mongoDB

Prof. Jose F. Rodrigues Jr.
University of Sao Paulo, Brazil

Edited and extended from the slides publicly distributed by:
A. Im, G. Cai, H. Tunc, J. Stevens, Y. Barve, S. Hei
Vanderbilt University

http://conteudo.icmc.usp.br/pessoas/junio/Site/index.htm

Content

History

- mongoDB = “Humongous DB”
  - Open-source
  - Document-based
  - “High performance, high availability”
  - Automatic scaling

-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1
-mongodb.org/manual

Motivations

- Problems with SQL
  - Rigid schema
  - Not easily scalable (designed for 90's technology or worse)
  - Requires unintuitive joins (despite its claims, Mongo does not do any better due to physical constraints)

- Perks of mongoDB
  - Easy interface with common languages (Java, Javascript, PHP, etc.)
  - Keeps essential features of RDBMS's while learning from key-value noSQL systems

http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

Company Using mongoDB

"MongoDB powers Under Armour's online store, and was chosen for its dynamic schema, ability to scale horizontally and perform multi-data center replication."

http://www.mongodb.org/about/production-deployments/

In Good Company

-Steve Francia, http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

Data Model

- Document-Based (max 16 MB each entry)
- Documents are in BSON format, consisting of field-value pairs
- Each document stored in a collection
- Collections
  - Like tables of relational db’s.
  - Documents do not have to have uniform structure

-docs.mongodb.org/manual/

JSON

- “JavaScript Object Notation”
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - ordered list of values

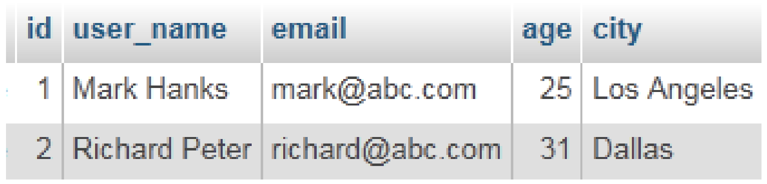
http://json.org/

BSON

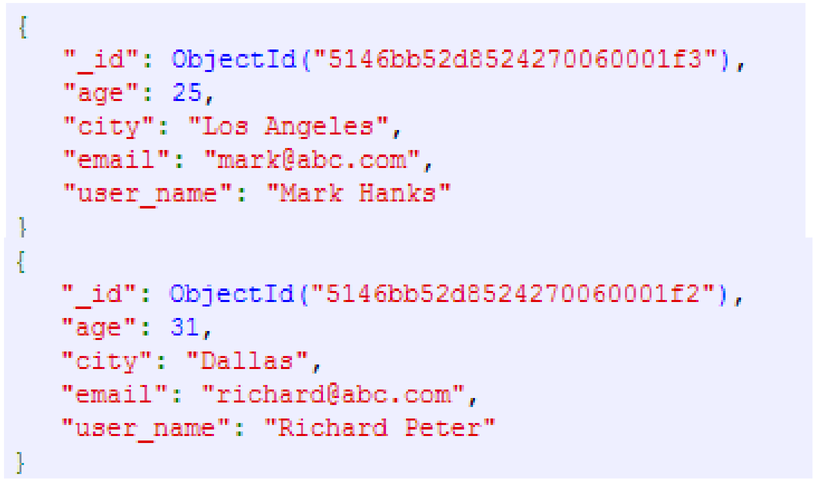
- “Binary JSON”
- Binary-encoded serialization of JSON-like docs
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

http://bsonspec.org/

BSON Example



id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas



```
{
    "_id": ObjectId("5146bb52d8524270060001f3"),
    "age": 25,
    "city": "Los Angeles",
    "email": "mark@abc.com",
    "user_name": "Mark Hanks"
}
{
    "_id": ObjectId("5146bb52d8524270060001f2"),
    "age": 31,
    "city": "Dallas",
    "email": "richard@abc.com",
    "user_name": "Richard Peter"
}
```

BSON Example

```
{
"_id" :       "37010"
"city" :      "ADAMS",
"pop" :       2660,
"state" :     "TN",
“congressmen:”: [“John”, “Willian”, “Adolf”]
“mayor” : {
               name: “John Smith”
               address: “13 Scenic Way”
             }
}
```

➔ Embedding and arrays, more similar to what we have in all-purpose programing languages

BSON Types

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

http://docs.mongodb.org/manual/reference/bson-types/
https://docs.mongodb.com/manual/reference/operator/query/type/

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

http://docs.mongodb.org/manual/reference/bson-types/

The _id Field

- Using the default _id:

```
db.collection.insert({city: “New York“, state:”NY”, pop:”5M”})
```

- Using your own _id:

```
db.collection.insert({_id: 10, city: “New York“, state:”NY”, pop:”5M”})
```

- Using your own composite _id:

```
db.collection.insert({_id: {city: “New York“, state:”NY”}, pop:”5M”})
```

The _id itself is a document.

http://docs.mongodb.org/manual/reference/bson-types/

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition
CRUD	DML

2. CRUD

Create, Read, Update, Delete

Getting Started with mongoDB

To install mongoDB, go to this link and click on the appropriate OS and architecture: http://www.mongodb.org/downloads

First, extract the files (preferably to the C drive).

Finally, create a data directory on C:\ for mongoDB to use

i.e. "md data" followed by "md data\db"

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

Install

- Unzip to any directory
- Find executable “mongod”
- Default connection at `localhost:27017`
- Run

```
mongod --dbpath .
```

- Or just

```
mongod
```

For the default dir (/var/lib/mongodb/ or c:\data\db)

- Or run from bin dir, and have data anywhere else

```
mongod --dbpath <any dir path>
```

- Visual interface

```
https://www.mongodb.com/products/compass
```

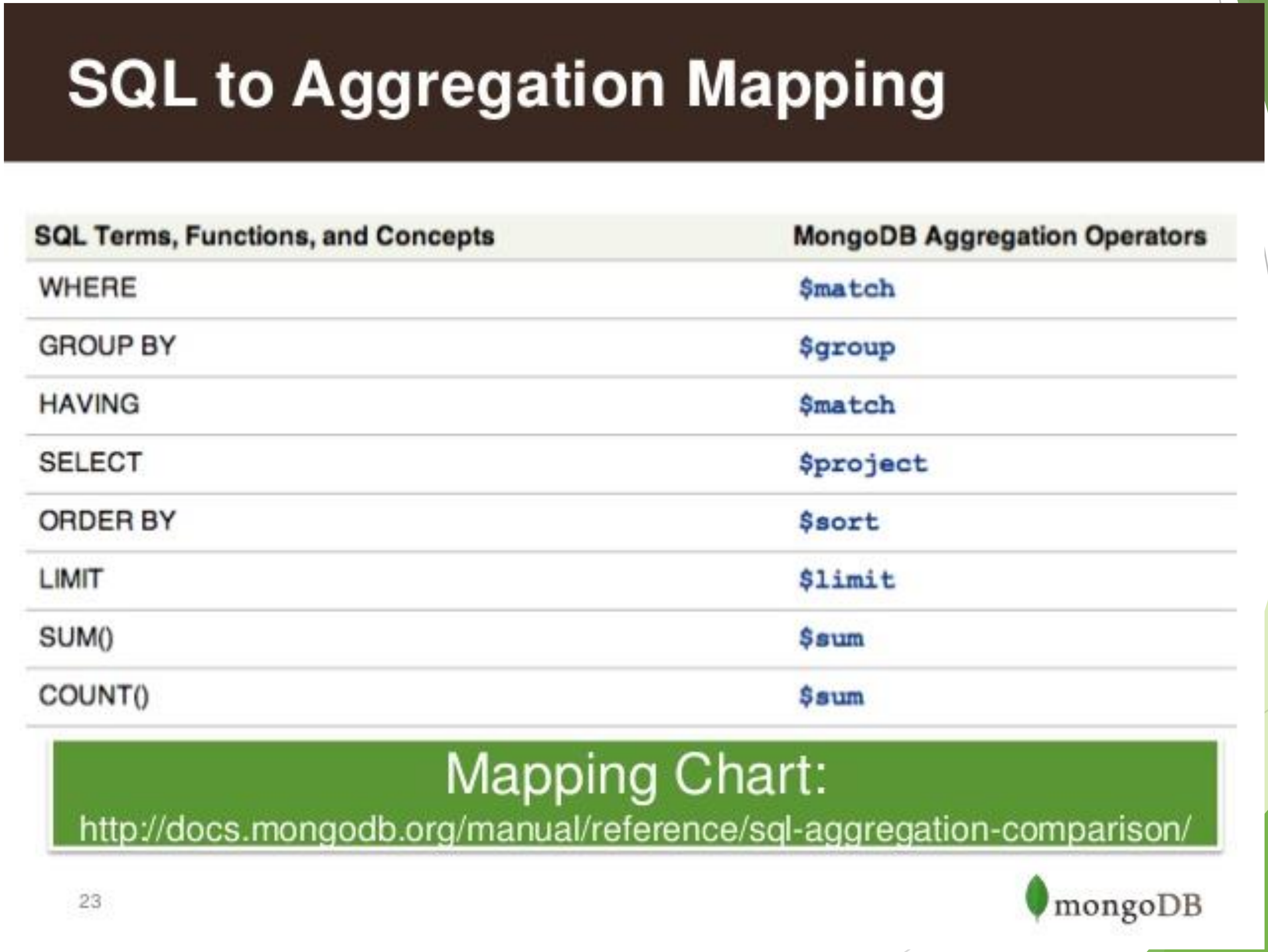
CRUD: Using the Shell

➔ To establish a connection to the server, open another command prompt window and go to the same directory, entering "mongo.exe"

To check which db you're using	`db`
Show all databases	`show dbs`
Switch db's/make a new one	`use <name>`
See what collections exist	`show collections`
Create collection	`db.createCollection("<name>")`

Note: db's are not actually created until you insert data!

CRUD – summary



SQL to Aggregation Mapping

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	`$match`
GROUP BY	`$group`
HAVING	`$match`
SELECT	`$project`
ORDER BY	`$sort`
LIMIT	`$limit`
SUM()	`$sum`
COUNT()	`$sum`

Mapping Chart:
http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/

23

mongoDB

CRUD: Using the Shell (cont.)

To insert documents into a collection/make a new collection:

```
db.<collection>.insert(<document>)
```

<=>

```
INSERT INTO <table>
VALUES(<attributevalues>);
```

CRUD: Inserting Data

Insert one document

```
db.<collection>.insert({<field>:<value>})
```

Inserting a document with a field name new to the collection is inherently supported by the BSON model.

To insert multiple documents, use an array.

CRUD: Querying

- Get all docs: `db.<collection>.find()`
  - Returns a cursor, which is iterated to display first 20 results.
  - Add ".limit(<number>)" to limit results

```
db.<collection>.find().limit(2)
```

  - SELECT * FROM <table>;

- Get one doc: db.<collection>.findOne(), the first in the disk physical order, usually the first inserted

CRUD: Querying

To match a specific value:

```
db.<collection>.find({<field>:<value>})
```

“AND”:

```
db.<collection>.find({<field1>:<value1>,
                      <field2>:<value2>
                      })
```

```
SELECT *
FROM <table>
WHERE <field1> = <value1> AND <field2> = <value2>;
```

CRUD: Querying

```
OR
db.<collection>.find({ $or: [
   {<field>:<value1>},
   {<field>:<value2>}          ]
})
```

```
SELECT *
FROM <table>
WHERE <field> = <value1> OR <field> = <value2>;
```

Checking for multiple values of a set:

```
db.<collection>.find({<field>: {$in [<value>, <value>]}})
SELECT *
FROM <table>
WHERE <field> IN (<value>,<value>);
```

CRUD: Querying

Including/excluding document fields

db.<collection>.find({ }, {<field1>: 1})

SELECT field1

FROM <table>;

0 false
>0 true

db.<collection>.find({<field1>:<value>}, {<field1>: 1})

SELECT field1

FROM <table>

WHERE <field1> = <value>;

CRUD: Querying

Including/excluding document fields

db.<collection>.find({<field1>:<value>}, {<field2>: 0})

SELECT <all fields but not field2>

FROM <table>

WHERE <field1> = <value>;

- notice that find() takes two parameters

CRUD: Updating

```
db.<collection>.update(
{<field1>:<value1>},        //all docs in which field = value
{$set: {<field2>:<value2>}},           //set field to value
{multi:true} )                         //update multiple docs
```

```
UPDATE <table>
SET <field2> = <value2>
WHERE <field1> = <value1>;
```

CRUD: Updating

To remove a field

db.<collection>.update({<field>:<value>},

{ $unset: { <field>: 1}})

ALTER TABLE DROP COLUMN <field>

“WHERE field = value”

CRUD: Removal

Remove all records where field = value

```
db.<collection>.remove({<field>:<value>})
```

```
DELETE FROM <table>
WHERE <field> = <value>;
```

As above, but only remove first document

```
db.<collection>.remove({<field>:<value>}, true)
```

CRUD: Isolation

- By default, all writes are atomic **only** on the level of a single document.
- This means that writes over multiple documents of the same collection can be interleaved with other operations.
- You can isolate writes on an **entire** collection by adding **“$isolated:1”** in the query area:

search criterium

```
db.foo.update(
    { status : "A" , $isolated : 1 },
    { $inc : { count : 1 } },
    { multi: true }
)                               --increments by 1 the field count of every document
                                --with status A in the collection foo
```

➔ In this example, the **$isolated :1** clause makes other clients wait to read and to write the collection until the command is completed

Access control included

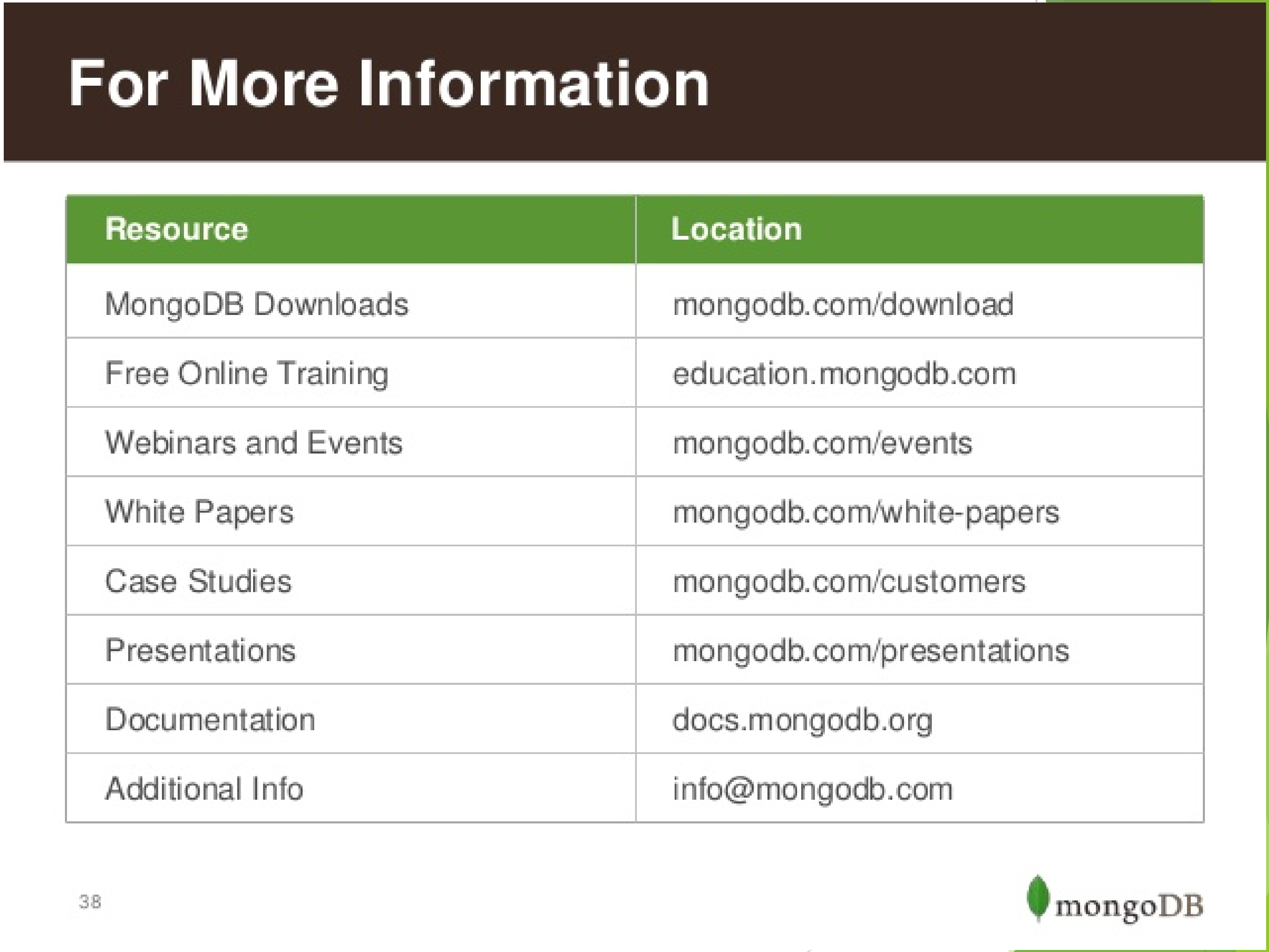
Authentication mode must be set during start up:

```
mongod --auth
```

```
Then, users must be created:

use admin /*as administrator*/
db.createUser(
 { user: "reportsUser", pwd: "12345678",
   roles: [
     { role: "read", db: "reporting" },
     { role: "read", db: "products" },
     { role: "readWrite", db: "accounts" }
   ]
  }
)
```

password



For More Information

Resource	Location
MongoDB Downloads	mongodb.com/download
Free Online Training	education.mongodb.com
Webinars and Events	mongodb.com/events
White Papers	mongodb.com/white-papers
Case Studies	mongodb.com/customers
Presentations	mongodb.com/presentations
Documentation	docs.mongodb.org
Additional Info	info@mongodb.com

mongoDB

MongoDB: The Definitive Guide,

By Kristina Chodorow and Mike Dirolf

Published: 9/24/2010

Pages: 216

Language: English

Publisher: O'Reilly Media, CA