MongoDB

Motivation

- **MongoDB is a powerful, flexible, and scalable general-purpose database**
- **MongoDB is a document-oriented database, and not relational**
  - Document oriented approach makes it possible to represent complex hierarchical relationships with a single record
- **No predefined schemas**
  - Document's keys and values are not of fixed types or sizes
  - Adding or removing fields as needed becomes easier

Scaling

Scaling a database comes down to the following two choices

- Scaling up (getting a bigger machine)
- Scaling out (partitioning data across more machines).

MongoDB was designed to scale out

- MongoDB automatically takes care of balancing data and load across a cluster
- Redistributing documents automatically
- Routing reads and writes to the correct machines

Scaling, Indexing

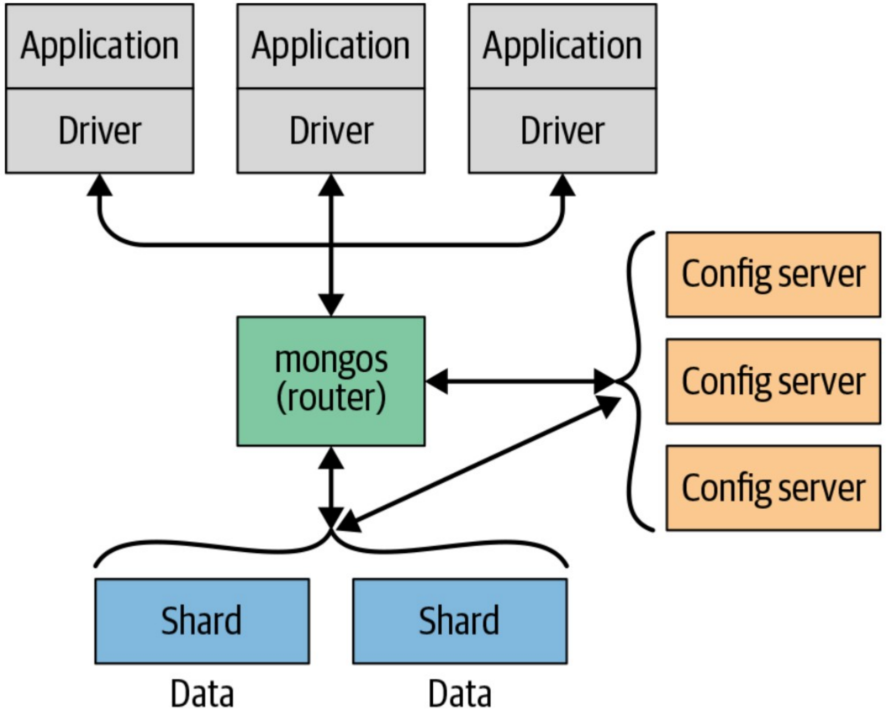
Scaling out MongoDB using sharding across multiple servers

The topology of a MongoDB cluster, or whether there is in fact a cluster rather than a single node at the other end of a database connection, is transparent to the application.

Indexing

MongoDB supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well.

Secondary indexes on hierarchical structures such as nested documents and arrays are also supported



Features

Aggregation

MongoDB provides an aggregation framework based on the concept of data processing pipelines.

Special collection and index types

MongoDB supports time-to-live (TTL) collections for data that should expire at a certain time, such as sessions and fixed-size (capped) collections, for holding recent data, such as logs.

MongoDB also supports partial indexes limited to only those documents matching a criteria filter in order to increase efficiency and reduce the amount of storage space required.

File storage

MongoDB supports an easy-to-use protocol for storing large files and file metadata.

Complex joins are not supported

MongoDB – Basic Concepts

Basic Concepts - 1

- **A document is the basic unit of data for MongoDB and is roughly equivalent to a row in a relational database management system (but much more expressive).**

  Document: An ordered set of keys with associated values.

  eg. {"greeting1" : "Hello, world!", "greeting2" : "Hello, MongoDB!"}

- **Every document has a special key, "_id", that is unique within a collection.**
- **A collection can be thought of as a table with a dynamic schema.**

  {"greeting" : "Hello, world!", "views": 3}

  {"signoff": "Good night, and good luck"}

- **MongoDB is type-sensitive and case-sensitive.**

  {"count" : 5}, {"count" : "5"}, {"Count" : "5"} are different

Basic Concepts: Documents

Keeping different kinds of documents in the same collection can be a nightmare for developers and admins

Documents of same schema together in the same collection allows for data locality

Documents of a single type into the same collection, helps efficient indexing

Documents can be embedded- otherwise called recursive specification

The flip side- there can be more data repetition with MongoDB

A single instance of MongoDB can host multiple independent databases, each of which contains its own collections.

Basic Concepts: Collections

Naming

A collection is identified by its name.

You should not create any collections with names that start with system., a prefix reserved for internal collections.

User-created collections should not contain the reserved character $ in their names

Subcollections

Organize collections is to use namespaced subcollections separated by the . character.

Basic Concepts: Databases

MongoDB groups collections into databases.

A single instance of MongoDB can host several databases, each grouping together zero or more collections.

A good rule of thumb is to store all data for a single application in the same database.

Databases are identified by name- case insensitive

admin, local, config are some of reserved database names

Database.collection is called a namespace, eg., NewOffice.Employees, cms.blog.posts

Installing Mongodb

Follow the instruction on the webpage

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

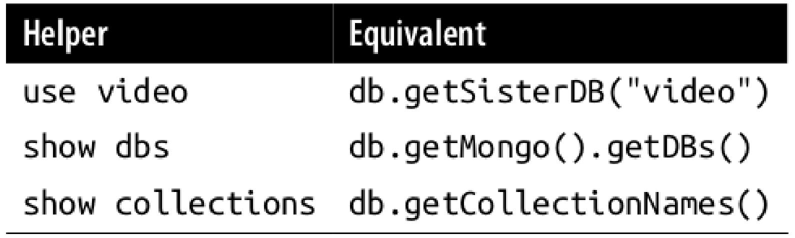
Installing sample databases

https://restheart.org/docs/mongodb-rest/sample-data

mongosh commands

```
show databases; use sample_analytics;
show collections; db.<collection>.find()
```

Table 2-1. JavaScript equivalents to shell helpers



Helper	Equivalent
`use video`	`db.getSisterDB("video")`
`show dbs`	`db.getMongo().getDBs()`
`show collections`	`db.getCollectionNames()`

Basic Operations with the Shell (CRUD)

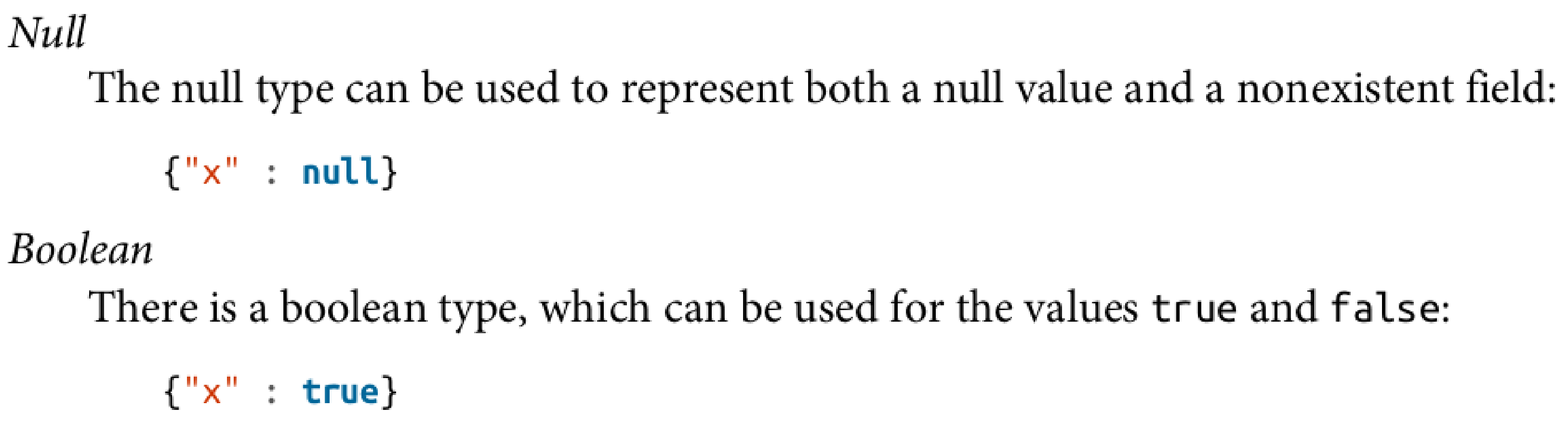
Create, Read, Update, Delete (CRUD)

```
mydoc={“field1”: “value1”, “field2”: “value2”, “field3”: “value3”}
mydocs={[{“field1”: “value1”}, {“field2”: “value2”}, {“field3”: “value3”}]}
db.myCollection.insertOne(mydoc)  #insert one doc into the collection
db.myCollection.insertMany(mydocs)   #insert multiple docs into the collection
db.myCollection.findOne()            # show the first document
db.myCollection.find().pretty()      # for pretty printing
db.myCollection.updateOne({field1: “value1”}, {$set : {field2: []}}) #unset field2
db.movies.deleteOne({field1: “value1”})
```

Datatypes

Documents in MongoDB can be thought of as “JSON-like”

MongoDB adds support for a number of additional data types while keeping JSON’s essential key/value–pair nature



Null

The null type can be used to represent both a null value and a nonexistent field:

```
{"x" : null}
```

Boolean

There is a boolean type, which can be used for the values `true` and `false`:

```
{"x" : true}
```

Datatypes

Number

The shell defaults to using 64-bit floating-point numbers. Thus, these numbers both look "normal" in the shell:

```
{"x" : 3.14}

{"x" : 3}
```

For integers, use the `NumberInt` or `NumberLong` classes, which represent 4-byte or 8-byte signed integers, respectively.

```
{"x" : NumberInt("3")}
{"x" : NumberLong("3")}
```

String

Any string of UTF-8 characters can be represented using the string type:

```
{"x" : "foobar"}
```

Datatypes

Date

MongoDB stores dates as 64-bit integers representing milliseconds since the Unix epoch (January 1, 1970). The time zone is not stored:

```
{"x" : new Date()}
```

Regular expression

Queries can use regular expressions using JavaScript's regular expression syntax:

```
{"x" : /foobar/i}
```

Array

Sets or lists of values can be represented as arrays:

```
{"x" : ["a", "b", "c"]}
```

Datatypes

Embedded document

Documents can contain entire documents embedded as values in a parent document:

```
{"x" : {"foo" : "bar"}}
```

Object ID

An object ID is a 12-byte ID for documents:

```
{"x" : ObjectId()}
```

Binary data

Binary data is a string of arbitrary bytes. It cannot be manipulated from the shell. Binary data is the only way to save non-UTF-8 strings to the database.

Datatypes – a note about arrays

One of the great things about arrays in documents is that MongoDB “understands” their structure and knows how to reach inside of arrays to perform operations on their contents.

This allows us to query on arrays and build indexes using their contents.

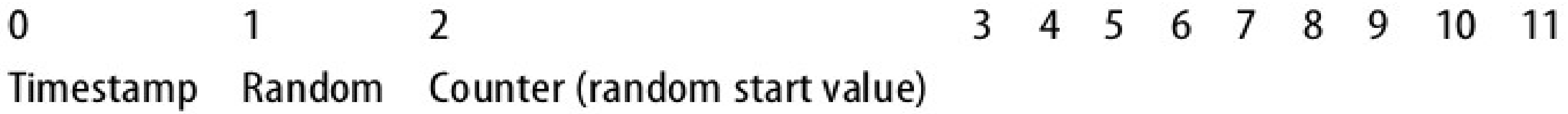
For instance, MongoDB can query for all documents where 3.14 is an element of the "things" array.

If this is a common query, one can even create an index on the "things" key to improve the query’s speed.

Datatypes – a note about objectids

MongoDB was designed to be a distributed database, it was important to be able to generate unique identifiers in a sharded environment.

ObjectIds use 12 bytes of storage, which gives them a string representation that is 24 hexadecimal digits: 2 digits for each byte



0	1	2	3	4	5	6	7	8	9	10	11
Timestamp	Random	Counter (random start value)									

This allows for up to 2563 (16,777,216) unique ObjectIds to be generated per process in a single second.

More on CRUD

In insertMany if there is error in inserting one document then the whole lot is abandoned if "ordered" is set to true in the options argument

Look for deleteOne, deleteMany, drop, replaceOne,

Note: The result of findOne() can be stored in a variable and using the dot notation the fields and thereby the values of those fields can be accessed.

```
joe = db.people.findOne({"name" : "joe", "age" : 20});
{"_id" : ObjectId("4b2b9f67a1f631733d917a7c"),
"name" : "joe",
"age" : 20}
```

```
> joe.age++;
> db.people.replaceOne({"name" : "joe"}, joe);
```

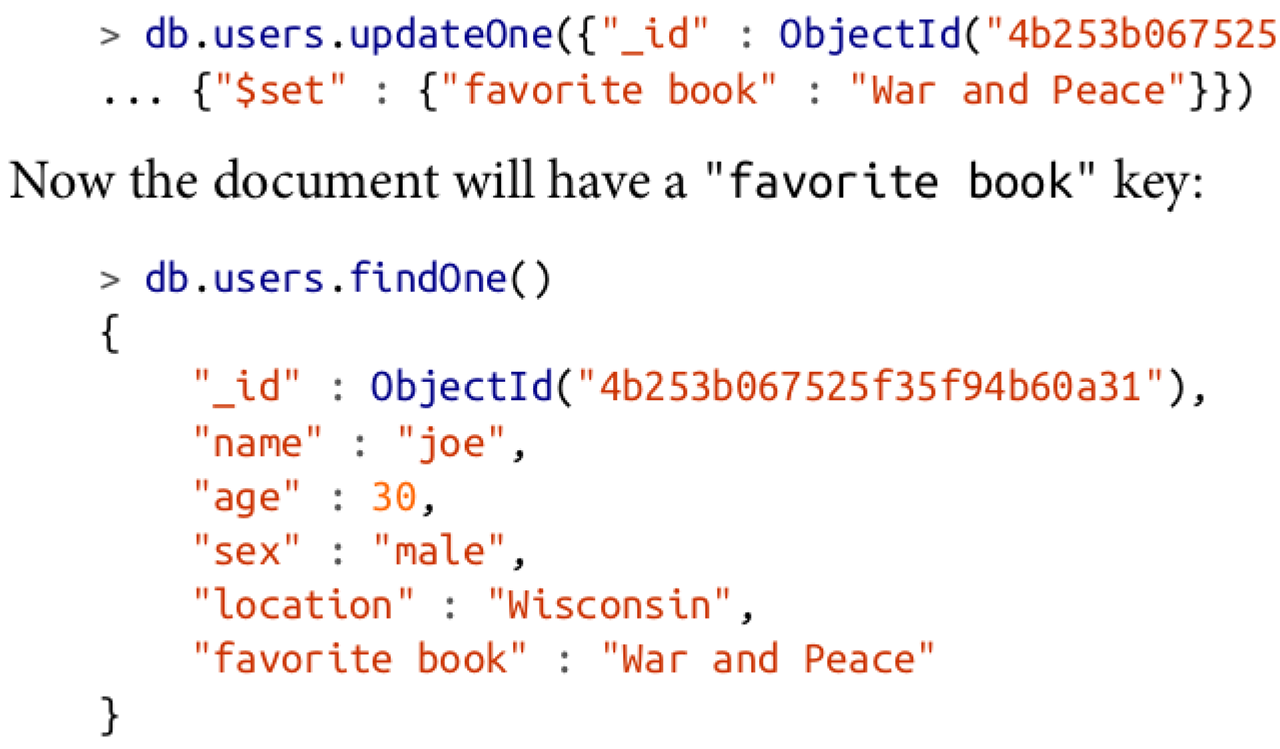
“$set” modifier

"$set" sets the value of a field. If the field does not yet exist, it will be created.

Example:

```
> db.users.findOne()
{
    "_id" : ObjectId("4b253b067525f35f94b60a31"),
    "name" : "joe",
    "age" : 30,
    "sex" : "male",
    "location" : "Wisconsin"
}
```

“$set” modifier



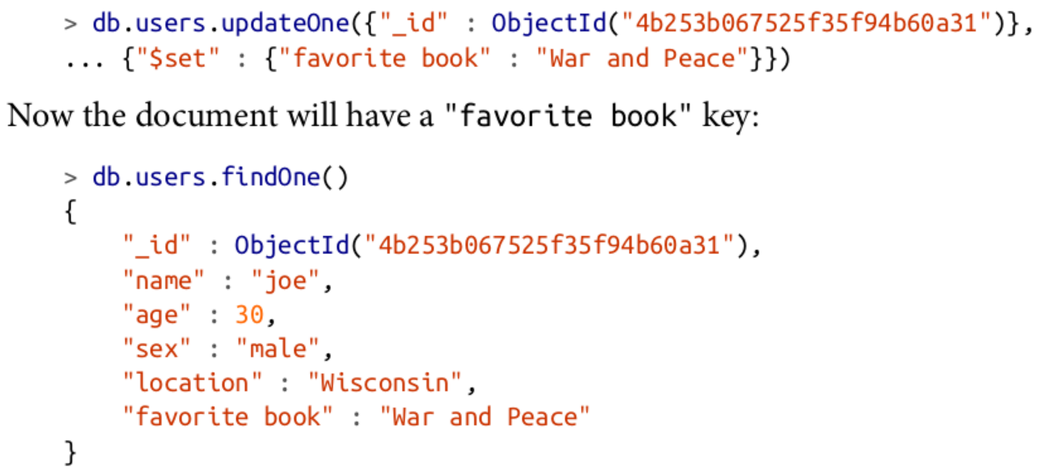
```
> db.users.updateOne({"_id" : ObjectId("4b253b067525f35f94b60a31")},
... {"$set" : {"favorite book" : "War and Peace"}})
```

Now the document will have a "favorite book" key:

```
> db.users.findOne()
{
    "_id" : ObjectId("4b253b067525f35f94b60a31"),
    "name" : "joe",
    "age" : 30,
    "sex" : "male",
    "location" : "Wisconsin",
    "favorite book" : "War and Peace"
}
```

A new key-value pair added.

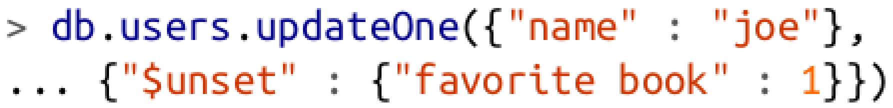
“$set” modifier



```
> db.users.updateOne({"_id" : ObjectId("4b253b067525f35f94b60a31")},
... {"$set" : {"favorite book" : "War and Peace"}})
```

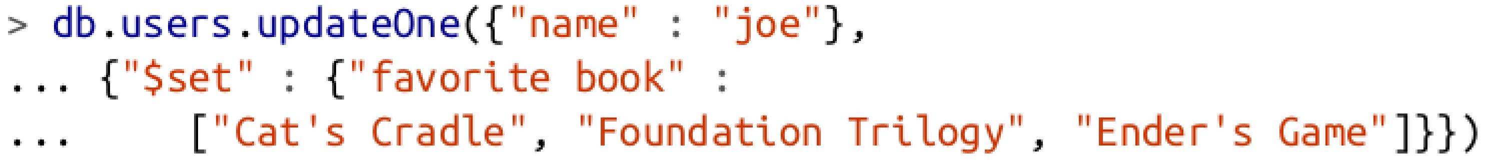
Now the document will have a "favorite book" key:

```
> db.users.findOne()
{
    "_id" : ObjectId("4b253b067525f35f94b60a31"),
    "name" : "joe",
    "age" : 30,
    "sex" : "male",
    "location" : "Wisconsin",
    "favorite book" : "War and Peace"
}
```



```
> db.users.updateOne({"name" : "joe"},
... {"$unset" : {"favorite book" : 1}})
```

Doc is same as before, w/o newly added field

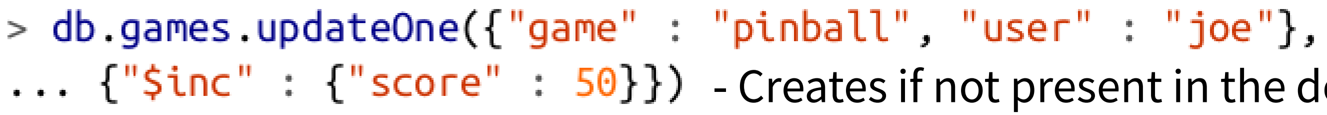


```
> db.users.updateOne({"name" : "joe"},
... {"$set" : {"favorite book" :
...     ["Cat's Cradle", "Foundation Trilogy", "Ender's Game"]}})
```

We can change the type of the field

Operators

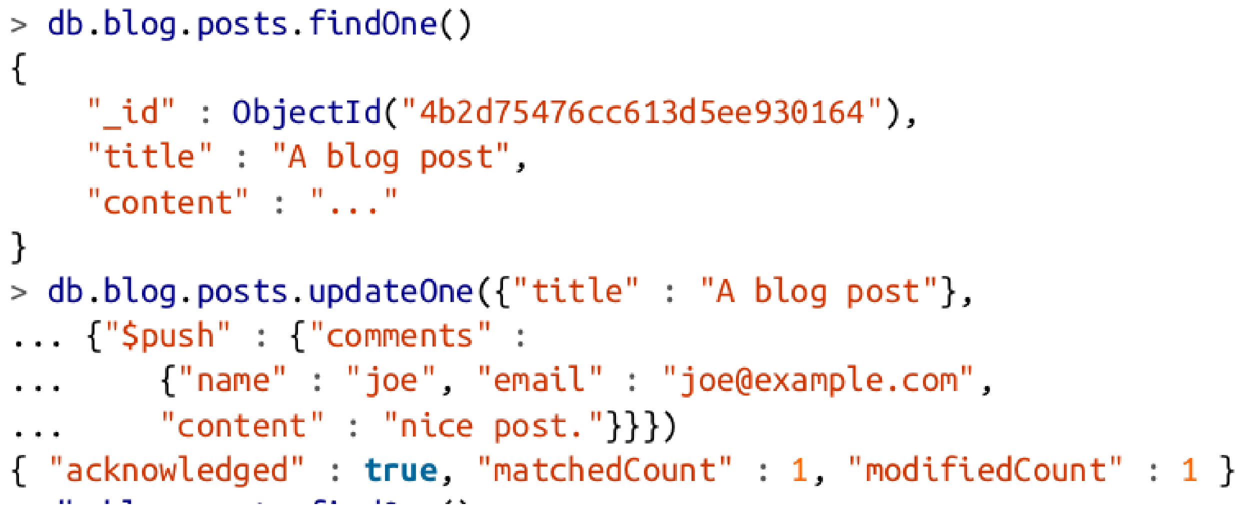
$inc



```
> db.games.updateOne({"game" : "pinball", "user" : "joe"},
... {"$inc" : {"score" : 50}})
```

- Creates if not present in the doc

"$push" adds elements to the end of an array if the array exists and creates a new array if it does not.



```
> db.blog.posts.findOne()
{
    "_id" : ObjectId("4b2d75476cc613d5ee930164"),
    "title" : "A blog post",
    "content" : "..."
}
> db.blog.posts.updateOne({"title" : "A blog post"},
... {"$push" : {"comments" :
...     {"name" : "joe", "email" : "joe@example.com",
...     "content" : "nice post."}}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```