MODULE 8

MONGODB

Terminology

- 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).
- A *collection* can be thought of as a table with a dynamic schema.
- A single instance of MongoDB can host multiple independent *databases*, each of which can have its own collections.
- Every document has a special key, "_id", that is unique within a collection.
- MongoDB comes with a simple but powerful JavaScript *shell*, which is useful for the administration of MongoDB instances and data manipulation.

Key-Value Documents

- An ordered set of keys with associated values.
- The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.
- In JavaScript, for example, documents are represented as objects:

```
{"greeting" : "Hello, world!"}
```

- multiple key/value pairs:

```
{"greeting" : "Hello, world!",
 "foo" : 3}
```

Keys in Documents

- The keys in a document are strings.
- Any UTF-8 character is allowed in a key, with a few notable exceptions:
- Keys must not contain the character **\0** (the null character).
  - This character is used to signify the end of a key.
- The **.** and **$** characters have some special properties and should be used only in certain circumstances.
  - In general, they should be considered reserved, and drivers will complain if they are used inappropriately.

Type-sensitive and Case-sensitive

> MongoDB is type-sensitive and case-sensitive.

> For example, these documents are distinct:

```
{"foo" : 3}
{"foo" : "3"}
```

> as are as these:

```
{"foo" : 3}
{"Foo" : 3}
```

Duplicate Keys

> MongoDB cannot contain duplicate keys.

> For example, the following is not a legal document:

```
{
  "greeting" : "Hello, world!",
  "greeting" : "Hello, MongoDB!"
}
```

Ordered Key-Value Pairs

> Key/value pairs in documents are ordered:

```
{"x" : 1, "y" : 2}
```

> is not the same as

```
{"y" : 2, "x" : 1}
```

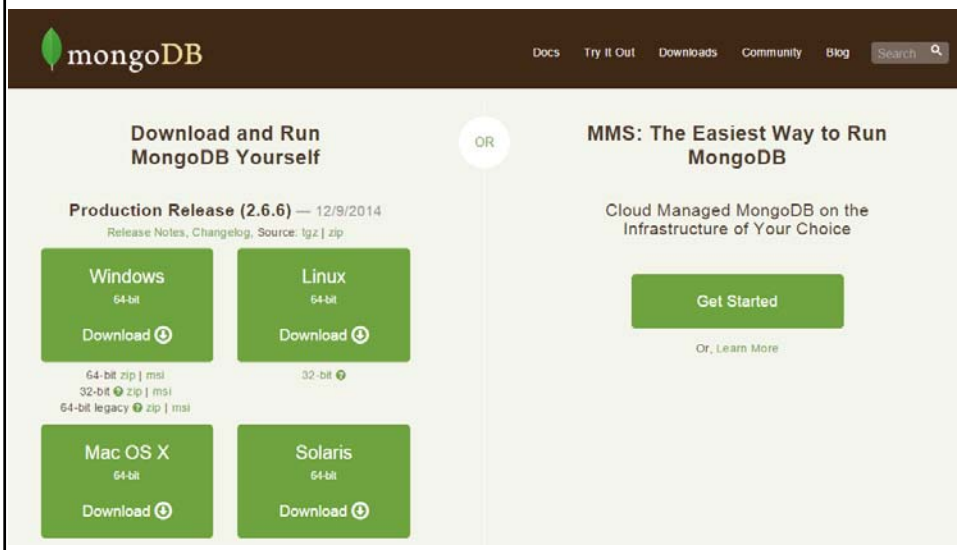
> Field order does not usually matter and you should not design your schema to depend on a certain ordering of fields (MongoDB may reorder them).

Module 5

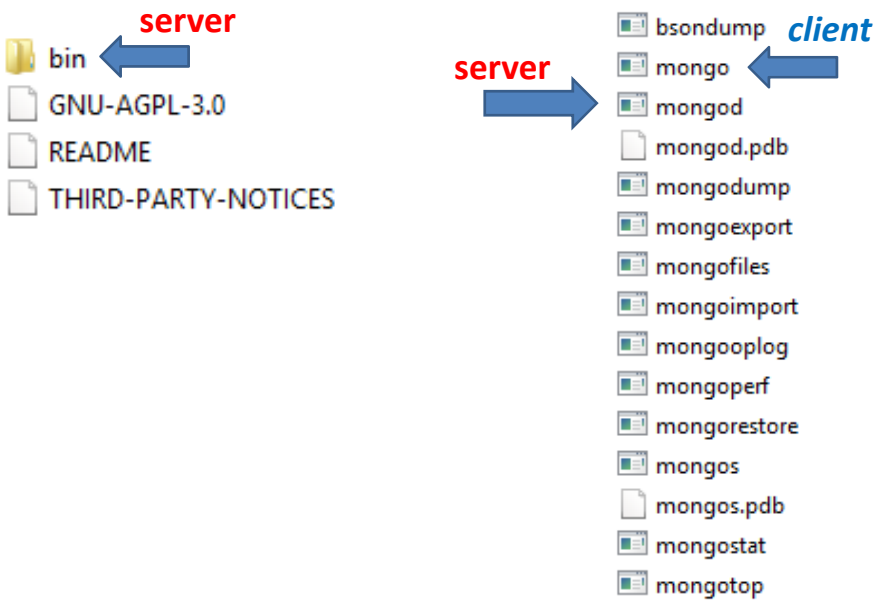
MongoDB

INSTALLATION

http://www.mongodb.org/downloads



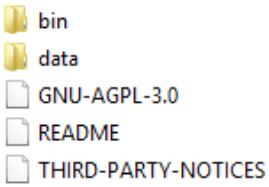
Installation Folder



Data folder

> Create a data folder

```
data\db
```



Running the server

```
set MONGO_HOME=c:\opt64\mongodb-2.6.6
set PATH=%MONGO_HOME%\bin;%PATH%
start mongod --dbpath=%MONGO_HOME%\data
```

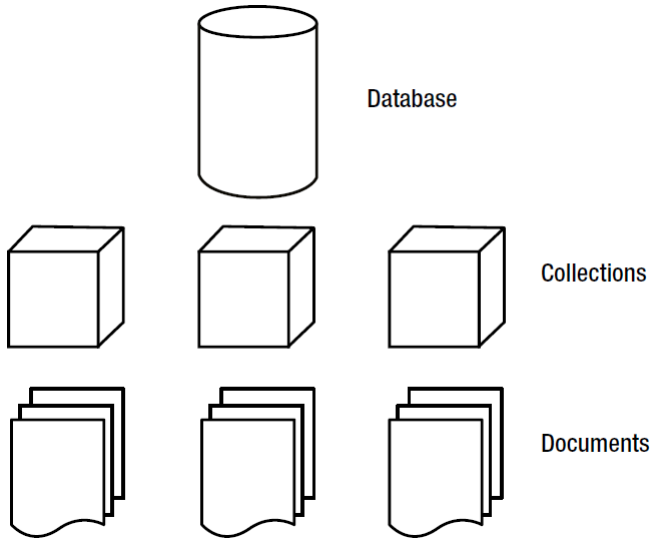
MongoDB Applications

bsondump	Reads contents of BSON-formatted rollback files
mongo	The database shell
mongod	The core database server
mongodump	Database backup utility
mongoexport	Export utility (JSON, CSV, TSV), not reliable for backup
mongofiles	Manipulates files in GridFS objects
mongoimport	Import utility (JSON, CSV, TSV), not reliable for recoveries
mongooplog	Pulls oplog entries from another mongod instance
mongoperf	Check disk I/O performance
mongorestore	Database backup restore utility
mongos	Mongodb sharding routerprocess
mongosniff	Sniff/traces MongoDB database activity in real time, Unix-like systems only
mongostat	Returns counters of database operation
mongotop	Tracks/reports MongoDB read/write activities
mongorestore	Restore/import utility

Connecting to the server

```
cmd> mongo
MongoDB shell version: 2.6.6
connecting to: test
>
```

The MongoDB database model



Basic Commands within the MongoDB Shell

Command	Function
`show dbs`	Shows the names of the available databases.
`show collections`	Shows the collections in the current database.
`show users`	Shows the users in the current database.
`use <db name>`	Sets the current database to <db name>.

The Shell

```
> Math.sin(Math.PI/4)
0.7071067811865475
> Math.tan(Math.PI/4)
0.9999999999999999
> function fact(n){
... if (n==0 || n==1) return 1;
... return n * fact(n-1);
... }
> fact(5)
120
```

Creating a database

```
MongoDB shell version: 2.6.6
connecting to: test
> show dbs
admin  (empty)
local  0.078GB
> use world
switched to db world
> show dbs
admin  (empty)
local  0.078GB
> db
world
```

Inserting to a database

```
> db.world.insert({code: "TUR", name: "Turkey",
population: 77000000});
WriteResult({ "nInserted" : 1 })
> var country= {};
> country.code="USA";
> country.name="United States of America";
> country.population=310000000;
310000000
> country
{
        "code" : "USA",
        "name" : "United States of America",
        "population" : 310000000
}
```

Inserting to a database

```
> db.world.save(country);
WriteResult({ "nInserted" : 1 })
> db.world.find()
{ "_id" : ObjectId("5392b9aba8b5a76657943b78"),
"code" : "TUR", "name" : "Turkey", "popula
tion" : 77000000 }
{ "_id" : ObjectId("5392bac5a8b5a76657943b79"),
"code" : "USA", "name" : "United States of
 America", "population" : 310000000 }
```

Delete

- **remove** permanently deletes documents from the database.
- Called with no parameters, it removes all documents from a collection.
- It can also take a document specifying criteria for removal.

```
> db.world.remove({ "code": "TUR"})
WriteResult({ "nRemoved" : 1 })
> db.world.remove({})
```

Update

- update takes (at least) two parameters
  - the criteria to find which document to update
  - the new document.

```
> var tr = db.world.findOne({ _id: "TUR"})
> tr.population = Number(tr.population)+1
70000001
> db.world.update({"_id": tr._id},tr);
WriteResult({ "nMatched" : 1, "nUpserted" : 0,
"nModified" : 1 })
> var tr = db.world.findOne({ _id: "TUR"})
> tr
{ "_id" : "TUR", "name" : "Turkiye",
"population" : 70000001 }
```

Basic Data Types

- Documents in MongoDB can be thought of as “JSON-like” in that they are conceptually similar to objects in JavaScript.
- JSON is a simple representation of data
  - the specification lists only six data types.
- MongoDB adds support for a number of additional data types while keeping JSON’s essential key/value pair nature.

Basic Data Types

- null
  - Null 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}
```

Basic Data Types

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

```
{"x" : 3.14}
```

or:

```
{"x" : 3}
```

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

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

Basic Data Types

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

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

- date
  - Dates are stored as milliseconds since the epoch.
  - 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}
```

Basic Data Types

> array

– Sets or lists of values can be represented as arrays:

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

– 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.

Basic Data Types

> embedded document

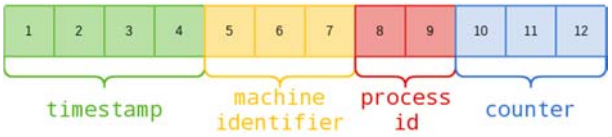
– Documents can contain entire documents embedded as values in a parent document:

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

Basic Data Types

- *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.
- *code*
  - Queries and documents can also contain arbitrary JavaScript code:

```
{"x" : function() { /* ... */ }}
```



Module 5

MongoDB

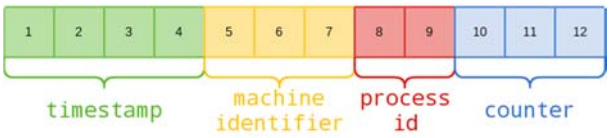
OBJECTIDS

"_id"

- Immutable and unique
  - You cannot change after the document is created
  - Two different documents cannot have the same `_id` attribute value

"_id"

- **`ObjectId`** (12-Byte) (Big-Endian)
  - Time the data created (4-Byte)
  - Process Id (2-Byte)
  - Machine Id (3-Byte)
  - Incremental number (3-Byte)



ObjectId

```
> new ObjectId
ObjectId("5392be94a8b5a76657943b7a")
> new ObjectId
ObjectId("5392be95a8b5a76657943b7b")
> new ObjectId
ObjectId("5392be96a8b5a76657943b7c")
```

_id.getTimestamp()

```
> db.world.find()[0]
{
  "_id" : ObjectId("5392b9aba8b5a76657943b78"),
        "code" : "TUR",
        "name" : "Turkey",
        "population" : 77000000
}
> db.world.find()[0]._id
ObjectId("5392b9aba8b5a76657943b78")
> db.world.find()[0]._id.getTimestamp()
ISODate("2014-06-07T07:05:15Z")
```

_id.str

- The hexadecimal string representation of the object.
- **toString()**

  Returns the JavaScript representation in the form of a string literal
- **valueOf()**
  - Returns the representation of the object as a hexadecimal string.
  - The returned string is the **str** attribute.

Relations

- You can choose either to embed information into a document or reference that information from another document

```
|_media
    |_cds
        |_id, artist, title, genre, releasedate
    |_ cd_tracklists
        |_cd_id, songtitle, length


|_media
    |_items
        |_<document>
```

Relations

> In the non-relational approach, the document might look like:

```
{
    "Type": "CD",
    "Artist": "Nirvana",
    "Title": "Nevermind",
    "Genre": "Grunge",
    "Releasedate": "1991.09.24",
    "Tracklist": [
        {
        "Track" : "1",
        "Title" : "Smells Like Teen Spirit",
        "Length" : "5:02"
        },
        {
        "Track" : "2",
        "Title" : "In Bloom",
        "Length" : "4:15"
        }
    ]
}
```

Relations

> When information is retrieved for a given CD, that information only needs to be loaded from one document into RAM, not from multiple documents.

> Remember that every reference requires another query in the database.

> The rule of thumb

– when using MongoDB is to embed data whenever you can.

– This approach is far more efficient and almost always viable.

Importing json

```
mongoimport --db world
        --collection countries1
        --type json
        --quiet
        --file countries_v1.json
        --jsonArray
```

db.cities2.stats()


```
{
        "ns" : "world.cities2",
        "count" : 4078,
        "size" : 462880,
        "avgObjSize" : 113,
        "storageSize" : 696320,
        "numExtents" : 4,
        "nindexes" : 1,
        "lastExtentSize" : 524288,
        "paddingFactor" : 1,
        "systemFlags" : 1,
        "userFlags" : 1,
        "totalIndexSize" : 122640,
        "indexSizes" : {
                "_id_" : 122640
        },
        "ok" : 1
}
```

db.cities2.storageSize()

```
> db.cities2.storageSize()
696320
> db.cities2.totalSize()
818960
> db.cities2.totalIndexSize()
122640
```

Queries

```
> db.countries.find(
  { population : { $gt : 50000000 } },
  {name: true}).count()
24
```

Queries

```
> db.countries.find({ population : { $gt :
50000000 , $lt: 100000000}},{name: true})
{ "_id" : "DEU", "name" : "Germany" }
{ "_id" : "EGY", "name" : "Egypt" }
{ "_id" : "ETH", "name" : "Ethiopia" }
{ "_id" : "FRA", "name" : "France" }
{ "_id" : "GBR", "name" : "United Kingdom" }
{ "_id" : "IRN", "name" : "Iran" }
{ "_id" : "ITA", "name" : "Italy" }
{ "_id" : "MEX", "name" : "Mexico" }
{ "_id" : "PHL", "name" : "Philippines" }
{ "_id" : "THA", "name" : "Thailand" }
{ "_id" : "TUR", "name" : "Turkey" }
{ "_id" : "UKR", "name" : "Ukraine" }
{ "_id" : "VNM", "name" : "Vietnam" }
```

Queries

```
> db.countries.find(
 {  'cities.city.name' :
    { $in : [ 'Istanbul', 'New York']}
 },
 { name: true }
)
{ "_id" : "TUR", "name" : "Turkey" }
{ "_id" : "USA", "name" : "United States" }
```

Distinct

```
> db.countries.distinct('continent')
[
        "North America",
        "Asia",
        "Africa",
        "Europe",
        "Oceania",
        "South America"
]
```

Grouping

```
> db.countries.group({
 key: { continent: 1},
 initial: { count: 0},
 reduce: function (doc,o){ o.count=o.count +1;}
});
```

Queries

```
[
        {
                "continent" : "North America",
                "count" : 37
        },
        {
                "continent" : "Asia",
                "count" : 50
        },
        {
                "continent" : "Africa",
                "count" : 57
        },
        {
                "continent" : "Europe",
                "count" : 46
        },
        . . .
]
```

Grouping

```
> db.countries.group({
   key: { name: 1},
   initial: { numberOfCities: 0},
   reduce: function (doc,o){
     o.numberOfCities =
          doc.cities.city.length }
   }
);
```

Regular Expression

```
> db.countries1.find(
 { name : /^....$/},
 { name: 1 , _id: 0}
)
{ "name" : "Cuba" }
{ "name" : "Guam" }
{ "name" : "Iran" }
{ "name" : "Laos" }
{ "name" : "Mali" }
{ "name" : "Niue" }
{ "name" : "Oman" }
{ "name" : "Peru" }
{ "name" : "Chad" }
```

Sorting

```
> db.countries.find(
{ continent : 'Asia'},
{ name: 1, population:1,_id :0})
.sort({population: -1})
.limit(10)
{ "name" : "China", "population" : 1277558000 }
{ "name" : "India", "population" : 1013662000 }
{ "name" : "Indonesia", "population" : 212107000 }
{ "name" : "Pakistan", "population" : 156483000 }
{ "name" : "Bangladesh", "population" : 129155000 }
{ "name" : "Japan", "population" : 126714000 }
{ "name" : "Vietnam", "population" : 79832000 }
{ "name" : "Philippines", "population" : 75967000 }
{ "name" : "Iran", "population" : 67702000 }
{ "name" : "Turkey", "population" : 66591000 }
```

Pagination

```
> db.countries.find(
{ continent : 'Asia'},
{ name: 1, population:1,_id :0})
.sort({population: -1})
.skip(10)
.limit(10)
```

```
{ "name" : "Thailand", "population" : 61399000 }
{ "name" : "South Korea", "population" : 46844000 }
{ "name" : "Myanmar", "population" : 45611000 }
{ "name" : "Uzbekistan", "population" : 24318000 }
{ "name" : "North Korea", "population" : 24039000 }
{ "name" : "Nepal", "population" : 23930000 }
{ "name" : "Afghanistan", "population" : 22720000 }
{ "name" : "Taiwan", "population" : 22256000 } ...
```