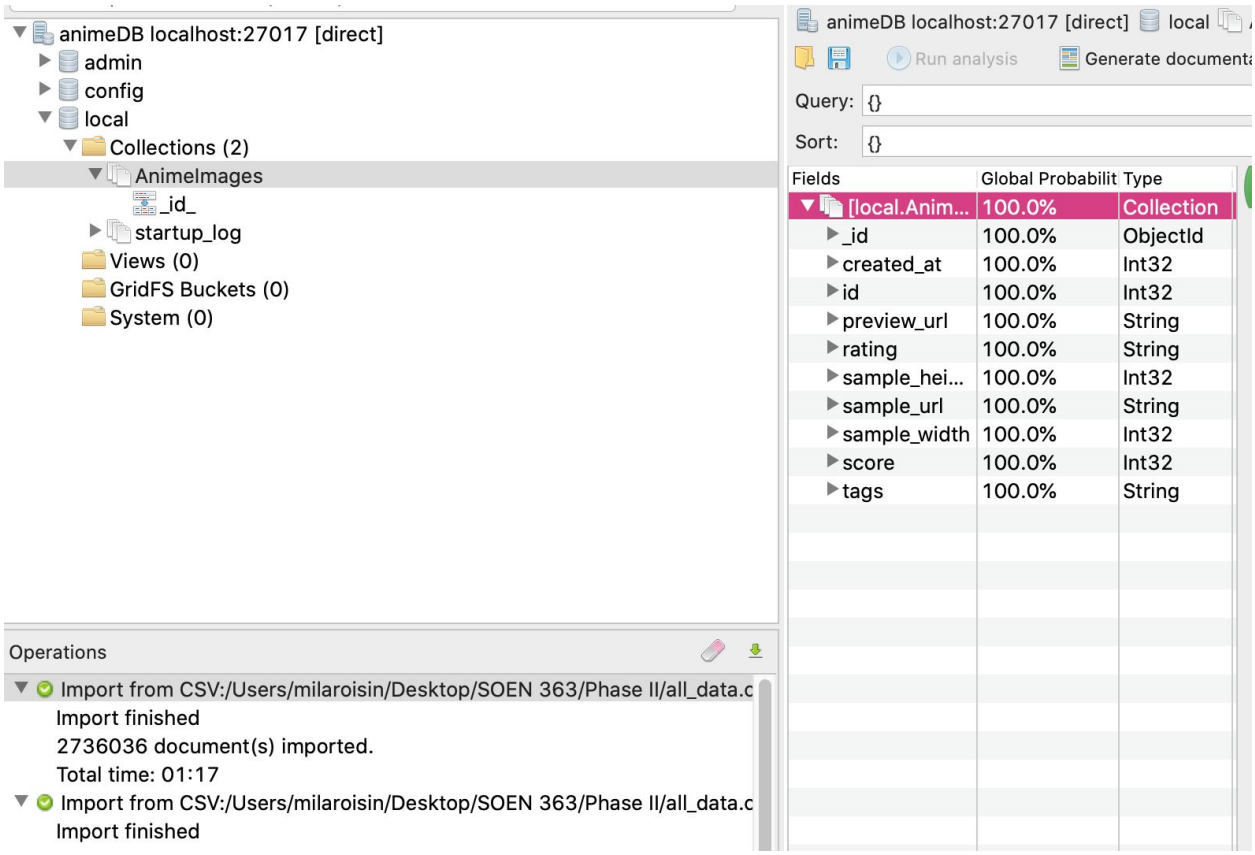


(c) Create a noSQL database for a real dataset of your choice.

We created the database ‘animeDB’ to load our big dataset for MongoDB with the document collection as ‘AnimeImages’.

(d) Load the dataset into your NoSQL database.

See screenshot below for successful loading of our CSV file into our NoSQL database with the correct field data type.



(e) Write at least 10 different queries that show some useful information about the dataset. This should include different aspects of your NoSQL.

For this part we used a neat feature that is part of the **Studio3T IDE**: the Visual Query Builder.

1. Images that have ratings other than ‘s’:

   Query Code:

```
use local;
db.getCollection("AnimeImages").find(
   {
      "rating" : {
         "$ne" : "s"
      }
   }
);
```

Results: 14,522 documents rendered in 2.893 seconds.

2. Images with net score > 20

```
use local;
db.getCollection("AnimeImages").find(
   {
      "score" : {
         "$gt" : NumberInt(20)
      }
   }
);
```

Results: 47 documents rendered in 3.384 seconds.

3. Image Height greater or equal to 1000 pixels AND Image Width greater or equal to 1000 pixels

Query:

```
// Requires official MongoShell 3.6+
use local;
db.getCollection("AnimeImages").find(
   {
      "sample_width" : {
         "$gte" : NumberInt(1000)
      },
      "sample_height" : {
         "$gte" : NumberInt(1000)
      }
   }
);
```

Result: 179, 083 documents rendered in 0.508 seconds.

4. Images that contain tags with an underscore “_”, height greater than 1000 pixels and width greater than 1000 pixels.

Query:

```
use local;
db.getCollection("AnimeImages").find(
   {
      "$or" : [
         {
            "tags" : /.*_.*/i
         },
         {
            "sample_height" : {
               "$gt" : NumberInt(1000)
            }
         }
      ],
      "sample_width" : {
         "$gt" : NumberInt(1000)
      }
   }
);
```

Results: 127, 452 documents rendered in 0.430 seconds

5. Images with width that equal 850 pixels

Query:

```
use local;
db.getCollection("AnimeImages").find(
   {
      "$or" : [
         {
            "sample_width" : NumberInt(850)
         }
      ]
   }
);
```

Result: 887,704 documents rendered in 0.187 seconds.

6. File types in Portable Networks Graphic (PNG) format

Query:

```
// Requires official MongoShell 3.6+
use local;
```

```
db.getCollection("AnimeImages").find(
   {
      "preview_url" : /.*\.png.*/i
   }
);
```

Results: 697, 283 documents rendered in 0.427 seconds.

7. File types in Joint Photographic Experts Group (JPG) format

Query:

```
// Requires official MongoShell 3.6+
use local;
db.getCollection("AnimeImages").find(
   {
      "sample_url" : /.*jpg.*/i
   },
   {
      "" : 1.0
   }
);
```

Results: 1, 928, 628 documents rendered in 0.157 seconds.

8. Images that contain brown eyes and brown hair:

Query:

```
// Requires official MongoShell 3.6+
use local;
db.getCollection("AnimeImages").find(
   {
      "$and" : [
         {
            "tags" : /.*brown_eyes.*/i
         },
         {
            "tags" : /.*brown_hair.*/i
         }
      ]
   }
);
```

Results: 188, 879 documents rendered in 1.148 seconds.

9. Images that are created in vector format (meaning artwork created with lines, points and curves made from mathematical equations rather than pixels)

Query:

```
// Requires official MongoShell 3.6+
use local;
db.getCollection("AnimeImages").find(
   {
      "tags" : /.*vector.*/i
   }
);
```

Results: 5,064 documents rendered in 17.457 seconds.

10. Images correlation for socks and stripes:

```
// Requires official MongoShell 3.6+
use local;
db.getCollection("AnimeImages").find(
   {
      "$and" : [
         {
            "tags" : /.*socks.*/i
         },
         {
            "tags" : /.*stripes.*/i
         }
      ]
   }
);
```

Results: 680 documents rendered in 25.896 seconds.

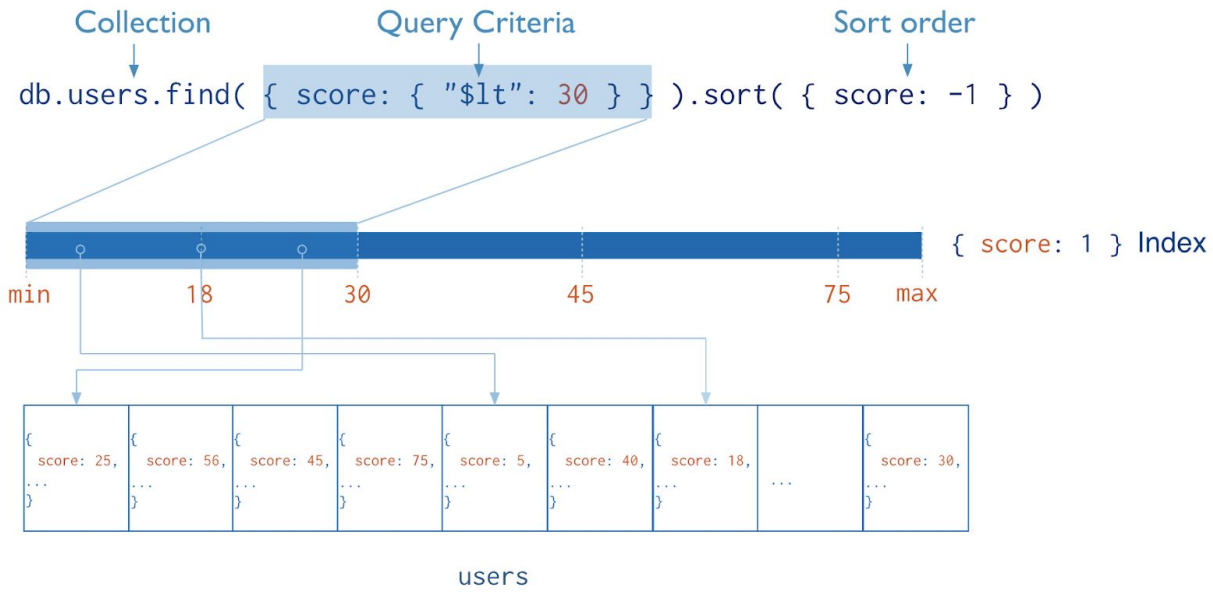
**Note: All query results are extracted in CSV format and zipped along with this report.

(f) Investigate the balance between the consistency and availability in your NoSQL system.

So looking further into this question, I came across the concept of the CAP theorem. The theorem suggests that a distributed database system relies on two of the three factors for a successful system: consistency (c), availability (a), and partition tolerance (p). For every choice made in a system, there is an opportunity cost that is made. There will be a trade off made between making choices between the three factors. IBM provided a clever expression to explain the CAP theorem: “Cheap, Fast, and Good: Pick Two”. MongoDB utilizes two factors of the CAP theorem: consistency and partition tolerance. Availability is not considered in this system. As it uses one primary node to conduct its queries, if it is preoccupied while doing a transaction, the system is put on hold until it frees up again.

(g) Investigate the indexing techniques available in your NoSQL system.

So according to the MongoDB documentation, “the best indexes for [an] application must take a number of factors into account, including the kinds of queries you expect, the ratio of reads to writes, and the amount of free memory on your system.” It works similar to other database systems except there’s a concept of indexes at the collections level and it works like a B-tree data structure. The primary key equivalence would be the unique index called _id. This is to prevent duplicate versions of the data and it uses this one index to evaluate queries.



Furthermore, MongoDB implements multikey indexes to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array. These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays. These indexes must make sure that they fit the RAM size so it can deliver strong performance and efficient processing.

More details for their indexing techniques can be located here in the MongoDB documentation manual: Indexing Strategies.