PART 1 - READING ASSIGNMENT

https://learning.oreilly.com/library/view/mongodb-in-action/9781617291609/?ar (Links to an external site.)
Chapter 1. A database for the modern web
Chapter 2. MongoDB through the JavaScript shell

PART 2 - READING ASSIGNMENT (Papers Attached)

Read the following papers and provide a short summary for each paper.
- Application of NoSQL Database in Web Crawling
- Comparing NoSQL MongoDB to an SQL
- Data Aggregation System

Application of NoSQL Database in Web Crawling

The principle for web crawling is with the underlying process that obtains the information from webpages and provide a link between them. The web crawling process consists of a spider which is used to crawl the pages from the website. These spiders can be also used in parallel to Improve the crawl rate. The Controller is to control the control the spider until the URL database is empty. Meteorological BBS information collection system provides a professional search engine database of meteorological information. It uses relational (MySQL) and NoSQL (MongoDB) database for it. The posts in BBS are stores in the database as text files which have a uniform format. The traditional database stores it in the table format and offers less flexibility to store these text documents. The MongoDB is a NOSQL database which has higher performance and scalability. MongoDB also supports the master/Slave replication set.TO finalize the solution, MongoDB is schema free and we do not need to design the structure beforehand. In Relational databases we have to store data across multiple tables and then we need to join and refer it back. Next the query, we can query with a single query id and then we can get to know all the post and its corresponding floors. Comparing the data size, the query speed becomes slower in RDBMS and MongoDB is much faster. Also to add, scalability in mongoDB is achieved much easily with auto sharding while horizontal scalability is not good across RDBMS. Hence to summarize, MongoDB works better with large sized documents as is the result with web crawling.

Comparing NoSQL MongoDB to an SQL

As the data collected today is rapidly increasing, NOSQL database solutions are much more in use and becoming more useful. If the data is non structured and large then NoSQL database would be a good choice. The computation and processing is simpler, more affordable and more flexible in MongoDB for key value pair of data. MongoDB does not require join operation. One option can be storing the data in such a way that it nested in the same document. The second option is to store a reference to the other document rather than nesting the entire document. There are some disadvantages for using a NoSQL over an RDBMS. MongoDB only provides atomic operations within a single document and it's also does not have much simple aggregate functions. So MongoDB used Map reduce to its rescue. In the map phase we emit the key, value pair and in the reduce phase we apply aggregation to get the results. The paper carried out an experiment to showcase the difference between MongoDB and SQL server. It compared the difference in insert speed between the two database systems. MongoDB outperformed SQL in all cases in which the update involved using the primary key. The better performance in MongoDB having a pre-built index on the primary key of the document which

is faster than SQL Server's primary key clustered index. For the select operation, MongoDB performs well on the complex queries, except when it comes to using aggregate functions. It can be concluded that MongoDB could be preferred for application where schema keeps changing frequently, high availability is required and don't need use aggregation function on frequent basis.

Data Aggregation System:

The data aggregation removes complication by giving us a single point of access to the data and for the data services it issues a caching layer.

Architecture of DAS:

- Web server is used to handle the user sessions
- Cached Server receives the queries made to the cache server to process. Cache server has multiple worker nodes to handle the queries. Cache servers also have a limited amount of data as it can regenerate the cache anytime from the raw data
- DAS uses Json for the internal representation of primary records and for storing the server logs, the analytical data
- GridFS is used when the document size is large after merging the document together
- Analytical Server is used for the scheduling of the tasks
- DAS is only a read-only system

DAS Query Language:

- The query are structured as : Conditions | filters | aggregators or map-reduce
- Conditions consists of one or more DAS Keys. Filters are used to filter the records. Aggregators are used to summarize the records.

Services:

- The services used by DAS returns the JSON or XML format and are accessed over HTTP.

Caching and Merging:

- There are two separate collection which comprises of the raw and merged collection
- The raw caches stores the document returned by the specific API and also the current status of the queries

Analytics:

- The DAS Analytical is a daemon is use access the DAS document store and which in turn is used to schedule and also execute the tasks

Benchmarks:

- To benchmark DAS execution, we utilized a 64-cycle linux hub with 8 centers (each 2.33GhZ) and 16GB of RAM, which we would hope to be regular of the equipment DAS would use underway. All DAS frameworks and MongoDB share this hub

PART 3 - PROGRAMMING ASSIGNMENT

Create a database for a Contact Management System in MongoDB

```
C:\mongodb\bin\mongo.exe
> use contactManagementSystem
switched to db contactManagementSystem
> db
contactManagementSystem
>
```

You could use any attributes you like, first name, last name, phone, address, zip, etc.

Create 5 records with different attributes and values you choose.

```
C:\mongodb\bin\mongo.exe
> use contactManagementSystem
switched to db contactManagementSystem
> db
contactManagementSystem
> try {
...    db.users.insertMany( [
...       { Name: { firstName: "A", lastName: "B"}, phone: 1234, address: "AB street, Boston, MA" , gender: "M", age : 25 },
...       { Name: { firstName: "C", lastName: "D"}, phone: 5678, address: "CD street, Boston, MA"  , gender: "M", age : 30  },
...       { Name: { firstName: "E", lastName: "F"}, phone: 1011, address: "EF street, Boston, MA" , gender: "F", age : 35  },
...   { Name: { firstName: "G", lastName: "H"}, phone: 1314, address: "GH street, Boston, MA" , gender: "F", age : 40  },
...   { Name: { firstName: "I", lastName: "J"}, phone: 1213, address: "IJ street, Boston, MA" , gender: "F", age : 45  }
...
...    ] );
... } catch (e) {
...    print (e);
... }
{
       "acknowledged" : true,
       "insertedIds" : [
              ObjectId("5f6ae2068853368b2acedbd2"),
              ObjectId("5f6ae2068853368b2acedbd3"),
              ObjectId("5f6ae2068853368b2acedbd4"),
              ObjectId("5f6ae2068853368b2acedbd5"),
              ObjectId("5f6ae2068853368b2acedbd6")
       ]
}
>
```

Then delete any one record of your choice.

```
}
> db.users.deleteOne( { age: 25 } )
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

Then update some information from any one of the records of your choice

```
> db.users.update(
... {"gender" : "M"},
... {$set: { "gender" : "Male"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find()
{ "_id" : ObjectId("5f6ae2068853368b2acedbd3"), "Name" : { "firstName" : "C", "lastName" : "D" }, "pho
ne" : 5678, "address" : "CD street, Boston, MA", "gender" : "Male", "age" : 30 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd4"), "Name" : { "firstName" : "E", "lastName" : "F" }, "pho
ne" : 1011, "address" : "EF street, Boston, MA", "gender" : "F", "age" : 35 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd5"), "Name" : { "firstName" : "G", "lastName" : "H" }, "pho
ne" : 1314, "address" : "GH street, Boston, MA", "gender" : "F", "age" : 40 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd6"), "Name" : { "firstName" : "I", "lastName" : "J" }, "pho
ne" : 1213, "address" : "IJ street, Boston, MA", "gender" : "F", "age" : 45 }
>
```

PART 4 - PROGRAMMING ASSIGNMENT

Create a collection called ‘games’. We’re going to put some games in it.
Add 5 games to the database.
Give each document the following properties: name, genre, rating (out of 100)

```
db.games.insertMany([{"name":"a", "genre":"Comedy" ,"rating(out of 100)": 90},{"name":"b", "genre":"Action" ,"rating(out of 100)": 95},
{"name":"c", "genre":"Drama" ,"rating(out of 100)": 70}, {"name":"d", "genre":"Horror" ,"rating(out of 100)": 75},
{"name":"e", "genre":"Thriller" ,"rating(out of 100)": 80}, {"name":"f", "genre":"Romance" ,"rating(out of 100)": 20}])
```

0.016 sec.

Key	Value	Type
(1)	{ 2 fields }	Object
acknowledged	true	Boolean
insertedIds	[6 elements]	Array
[0]	ObjectId("5f6f750d30050a13158cf388")	ObjectId
[1]	ObjectId("5f6f750d30050a13158cf389")	ObjectId
[2]	ObjectId("5f6f750d30050a13158cf38a")	ObjectId
[3]	ObjectId("5f6f750d30050a13158cf38b")	ObjectId
[4]	ObjectId("5f6f750d30050a13158cf38c")	ObjectId
[5]	ObjectId("5f6f750d30050a13158cf38d")	ObjectId

If you make some mistakes and want to clean it out, use remove() on your collection.

```
db.games.remove( { genre: "Horror" })
```

0.005 sec.

```
Removed 1 record(s) in 4ms
```

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
achievements	[2 elements]	Array
(2) ObjectId("5f6f750d30050a13158cf389")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf389")	ObjectId
name	b	String
genre	Action	String
rating(out of 100)	95.0	Double
achievements	[2 elements]	Array
(3) ObjectId("5f6f750d30050a13158cf38a")	{ 2 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38a")	ObjectId
achievements_using_save	[2 elements]	Array
(4) ObjectId("5f6f750d30050a13158cf38c")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38c")	ObjectId
name	e	String
genre	Thriller	String
rating(out of 100)	80.0	Double
(5) ObjectId("5f6f750d30050a13158cf38d")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38d")	ObjectId
name	f	String
genre	Romance	String
rating(out of 100)	20.0	Double

Write a query that returns all the games.

```
db.getCollection('games').find({})
```

games 0.001 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
(2) ObjectId("5f6f750d30050a13158cf389")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf389")	ObjectId
name	b	String
genre	Action	String
rating(out of 100)	95.0	Double
(3) ObjectId("5f6f750d30050a13158cf38a")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38a")	ObjectId
name	c	String
genre	Drama	String
rating(out of 100)	70.0	Double
(4) ObjectId("5f6f750d30050a13158cf38b")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38b")	ObjectId
name	d	String
genre	Horror	String
rating(out of 100)	75.0	Double
(5) ObjectId("5f6f750d30050a13158cf38c")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38c")	ObjectId
name	e	String
genre	Thriller	String
rating(out of 100)	80.0	Double
(6) ObjectId("5f6f750d30050a13158cf38d")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38d")	ObjectId
name	f	String
genre	Romance	String
rating(out of 100)	20.0	Double

Write a query to find one of your games by name without using limit()

```
db.getCollection('games').find({name:"a"})
```

games 0.001 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
achievements	[2 elements]	Array

Use the findOne method. Look how much nicer it's formatted!

```
db.getCollection('games').findOne({})
```

0.001 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double

Write a query that returns the 3 highest rated games.

```
db.games.find().sort({"rating(out of 100)":-1}).limit(3)
```

games 0.001 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf389")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf389")	ObjectId
name	b	String
genre	Action	String
rating(out of 100)	95.0	Double
(2) ObjectId("5f6f750d30050a13158cf388")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
(3) ObjectId("5f6f750d30050a13158cf38c")	{ 4 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38c")	ObjectId
name	e	String
genre	Thriller	String
rating(out of 100)	80.0	Double

Update your two favourite games to have two achievements called ‘Game Master’ and ‘Speed Demon’, each under a single key.
Show two ways to do this.
Do the first using update() and do the second using save().
Hint: for save, you might want to query the object and store it in a variable first.

Using Update:

```
db.games.updateMany(
   { "name":{$in:["a","b"]}},
   { $set : { achievements: ["Game Master","Speed Demon"]}});

db.getCollection('games').find({})
```

0 sec.

Key	Value	Type
(1)	{ 3 fields }	Object

games 0 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
achievements	[2 elements]	Array
[0]	Game Master	String
[1]	Speed Demon	String
(2) ObjectId("5f6f750d30050a13158cf389")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf389")	ObjectId
name	b	String
genre	Action	String
rating(out of 100)	95.0	Double
achievements	[2 elements]	Array
[0]	Game Master	String
[1]	Speed Demon	String
(3) ObjectId("5f6f750d30050a13158cf38a")	{ 4 fields }	Object
(4) ObjectId("5f6f750d30050a13158cf38b")	{ 4 fields }	Object
(5) ObjectId("5f6f750d30050a13158cf38c")	{ 4 fields }	Object
(6) ObjectId("5f6f750d30050a13158cf38d")	{ 4 fields }	Object

Using Save:

```
db.games.save( { _id : ObjectId("5f6f750d30050a13158cf38a"), "achievements_using_save":["Game Master","Speed Demon"] } )
```

0.006 sec.

```
Updated 1 existing record(s) in 4ms
```

```
db.getCollection('games').find({})
```

games 0.004 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
(2) ObjectId("5f6f750d30050a13158cf389")	{ 5 fields }	Object
(3) ObjectId("5f6f750d30050a13158cf38a")	{ 2 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf38a")	ObjectId
achievements_using_save	[2 elements]	Array
[0]	Game Master	String
[1]	Speed Demon	String
(4) ObjectId("5f6f750d30050a13158cf38b")	{ 4 fields }	Object
(5) ObjectId("5f6f750d30050a13158cf38c")	{ 4 fields }	Object
(6) ObjectId("5f6f750d30050a13158cf38d")	{ 4 fields }	Object

Write a query that returns all the games that have both the ‘Game Master’ and the ‘Speed Demon’ achievements.

```
db.games.find({$and : [{"achievements":"Game Master"},{"achievements":"Speed Demon"}]})
```

games 0.001 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
achievements	[2 elements]	Array
[0]	Game Master	String
[1]	Speed Demon	String
(2) ObjectId("5f6f750d30050a13158cf389")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf389")	ObjectId
name	b	String
genre	Action	String
rating(out of 100)	95.0	Double
achievements	[2 elements]	Array
[0]	Game Master	String
[1]	Speed Demon	String

Write a query that returns only games that have achievements.
Not all of your games should have achievements, obviously.

```
db.games.find({"achievements":{'$exists': 1}})
```

games 0.001 sec.

Key	Value	Type
(1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf388")	ObjectId
name	a	String
genre	Comedy	String
rating(out of 100)	90.0	Double
achievements	[2 elements]	Array
(2) ObjectId("5f6f750d30050a13158cf389")	{ 5 fields }	Object
_id	ObjectId("5f6f750d30050a13158cf389")	ObjectId
name	b	String
genre	Action	String
rating(out of 100)	95.0	Double
achievements	[2 elements]	Array

PART 5 - PROGRAMMING ASSIGNMENT

Execute 5 commands of your choice from each of the following groups, and paste the screenshots in a word document

mongo> help [5 commands]

```
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        sh.help()                    sharding helpers
        rs.help()                    replica set helpers
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce

        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
        show profile                 show most recent system.profile entries with
        show logs                    show the accessible logger names
        show log [name]              prints out the last segment of log in memory
        use <db_name>                set current database
        db.foo.find()                list objects in collection foo
        db.foo.find( { a : 1 } )     list objects in foo where a == 1
        it                           result of the last line evaluated; use to fu
        DBQuery.shellBatchSize = x   set default number of items to display on sh
        exit                         quit the mongo shell
```

Show dbs

```
> show dbs
admin                   0.000GB
config                  0.000GB
contactManagementSystem 0.000GB
local                   0.000GB
moviedb                 0.000GB
nyseDB                  0.055GB
stocksDB                0.004GB
testdb                  0.000GB
userslab2               0.000GB
```

Use db

```
> use contactManagementSystem
switched to db contactManagementSystem
```

Show collections

```
> show collections
games
users
```

db.collection.find()

```
> db.users.find()
{ "_id" : ObjectId("5f6ae2068853368b2acedbd3"), "Name" : { "firstName" : "C", "lastName" : "D" }, "phone" : 5678, "address" : "CD street, Boston,
, "age" : 30 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd4"), "Name" : { "firstName" : "E", "lastName" : "F" }, "phone" : 1011, "address" : "EF street, Boston,
age" : 35 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd5"), "Name" : { "firstName" : "G", "lastName" : "H" }, "phone" : 1314, "address" : "GH street, Boston,
age" : 40 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd6"), "Name" : { "firstName" : "I", "lastName" : "J" }, "phone" : 1213, "address" : "IJ street, Boston,
age" : 45 }
```

Db.collection.find(condition)

```
> db.users.find({gender : "Male"})
{ "_id" : ObjectId("5f6ae2068853368b2acedbd3"), "Name" : { "firstName" : "C", "lastName" : "D" }, "phone" : 5678, "address" : "CD street, Boston, MA", "gender" : "Male"
, "age" : 30 }
```

mongo> db.help() [5 commands]

db.getName()

```
> db.getName()
contactManagementSystem
```

Db.stats()

```
> db.stats()
{
        "db" : "contactManagementSystem",
        "collections" : 2,
        "views" : 0,
        "objects" : 10,
        "avgObjSize" : 117.1,
        "dataSize" : 1171,
        "storageSize" : 73728,
        "numExtents" : 0,
        "indexes" : 2,
        "indexSize" : 49152,
        "fsUsedSize" : 225477861376,
        "fsTotalSize" : 247584882688,
        "ok" : 1
}
```

Db.getCollectionNames()

```
db.getCollectionNames()
"games", "users" ]
```

Db.hostinfo()

```
> db.hostInfo()
{
        "system" : {
                "currentTime" : ISODate("2020-09-27T08:09:19.581Z"),
                "hostname" : "SatwikPC",
                "cpuAddrSize" : 64,
                "memSizeMB" : 16218,
                "memLimitMB" : 16218,
                "numCores" : 8,
                "cpuArch" : "x86_64",
                "numaEnabled" : false
        },
        "os" : {
                "type" : "Windows",
                "name" : "Microsoft Windows 10",
                "version" : "10.0 (build 18362)"
        },
        "extra" : {
                "pageSize" : NumberLong(4096)
        },
        "ok" : 1
}
```

Db.logout()

```
> db.logout()
{ "ok" : 1 }
```

mongo> db.mycoll.help() [10 commands]

db.col.dataSize()

```
> db.users.dataSize()
575
```

Db.col.find().help()

```
> db.users.find().help()
find(<predicate>, <projection>) modifiers
        .sort({...})
        .limit(<n>)
        .skip(<n>)
        .batchSize(<n>) - sets the number of docs to return per getMore
        .collation({...})
        .hint({...})
        .readConcern(<level>)
        .readPref(<mode>, <tagset>)
        .count(<applySkipLimit>) - total # of objects matching query. by default ignores skip,limit
        .size() - total # of objects cursor would return, honors skip,limit
        .explain(<verbosity>) - accepted verbosities are {'queryPlanner', 'executionStats', 'allPlansExecution'}
        .min({...})
        .max({...})
        .maxScan(<n>)
        .maxTimeMS(<n>)
        .comment(<comment>)
        .tailable(<isAwaitData>)
        .noCursorTimeout()
        .allowPartialResults()
        .returnKey()
        .showRecordId() - adds a $recordId field to each returned object

Cursor methods
        .toArray() - iterates through docs and returns an array of the results
        .forEach(<func>)
        .map(<func>)
        .hasNext()
        .next()
        .close()
        .objsLeftInBatch() - returns count of docs left in current batch (when exhausted, a new getMore will be issued)
        .itcount() - iterates through documents and counts them
        .pretty() - pretty print each document, possibly over multiple lines
```

db.col.count()

```
}
> db.users.count()
4
```

Db.col.find({}).count()

```
> db.users.find({"gender":"F"}).count()
3
```

Db.col.find({})

```
> db.users.find({"gender":"F"}).limit(1)
{ "_id" : ObjectId("5f6ae2068853368b2acedbd4"), "Name" : { "firstName" : "E", "lastName" : "F" }, "phone" : 1011, "address" : "EF street, Boston, MA", "gender" : "F", "
age" : 35 }
```

Db.col.find({}).skip()

```
> db.users.find({"gender":"F"}).skip(1)
{ "_id" : ObjectId("5f6ae2068853368b2acedbd5"), "Name" : { "firstName" : "G", "lastName" : "H" }, "phone" : 1314, "address" : "GH street, Boston, MA", "gender" : "F", "
age" : 40 }
{ "_id" : ObjectId("5f6ae2068853368b2acedbd6"), "Name" : { "firstName" : "I", "lastName" : "J" }, "phone" : 1213, "address" : "IJ street, Boston, MA", "gender" : "F", "
age" : 45 }
```

Db.col.getDB()

```
> db.users.getDB()
contactManagementSystem
>
```

Db.col.insertOne()

```
> db.users.insertOne({"name": {"firstName": "K", "lastName":"L"},"phone":1111,"address":"inserted address","gender":"F"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5f704b7c8b4c697bf86cce20")
}
```

Db.col.totalSize()

```
> db.users.totalSize()
59632
```

Db.col.renameCollection()

```
> db.users.renameCollection( "UsersUpdated")
{ "ok" : 1 }
```