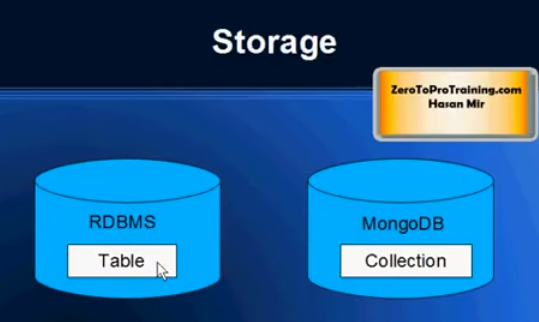
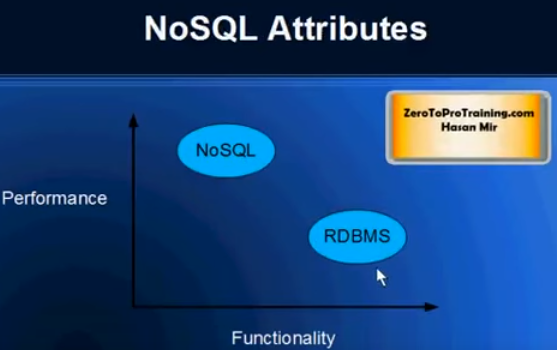
* It is **Document Oriented Database**. Think of MySQL but with JSON-like objects comprising the data mode, rather than RDBMS tables. MongoDB supports neither joins nor transactions.
* Classified as a [NoSQL](https://en.wikipedia.org/wiki/NoSQL) database. A **NoSQL** (originally referring to "non SQL" or "non relational" [[1]](https://en.wikipedia.org/wiki/NoSQL#cite_note-1)) database provides a mechanism for [storage](https://en.wikipedia.org/wiki/Computer_data_storage) and [retrieval](https://en.wikipedia.org/wiki/Data_retrieval) of data which is modeled in means other than the tabular relations used in [relational databases](https://en.wikipedia.org/wiki/Relational_database).
* It is not relational database
* It has **json** like data
* Dynamic schema.
* MongoDb supports rich query to fetch data from the database.
* GridFS specification of MongoDb supports storage of files.
* A MongoDB database stores its data in **collections** instead of tables, which are the rough equivalent of RDBMS tables.



* The data in collection is stored as document. This is equivalent to row in table of RDBMS.
* They are **horizontally scalable** means if more data increase the number of computers. So to double the performance double the number of computers.
* No Joins supported
* No complex transactions supported
* No constraints supported
* NoSql DB have a query language of their own.
* NoSql Db are better in performance in comparison to RDBMS.



* Every document should have id.
* In relational db all records going in one table must have same number of fields but in mongo db documents in one collection can have diff fields.
* RDBMS has sql, NoSql has document oriented query language.
* Indexing is supported, any field in a document can be supported.
* Master slave replication is supported.
* Javascript can be used in queries.
* Created by 10gen in 2007. Recent stable version is 2.4

Below given table shows the relationship of RDBMS terminology with MongoDB

|  |  |
| --- | --- |
| **RDBMS** | **MongoDB** |
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key \_id provided by mongodb itself) |

To install mongodb download msi from <https://www.mongodb.org/downloads#production>

Execute the downloaded msi file

Go to mongodb directory “C:\Program Files\MongoDB\Server\3.2\bin” and execute the command : “mongod.exe --dbpath E:\my-notes\mongodb”. This will start the mongodb server.

Next execute the mongodb client by running the mongo.exe file.

To list the valid commands for client run command “db.help()”

|  |  |
| --- | --- |
| **Command** | **Usage** |
| mongod.exe --dbpath E:\my-notes\mongodb | To start mongodb server |
| mongo.exe | To run the mongodb client |
| use DATABASE\_NAME | To create a db |
| Db | To check my current db |
| show dbs | If you want to check your databases list |
| db.movie.insert({"name":"tutorials point"}) | To insert a document. In mongodb you don't need to create collection. MongoDB creates collection automatically, when you insert some document. |
| db.dropDatabase() | This will delete the selected database. If you have not selected any database, then it will delete default 'test' database |
| db.createCollection(name, options)  eg1: db.createCollection("mycollection")  eg2: db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } ) | to create collection |
| show collections | To list created collection |
| db.COLLECTION\_NAME.drop()  eg: db.mycollection.drop() | To drop a collection |
| db.COLLECTION\_NAME.insert(document)  or  save | To insert document into collection |
| db.COLLECTION\_NAME.find() | find()method will display all the documents in a non structured way. |
| db.COLLECTION\_NAME.find().pretty() | To display the results in a formatted way |
| findOne() | returns only one document. |
| db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA}) | save() method replaces the existing document with the new document passed in save() method |
| db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA) | To remove document from collection |
| db.mycol.remove() | To truncate |
| db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1) | If there are multiple records and you want to delete only first record, then set justOne parameter in remove() method |
| db.COLLECTION\_NAME.find().limit(NUMBER) | To limit the records in MongoDB |
| db.COLLECTION\_NAME.find().sort({KEY:1}) | To sort documents in MongoDB, you need to use sort() method. sort() method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order. |

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS Equivalent** |
| Equality | {<key>:<value>} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

AND in MongoDB

Syntax

In the **find()** method if you pass multiple keys by separating them by ',' then MongoDB treats it **AND** condition. Basic syntax of **AND** is shown below −

>db.mycol.find({key1:value1, key2:value2}).pretty()

## OR in MongoDB

### Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Basic syntax of **OR** is shown below −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

**Update**

db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

By default mongodb will update only single document, to update multiple you need to set a paramter 'multi' to true.

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

# Commands

Use <db\_name>

set current db, crete if not exists

1. **Insert**

db.movies.insertOne({"title":"force", year:2016})

1. **Find**

db.movies.find({})

db.movies.find({}).pretty()

db.movies.find({title:"Jaws"})

//to show specific fields from the response

db.movies.find({title:"Jaws"},{title:true,\_id:false})

db.movies.findOne({})

db.movies.find({"year":{$gt:1985}})

db.movies.find({"year":{$lt:1985}})

//to check if a field exisis

db.movies.find({title:{$exists:true}})

db.movies.find({name:{$exists:true}})

//to check if field is of type string

db.movies.find({title:{$type:2}})

db.movies.find({title:{$regex:"^f"}})

//To combine 2 conditions using or

db.movies.find({$or:[{title:"Jaws"},{title:"Jurassic park"}]})

//to match multiple values of a field with value as array

db.movies.find({stars:{$all:["abc","acb"]}})

//to match multiple values of a field with value from array

db.movies.find({title:{$in:["Jaws","Jurassic park"]}})

1. **sort**

db.movies.find().sort({title:1})

//to skip the first 2 documents

db.movies.find().skip(2)

1. **count**

db.movies.count({})

db.movies.count({year:{$gt:1985}})

1. **Update**

//To update a document use the update, 1st arg is analogous to where clause & 2nd arg is the document, so the old one will be replaced with the new one

db.movies.update({title:"force"},{title:"force2", year:2017})

//To update specific field of a doc use $set

db.movies.update({title:"force"},{$set:{year:2013}})

//To increase value of a field having numeric value

db.movies.update({title:"force"},{$inc:{year:1}})

//To remove field from a document

db.movies.update({title:"force"},{$unset:{year:1}})

//To manipulate array elements

//To modify specific element in a array

db.movies.update({title:"force"},{$set:{"stars.1":"John"}})

//To add elements into an array use $push

db.movies.update({title:"force"},{$push:{heroes:"Abraham"}})

//to remove the right most element from the array use $pop

db.movies.update({title:"force"},{$pop:{heroes:1}})

//to remove the left most element from the array use $pop

db.movies.update({title:"force"},{$pop:{heroes:-1}})

//To push multiple elements use $pushAll

db.movies.update({title:"force"},{$pushAll:{heroes:["John","Abraham","Rambo"]}})

//To remove specific element from array

db.movies.update({title:"force"},{$pull:{heroes:"Abraham"}})

db.movies.update({title:"force"},{$pullAll:{heroes:["Abraham","John"]}})

//To add an element into array only if its not present

db.movies.update({title:"force"},{$addToSet:{heroes:"Abraham"}})

//Upsert: update if present or insert

db.movies.update({title:"Rock on"},{$set:{year:2018}},{upsert:true})

//If the query specifies incomplete info then insert will not happen

db.movies.update({year:{$gt:10}},{$set:{year:2018}},{upsert:true})

//TO update multiple docs set property {multi:true}

db.movies.update({year:{$gt:2017}},{$set:{year:2020}},{multi:true})

1. **Remove document**

//To delete a document use remove

db.movies.remove({year:2020})

//to remove all documents from collection

db.movies.remove({})

1. **drop collection**

db.movies.drop()

db.movies.find()

1. **Aggregation**
   * **project:**

Passes along the documents with only the specified fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

db.movies.aggregate([{$project:{title:1,"\_id":0}}])

db.movies.aggregate([{$project:{title:1,"star":"$title"}}])

* **match:**

Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

db.movies.aggregate([{$match:{title:"Force"}}])

db.movies.aggregate([{$match:{title:"Force"}},{$project:{title:1,"\_id":0}}])

db.movies.aggregate({$limit:2})

* **unwind:**

Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.

db.movies.aggregate([{$unwind:"$heroes"}])

db.movies.aggregate([{$unwind:{path:"$heroes",preserveNullAndEmptyArrays: true}}])

* **group**

db.movies.aggregate( [ { $group : { \_id : "$year" } } ] )

db.movies.aggregate( [ { $group : { \_id : "$year", name:{$push:"$title"} } } ] )

* **lookup:**

Performs a left outer join to an unsharded collection in the same database to filter in documents from the “joined” collection for processing. The $lookup stage does an equality match between a field from the input documents with a field from the documents of the “joined” collection.

1. **indexes:**

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

Indexes are special data structures that store a small portion of the collection’s data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index.

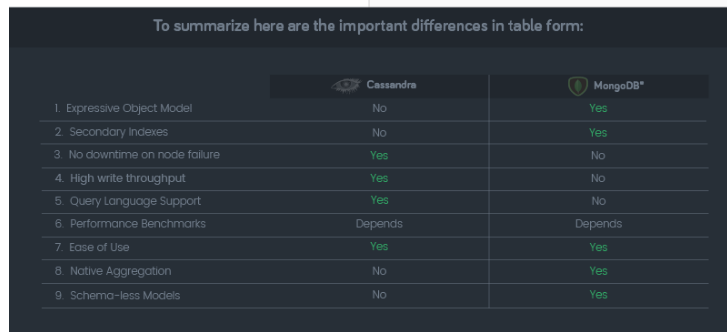
MongoDB creates a unique index on the \_id field during the creation of a collection. The \_id index prevents clients from inserting two documents with the same value for the \_id field.

* **Index on a Single Field**

db.movies.createIndex({title:1})

db.movies.getIndexes()

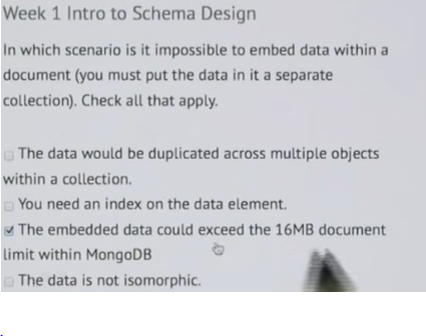
MongoDb vs Cassandra



RocksDB

CockroachDB

# Mongo 16mb problem



# Refernces

<https://www.youtube.com/watch?v=liQzIsFnCr0>

<https://www.youtube.com/watch?v=qI_g07C_Q5I>

mongo vs cassendra: https://scalegrid.io/blog/infographic-comparing-cassandra-vs-mongodb/