THE UNIVERSITY OF WESTERN AUSTRALIA

Topic 11: MongoDB

CITS3403 Agile Web Development

Getting MEAN with Mongo, Express, Angular and Node, Chapter 5, and www.tutorialspoint.com/mongodb

Semester 1, 2018

Mongo DB

- MongoDB (from humongous) is a free and open-source cross-platform document-oriented database.
- Classified as a NoSQL database, MongoDB avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas.
- As of July 2015, MongoDB is the fourth most popular type of database management system, and the most popular for document stores.

Document databases

Document databases don’t have tables or schemas.
Instead, they consist of *Collections* of *Documents.*
Each document in a collection may have different fields.
The fields of a document can be another document (a sub-document), but two documents cannot share a subdocument. i.e it is a tree
In Mongo, each document is represented as a JSON object.

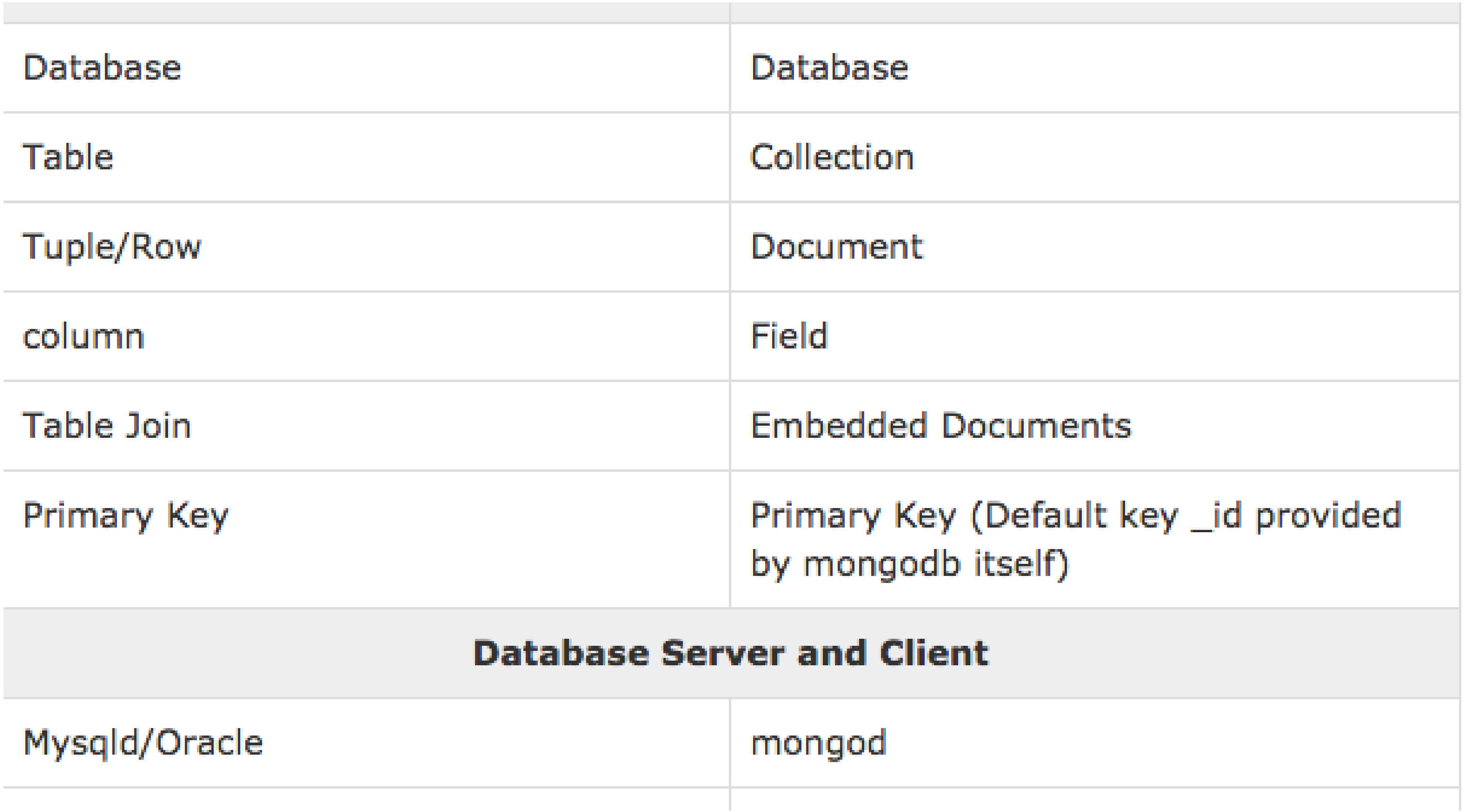
Databases, Collections and Documents

- Database - Database is a physical container for collections. Each database gets its own set of files on the file system.

- Collection - Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema.

- Document - A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure.

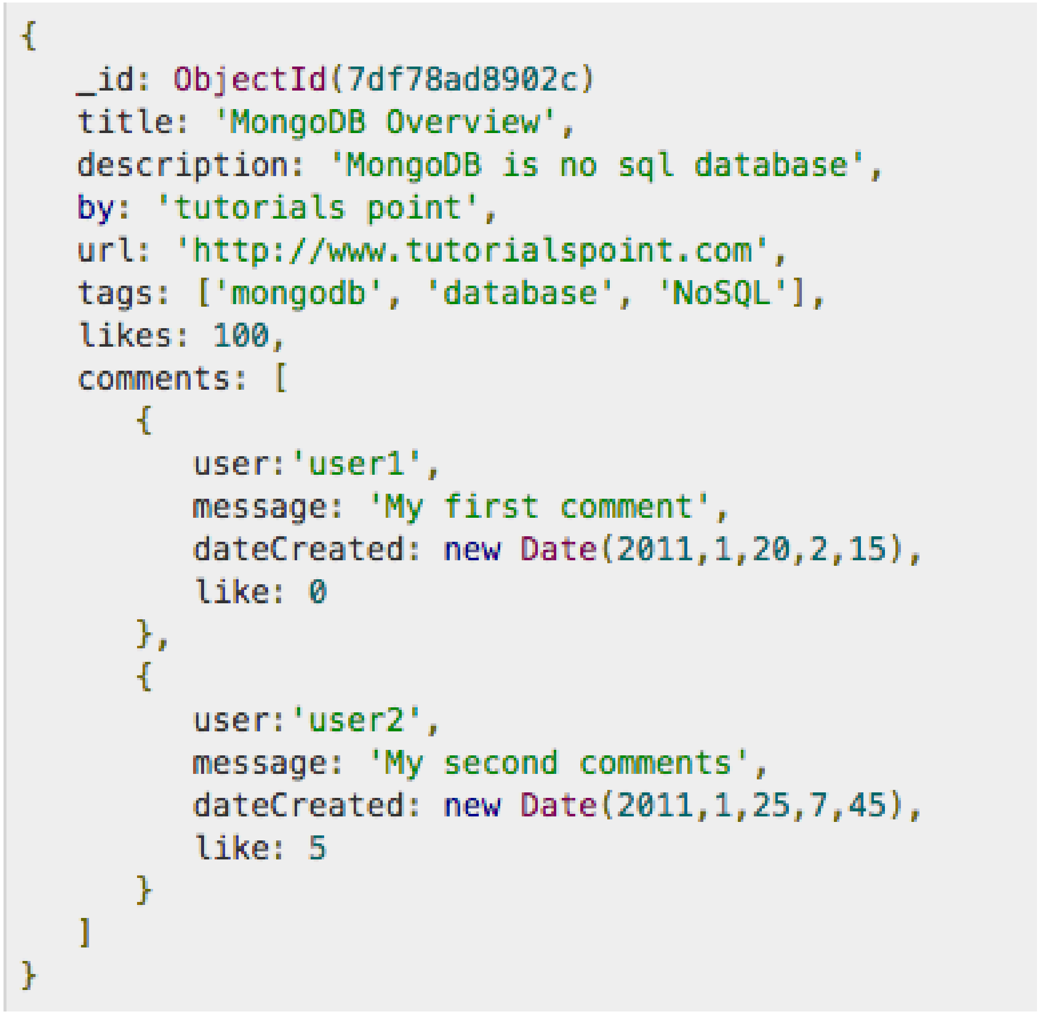
Mongo vs RDBMS



Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod

Sample document

- Below is a sample document. Every document has an id.
- The document is a javascript object.
- Any relational database has a number of tables and their relationships.
- In MongoDB there is no concept of relationship



```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

“Advantages” of Mongo

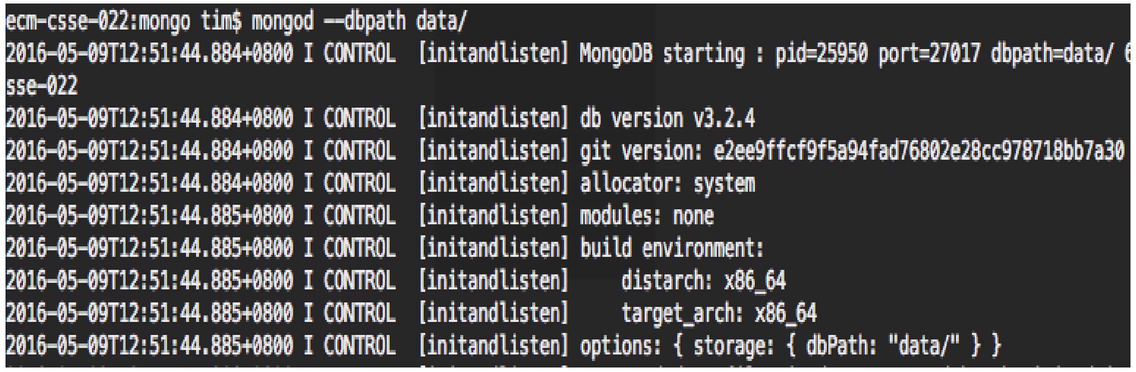
- Mongo is schema-less: different documents in a collection can have different fields.
- Documents are objects: saves conversion logic.
- No complex joins. No joins at all.
- Deep query ability: document based query language.
- Tunable and scalable.

… but

- data should be tree like.
- joins need to be done outside the database.

Initialising Mongo

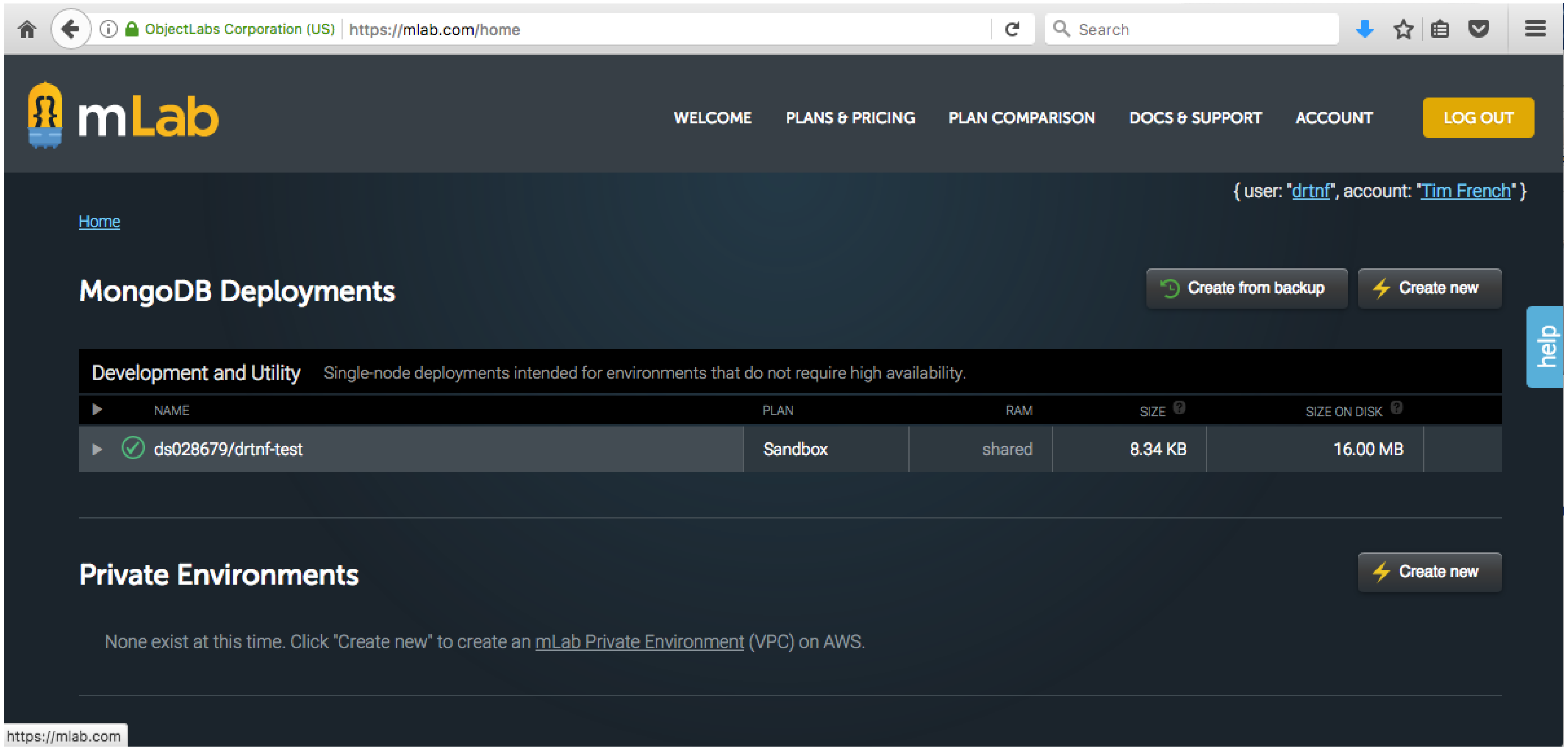
- To start mongo, you need a directory to hold the data, and a port to serve the data.
- Create a sub-directory `data` and then start the mongo daemon: `mongod –dbpath data`
- this creates a set of files in the data directory, and starts a process listening on port 27017
- Other processes can now create and access databases through this port
- e.g. mongo shell



```
ecm-csse-022:mongo tim$ mongod --dbpath data/
2016-05-09T12:51:44.884+0800 I CONTROL  [initandlisten] MongoDB starting : pid=25950 port=27017 dbpath=data/ 6
sse-022
2016-05-09T12:51:44.884+0800 I CONTROL  [initandlisten] db version v3.2.4
2016-05-09T12:51:44.884+0800 I CONTROL  [initandlisten] git version: e2ee9ffcf9f5a94fad76802e28cc978718bb7a30
2016-05-09T12:51:44.884+0800 I CONTROL  [initandlisten] allocator: system
2016-05-09T12:51:44.885+0800 I CONTROL  [initandlisten] modules: none
2016-05-09T12:51:44.885+0800 I CONTROL  [initandlisten] build environment:
2016-05-09T12:51:44.885+0800 I CONTROL  [initandlisten]     distarch: x86_64
2016-05-09T12:51:44.885+0800 I CONTROL  [initandlisten]     target_arch: x86_64
2016-05-09T12:51:44.885+0800 I CONTROL  [initandlisten] options: { storage: { dbPath: "data/" } }
```

Database as a service

- We will be using mLab as a free database as a service. This provides minimal instances hosted in the cloud.



{ user: "drtnf", account: "Tim French" }

Home

MongoDB Deployments

Development and Utility Single-node deployments intended for environments that do not require high availability.

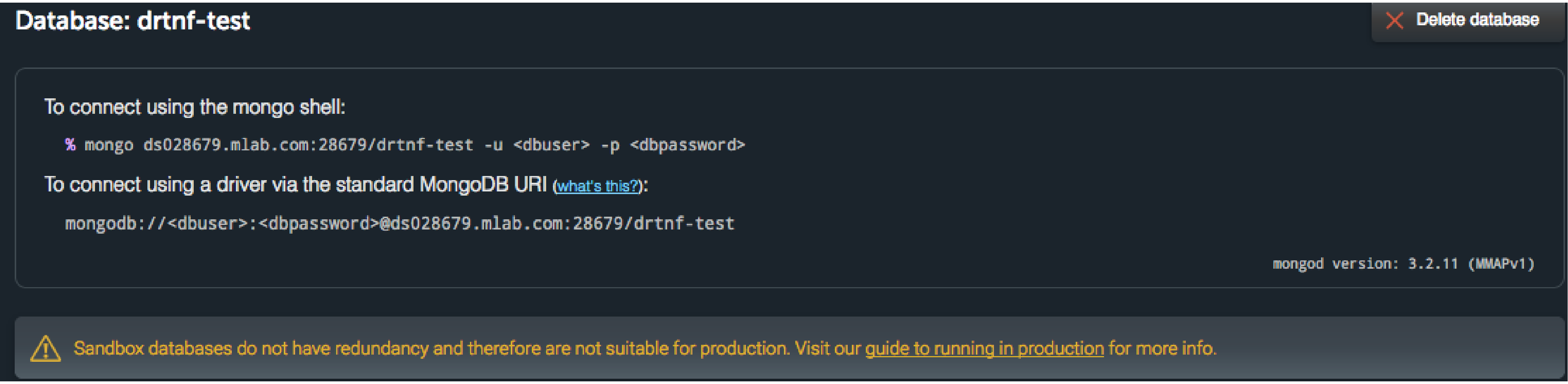
NAME	PLAN	RAM	SIZE	SIZE ON DISK
ds028679/drtnf-test	Sandbox	shared	8.34 KB	16.00 MB

Private Environments

None exist at this time. Click "Create new" to create an mLab Private Environment (VPC) on AWS.

Using mLab

- You can remotely access mLab instances…:



Database: drtnf-test

Delete database

To connect using the mongo shell:

```
% mongo ds028679.mlab.com:28679/drtnf-test -u <dbuser> -p <dbpassword>
```

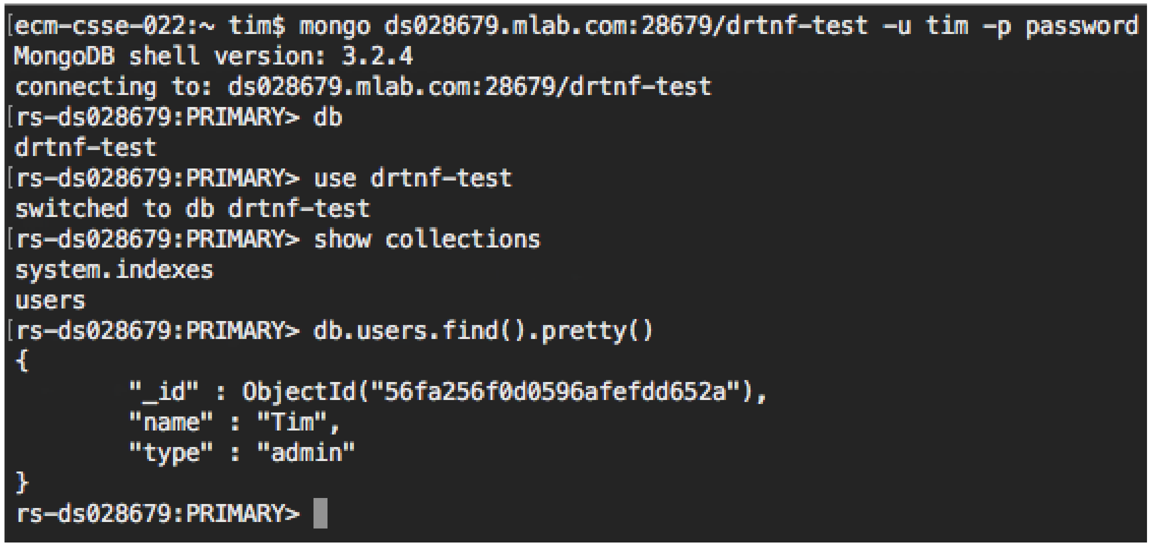
To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds028679.mlab.com:28679/drtnf-test
```

mongod version: 3.2.11 (MMAPv1)

Sandbox databases do not have redundancy and therefore are not suitable for production. Visit our guide to running in production for more info.

- …and access them through a shell environment.



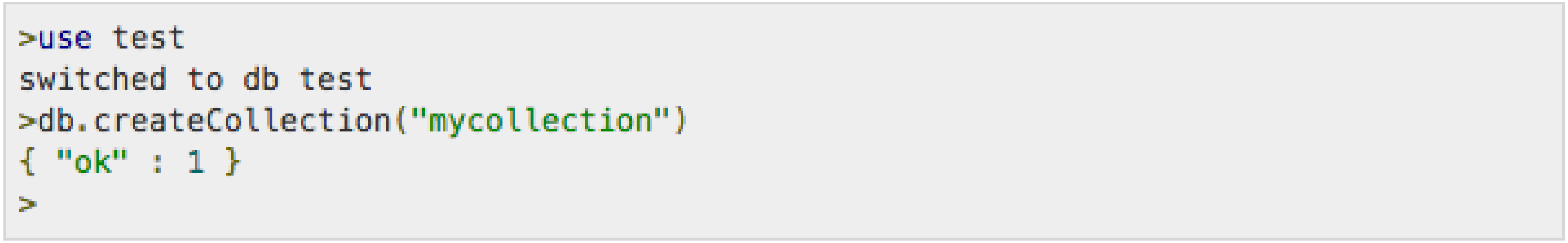
```
ecm-csse-022:~ tim$ mongo ds028679.mlab.com:28679/drtnf-test -u tim -p password
MongoDB shell version: 3.2.4
connecting to: ds028679.mlab.com:28679/drtnf-test
rs-ds028679:PRIMARY> db
drtnf-test
rs-ds028679:PRIMARY> use drtnf-test
switched to db drtnf-test
rs-ds028679:PRIMARY> show collections
system.indexes
users
rs-ds028679:PRIMARY> db.users.find().pretty()
{
        "_id" : ObjectId("56fa256f0d0596afefdd652a"),
        "name" : "Tim",
        "type" : "admin"
}
rs-ds028679:PRIMARY> 
```

Getting started – Mongo shell

- Once the daemon is running, you can access the database using the mongo shell.
- type `mongo` at the command line
- `use <db>` will allow you to access or create a database
- the command `db`, shows the current database; `show dbs` shows all the databases.
- To insert you use `db.<collection>.insert(<JSObj>)`
- To remove a datbase: `db.dropDatabase()`
- To stop the daemon: `use admin,` then `db.shutdownServer()`

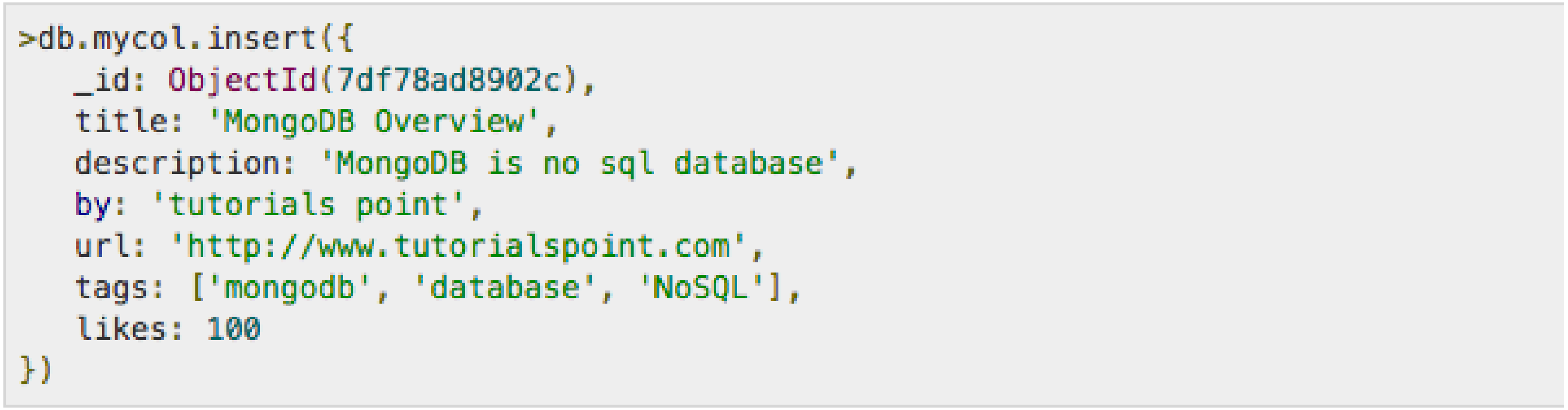
Adding data

- To create a new collection:
  `db.createCollection(<name>, <options>)`



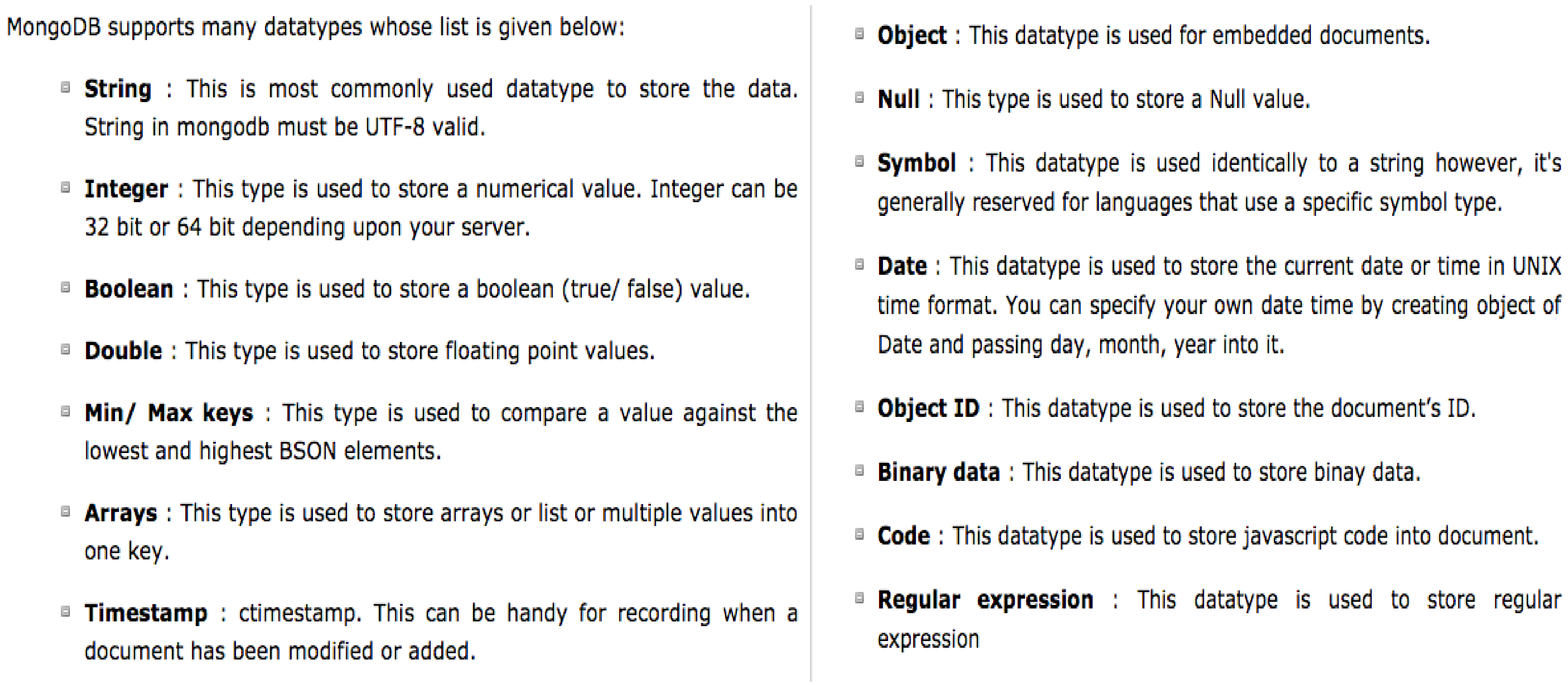
```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

- Remove a collection: `db.<cName>.drop()`
- To insert a document: `db.<cName>.insert(<doc>)`



```
>db.mycol.insert({
   _id: ObjectId(7df78ad8902c),
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
})
```

Mongo data types

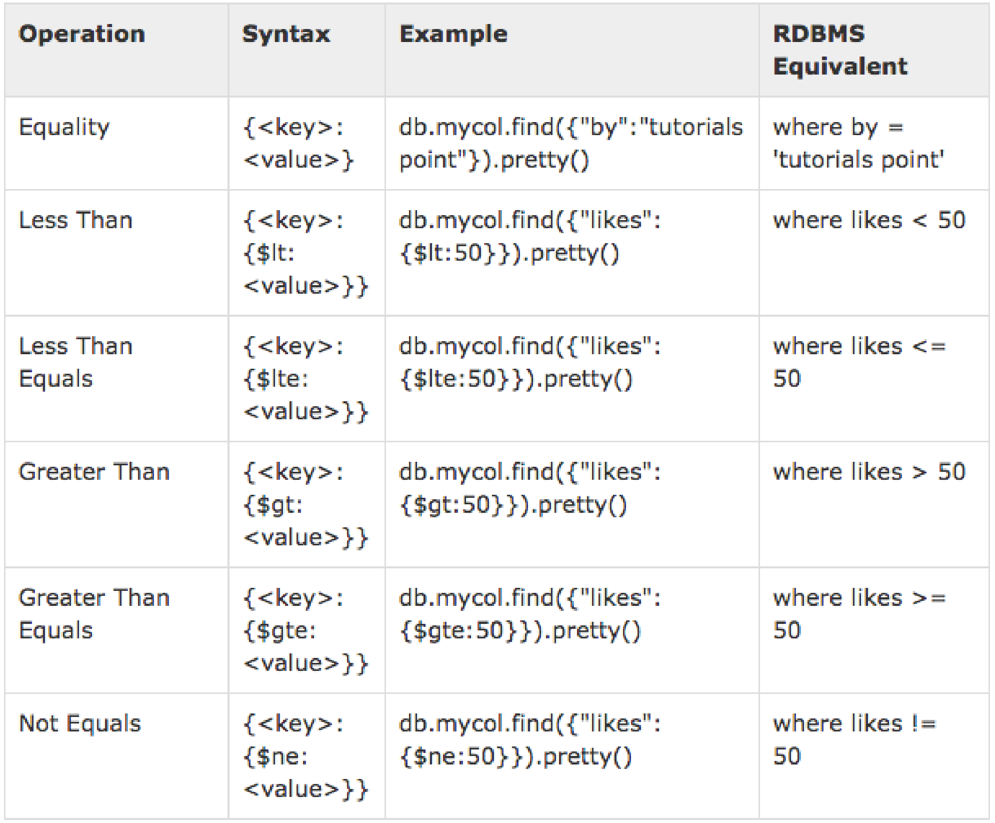


MongoDB supports many datatypes whose list is given below:

- **String** : This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.
- **Integer** : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** : This type is used to store a boolean (true/ false) value.
- **Double** : This type is used to store floating point values.
- **Min/ Max keys** : This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** : This type is used to store arrays or list or multiple values into one key.
- **Timestamp** : ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** : This datatype is used for embedded documents.
- **Null** : This type is used to store a Null value.
- **Symbol** : This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type.
- **Date** : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** : This datatype is used to store the document's ID.
- **Binary data** : This datatype is used to store binay data.
- **Code** : This datatype is used to store javascript code into document.
- **Regular expression** : This datatype is used to store regular expression

Querying data

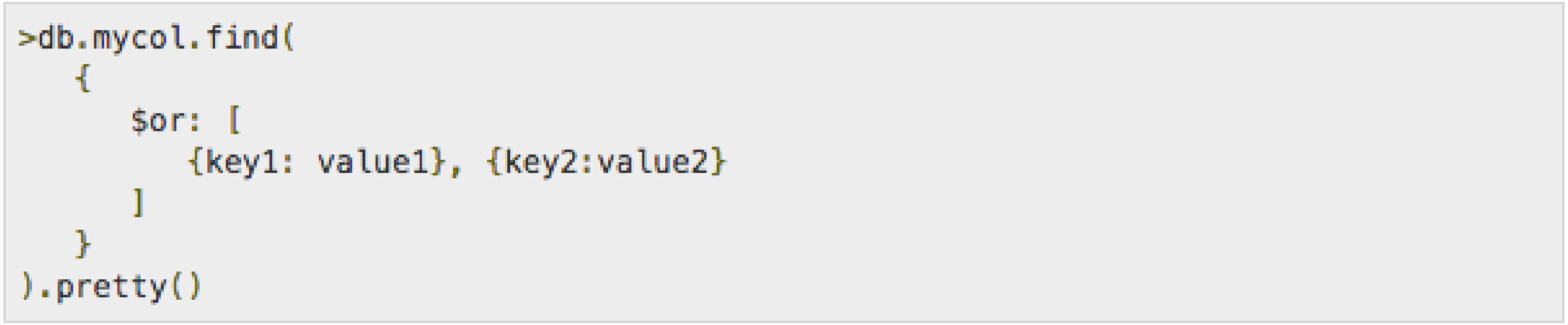
- Use `db.<cName>.find()` to return all documents in a collection.
- Use `db.<cName>.find().pretty()` for nice formatting.
  - To find particular documents, you can test fields.
  - A list of constraints will return their intersection (AND)



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

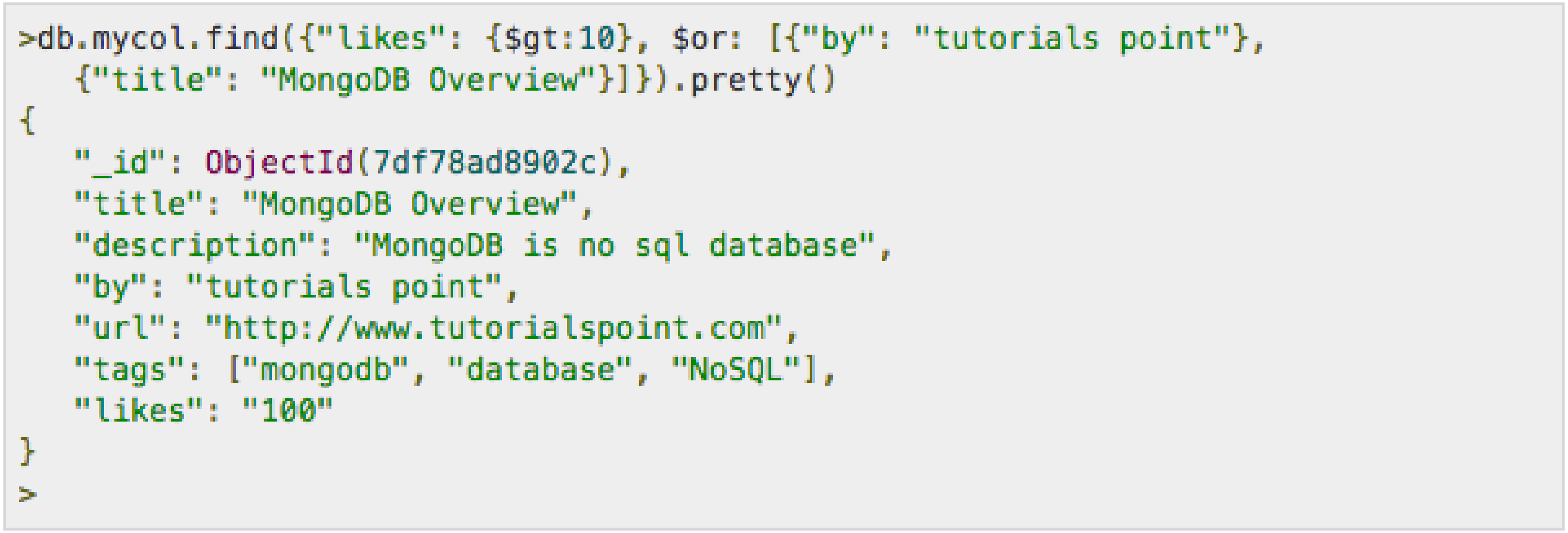
OR in mongo

- TO find the union of two constraints use $or:



```
>db.mycol.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

- AND and OR can be nested.

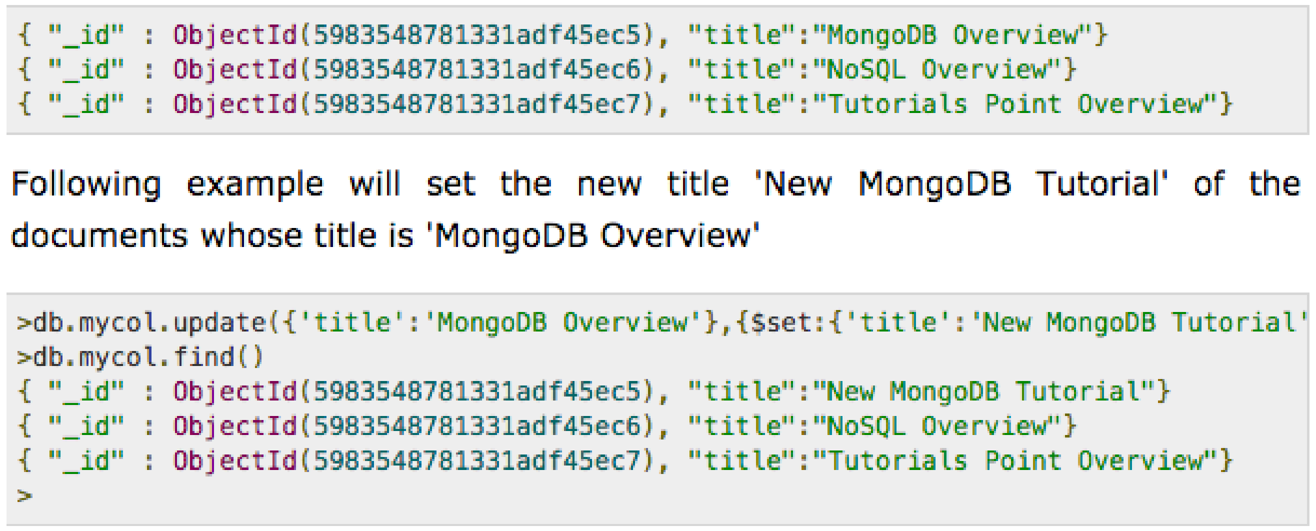


```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},
   {"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

Update a document

- Databases typically need to support the CRUD operations (create, read, update, delete)
- For updates we use `db.<cName>.update(criteria,data)`

- You can also use the `save` to update a document, given its document id.



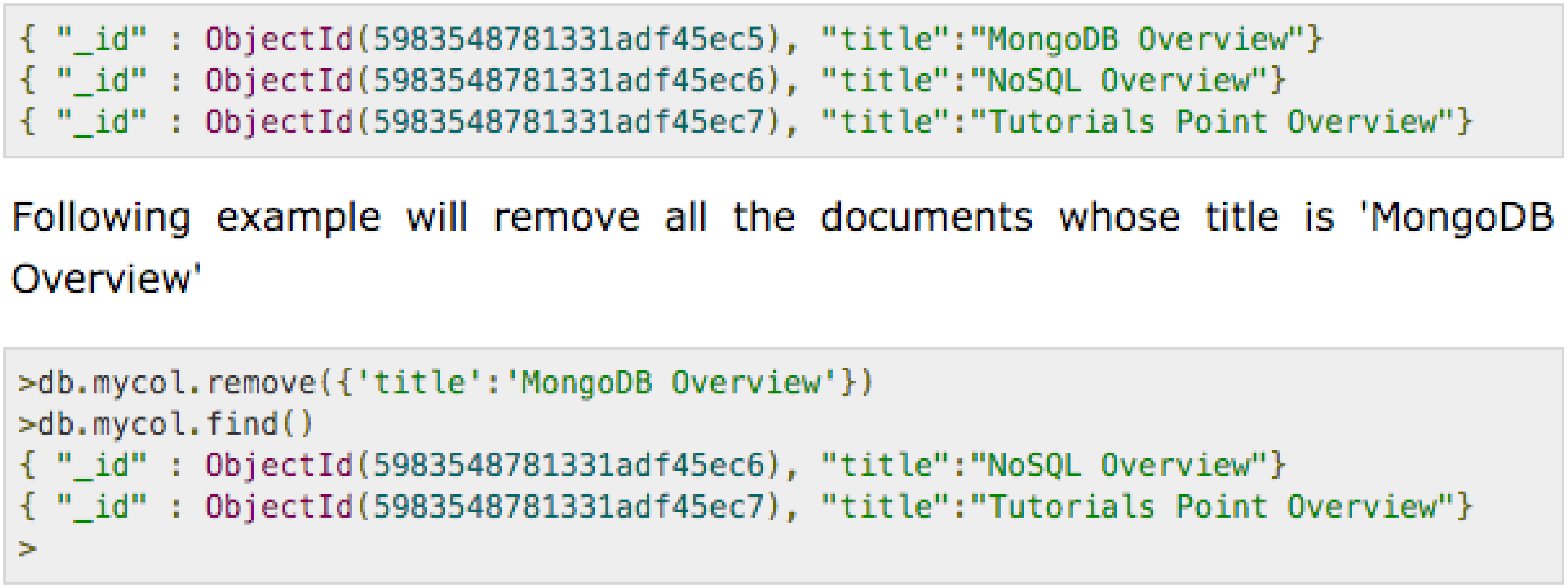
```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

Delete

- To delete documents from a collection, you use: `db.<cName>.remove()`
- You can provide a criteria for the documents to remove, in the same way they are specified in the find method.



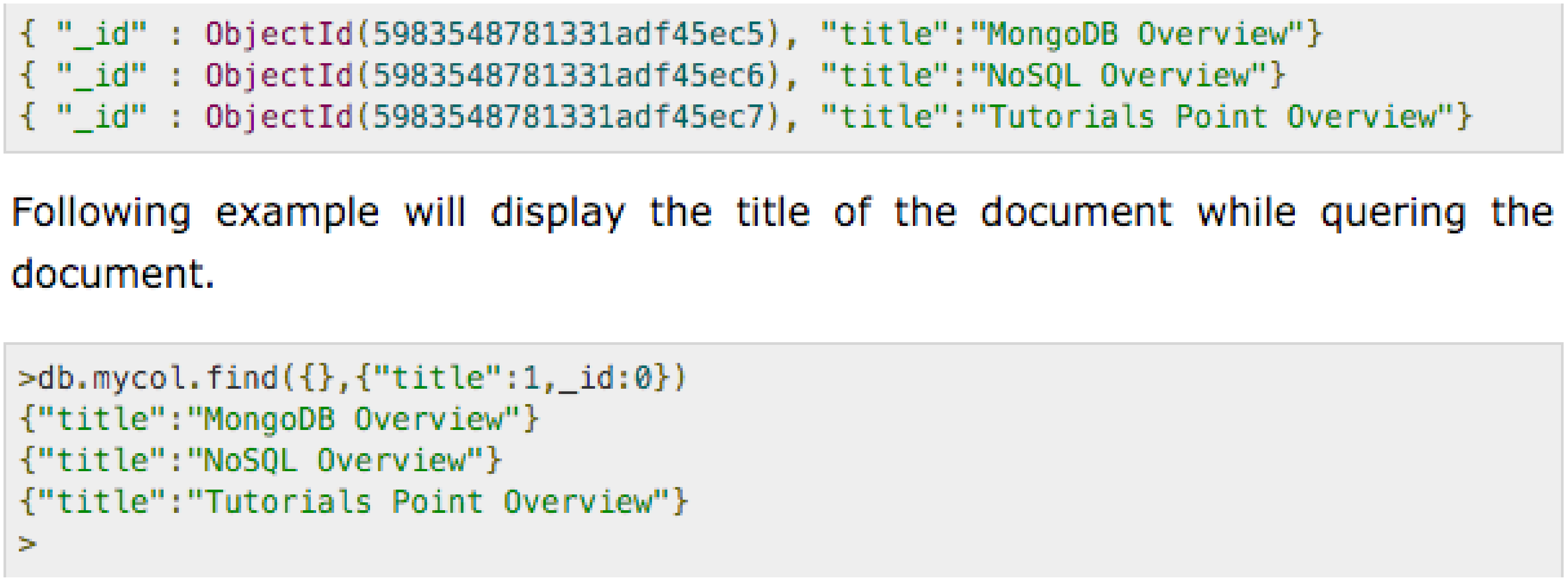
```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

Projection

- To project to a subset of fields, you can use an optional parameter in the `find` method.
- When two objects are given as a parameter, the first is the query constraint, the second is the fields you want returned (1 to return, 0 to not)



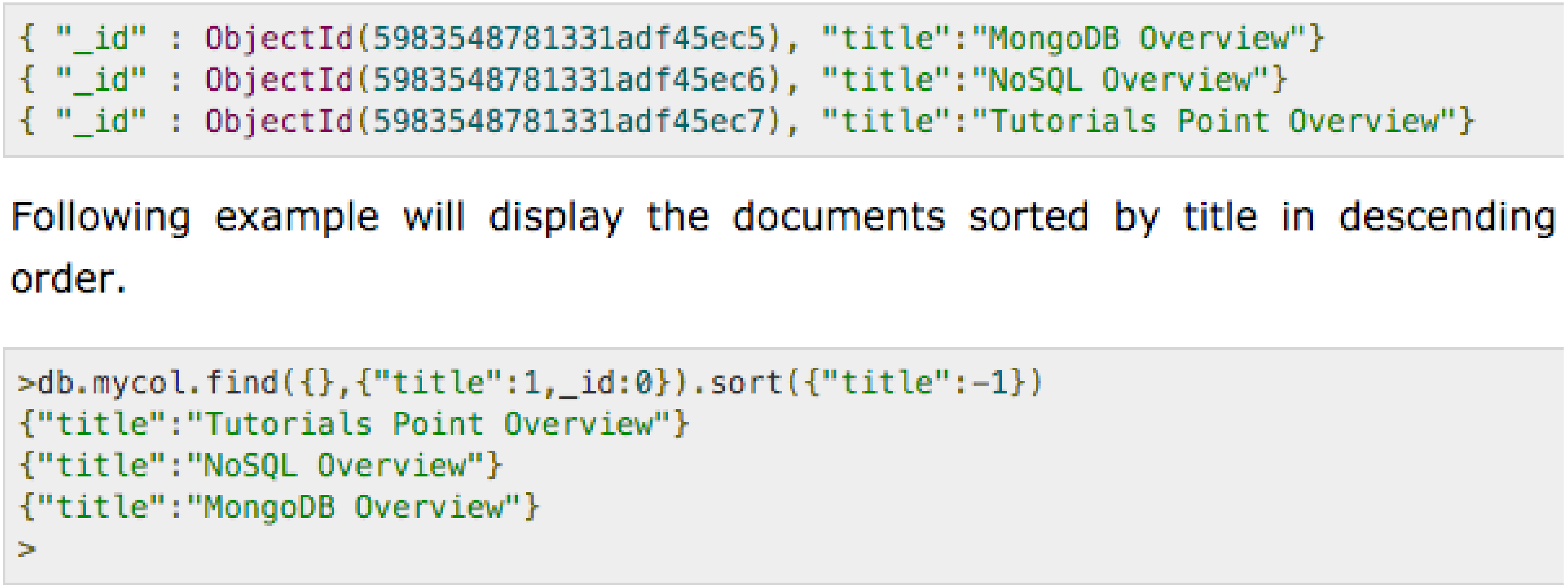
```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the title of the document while quering the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

Limiting and Sorting

- `limit(n)` and `skip(n)` can be used in conjunction with `find` to return (or skip) the first `n` results.
- Similarly `sort({key:1})` acts on the object returned by find to `sort` according to the given key (1 for ascending, -1 for descending).



```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the documents sorted by title in descending order.

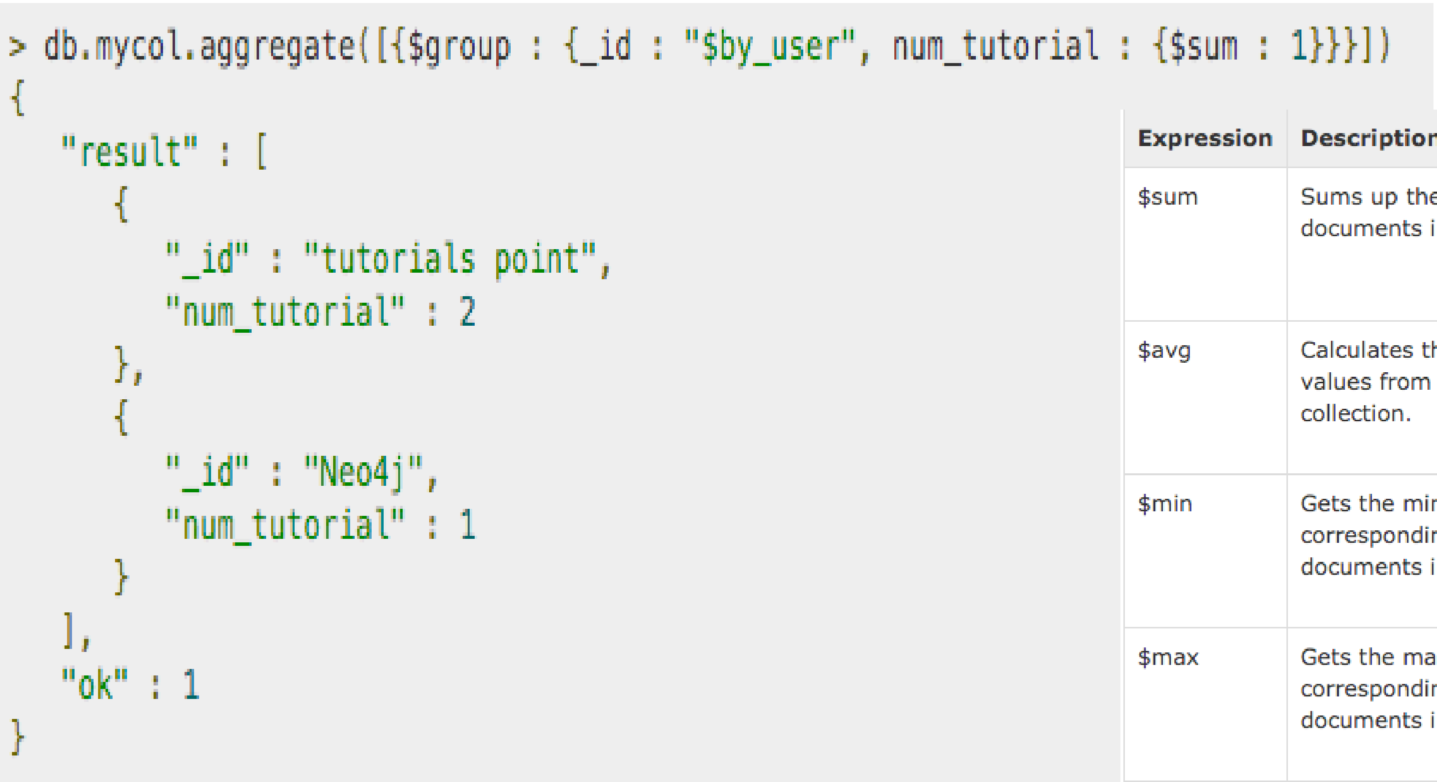
```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Indexing

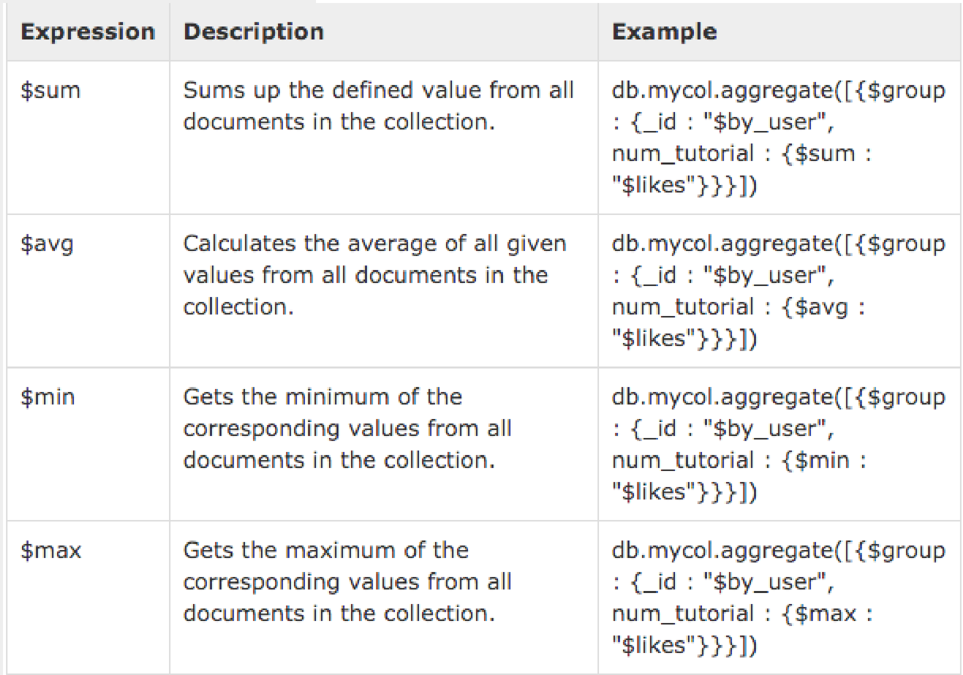
- In order for Mongo to efficiently search a collection, it needs the fields to be indexed. Otherwise it has to iterate over the entire collection.
- Use `db.<cName>.createIndex({key:1})` to create an index for fast searching (-1 for reverse order).
- Constructing the index can be time-consuming so there are options to optimise how the index is built and used.

Aggregation

- Aggregation operation process records and return computed results.
- `db.<cName>.aggregate([{$group:{…}}])`
- There are many options:



```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "tutorials point",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
```



Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])