Introduction to MongoDB

CS185C: Introduction to NoSQL Databases

Suneuy Kim

Reference

[1] https://docs.mongodb.com/manual/

[2] The Definitive Guide to MongoDB: A Complete Guide to Dealing with Big Data using MongoDB, Third Edition by David Howes, Peter Membrey, Eelco Plugge and Tim Hawkins, December 16, 2015

[3] MongoDB: The Definitive Guide, 2nd Edition, Powerful and Scalable Data Storage by Kristina Chodorow, May 2013

Using the Right Tool for the Right Job

- One size (relational databases) does not fit all (different types of data)
- MongoDB provides rich document-oriented database that’s optimized for speed and scalability
- Polygot Persistency (e.g. RDBMS for the accounting components and MongoDB for the document storage)

MongoDB Features

- WiredTiger
  - The default storage engine as of MongodDB3.2. (MMAP used to be the default.)
  - Document level locking as compared to collection level locking in MMAP
  - Compression
- Storage engine
  - A software module that a database management system uses to create, read, update data from a database.
  - Storage engine is where processes such as locking, index maintenance, and transactions occur.

JSON and MongoDB

- CSV is to store flat data, not for aggregate

```
Lastname, Firstname, Phone Number
Membrey, Peter, +852 1234 5678
Thielen, Wouter, +81 1234 5678
```

- [Q] What if someone has more than two phone numbers?

- JSON

```
{
   "firstname": "Peter",
   "lastname": "Membrey",
   "numbers": [
      {
         "phone": "+852 1234 5678"
      },
      {
         "fax": "+44 1234 565 555"
      }
   ]
}
```

- JSON allows complex data structures to be represented in a simple, human-readable text format.
- MongoDB **stores data in BSON** (Binary JSON), not in JSON

Using Document-Oriented Storage - BSON

- Binary JSON developed by MongoDB
- MongDB stores data in BSON.
- Easier to traverse and index very quickly at the cost of slightly more space than JSON (MongoDB is meant to be fast, rather than space-efficient.)
- It is easy and quick to convert BSON to a native data structure for each high level language (Python, Ruby, etc.)
- Extensions to JSON
  - Extended types for numeric data (int32 and int64) and support for binary data

Supporting Dynamic Queries

- SQL: static data and dynamic query
- CouchDB: dynamic data and static query or query in map-reduce functions
- MongDB: dynamic data and dynamic query

Note:

- Dynamic data means schema less
- Dynamic query means you can run a query without planning for it in advance.

Index

- All documents are automatically indexed on the _id key – unique index
- A user defined index allows duplicates (e.g. index on last name key)
- By default, an error occurs if you try to create a unique index on a key that has duplicate values
- Indexes on embedded documents (e.g. an index on the ZIP or postal code)
- Extensive support for indexing your documents
  - Composite Indexes (e.g. an index that combines both the lastname and firstname)
  - Geospatial Indexes
  - Many more …

Profiling Queries

- MongoDB's query planner `explain()`

```
db.media.find().explain();
```

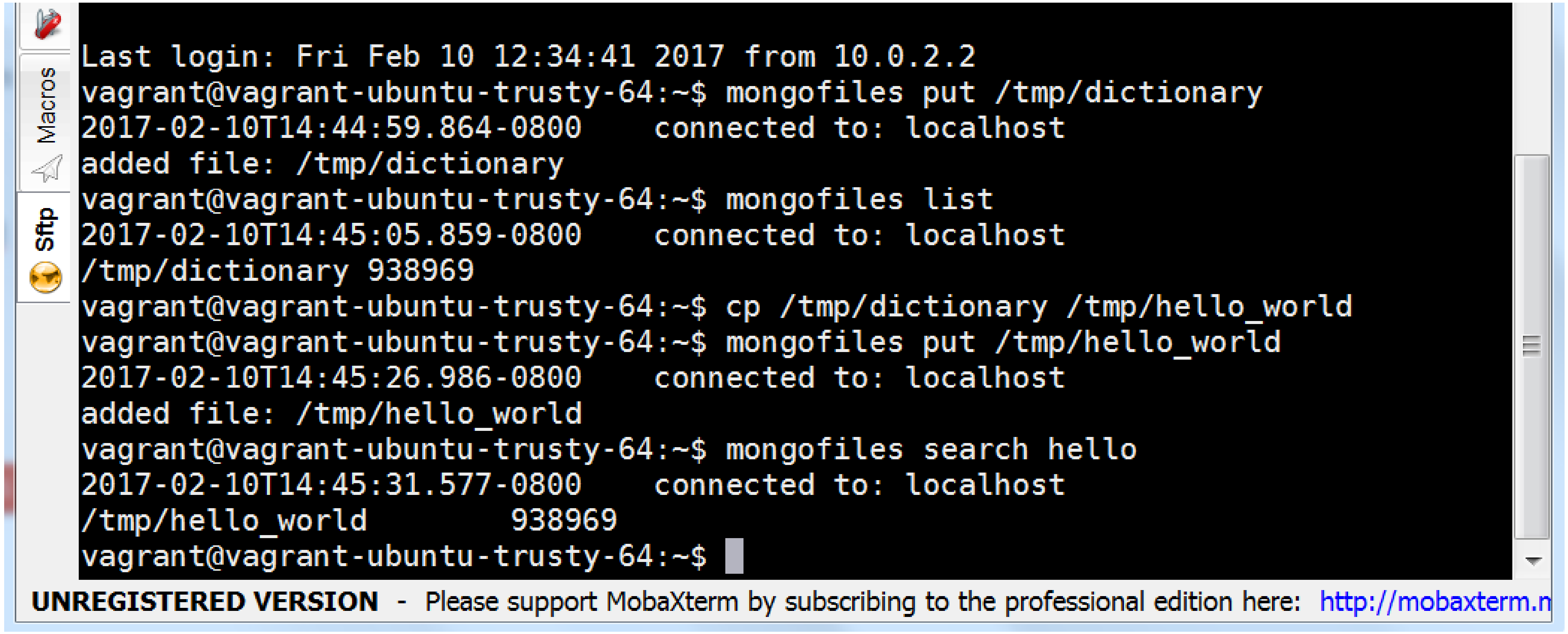
v.s.

```
db.media.find({"_id" :
ObjectId("58013ad941a51bd2d7599db9")}).explain();
```

GridFS

- The maximum size of a MongoDB document in BSON: 16 MB: not enough for movie clips, high-quality audio clips, etc.
- GridFS is to store large files and yet to access parts of the file without retrieving the entire thing.
- To MongoDB, files in GridFS are just normal collections containing documents.
- GridFS consists of two collections
  - Metadata are in the `files` collection
  - Data are broken down into chunks that are stored in the `chunks` collection – easy and scalable

GridFS command line tool: mongofiles

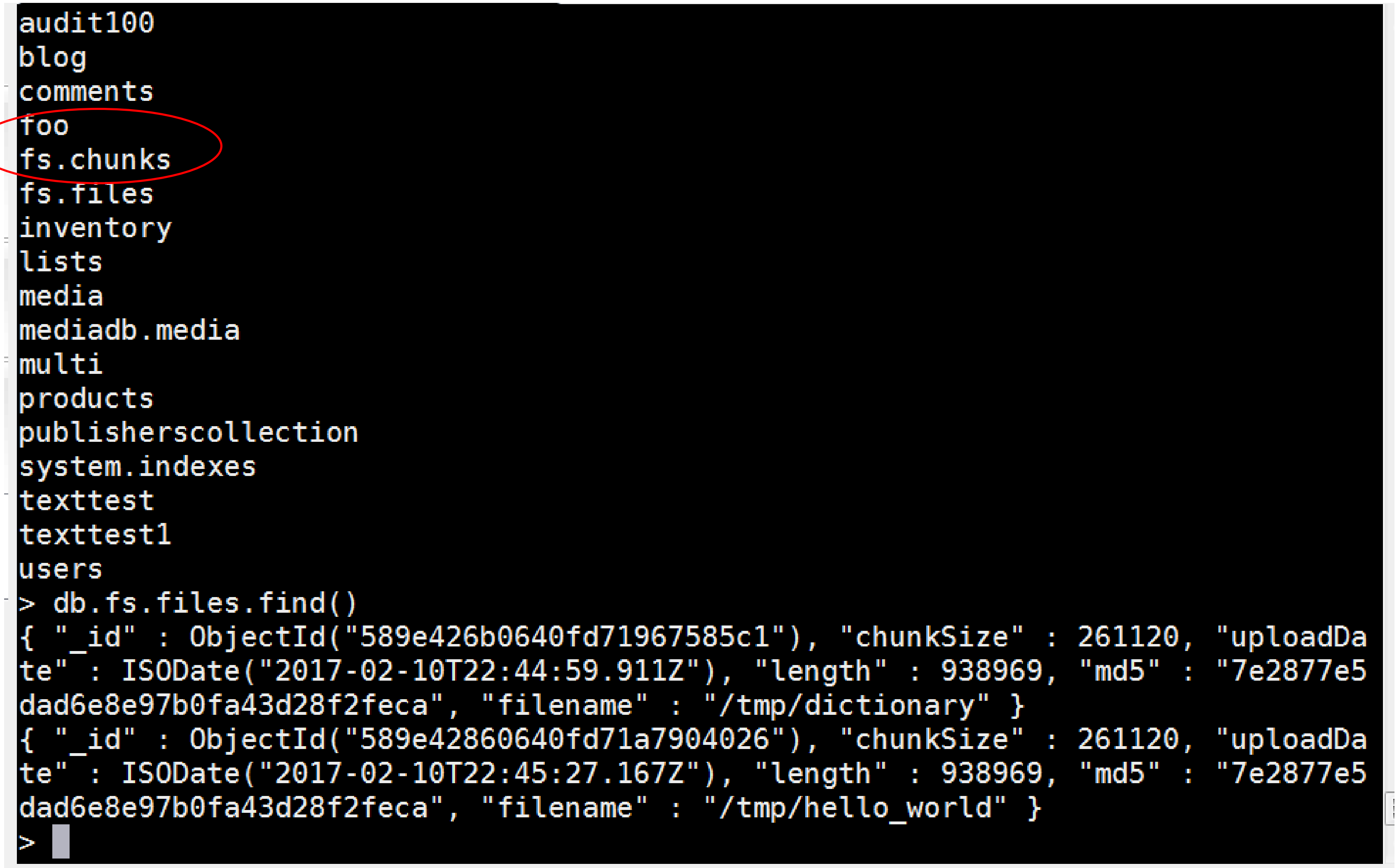


```
Last login: Fri Feb 10 12:34:41 2017 from 10.0.2.2
vagrant@vagrant-ubuntu-trusty-64:~$ mongofiles put /tmp/dictionary
2017-02-10T14:44:59.864-0800    connected to: localhost
added file: /tmp/dictionary
vagrant@vagrant-ubuntu-trusty-64:~$ mongofiles list
2017-02-10T14:45:05.859-0800    connected to: localhost
/tmp/dictionary 938969
vagrant@vagrant-ubuntu-trusty-64:~$ cp /tmp/dictionary /tmp/hello_world
vagrant@vagrant-ubuntu-trusty-64:~$ mongofiles put /tmp/hello_world
2017-02-10T14:45:26.986-0800    connected to: localhost
added file: /tmp/hello_world
vagrant@vagrant-ubuntu-trusty-64:~$ mongofiles search hello
2017-02-10T14:45:31.577-0800    connected to: localhost
/tmp/hello_world        938969
vagrant@vagrant-ubuntu-trusty-64:~$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.m

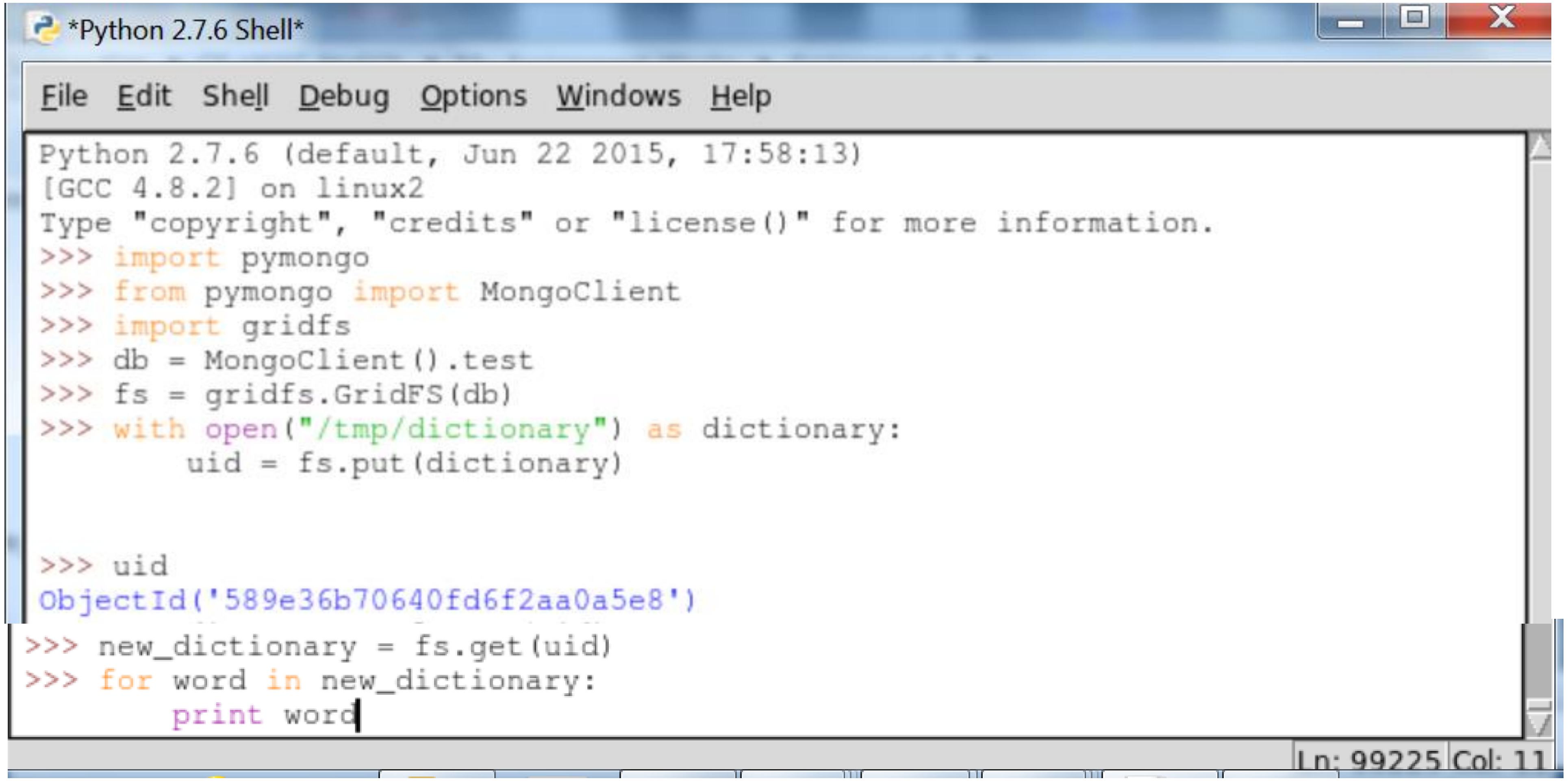
Example from [2] Chapter 5

To MongoDB, files in GridFS are just normal collections containing documents.



```
audit100
blog
comments
foo
fs.chunks
fs.files
inventory
lists
media
mediadb.media
multi
products
publisherscollection
system.indexes
texttest
texttest1
users
> db.fs.files.find()
{ "_id" : ObjectId("589e426b0640fd71967585c1"), "chunkSize" : 261120, "uploadDa
te" : ISODate("2017-02-10T22:44:59.911Z"), "length" : 938969, "md5" : "7e2877e5
dad6e8e97b0fa43d28f2feca", "filename" : "/tmp/dictionary" }
{ "_id" : ObjectId("589e42860640fd71a7904026"), "chunkSize" : 261120, "uploadDa
te" : ISODate("2017-02-10T22:45:27.167Z"), "length" : 938969, "md5" : "7e2877e5
dad6e8e97b0fa43d28f2feca", "filename" : "/tmp/hello_world" }
>
```

GridFS: from pymongo (python driver)



```
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> import pymongo
>>> from pymongo import MongoClient
>>> import gridfs
>>> db = MongoClient().test
>>> fs = gridfs.GridFS(db)
>>> with open("/tmp/dictionary") as dictionary:
        uid = fs.put(dictionary)


>>> uid
ObjectId('589e36b70640fd6f2aa0a5e8')
>>> new_dictionary = fs.get(uid)
>>> for word in new_dictionary:
        print word
```

Example from [2] Chapter 5

Replica Sets

- A replica set has one primary server.
- The primary server handles all the write requests from the clients.
- When a write occurs, it is logged in the primary’s oplog.
- The `oplog` is replicated by the secondary servers in the same replica set.
- When the primary fails, a new primary will be elected among surviving members of the replica.

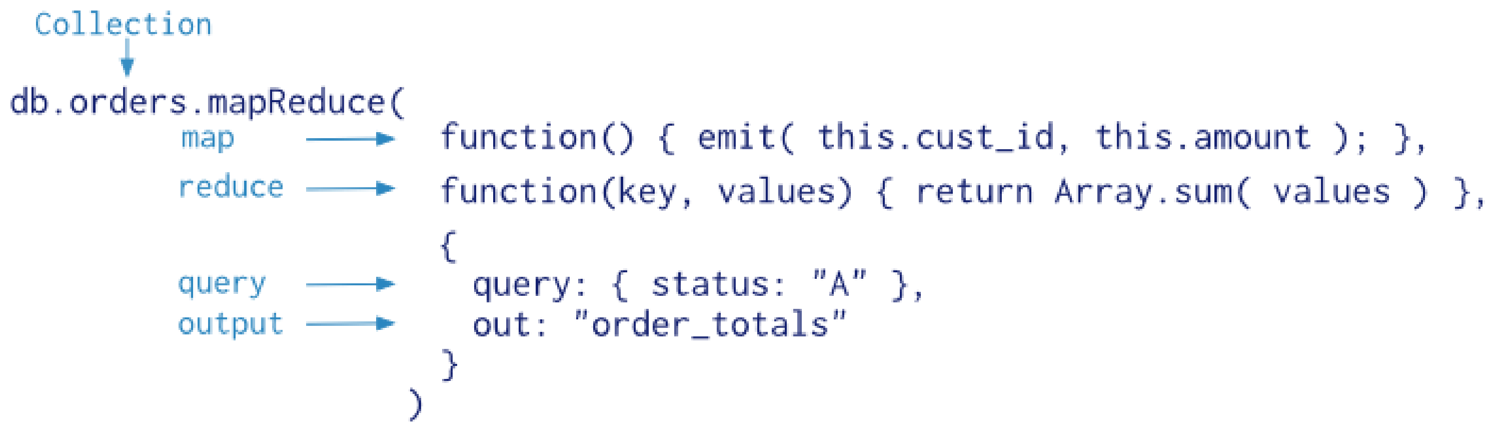
Sharding

- Individual documents are self-contained - BSON
- Sharding provides horizontal scalability - additional shards can be added to increase resource capacity without any changes to your application code.
- Auto-sharding: MongoDB takes care of all the data splitting and recombination for you.

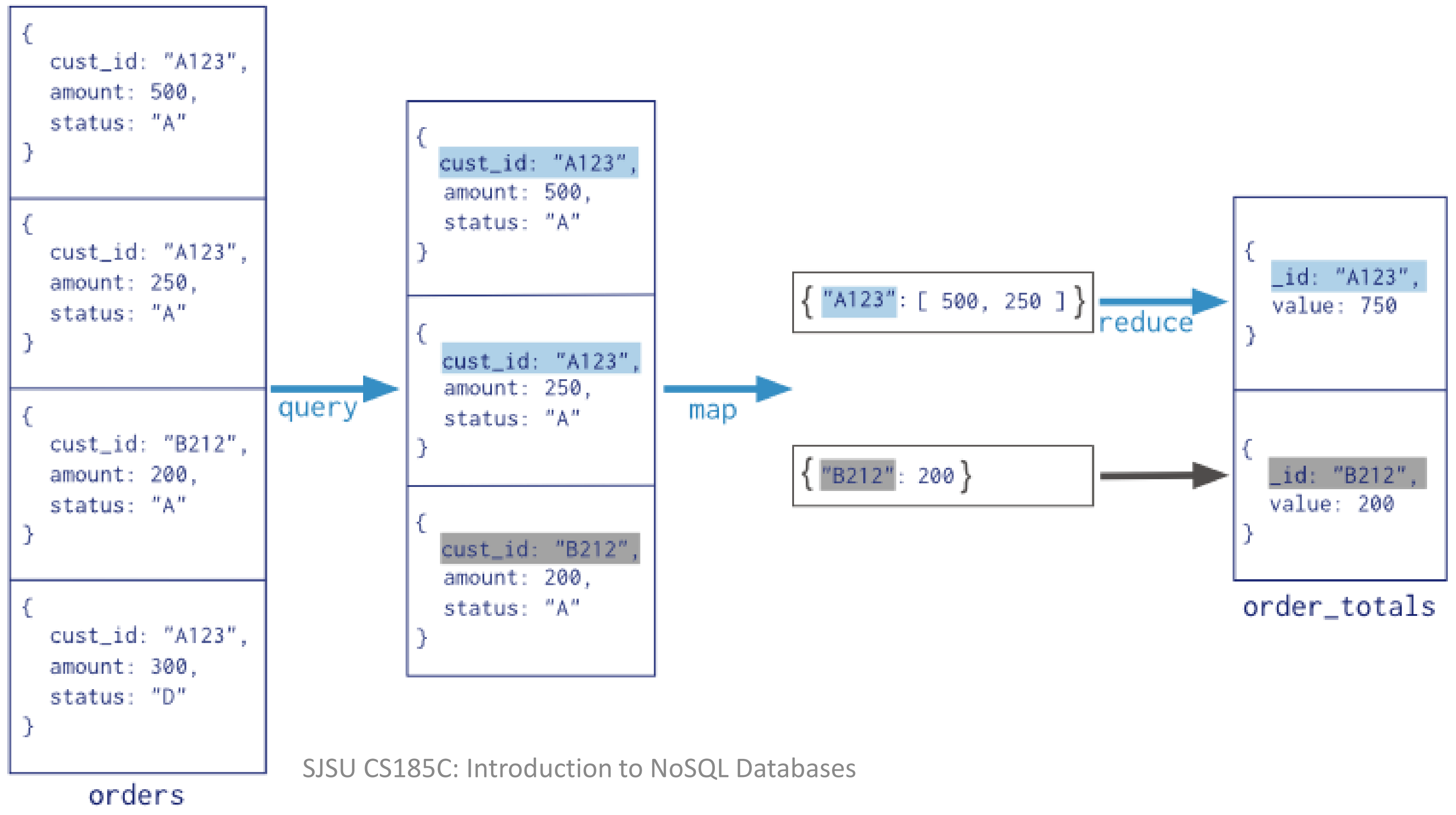
Advanced Queries

- Using Map and Reduce functions
  - Not required but provided.
  - If you would normally use GROUP BY in SQL, then the map and reduce functions may be the right tools for the job in MongoDB.
- The Aggregation Framework
  - Map-reduce can be slow.
  - Pipe line based aggregate operators implemented in C++: highly performant

MongoDB MapReduce function



```
db.orders.mapReduce(
                      function() { emit( this.cust_id, this.amount ); },
                      function(key, values) { return Array.sum( values ) },
                      {
                        query: { status: "A" },
                        out: "order_totals"
                      }
                    )
```



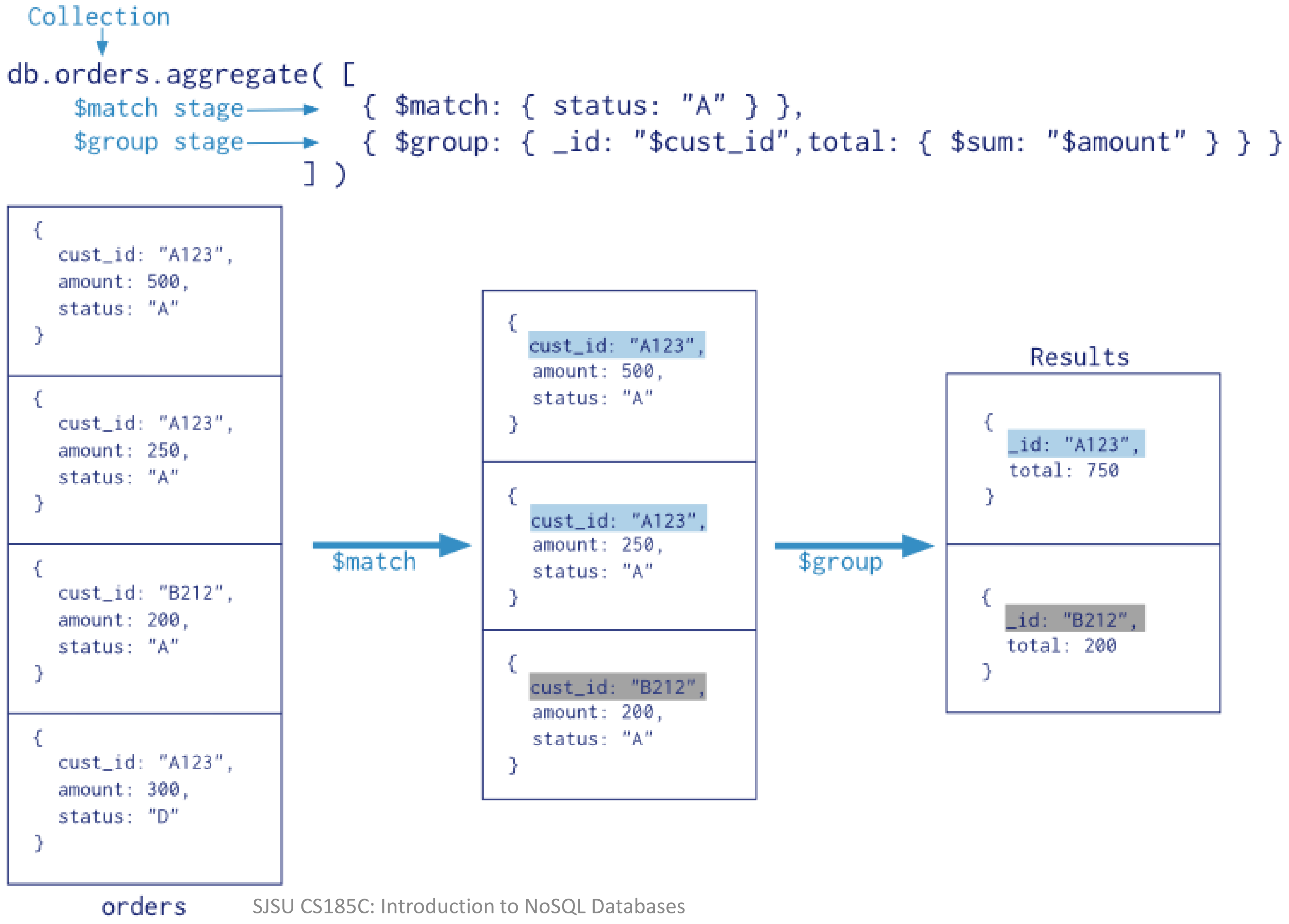
```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{ "A123": [ 500, 250 ] }
{ "B212": 200 }
```

```
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
```

Pipeline based aggregation



```
db.orders.aggregate( [
    { $match: { status: "A" } },
    { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
] )
```

Collection
$match stage
$group stage
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
$match
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
$group
Results
{ _id: "A123", total: 750 }
{ _id: "B212", total: 200 }

Getting Started

Document

- The unit of storage in MongoDB (v.s. a row in RDBMS)
- A document an ordered set of key-value pairs.

e.g. This document {"type" : "Book"} contains one key-value pair consisting of a key named " type", and its value, "Book“.

- Keys are strings
- Values can be any of the following types

String, Integer, Boolean, Double, Min/Max keys, Arrays, Timestamp, Object, Null, Symbol, Date, ObjectID, Binary data, Regular expression, Java Script code.

Document

- MongoDB is type-sensitive and case-sensitive

`{"foo",3}` is distinct from `{"foo":"3"}` and `{"Foo":3}`

- A MongoDB document cannot have duplicate keys
- Key/value pairs in documents are ordered.

`{"x":1,"y":2}` is not the same as `{"y":2,"x":1}`

- Not requires every document to have the same field, or that every field with the same name has the same type of value.

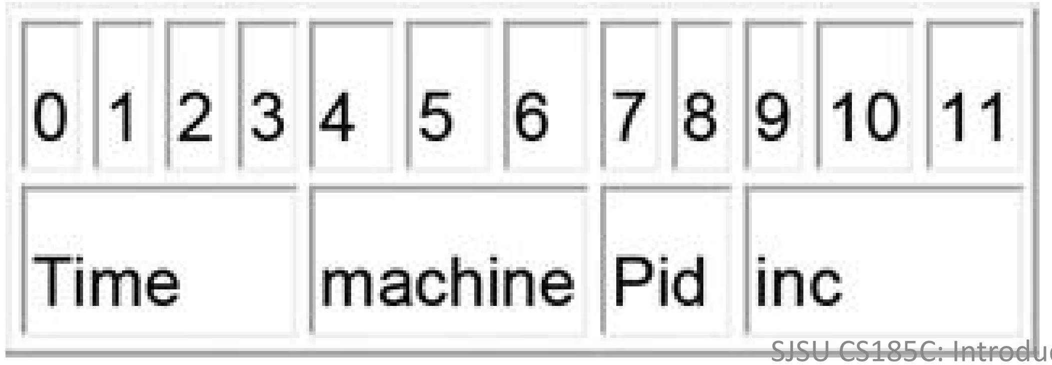
Document Example

Key Value

```
{
   "firstname": "Peter",
   "lastname": "Membrey",
   "phone_numbers": [
      "+852 1234 5678",
      "+44 1234 565 555"
   ]
}
```

`_id` field of document

- Each document has a unique identifier `_id` of which value is auto generated by default.
- `_id` type is `ObjectID` : a 12 byte unique id that can be generated independently in a distributed sharded environment.
- Automatically added to a new document
- The default value is a ObjectId BSON data type consisting of a 12-byte binary value
- Time and inc (=counter) fields are stored in Big Endian format.



0	1	2	3	4	5	6	7	8	9	10	11
Time				machine			Pid		inc		

Collections

- A group of similar documents.
- Dynamic Schemas: Documents within a single collection can have any number of different shapes (different keys and different types of values)
  - You do not need to predefine a structure for any of the document
  - Supports programming in a dynamic typed language such as Python or PHP
  - Still need to define collections and indexes

Collections

- It is common practice to group related types of documents together
    - Application code doesn't have to weed out irrelevant documents
    - Faster to get a list of collections than to extract a list of the types in a collection
    - Data locality
    - Efficient indexing

Collections

- Expandable collections (default) vs. capped collections
- Every collection should have a unique name
  - Should begin with a letter or an underscore (_)
  - $ is reserved by MongoDB
  - An empty string Is not allowed.
  - The null character cannot be used.
  - Cannot start with the `system.` string.
- Each collection accounts for at least two name spaces
  - One for the collection it self
  - One for the first index created in the collection.

Collections

- Sub-collection
  By convention, collections are organized using name spaced sub-collection separated by .
- Example: `blog.authors`, `blog.posts`

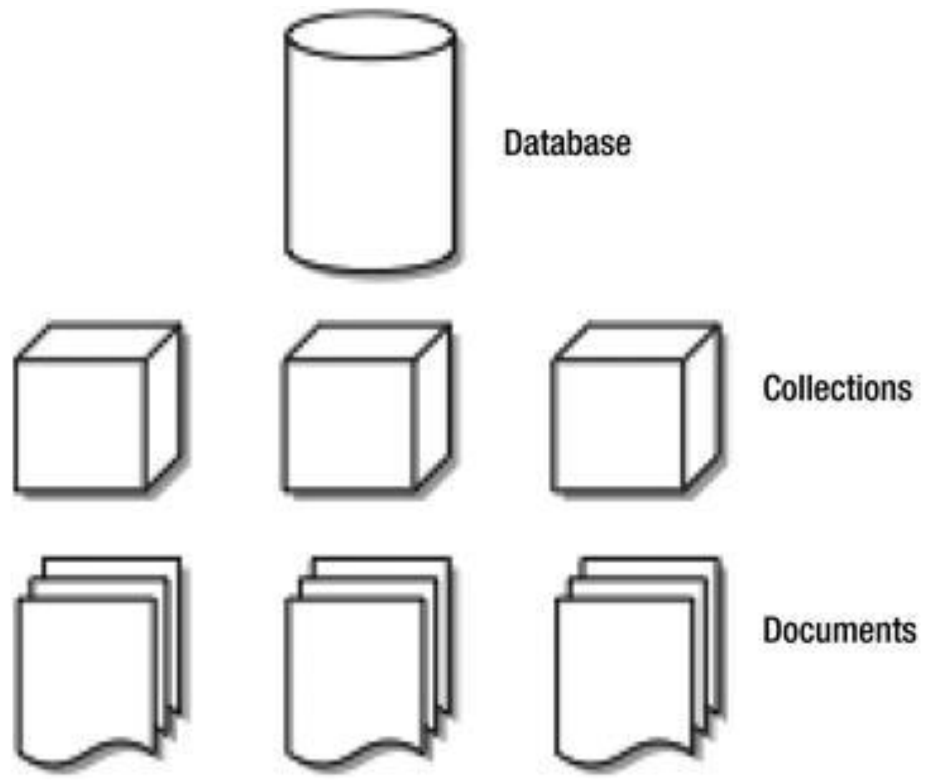
Databases

- A database is a group of collections.
- Rule of thumbs: to store all data for a single application in the same database
- Database names will actually end up as files on your file system. (Naming restrictions exist due to this fact.)
- Reserved database names: admin, local, cofig
- Namespace (fully qualified collection name) = database name + collection name

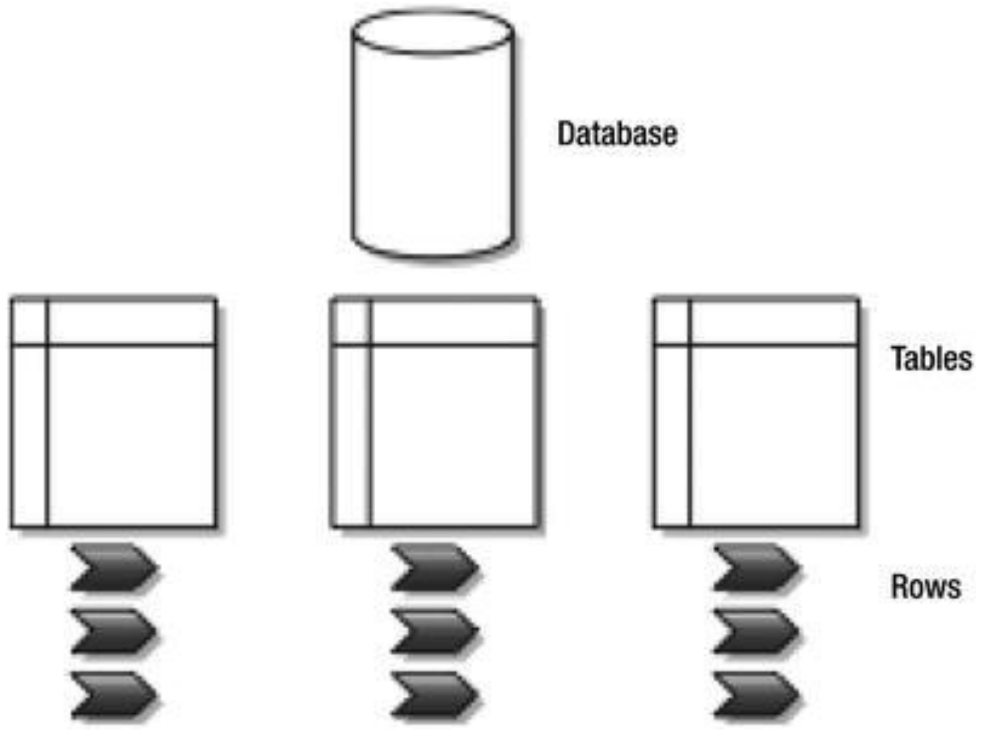
  e.g. cms.blog.posts

Data Model: MongoDB vs RDBMS

MongoDB Database Model



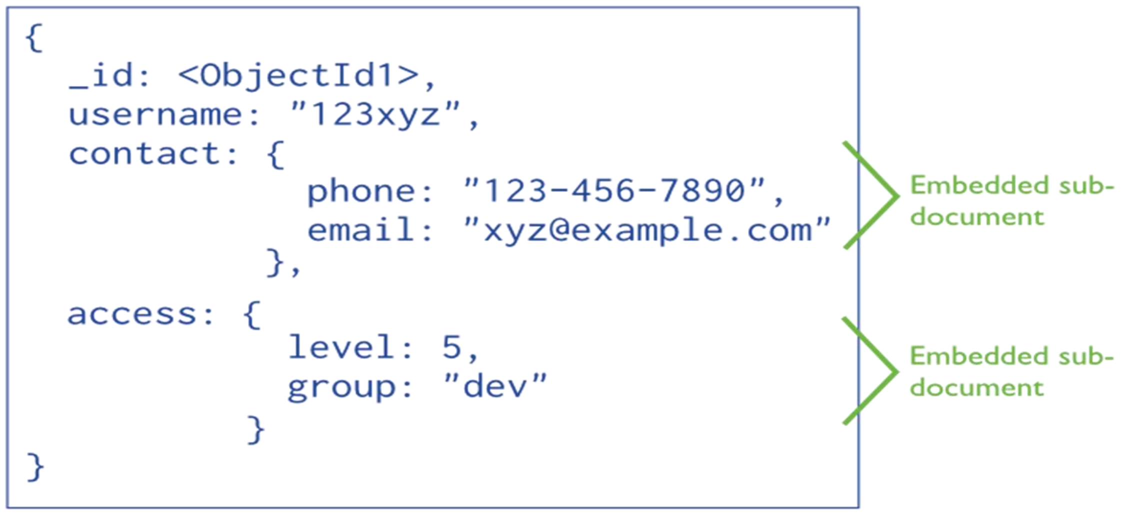
RDBMS Data Model



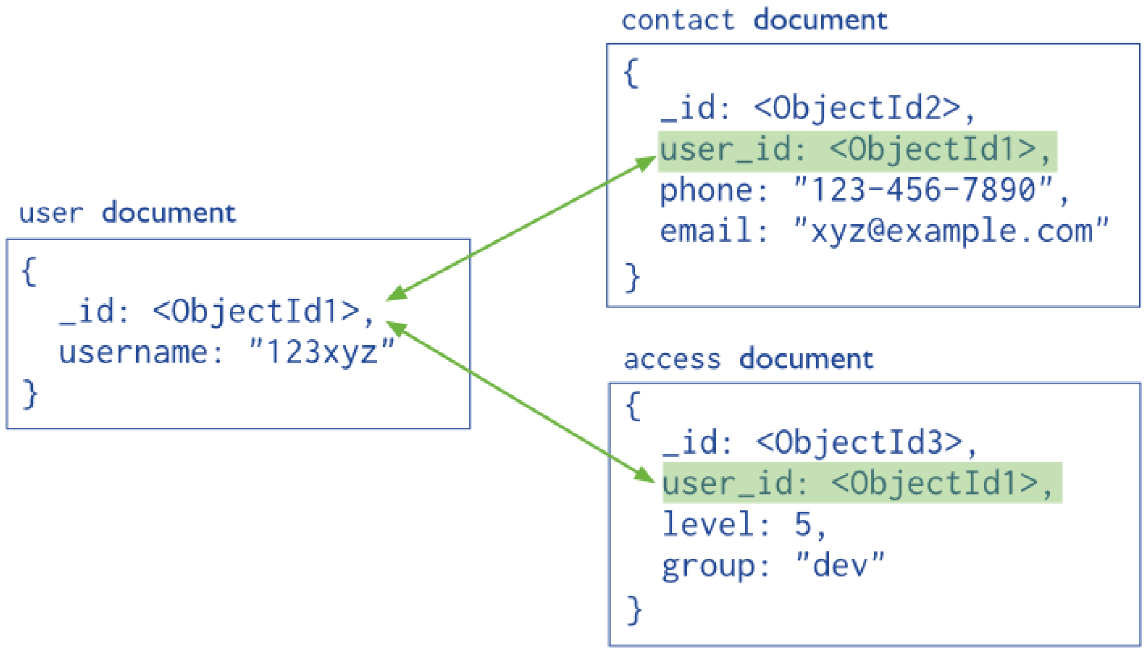
Embedding vs. Referencing Information in Documents

- Embedding information: you place a certain type of data (for example, an array containing more data) into the document itself.
- Referencing information: you create a reference to another document that contains that specific data (e.g. RDBMS foreign keys. Joins are needed to put information together.)
- With MongoDB, embed data whenever you can
  - To ensure that all related information is kept in one single document
  - Works much faster because the data are co-located in the disk

Embedding vs. Referencing



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```



user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Every reference needs another query in the database.

Basic Data Types

- JSON’s expressive capabilities are limited because the only types are null, boolean, numeric, string, array, and object.
- MongoDB adds support for a number of additional data types while keeping JSON’s essential key/value pair nature.

Basic Data Types

Type	Example
null	{"x" : null}
boolean	{"x" : true}
number	{"x" : 3.14} {"x" : NumberInt("3")}
string	{"x" : "foobar"}
date	{"x" : new Date()}
regular expression	{"x" : /foobar/i}
array	{"x" : ["a", "b", "c"]}
embedded document	{"x" : {"foo" : "bar"}}
code	{"x" : function() { /* ... */ }}

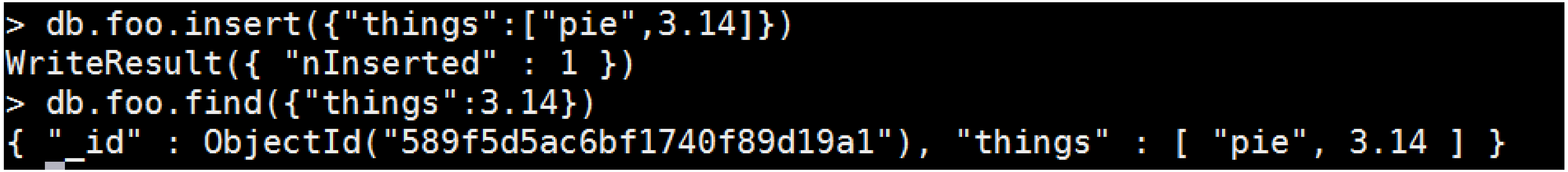
Arrays

- Arrays can contain different data types as values

  e.g.) {"things" : ["pie", 3.14]}

- MongoDB knows how to reach inside of arrays to perform operations on their contents.

  e.g.) Find all documents where 3.14 is an element of the "things" array.



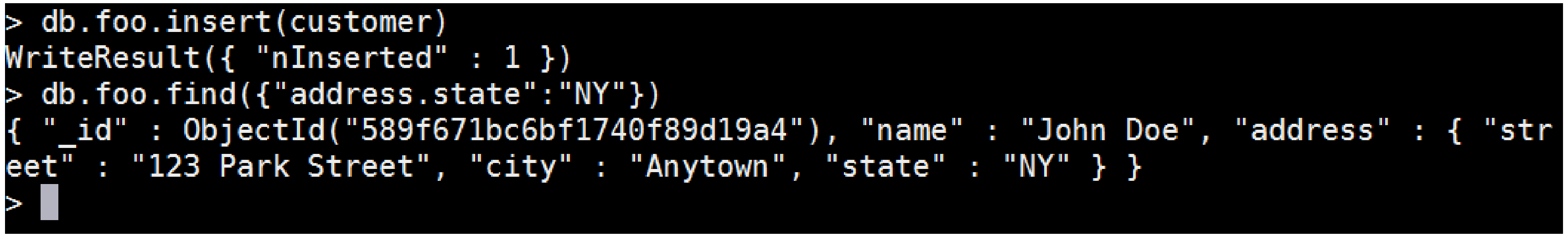
```
> db.foo.insert({"things":["pie",3.14]})
WriteResult({ "nInserted" : 1 })
> db.foo.find({"things":3.14})
{ "_id" : ObjectId("589f5d5ac6bf1740f89d19a1"), "things" : [ "pie", 3.14 ] }
```

Embedded Documents

- Documents can be used as the value for the key.
- Embedded "address" document

```
{
       "name" : "John Doe",
       "address" : {
               "street" : "123 Park Street",
               "city" : "Anytown",
               "state" : "NY"
        }
}
```

- MongoDB is able to reach inside embedded documents to build indexes, perform queries, or make updates.



```
> db.foo.insert(customer)
WriteResult({ "nInserted" : 1 })
> db.foo.find({"address.state":"NY"})
{ "_id" : ObjectId("589f671bc6bf1740f89d19a4"), "name" : "John Doe", "address" : { "str
eet" : "123 Park Street", "city" : "Anytown", "state" : "NY" } }
>
```

To Setup and Deploy MongoDB

- Refer to a separate document from the course web site.

To start/stop the server

- `$sudo service mongod status`
- `$sudo service mongod start [restart|stop]`
- The default data directory `mongod` uses: `/data/db`
- Make sure to create `/data/db` and the user has a write permission to this directory.
- The default port: `27017`

mongo – Mongo Shell

- A JavaScript shell
- JavaScript equivalents to shell helpers

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