#### **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

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.

Then delete any one record of your choice.

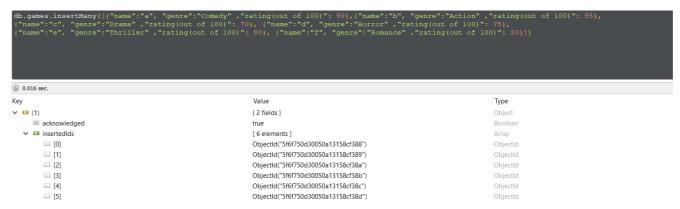
```
}
> db.users.deleteOne( { age: 25 } )
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

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

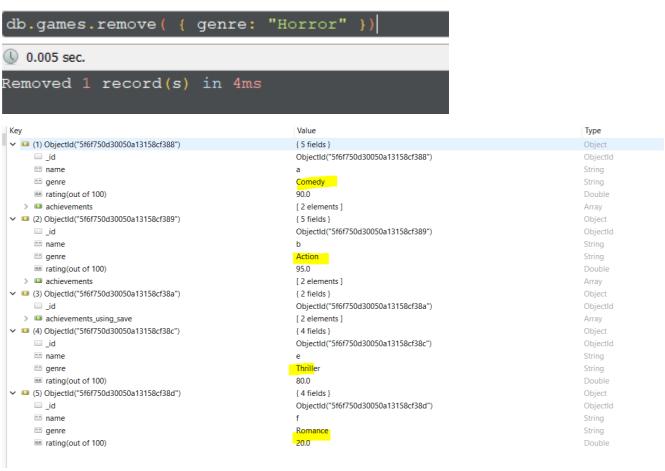
```
> db.users.update(
... {"gender" : "M"},
... {$set: { "gender" : "Male"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find()
{ "_id" : ObjectId("5f6ae2068853368b2acedbd3"), "Name" : { "firstName" : "C", "lastName" : "D" }, "pho
ne" : 5678, "address" : "CD street, 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 }
>
```

Create a collection called 'games'. We're going to put some games in it. Add 5 games to the database.

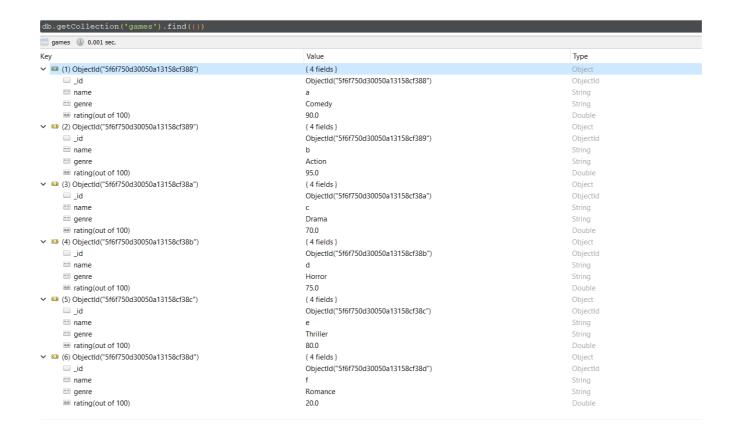
Give each document the following properties: name, genre, rating (out of 100)



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



Write a query that returns all the games.



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



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



Write a query that returns the 3 highest rated games.



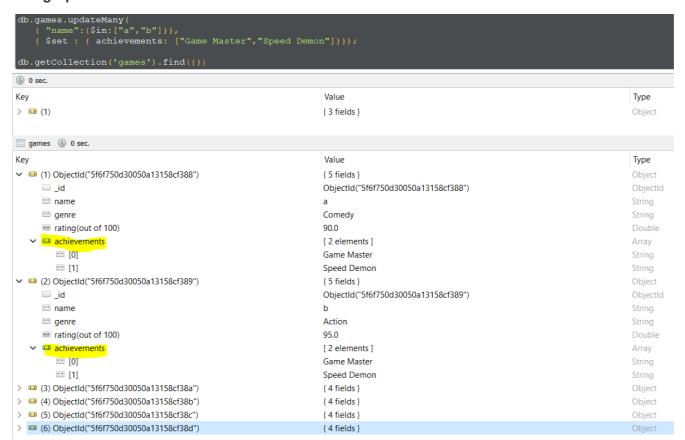
Update your two favourite games to have two achievements called 'Game Master' and 'Speed Demon', each under a single key.

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.

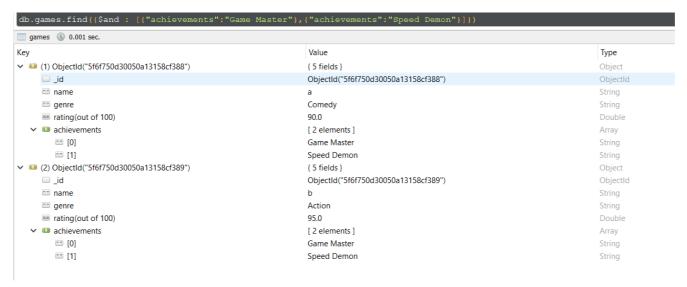
### **Using Update:**



# **Using Save:**



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



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

■ games 🕔 0.001 sec.		
Key	Value	Туре
✓ 💴 (1) ObjectId("5f6f750d30050a13158cf388")	{ 5 fields }	Object
	ObjectId("5f6f750d30050a13158cf388")	ObjectId
"" name	a	String
"" genre	Comedy	String
ma rating(out of 100)	90.0	Double
> • achievements	[ 2 elements ]	Array
✓ ☑ (2) ObjectId("5f6f750d30050a13158cf389")	{ 5 fields }	Object
	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
                                    replica set helpers
       rs.help()
       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
                                    list objects in collection foo
       db.foo.find()
       db.foo.find( { a : 1 } ) list objects in foo where a == 1
                                    result of the last line evaluated; use to fur
       DBQuery.shellBatchSize = x
                                    set default number of items to display on she
       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 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)

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.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>)
           .maximmens(\tag{\text{NP}}\)
.comment(\text{comment}\)
.tailable(\text{sisAwaitData}\)
.noCursorTimeout()
.allowPartialResults()
           .returnKey()
.showRecordId() - adds a $recordId field to each returned object
           .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.col.totalSize()

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

# Db.col.renameCollection()

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