MondgoDB

Kayartaya Vinod

Introduction

- MongoDB is an open-source document database.
    - Hu**mongo**us DB
- Falls into the category of NOSQL databases

Introduction

- Stores the data in un-normalized format
- Data is stored as collections of documents

Document database

- A record in an RDBMS is equivalent to a document in MongoDB
  - A data structure composed of field and value pairs.
- MongoDB documents are similar to JSON objects.
  - The values of fields may include scalar data, other documents, arrays, and arrays of documents.
  - Internally, MongoDB stores these documents in the binary format, called BSON (Binary JSON)

Document

```
{
        id      : 7788,
        name    : "Vinod Kumar",
        phones  : [ "9731424784", "9844083934"],
        emails  : [
                {
                        type    : "personal",
                        address : "kayartaya.vinod@gmail.com"
                },
                {
                        type    : "official",
                        address : "vinod@knowledgeworksindia.com"
                }
        ]
}
```

A Java equivalent of the document

```
public class Email{
        private String type;
        private String address;
        // constructors
        // getters/setters
}

public class Person {
        private int id;
        private String name;
        private String[] phones;
        private List<Email> emails = new ArrayList<Email>();
        // constructors
        // getters/setters
}
```

A Java equivalent of the document

```
Person p1 = new Person();

p1.setId(7788);
p1.setName("Vinod Kumar");
p1.setPhones(new String[]{"9731424784","9844093934"});

p1.getEmails().add(
        new Email("personal", "kayartaya.vinod@gmail.com"));
p1.getEmails().add(
        new Email("official", "vinod@knowledgeworksindia.com"));
```

Document Model

- Data in MongoDB has a flexible schema
- Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB’s collections do not enforce document structure
- This flexibility facilitates the mapping of documents to an entity or an object

Document Model

- Each document can match the data fields of the represented entity, even if the data has substantial variation
- In practice, however, the documents in a collection share a similar structure
- When designing data models, always consider the application usage of the data (i.e. queries, updates, and processing of the data)

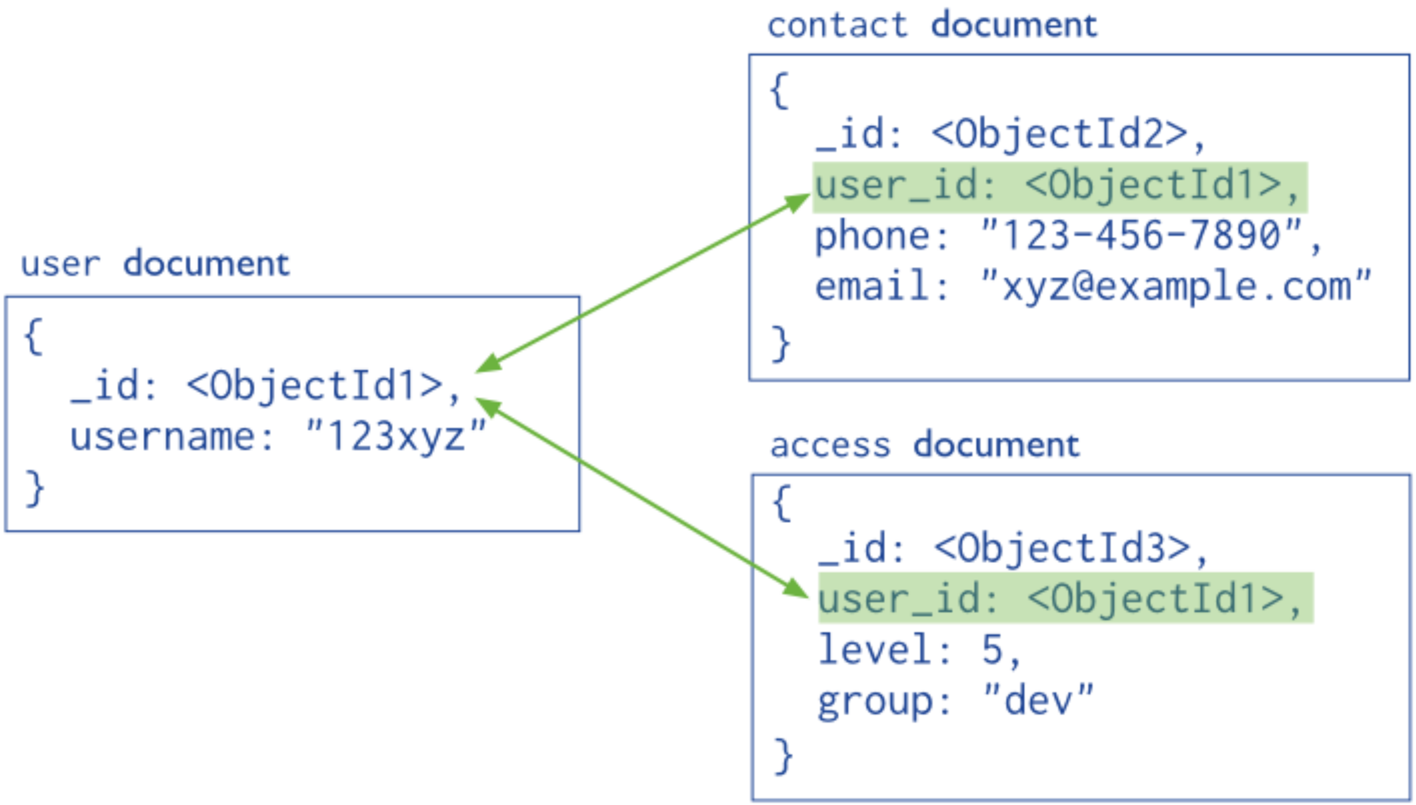
Document Model

- There are two tools that allow applications to represent these relationships:
  - references and
  - embedded documents

References

- References store the relationships between data by including links or references from one document to another
- Applications can resolve these references to access the related data
- Broadly, these are normalized data models

References



user document

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

contact document

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

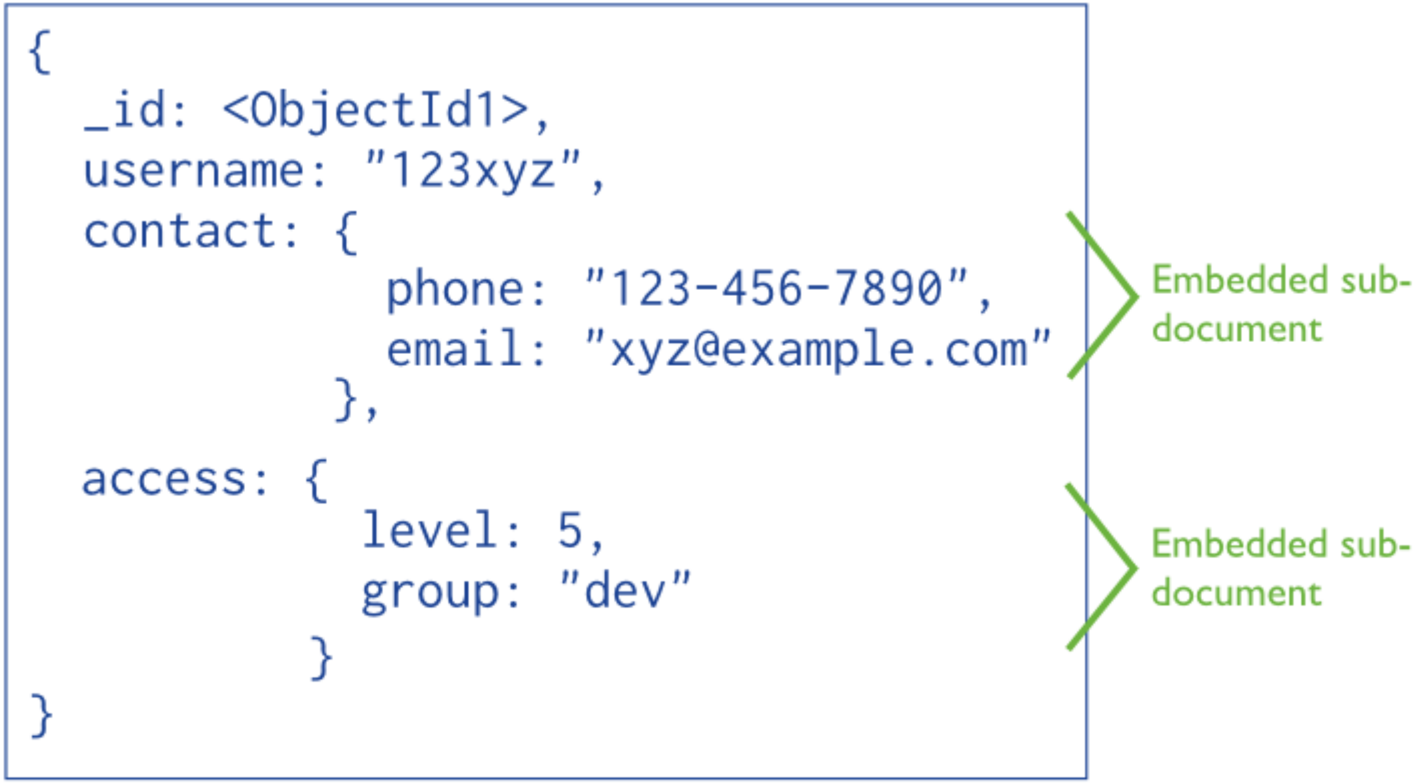
access document

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

Embedded Data

- Embedded documents capture relationships between data by storing related data in a single document structure
- MongoDB documents make it possible to embed document structures in a field or array within a document
- These denormalized data models allow applications to retrieve and manipulate related data in a single database operation

References



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

Embedded sub-document

Embedded sub-document

Atomicity of Write Operations

- In MongoDB, write operations are atomic at the document level
- No single write operation can atomically affect more than one document or more than one collection
- A denormalized data model with embedded data combines all related data for a represented entity in a single document

Atomicity of Write Operations

- This facilitates atomic write operations since a single write operation can insert or update the data for an entity
- Normalizing the data would split the data across multiple collections and would require multiple write operations that are not atomic collectively

Advantages of Mongodb

- Documents correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism.

High performance

- Support for embedded data models reduces I/O activity on database system.
- Indexes support faster queries and can include keys from embedded documents and arrays.

High availability

- MongoDB’s replication facility, called replica sets, provide:
  - automatic failover.
  - data redundancy.
- A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

Automatic scaling

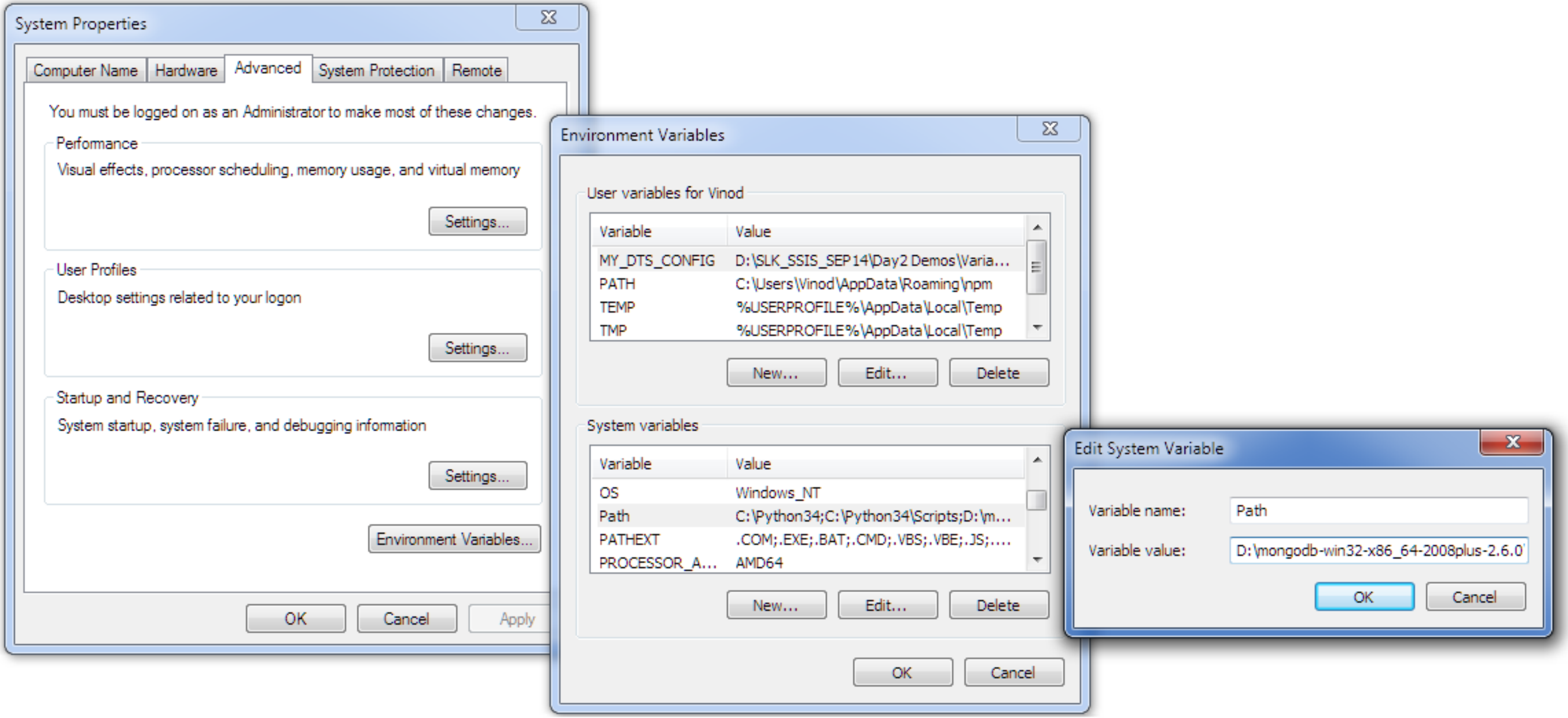
- MongoDB provides horizontal scalability as part of its core functionality.
  - Automatic sharding distributes data across a cluster of machines.
  - Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

Installation/setup

- Download the binary for your operating system
  - For Windows 7 64bit:
    - https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-2.6.0.zip
  - All Windows downloads:
    - https://www.mongodb.org/dl/win32/x86_64
  - All Linux downloads:
    - https://www.mongodb.org/dl/linux
- Unzip to a drive
  - In my computer:
    - D:\mongodb-win32-x86_64-2008plus-2.6.0

Installation/setup

- Add the D:\mongodb-win32-x86_64-2008plus-2.6.0\bin to your PATH variable



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your logon

Startup and Recovery

System startup, system failure, and debugging information

User variables for Vinod

Variable	Value
MY_DTS_CONFIG	D:\SLK_SSIS_SEP14\Day2 Demos\Varia...
PATH	C:\Users\Vinod\AppData\Roaming\npm
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
OS	Windows_NT
Path	C:\Python34;C:\Python34\Scripts;D:\m...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Variable name: Path

Variable value: D:\mongodb-win32-x86_64-2008plus-2.6.0

Default DB location

- MongoDB requires a data directory to store all data.
- MongoDB’s default data directory path is \data\db.
  - You can create this folder structure in the same drive as mongodb’s installation drive
  - Or specify another location when starting the server

Starting the server

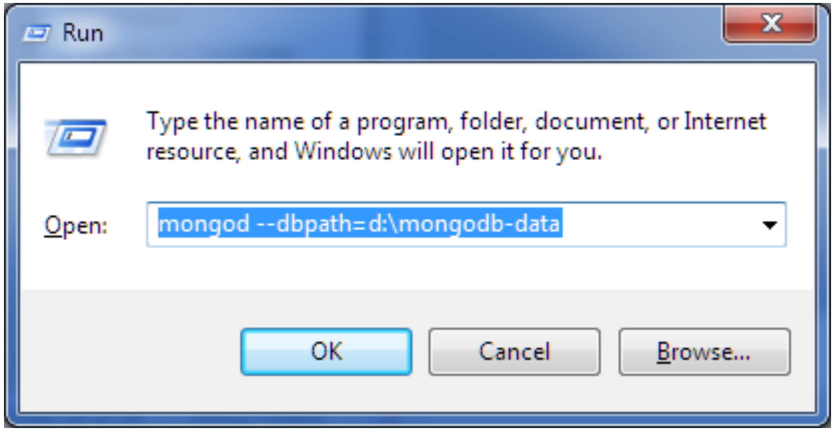
- The server can be started by running the mongod.exe executable
- Some of the useful options are:

  ```
  --dbpath=PATH-TO-YOUR-DB
  --port=27017
  ```

- Example:

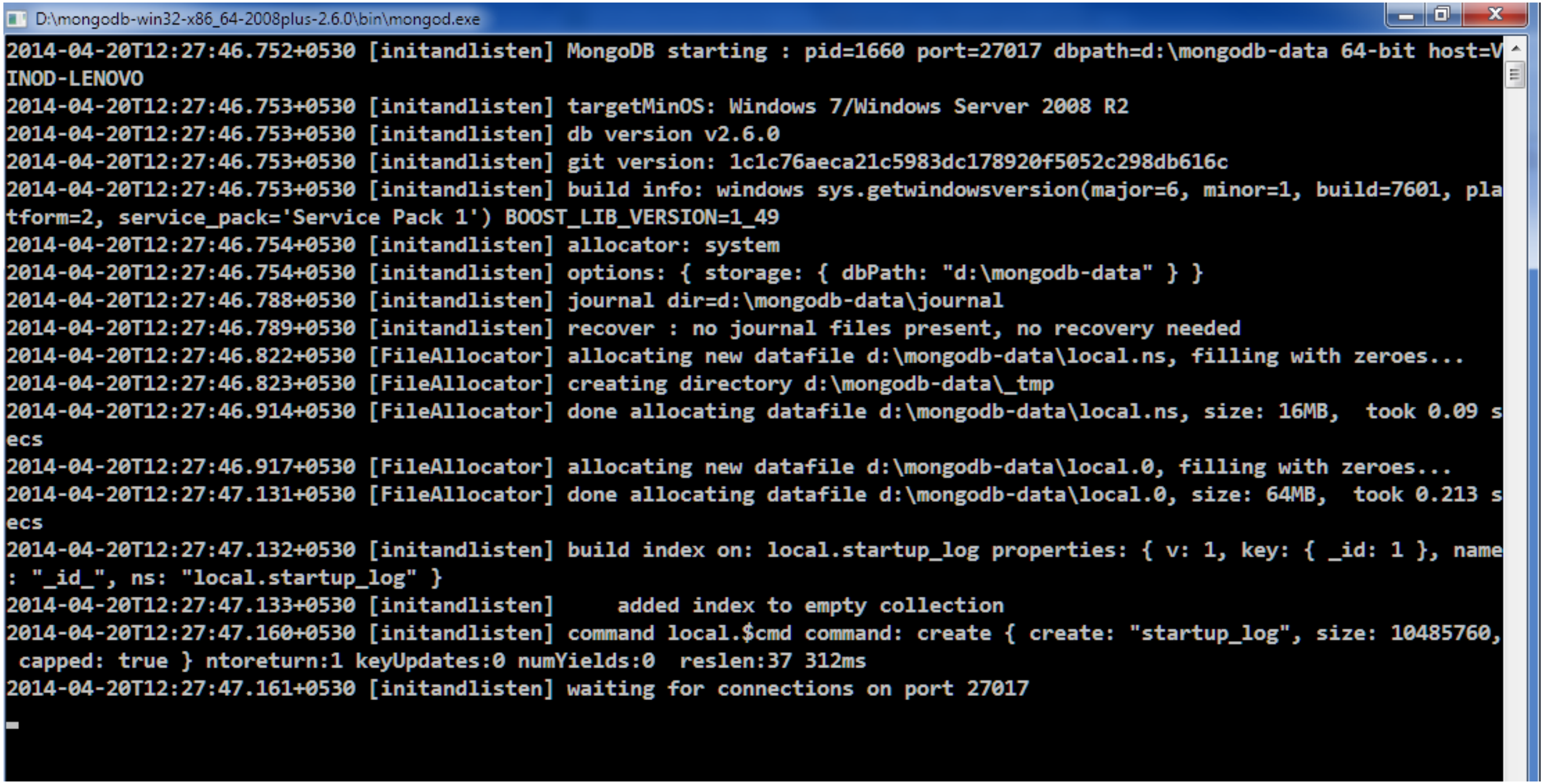
  ```
  mongod --dbpath=d:\mongodb-data
  ```

  (Note: you must create the folder manually before running this command)



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: mongod --dbpath=d:\mongodb-data

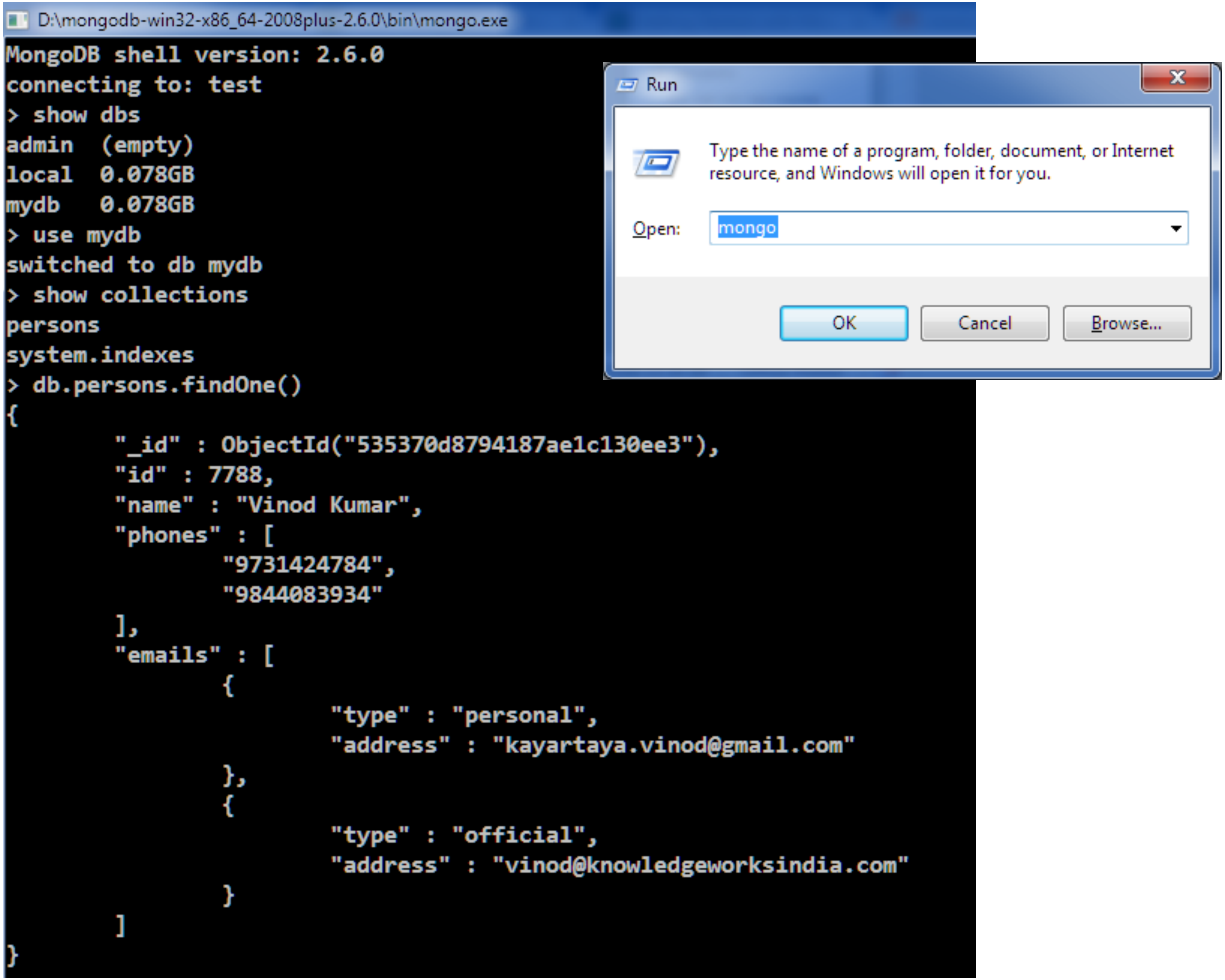


```
2014-04-20T12:27:46.752+0530 [initandlisten] MongoDB starting : pid=1660 port=27017 dbpath=d:\mongodb-data 64-bit host=V
INOD-LENOVO
2014-04-20T12:27:46.753+0530 [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2014-04-20T12:27:46.753+0530 [initandlisten] db version v2.6.0
2014-04-20T12:27:46.753+0530 [initandlisten] git version: 1c1c76aeca21c5983dc178920f5052c298db616c
2014-04-20T12:27:46.753+0530 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, pla
tform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2014-04-20T12:27:46.754+0530 [initandlisten] allocator: system
2014-04-20T12:27:46.754+0530 [initandlisten] options: { storage: { dbPath: "d:\mongodb-data" } }
2014-04-20T12:27:46.788+0530 [initandlisten] journal dir=d:\mongodb-data\journal
2014-04-20T12:27:46.789+0530 [initandlisten] recover : no journal files present, no recovery needed
2014-04-20T12:27:46.822+0530 [FileAllocator] allocating new datafile d:\mongodb-data\local.ns, filling with zeroes...
2014-04-20T12:27:46.823+0530 [FileAllocator] creating directory d:\mongodb-data\_tmp
2014-04-20T12:27:46.914+0530 [FileAllocator] done allocating datafile d:\mongodb-data\local.ns, size: 16MB,  took 0.09 s
ecs
2014-04-20T12:27:46.917+0530 [FileAllocator] allocating new datafile d:\mongodb-data\local.0, filling with zeroes...
2014-04-20T12:27:47.131+0530 [FileAllocator] done allocating datafile d:\mongodb-data\local.0, size: 64MB,  took 0.213 s
ecs
2014-04-20T12:27:47.132+0530 [initandlisten] build index on: local.startup_log properties: { v: 1, key: { _id: 1 }, name
: "_id_", ns: "local.startup_log" }
2014-04-20T12:27:47.133+0530 [initandlisten]     added index to empty collection
2014-04-20T12:27:47.160+0530 [initandlisten] command local.$cmd command: create { create: "startup_log", size: 10485760,
 capped: true } ntoreturn:1 keyUpdates:0 numYields:0  reslen:37 312ms
2014-04-20T12:27:47.161+0530 [initandlisten] waiting for connections on port 27017
```

JavaScript Shell

- The executable “mongo.exe” provides an interface to issue direct commands on the db.
- By default the “mongo” command tries to connect to “localhost” and port “27017”
- You can connect to different ones using --host and --port options:

```
mongo --port 12345 --host vinod_homepc
```



D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongo.exe

```
MongoDB shell version: 2.6.0
connecting to: test
> show dbs
admin  (empty)
local  0.078GB
mydb   0.078GB
> use mydb
switched to db mydb
> show collections
persons
system.indexes
> db.persons.findOne()
{
        "_id" : ObjectId("535370d8794187ae1c130ee3"),
        "id" : 7788,
        "name" : "Vinod Kumar",
        "phones" : [
                "9731424784",
                "9844083934"
        ],
        "emails" : [
                {
                        "type" : "personal",
                        "address" : "kayartaya.vinod@gmail.com"
                },
                {
                        "type" : "official",
                        "address" : "vinod@knowledgeworksindia.com"
                }
        ]
}
```

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: mongo

Some commands to start with..

- show dbs
  - displays the list of databases
- use mydb
  - switches to the database “mydb” if exists, or creates a new with the same name and switches to it
- db
  - displays the current database in use
- db.dropDatabase()
  - Deletes the current database

Importing external data

- Use the mongoimport.exe tool to import external data into a database

```
mongoimport
        --host localhost
        --port 27017
        --db mydb
        --jsonArray
        --collection orders
        --file d:\orders.json
```

Some commands to start with..

- db.<collection>.findOne()
  - Displays the first document in the collection

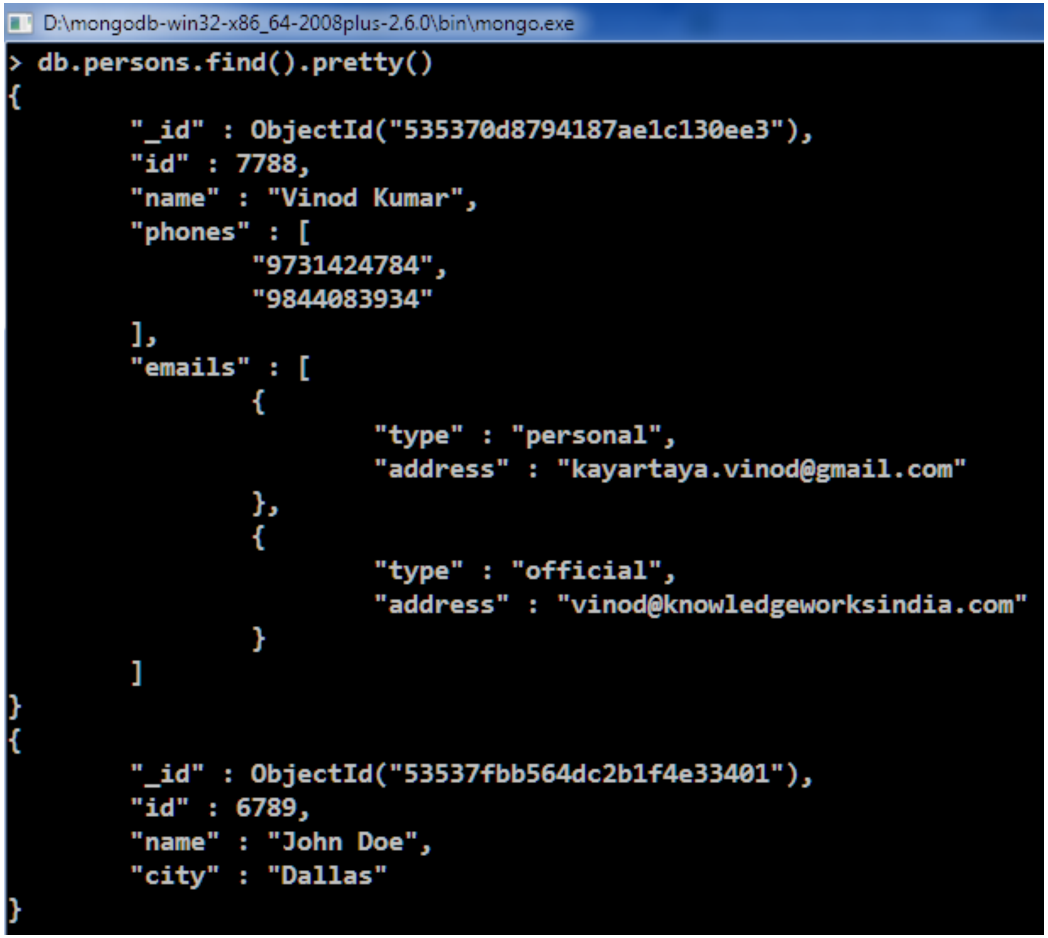
- db.<collection>.find().pretty()
  - displays the first 20 documents in an indented format

Some commands to start with..

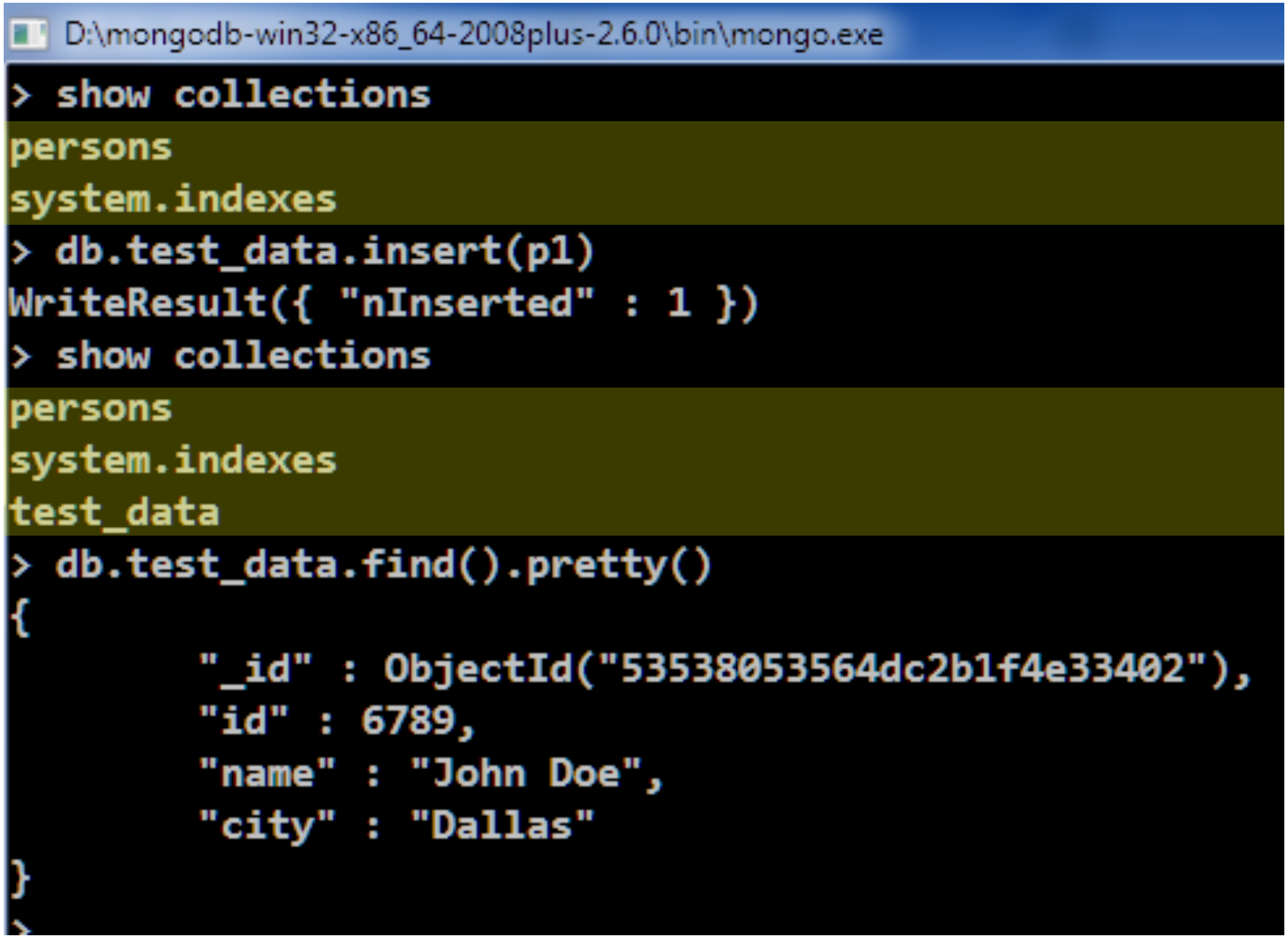
- show collections
  - Displays the list of collections (tables in RDBMS)
- db.<collection>.insert(data)
  - If the collection exists, inserts the data, else creates a new collection with the same name and inserts the data

```
p1 = {
        id      : 6789,
        name    : "John Doe",
        city    : "Dallas"
}

db.persons.insert(p1);
```



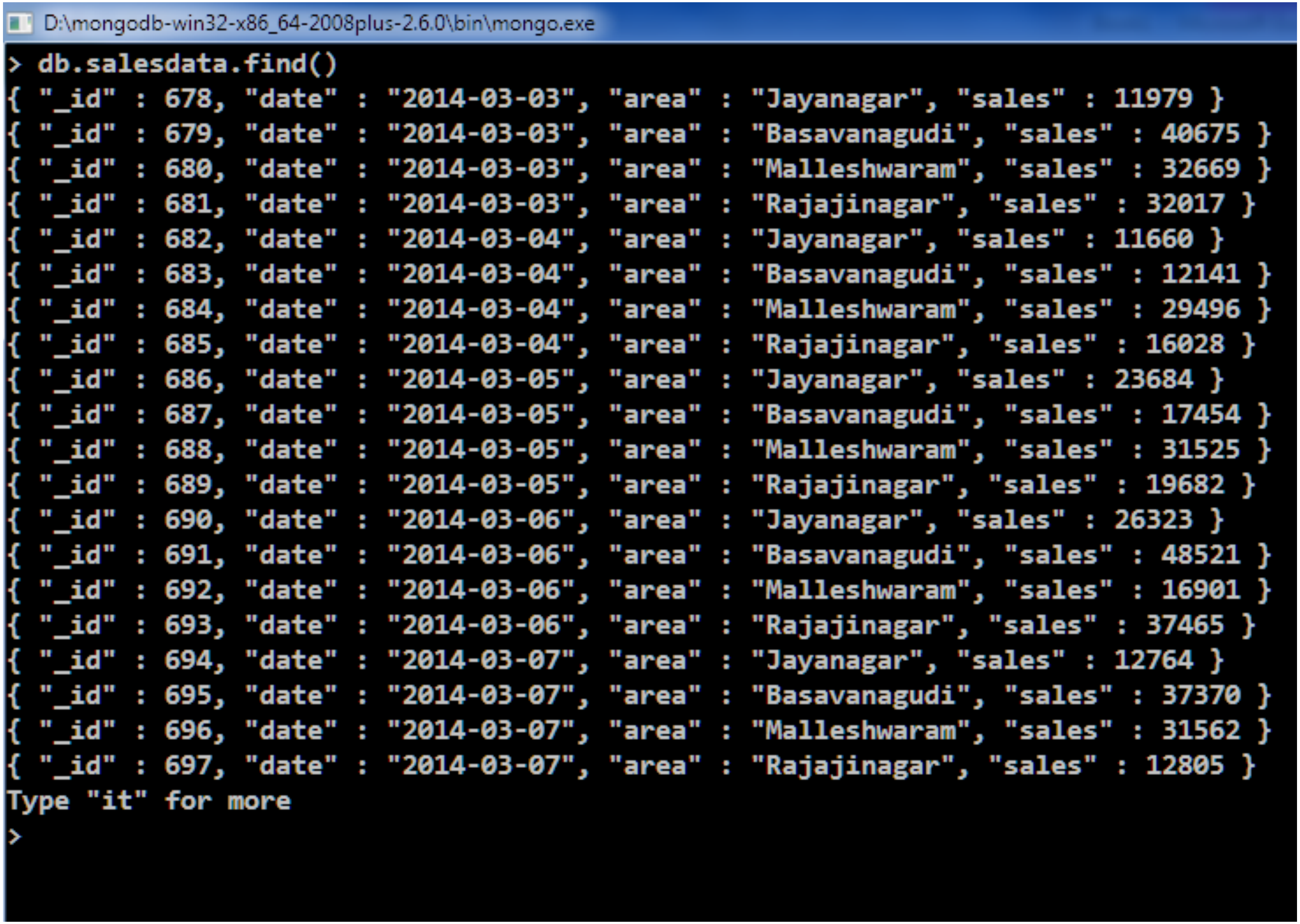
```
> db.persons.find().pretty()
{
        "_id" : ObjectId("535370d8794187ae1c130ee3"),
        "id" : 7788,
        "name" : "Vinod Kumar",
        "phones" : [
                "9731424784",
                "9844083934"
        ],
        "emails" : [
                {
                        "type" : "personal",
                        "address" : "kayartaya.vinod@gmail.com"
                },
                {
                        "type" : "official",
                        "address" : "vinod@knowledgeworksindia.com"
                }
        ]
}
{
        "_id" : ObjectId("53537fbb564dc2b1f4e33401"),
        "id" : 6789,
        "name" : "John Doe",
        "city" : "Dallas"
}
```



```
> show collections
persons
system.indexes
> db.test_data.insert(p1)
WriteResult({ "nInserted" : 1 })
> show collections
persons
system.indexes
test_data
> db.test_data.find().pretty()
{
        "_id" : ObjectId("53538053564dc2b1f4e33402"),
        "id" : 6789,
        "name" : "John Doe",
        "city" : "Dallas"
}
>
```

Some commands to start with..

- db.<collection>.find()
  - Returns a cursor to the result
  - Displays the first 20 documents on the screen
  - type "it" to iterate again and get 20 more documents



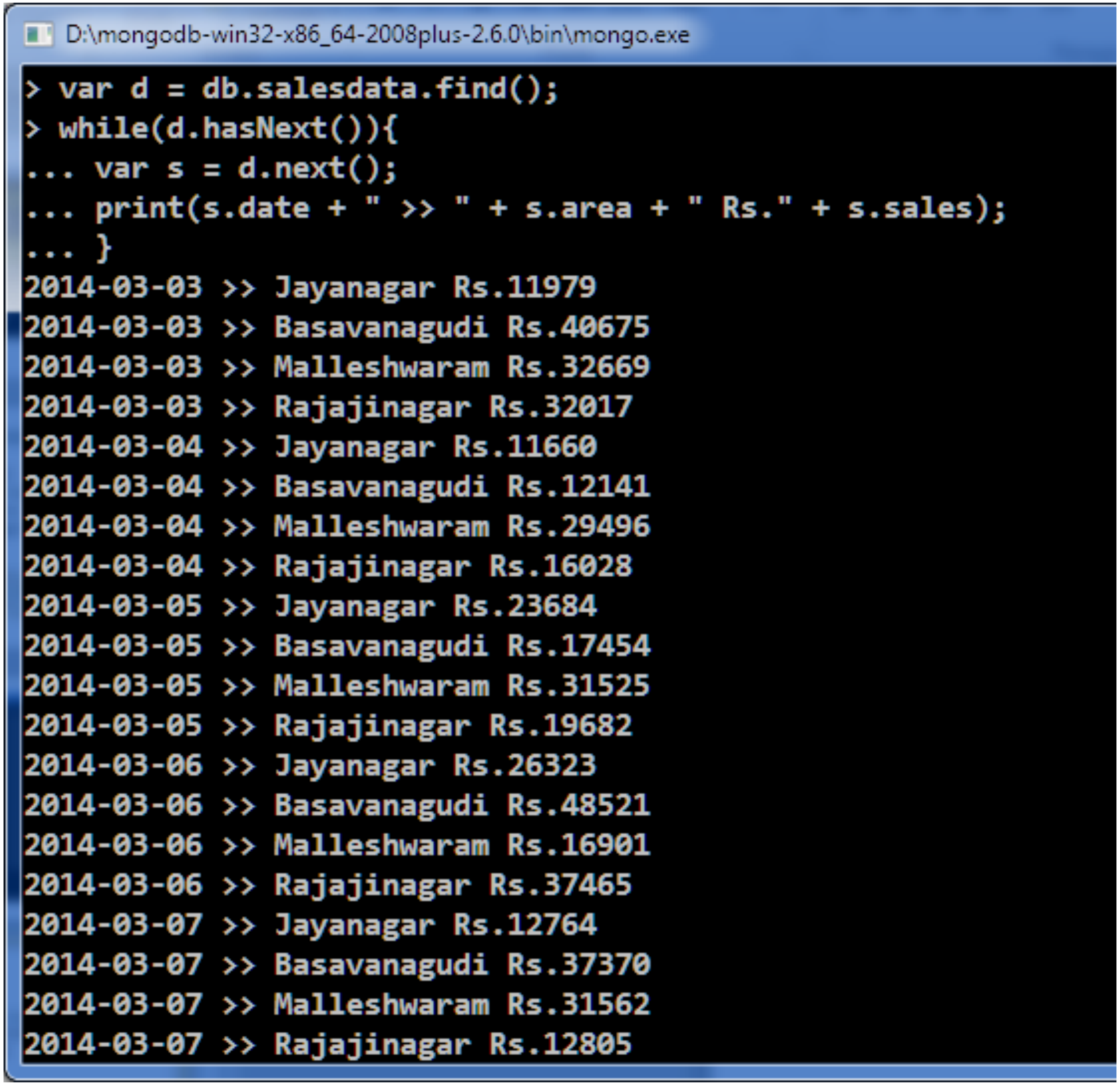
```
> db.salesdata.find()
{ "_id" : 678, "date" : "2014-03-03", "area" : "Jayanagar", "sales" : 11979 }
{ "_id" : 679, "date" : "2014-03-03", "area" : "Basavanagudi", "sales" : 40675 }
{ "_id" : 680, "date" : "2014-03-03", "area" : "Malleshwaram", "sales" : 32669 }
{ "_id" : 681, "date" : "2014-03-03", "area" : "Rajajinagar", "sales" : 32017 }
{ "_id" : 682, "date" : "2014-03-04", "area" : "Jayanagar", "sales" : 11660 }
{ "_id" : 683, "date" : "2014-03-04", "area" : "Basavanagudi", "sales" : 12141 }
{ "_id" : 684, "date" : "2014-03-04", "area" : "Malleshwaram", "sales" : 29496 }
{ "_id" : 685, "date" : "2014-03-04", "area" : "Rajajinagar", "sales" : 16028 }
{ "_id" : 686, "date" : "2014-03-05", "area" : "Jayanagar", "sales" : 23684 }
{ "_id" : 687, "date" : "2014-03-05", "area" : "Basavanagudi", "sales" : 17454 }
{ "_id" : 688, "date" : "2014-03-05", "area" : "Malleshwaram", "sales" : 31525 }
{ "_id" : 689, "date" : "2014-03-05", "area" : "Rajajinagar", "sales" : 19682 }
{ "_id" : 690, "date" : "2014-03-06", "area" : "Jayanagar", "sales" : 26323 }
{ "_id" : 691, "date" : "2014-03-06", "area" : "Basavanagudi", "sales" : 48521 }
{ "_id" : 692, "date" : "2014-03-06", "area" : "Malleshwaram", "sales" : 16901 }
{ "_id" : 693, "date" : "2014-03-06", "area" : "Rajajinagar", "sales" : 37465 }
{ "_id" : 694, "date" : "2014-03-07", "area" : "Jayanagar", "sales" : 12764 }
{ "_id" : 695, "date" : "2014-03-07", "area" : "Basavanagudi", "sales" : 37370 }
{ "_id" : 696, "date" : "2014-03-07", "area" : "Malleshwaram", "sales" : 31562 }
{ "_id" : 697, "date" : "2014-03-07", "area" : "Rajajinagar", "sales" : 12805 }
Type "it" for more
>
```

Some commands to start with..

- Since “mongo.exe” is a JavaScript shell, you can use a script to process the cursor returned by the find() method

```
var d = db.salesdata.find(); // returns the cursor

// now loop through the cursor to get one document at a time
while(d.hasNext()){
        var s = d.next();
        print(s.date + " >> " + s.area + " Rs." + s.sales);
}
```



```
> var d = db.salesdata.find();
> while(d.hasNext()){
... var s = d.next();
... print(s.date + " >> " + s.area + " Rs." + s.sales);
... }
2014-03-03 >> Jayanagar Rs.11979
2014-03-03 >> Basavanagudi Rs.40675
2014-03-03 >> Malleshwaram Rs.32669
2014-03-03 >> Rajajinagar Rs.32017
2014-03-04 >> Jayanagar Rs.11660
2014-03-04 >> Basavanagudi Rs.12141
2014-03-04 >> Malleshwaram Rs.29496
2014-03-04 >> Rajajinagar Rs.16028
2014-03-05 >> Jayanagar Rs.23684
2014-03-05 >> Basavanagudi Rs.17454
2014-03-05 >> Malleshwaram Rs.31525
2014-03-05 >> Rajajinagar Rs.19682
2014-03-06 >> Jayanagar Rs.26323
2014-03-06 >> Basavanagudi Rs.48521
2014-03-06 >> Malleshwaram Rs.16901
2014-03-06 >> Rajajinagar Rs.37465
2014-03-07 >> Jayanagar Rs.12764
2014-03-07 >> Basavanagudi Rs.37370
2014-03-07 >> Malleshwaram Rs.31562
2014-03-07 >> Rajajinagar Rs.12805
```

Some commands to start with..

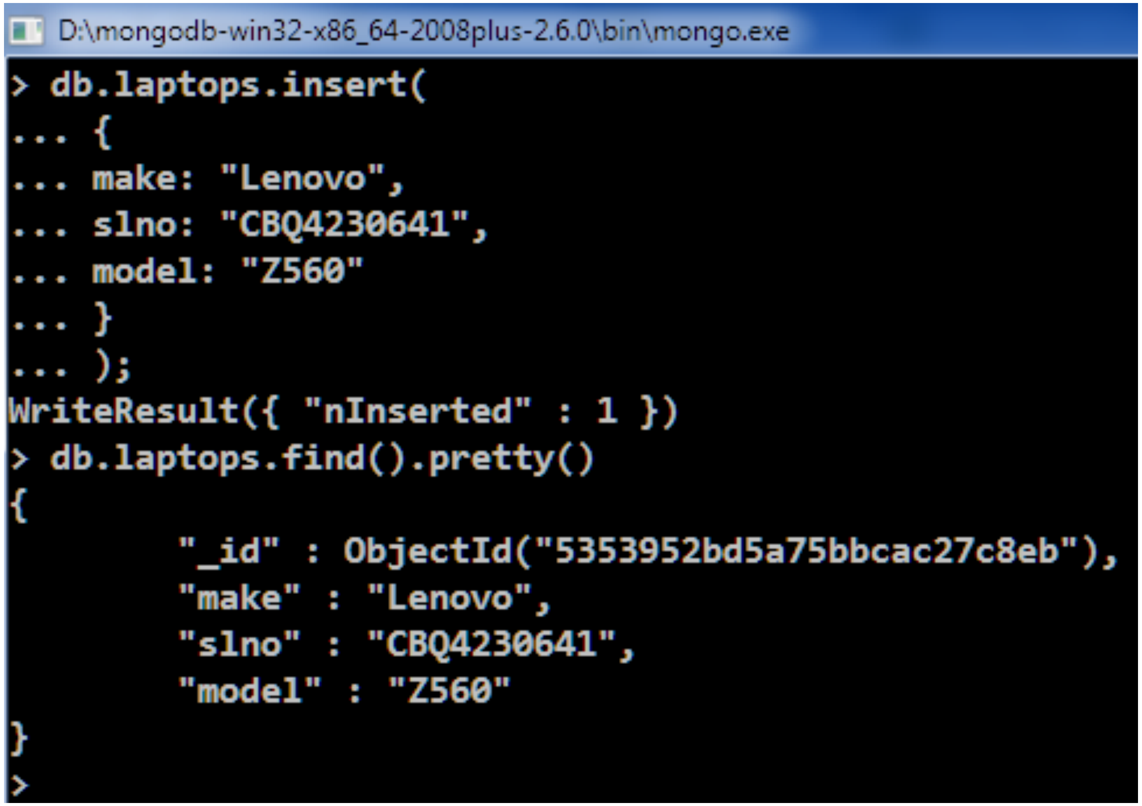
- You can use array operator on a cursor returned by find() method

```
var cur = db.salesdata.find();
var sales1 = cur[3];  // 4th element
var arr = cur.toArray(); // loads all data to RAM
```

Performing insert/update

- A new document can be added to a collection using the following methods:
  - db.<collection>.insert(doc)
  - db.<collection>.update(doc, {upsert: true})
  - db.<collection>.save(doc)
- MongoDB does not support transactions
  - Once data is inserted/modified/deleted, it is reflected to all the clients
    - No concept of commit, rollback or savepoints
  - Client applications (such as Java apps) can make use of external transaction managers.

Performing insert/update



D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongo.exe

```
> db.laptops.insert(
... {
... make: "Lenovo",
... slno: "CBQ4230641",
... model: "Z560"
... }
... );
WriteResult({ "nInserted" : 1 })
> db.laptops.find().pretty()
{
        "_id" : ObjectId("5353952bd5a75bbcac27c8eb"),
        "make" : "Lenovo",
        "slno" : "CBQ4230641",
        "model" : "Z560"
}
>
```

Performing insert/update

- If you add a new document without the _id field, the client library or the mongod instance adds an _id field and populates the field with a unique ObjectId.
  - A special 12-byte BSON type that guarantees uniqueness within the collection.
  - The ObjectId is generated based on timestamp, machine ID, process ID, and a process-local incremental counter.
  - The _id field is immutable

Performing insert/update

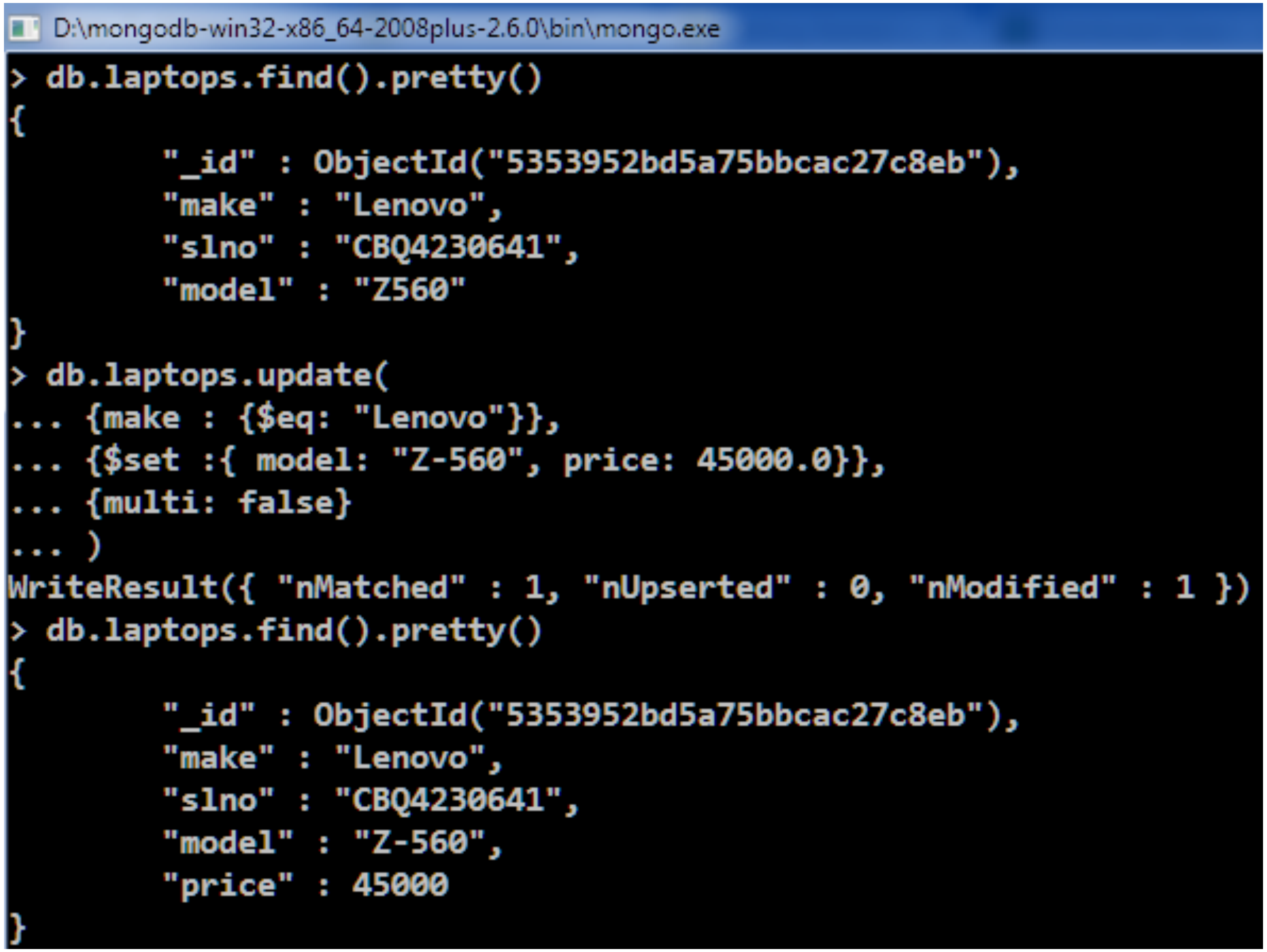
- An existing document can be modified using the update() or save() methods

```
db.<collection>.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>
   }
)
```

Update explained

Parameter	Type	Description
query	document	The selection criteria for the update.
update	document	The modifications to apply.
upsert	boolean	Optional. If set to true, creates a new document when no document matches the query criteria. The default value is false, which does *not* insert a new document when no match is found.
multi	boolean	Optional. If set to true, updates multiple documents that meet thequery criteria. If set to false, updates one document. The default value is false.

Performing insert/update



D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongo.exe

```
> db.laptops.find().pretty()
{
        "_id" : ObjectId("5353952bd5a75bbcac27c8eb"),
        "make" : "Lenovo",
        "slno" : "CBQ4230641",
        "model" : "Z560"
}
> db.laptops.update(
... {make : {$eq: "Lenovo"}},
... {$set :{ model: "Z-560", price: 45000.0}},
... {multi: false}
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.laptops.find().pretty()
{
        "_id" : ObjectId("5353952bd5a75bbcac27c8eb"),
        "make" : "Lenovo",
        "slno" : "CBQ4230641",
        "model" : "Z-560",
        "price" : 45000
}
```

Update operators

Name	Description
$inc	Increments the value of the field by the specified amount.
$mul	Multiplies the value of the field by the specified amount.
$rename	Renames a field.
$setOnInsert	Sets the value of a field upon document creation during an upsert. Has no effect on update operations that modify existing documents.
$set	Sets the value of a field in an existing document.
$unset	Removes the specified field from an existing document.
$min	Only updates if the existing field value is less than the specified value.
$max	Only updates if the existing field value is greater than the specified value.
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp.

Update examples

```
db.laptops.update(
        {make: {$eq: "Lenovo"}},
        {
                $currentDate:{ dop: true },
                $set: { price: 46500 }
        }
)
```

- Adds a new property “dop” with the current date/time as the value
- Changes the value of the property “price” to 46500

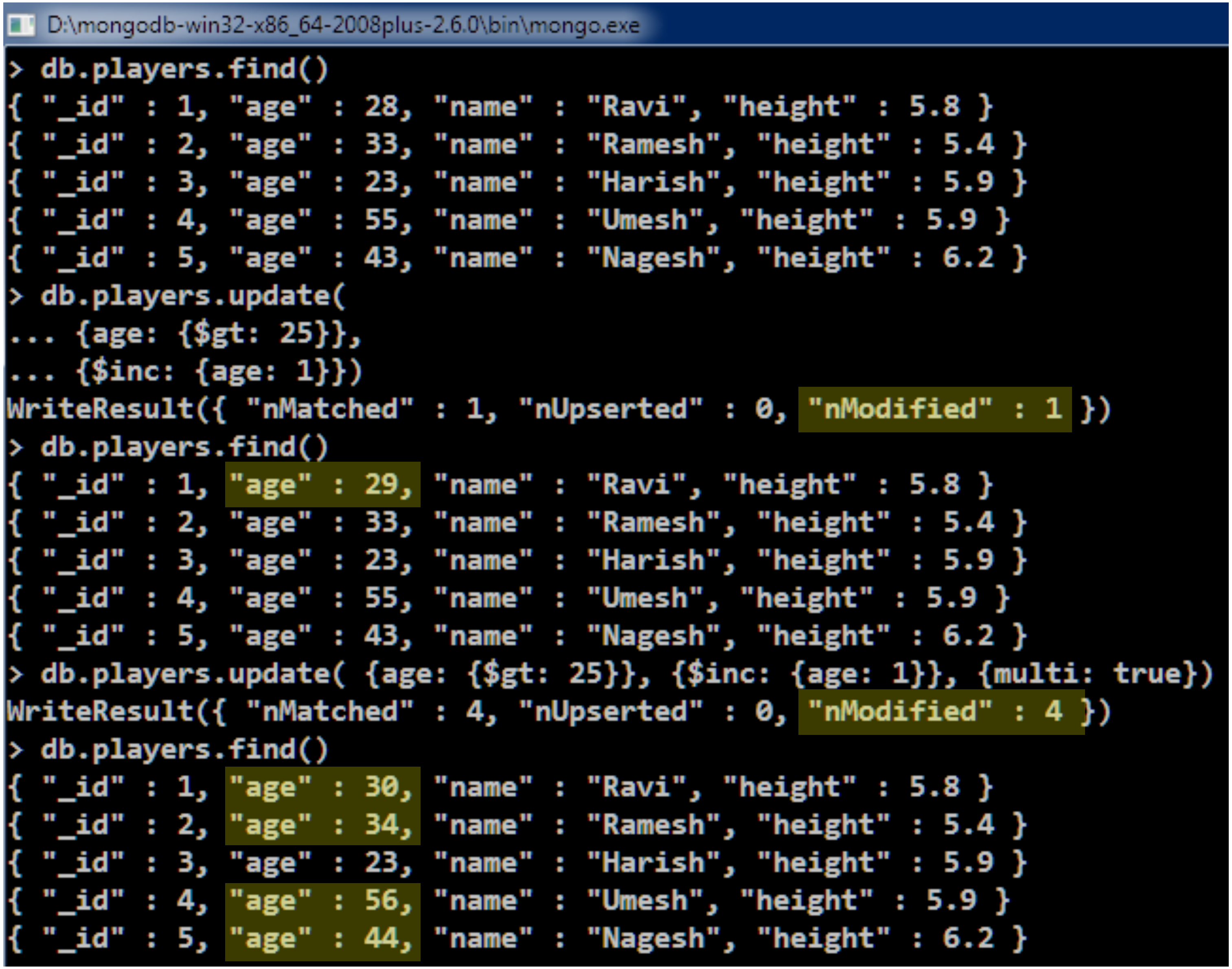
Update examples

```
db.laptops.update(
        {make: {$eq: "Lenovo"}},
        {$inc: { price: 1500 }}
)
```

- Increments the “price” by 1500 for the first matched document

```
db.laptops.update(
        {make: {$eq: "Lenovo"}},
        {$inc: { price: 1500 }},
        {multi: true}
)
```

- Increments the “price” by 1500 for all the matched documents



D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongo.exe

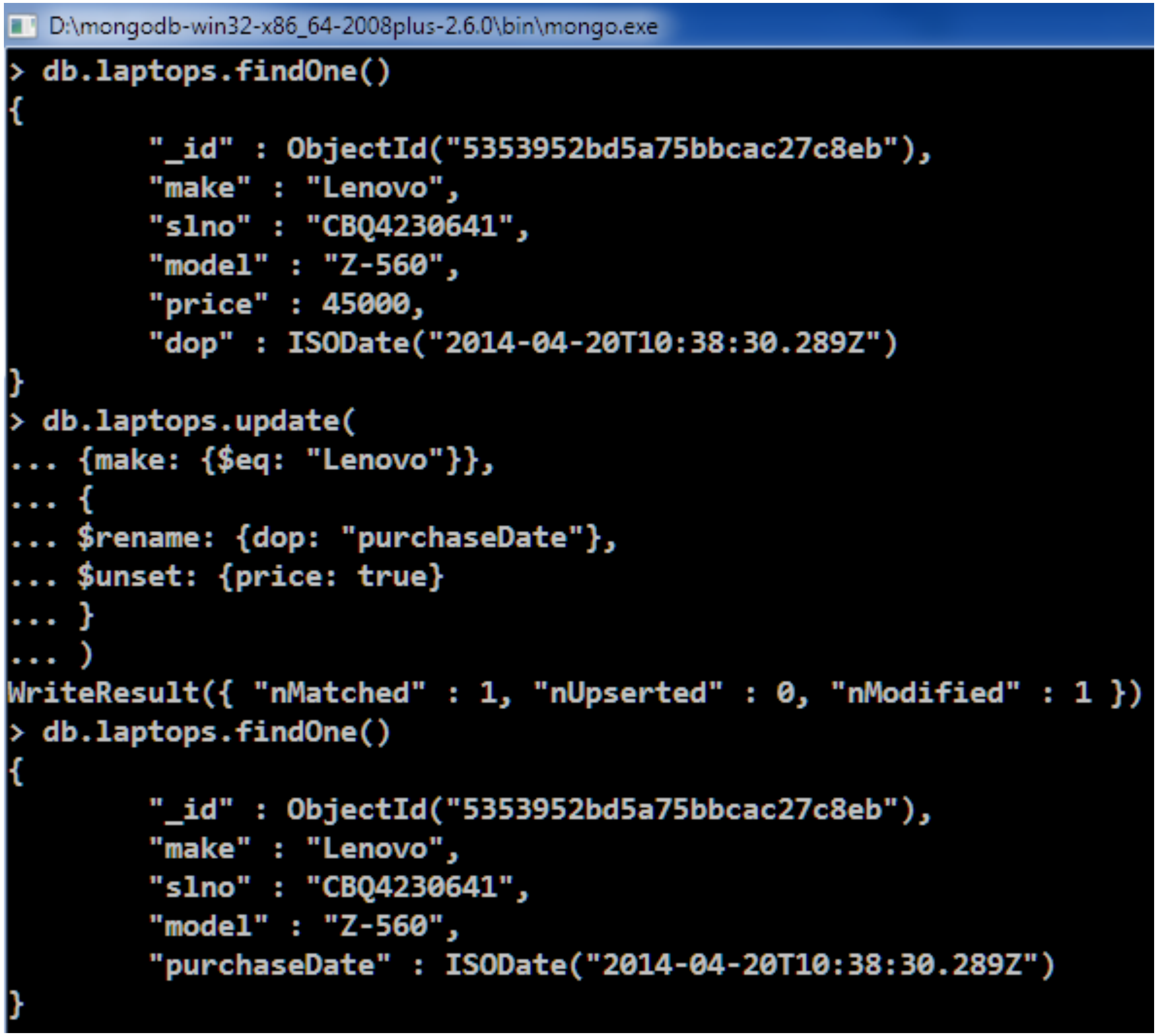
```
> db.players.find()
{ "_id" : 1, "age" : 28, "name" : "Ravi", "height" : 5.8 }
{ "_id" : 2, "age" : 33, "name" : "Ramesh", "height" : 5.4 }
{ "_id" : 3, "age" : 23, "name" : "Harish", "height" : 5.9 }
{ "_id" : 4, "age" : 55, "name" : "Umesh", "height" : 5.9 }
{ "_id" : 5, "age" : 43, "name" : "Nagesh", "height" : 6.2 }
> db.players.update(
... {age: {$gt: 25}},
... {$inc: {age: 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.players.find()
{ "_id" : 1, "age" : 29, "name" : "Ravi", "height" : 5.8 }
{ "_id" : 2, "age" : 33, "name" : "Ramesh", "height" : 5.4 }
{ "_id" : 3, "age" : 23, "name" : "Harish", "height" : 5.9 }
{ "_id" : 4, "age" : 55, "name" : "Umesh", "height" : 5.9 }
{ "_id" : 5, "age" : 43, "name" : "Nagesh", "height" : 6.2 }
> db.players.update( {age: {$gt: 25}}, {$inc: {age: 1}}, {multi: true})
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })
> db.players.find()
{ "_id" : 1, "age" : 30, "name" : "Ravi", "height" : 5.8 }
{ "_id" : 2, "age" : 34, "name" : "Ramesh", "height" : 5.4 }
{ "_id" : 3, "age" : 23, "name" : "Harish", "height" : 5.9 }
{ "_id" : 4, "age" : 56, "name" : "Umesh", "height" : 5.9 }
{ "_id" : 5, "age" : 44, "name" : "Nagesh", "height" : 6.2 }
```

Update examples

```
db.laptops.update(
        {make: {$eq: "Lenovo"}},
        {
                $rename: {dop: "purchaseDate"},
                $unset: {price: true}
        }
)
```

- Renames the property “dop” to “purchaseDate”
- Removes the property “price” from the document

Update examples



D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongo.exe

```
> db.laptops.findOne()
{
        "_id" : ObjectId("5353952bd5a75bbcac27c8eb"),
        "make" : "Lenovo",
        "slno" : "CBQ4230641",
        "model" : "Z-560",
        "price" : 45000,
        "dop" : ISODate("2014-04-20T10:38:30.289Z")
}
> db.laptops.update(
... {make: {$eq: "Lenovo"}},
... {
... $rename: {dop: "purchaseDate"},
... $unset: {price: true}
... }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.laptops.findOne()
{
        "_id" : ObjectId("5353952bd5a75bbcac27c8eb"),
        "make" : "Lenovo",
        "slno" : "CBQ4230641",
        "model" : "Z-560",
        "purchaseDate" : ISODate("2014-04-20T10:38:30.289Z")
}
```

Update examples

```
db.laptops.update(
        {make: {$eq: "Apple"}},
        {
                $set : {
                        slno: "ZXE093745",
                        model: "MacBook Pro",
                        price: 76500
                }
        }
)
```

Only updates the document. If the document does not exist, then no new document is created.

```
db.laptops.update(
        {make: {$eq: "Apple"}},
        {
                $set : {
                        slno: "ZXE093745",
                        model: "MacBook Pro",
                        price: 76500
                }
        },
        { upsert: true}
)
```

Tries to update the document.

Effect of “upsert: true” → If the document does not exist, then a new document is created.

Deleting a document

- To delete a document from a collection, use a delete criteria and issue the following command:

  ```
  db.<collection>.remove(
      <query>,
      <justOne>
  )
  ```

Deleting a document

```
db.laptops.remove(
        { make: "Lenovo"},
        { justOne: true }
)
```

- **Removes the first matched document**

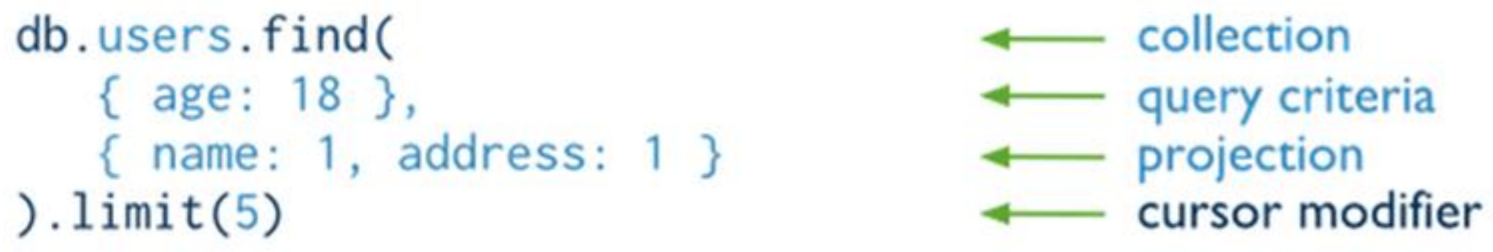
```
db.laptops.remove(
        { make: "Lenovo"}
)
```

- **Removes all the matched documents**

Finding documents

- For query operations, MongoDB provide a db.collection.find() method.
- The method accepts both the query criteria and projections and returns a cursor to the matching documents.
- You can optionally modify the query to impose limits, skips, and sort orders.

MongoDB query operation



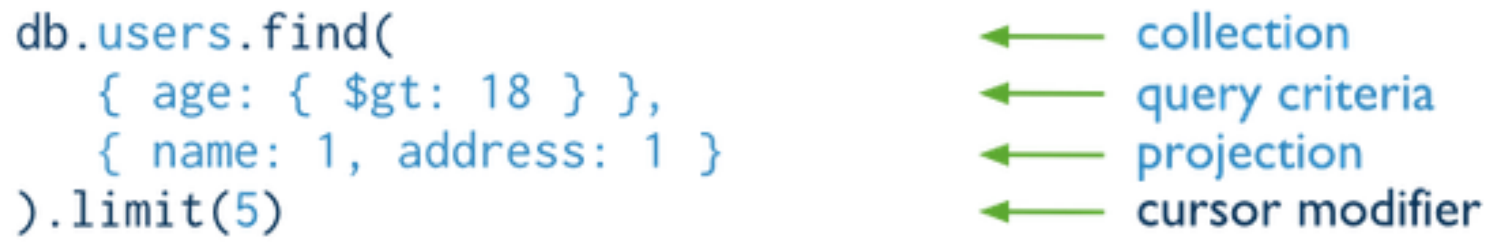
```
db.users.find(
   { age: 18 },
   { name: 1, address: 1 }
).limit(5)
```

collection

query criteria

projection

cursor modifier



```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

collection

query criteria

projection

cursor modifier

MongoDB query operation

- All queries in MongoDB address a single collection
- You can modify the query to impose limits, skips, and sort orders
- The order of documents returned by a query is not defined unless you specify a sort()
- MongoDB update/remove methods use the same query syntax

Ordering the query results

- Use the sort() method to achieve it

```
db.orders.find({“customer.customer_id”: "ANTON"})
         .sort({order_date: 1})
```

Query operators

Name	Description
$gt	Matches values that are greater than the value specified in the query.
$gte	Matches values that are equal to or greater than the value specified in the query.
$in	Matches any of the values that exist in an array specified in the query.
$lt	Matches values that are less than the value specified in the query.
$lte	Matches values that are less than or equal to the value specified in the query.
$ne	Matches all values that are not equal to the value specified in the query.
$nin	Matches values that **do not** exist in an array specified to the query.

More operators

Name	Description
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$exists	Matches documents that have the specified field.
$type	Selects documents if a field is of the specified type.

Using $where operator

```
// Get count of orders having
// more than 5 products

db.orders.find({
        $where: "this.products.length > 5"
}).count();
```

Using $exists operator

```
// Get count of orders having
// more than 5 products

db.orders.find({
        "products.5": {$exists: 1}
}).count();
```

Projection

```
db.orders.find(
     {customer.city: "London"},
     {
          _id: 0,
          order_id: 1,
          order_date: 1,
          customer.customer_name: 1,
          employee.name: 1
     });
```

Grouping

- Can be done in several ways
  - aggregate function
  - mapReduce function

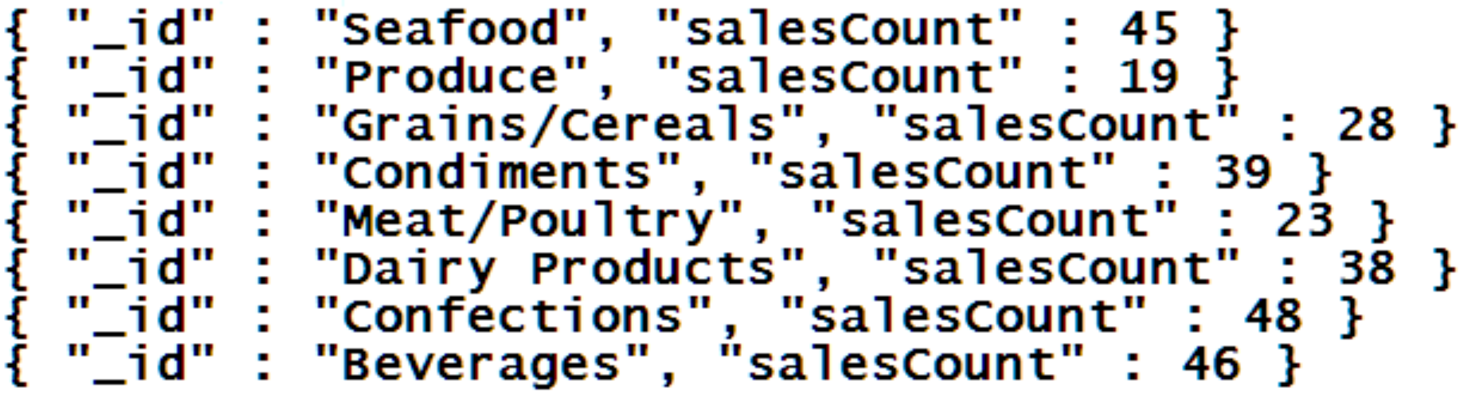
Using aggregate function

- Syntax:

```
db.collection.aggregate(
    {GROUP_OPTIONS, HAVING_OPTIONS})
```

Examples

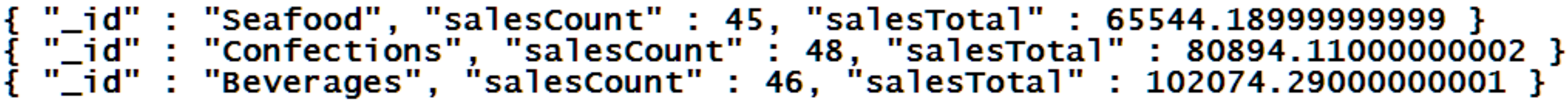
```
db.sales.aggregate(
{
            $group: {
                        _id: "$category",
                        salesCount: {$sum: 1}
            }
});
```



```
{ "_id" : "Seafood", "salesCount" : 45 }
{ "_id" : "Produce", "salesCount" : 19 }
{ "_id" : "Grains/Cereals", "salesCount" : 28 }
{ "_id" : "Condiments", "salesCount" : 39 }
{ "_id" : "Meat/Poultry", "salesCount" : 23 }
{ "_id" : "Dairy Products", "salesCount" : 38 }
{ "_id" : "Confections", "salesCount" : 48 }
{ "_id" : "Beverages", "salesCount" : 46 }
```

Examples

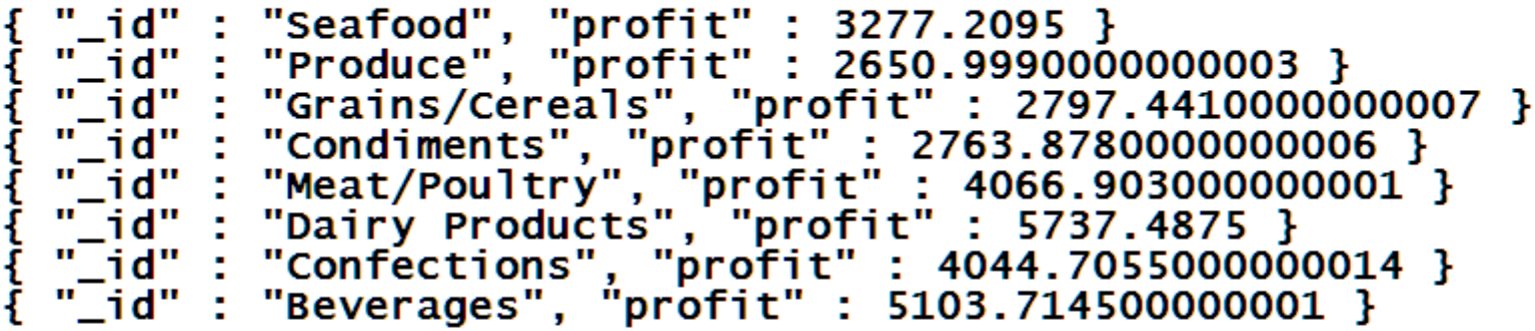
```
db.sales.aggregate(
{ $group: {
        _id: "$category",
        salesCount: {$sum: 1},
        salesTotal: {$sum: "$sales"}
        }
},
{ $match: { salesCount: { $gte: 40}}}
);
```



```
{ "_id" : "Seafood", "salesCount" : 45, "salesTotal" : 65544.18999999999 }
{ "_id" : "Confections", "salesCount" : 48, "salesTotal" : 80894.1100000002 }
{ "_id" : "Beverages", "salesCount" : 46, "salesTotal" : 102074.2900000001 }
```

Examples

```
db.sales.aggregate({
            $group : {
                        _id: "$category",
                        profit: { $sum: {$multiply: ["$sales", 0.05]}}
            }
});
```



```
{ "_id" : "Seafood", "profit" : 3277.2095 }
{ "_id" : "Produce", "profit" : 2650.9990000000003 }
{ "_id" : "Grains/Cereals", "profit" : 2797.4410000000007 }
{ "_id" : "Condiments", "profit" : 2763.8780000000006 }
{ "_id" : "Meat/Poultry", "profit" : 4066.903000000001 }
{ "_id" : "Dairy Products", "profit" : 5737.4875 }
{ "_id" : "Confections", "profit" : 4044.7055000000014 }
{ "_id" : "Beverages", "profit" : 5103.714500000001 }
```

Examples- Sorting the group result

```
db.salesdata.aggregate(
        {
                $group: {
                        _id: "$quarter",
                        salesCount: { $sum: 1},
                        salesTotal: { $sum: "$amount"},
                        salesAvg: { $avg: "$amount"},
                        maxSales: { $max: "$amount"},
                        minSales: { $min: "$amount"}
                }
        },
        {
                $sort: {
                        salesCount: 1,
                        salesTotal: 1
                }
        }
).pretty()
```

Operators with $group

- Following are some of the accumulator operators that could be used along with $group operator:
  - $avg, $first, $last, $max, $min, $push, $sum

Using mapReduce

- Syntax:

```
db.collection.mapReduce(
        mapFunction,
        reduceFunction,
        options);
```

Using mapReduce

- mapFunction
  - a callback function
  - has access to a single document via “this”
  - should emit two properties from “this” or values derived out of “this” properties

Using mapReduce

- emit(a, b)
  - “a” will be used as a key representing an array of “b” values
- Example:
  - emit(this.category, this.sales) will create a dictionary with “category” as key and each “category” representing an array of corresponding “sales” values .

Example

- Consider the following data

```
{name: "ram", gender: "male"}
{name: "shyam", gender: "male"}
{name: "sita", gender: "female"}
{name: "gita", gender: “female"}
```

Example

- emit(this.gender, this.name)

- would create a collection like this

```
[
        {key: "male", values: [ "ram", "shyam" ] },
        {key: "female", values: [ "sita", "gita" ] }
]
```

Example

- The reduceFunction will receive the key and values from each document separately

Using mapReduce

- reduceFunction
  - callback function
  - called for each of the key generated by the emit function
  - receives two arguments, key and an array of values

Example

```
var op=db.persons.mapReduce(function(){
        var g = "m";
        var t = "Mr.";
        if(this.gender=="female"){
                g = "f";
                t = "Ms.";
        }
        emit(g, t + this.name);
}, function(k, v){
        return v.join();
}, {
        out: { inline: true}
});
```

Example

```
After calling emit(g, t + this.name)
key --> values
"m" --> ["Mr.ram", "Mr.shyam"]
"f" --> ["Ms.sita", "Ms.gita"]

After the reduceFunction is called,
key --> values
"m" --> "Mr.ram, Mr.shyam"
"f" --> "Ms.sita, Ms.gita"
```

Indexes

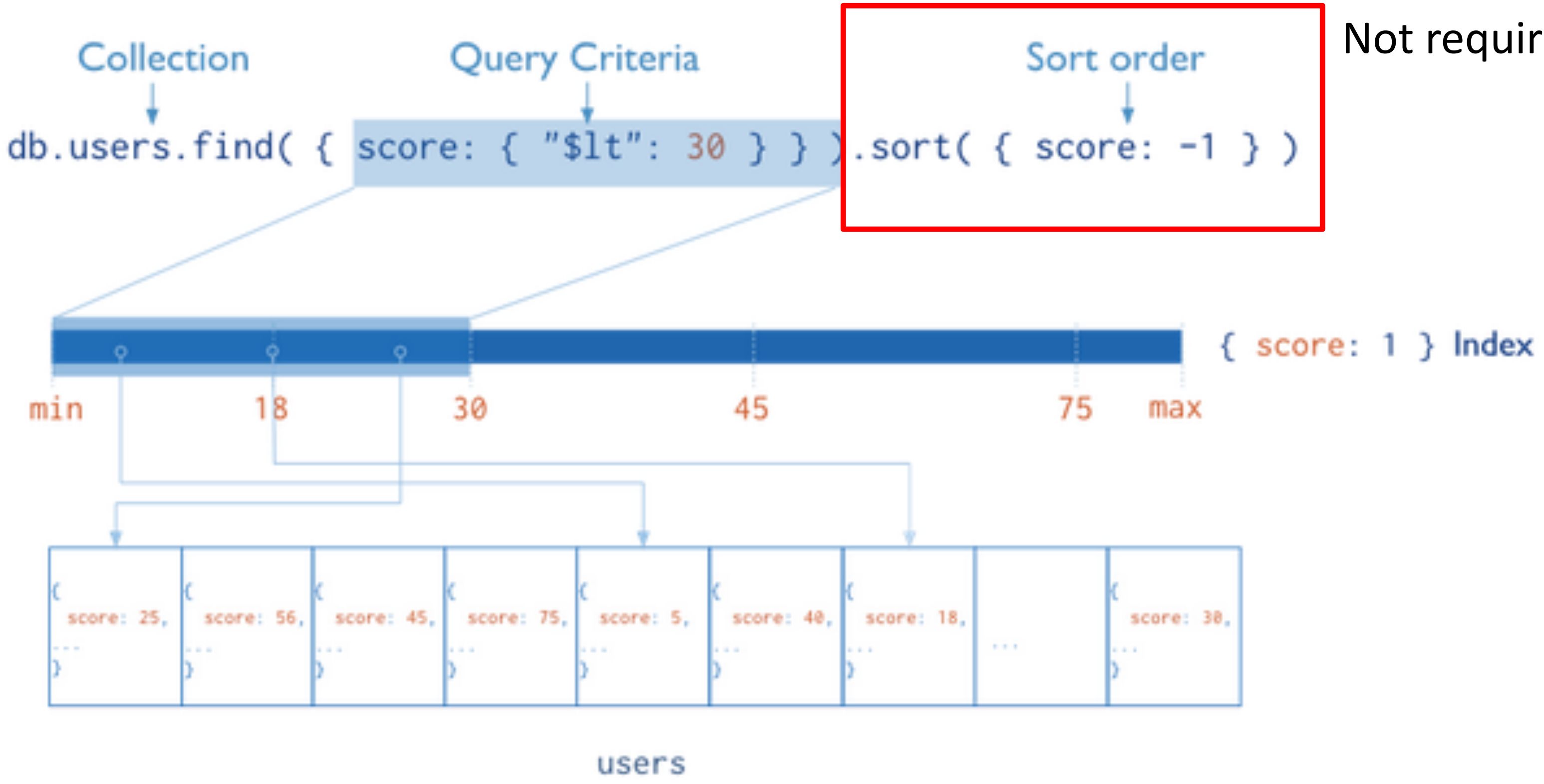
- Indexes help efficient execution of queries.
  - Without indexes MongoDB must scan every document in a collection to select those documents that match the query statement.
  - These collection scans are inefficient because they require mongod to process a larger volume of data than an index for each operation.

Indexes

- Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.
  - The index stores the value of a specific field or set of fields, ordered by the value of the field.

Indexes

MongoDB can use indexes to return documents sorted by the index key directly from the index without requiring an additional sort phase.

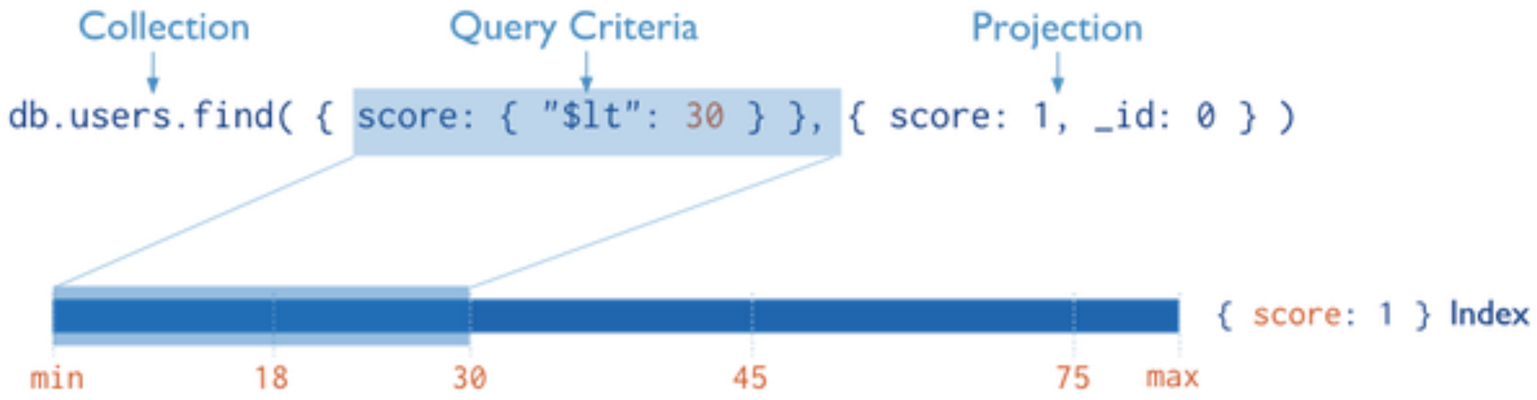


```
db.users.find( { score: { "$lt": 30 } } ).sort( { score: -1 } )
```

Not required

Indexes

When the query criteria and the projection of a query include only the indexed fields, MongoDB will return results directly from the index without scanning any documents or bringing documents into memory.



Index types

- Single field index
- Compound index
- Multi key index
- Geospatial index
- Text index
- Hashed index

http://docs.mongodb.org/v2.6/core/index-types/

http://docs.mongodb.org/manual/core/index-types/

Index creation

- Index on single field:

```
db.people.ensureIndex({"phone-number": 1})
```

- Compound index

```
db.products.ensureIndex(
          {item: 1, category: 1, price: 1})
```

- Text index

```
db.reviews.ensureIndex(
          { comments: "text" } )
```

Index creation

- Hashed index

```
db.active.ensureIndex(
        { productName: "hashed" } )
```

- Geospatial index

```
db.places.ensureIndex({coords: "2dsphere"})
db.places.ensureIndex({coords: "2d"})
```

Geospatial Index Example

```
db.atms.ensureIndex({coords: "2dsphere"});

db.atms.find({
        coords: {
                $near: [12.9461, 77.5703]
        }
}, {_id: 0}).limit(3).pretty();
```

Sharding

- Storing data across multiple machines.
- MongoDB uses sharding to support deployments with
  - very large data sets and
  - high throughput operations

Problems

- Database systems with large data sets and high throughput applications can challenge the capacity of a single server.
  - High query rates can exhaust the CPU capacity of the server.
  - Larger data sets exceed the storage capacity of a single machine.
  - Working set sizes larger than the system’s RAM stress the I/O capacity of disk drives.

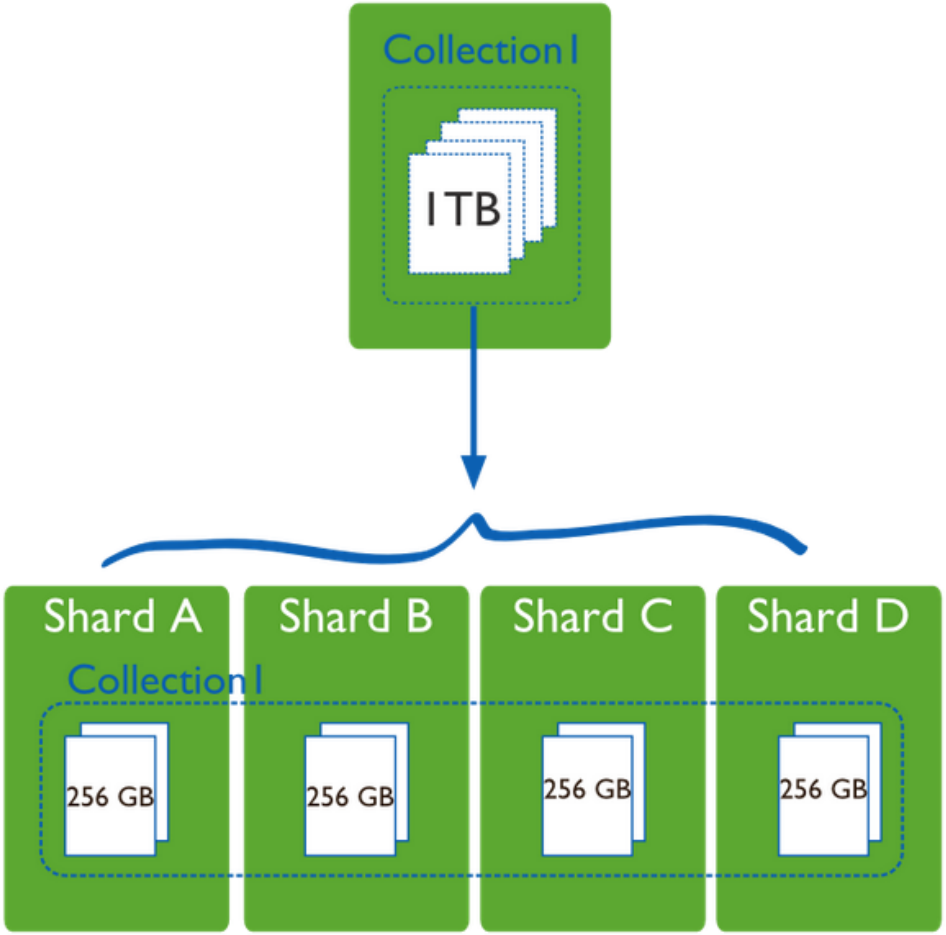
Solution 1

- Vertical scaling
  - Increase capacity by adding more CPU and storage resources
    - Limitations: high performance systems with large numbers of CPUs and large amount of RAM are disproportionately more expensive than smaller systems
    - Practical maximum capability
      - Cloud-based providers may only allow users to provision smaller instances

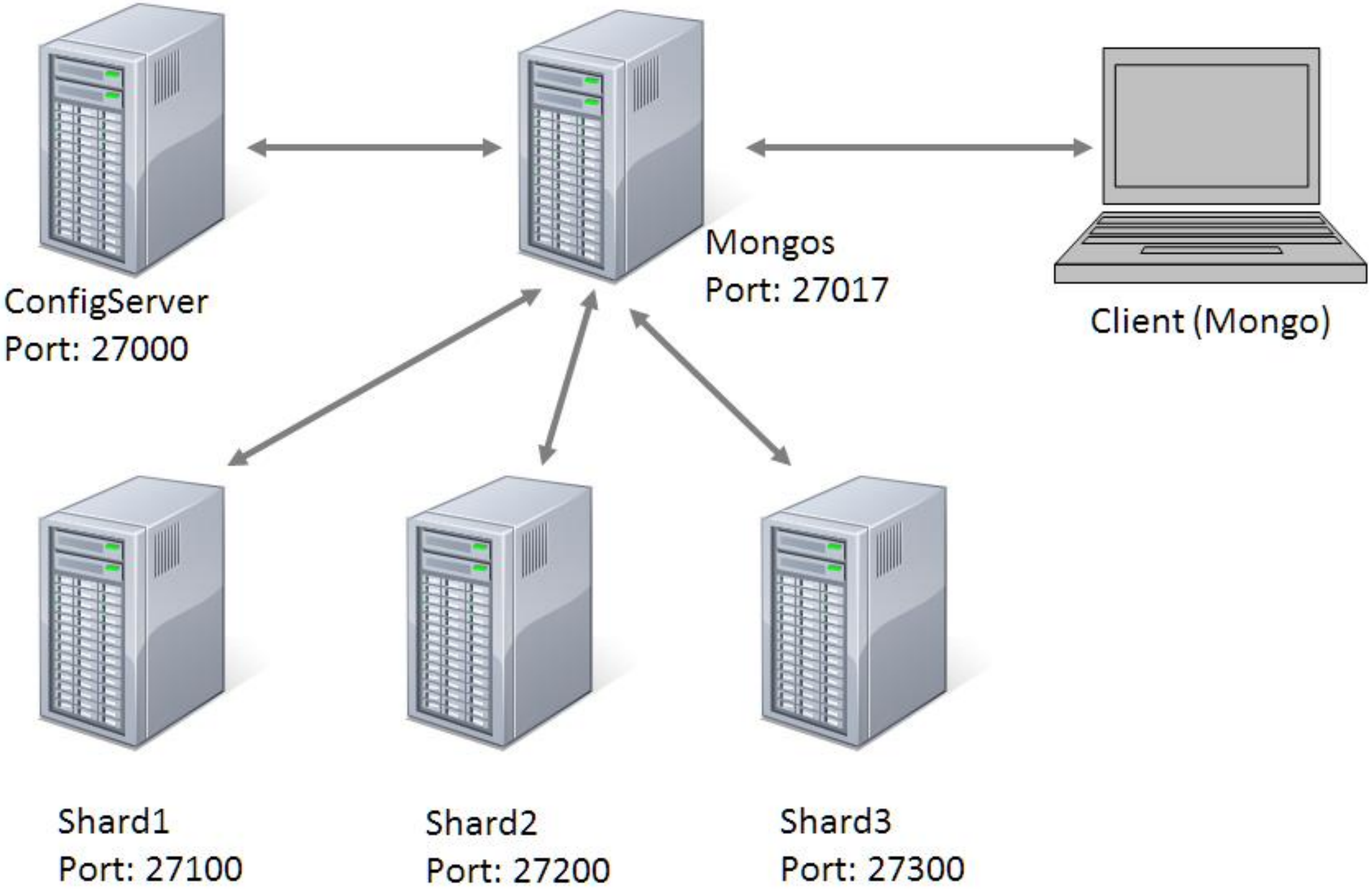
Solution 2

- Sharding
  - Horizontal scaling
  - Divides the data set and distributes the data over multiple servers, or shards.
    - Each shard is an independent database,
    - Collectively, the shards make up a single logical database.

Sharding


Collection I
ITB
Shard A
Shard B
Shard C
Shard D
Collection I
256 GB
256 GB
256 GB
256 GB

Sharding

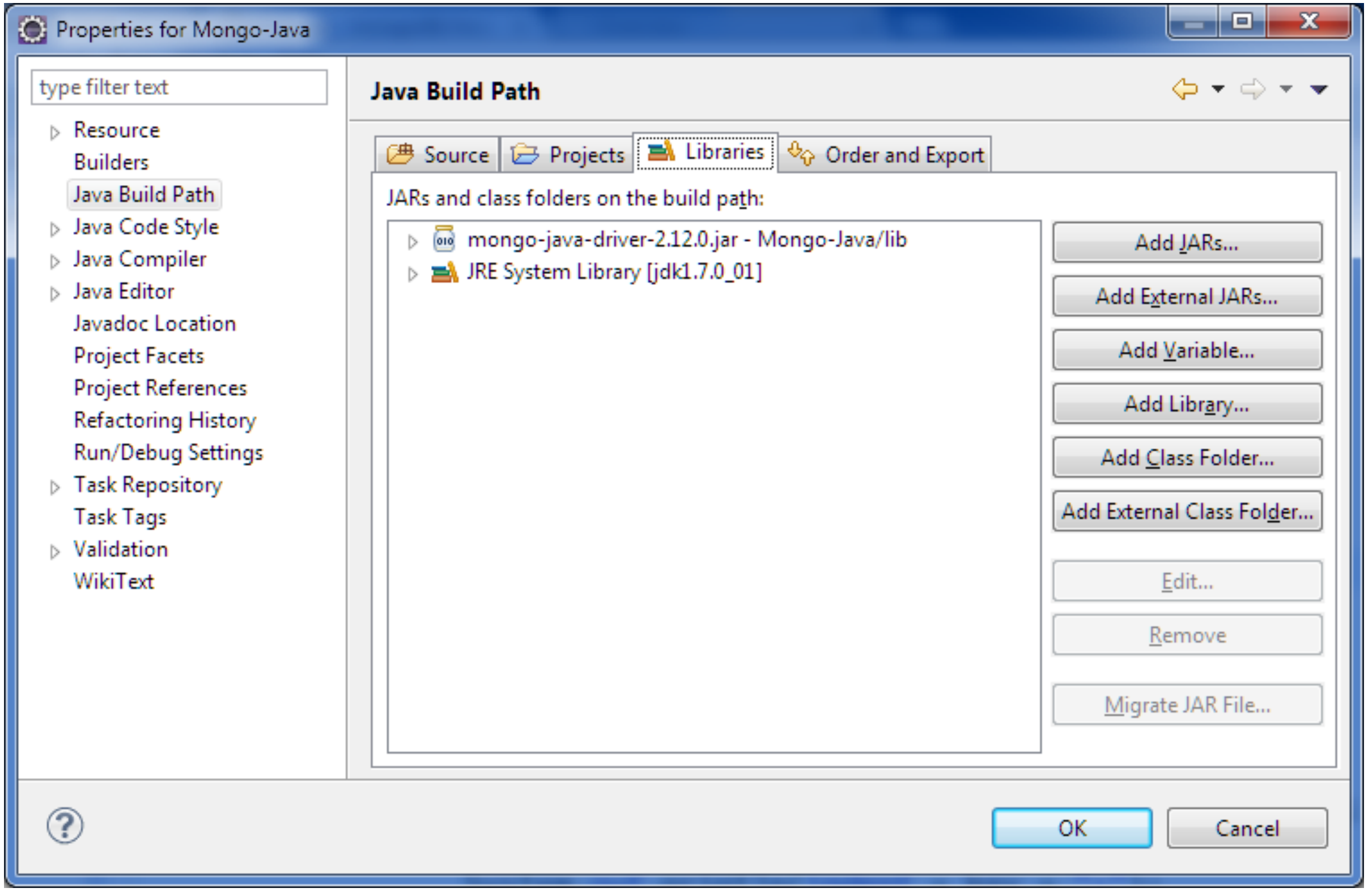

ConfigServer
Port: 27000
Mongos
Port: 27017
Client (Mongo)
Shard1
Port: 27100
Shard2
Port: 27200
Shard3
Port: 27300

Using JavaDriver

- Download the latest version of JavaDriver
  - https://github.com/mongodb/mongo-java-driver/releases/download/r2.12.0/mongo-java-driver-2.12.0.jar

Using JavaDriver

- Add to buildpath (classpath)



JARs and class folders on the build path:

mongo-java-driver-2.12.0.jar - Mongo-Java/lib

JRE System Library [jdk1.7.0_01]

Use the API

- com.mongodb.MongoClient
  - A MongoDB client with internal connection pooling.
  - For most applications, you should have one MongoClient instance for the entire JVM.
  - The MongoClient class is designed to be thread safe and shared among threads
  - Inherits from com.mongodb.Mongo

Connecting to server

```
MongoClient client = new MongoClient();

MongoClient client = new MongoClient("localhost");

MongoClient client = new MongoClient("localhost", 27017);

MongoClient client = new MongoClient(
       new ServerAddress("localhost"));

MongoClient client = new MongoClient(
       new ServerAddress("localhost"),
       new MongoClientOptions.Builder().build());
```

Connecting to replica set

- You can connect to a replica set using the Java driver by passing a ServerAddress list to the Mongo constructor

```
Mongo mongo = new Mongo(Arrays.asList(
        new ServerAddress("localhost", 27017),
        new ServerAddress("localhost", 27018),
        new ServerAddress("localhost", 27019))
);
```

Some useful methods

- List<String> getDatabaseNames()
- DB getDB(String dbName)
- void dropDatabase(String dbName)
- void close()

Query for list of databases

```
MongoClient client = new MongoClient("localhost", 27017);
List<String> dbNames = client.getDatabaseNames();

System.out.println("Following databases were found: ");
for (String dbName : dbNames) {
        System.out.println(dbName);
}

client.close();
```

Mongo Database

- com.mongodb.DB
- A thread-safe client view of a logical database in a MongoDB cluster.

```
MongoClient client = new MongoClient();
DB db = client.getDB("mydb");
```

Some useful methods

- DBCollection createCollection(
  String name, DBObject options)
- DBCollection getCollection(String name)
- Set<String> getCollectionNames()
- void dropDatabase()

Query for list of collections

```
MongoClient client = new MongoClient("localhost", 27017);
DB db = client.getDB("mydb");
Set<String> collections = db.getCollectionNames();

System.out.println("'mydb' contains following
collections");
for (String collection : collections) {
      System.out.println(collection);
}
client.close();
```

Mongo Collection

- com.mongodb.DBCollection
- This class provides a skeleton implementation of a database collection

```
MongoClient client = new MongoClient();
DB db = client.getDB("mydb");
DBCollection orders = db.getCollection("orders");
```

Adding a document

```
MongoClient mc = new MongoClient();
DB db = mc.getDB("vindb");

DBObject doc1 = new BasicDBObject("name", "Cellphone")
          .append("price", 25000.0)
          .append("make", "Lenovo");

DBCollection items = db.getCollection("items");
items.save(doc1);
```

Adding a json string

```
MongoClient mc = new MongoClient();
DB db = mc.getDB("vindb");

String itemStr = "{" +
        "name : \"Wrist Watch\"," +
        "make : \"Titan\"," +
        "price: 5600.0" +
        "}";

DBObject doc1 = (DBObject) JSON.parse(itemStr);
DBCollection items = db.getCollection("items");
items.save(doc1);
```

Getting data

```
MongoClient mc = new MongoClient();
DB db = mc.getDB("vindb");
DBCollection sales = db.getCollection("sales");

DBObject first = sales.findOne();

System.out.println(first);

System.out.println("Sales amount = $" + first.get("sales"));
System.out.println("Quarter      = " + first.get("quarter"));
System.out.println("Category     = " + first.get("category"));
System.out.println("Name         = " + first.get("product"));
```

Get all data

```
MongoClient mc = new MongoClient();
DB db = mc.getDB("vindb");
DBCollection books = db.getCollection("books");
DBCursor cursor = books.find();

String booksJson = JSON.serialize(cursor);
System.out.println(booksJson);
```

Querying

```
MongoClient mc = new MongoClient();
DB db = mc.getDB("vindb");
DBCollection books = db.getCollection("sales");

DBObject query = new BasicDBObject(
        "category", "Beverages");

DBCursor cursor = books.find(query);
```