MongoDB

Contents

- **Overview of MongoDB**
- **MongoDB Server**
- **MongoDB Client - Interactive shell**
- **Validation**

Introduction

- **humongous - extremely large : huge**
- **http://www.mongodb.org/**
- **An open source, high-performance, schema-free, document-oriented NoSQL database system.**

- **A record in MongoDB is a document, which is a data structure composed of field and value pairs.**

Relational v. Mongo

Relational	**Mongo**
database	**database**
table	**collection**
row	**document**
column	**field**

Mongo is a Schema-less Database

- **Fields are associated with documents rather than collections.**

- **In other words different documents in a collection can have different fields.**

- **MongoDB is a schema-less database.**

Large Data Sets – Scalability Issues

- **Large data sets and high throughput applications challenge the capacity of a single server:**
  - **High query rates can exhaust CPU**
  - **Large data sets can exceed storage capacity of a single machine/ can also stress RAM**

Large Data Sets – Scalability Issues

- **Database systems have 2 approaches to address these scalability issues**
  1. **Vertical Scaling – "scale up" – increase CPU, increase storage capacity. Practical maximum for vertical scaling, cost of these high performance systems versus smaller systems**
  2. **Sharding or Horizontal scaling distributes data over multiple servers, or shards. Each shard is an independent database, and collectively, the shards make up a single logical database.**

Sharding

- **MongoDB's path to scaling out to huge volumes.**
- **MongoDB provides horizontal scalability as part of its core functionality**
- **Automatic sharding distributes data across a cluster of machines**

Sharding

- **Sharding is a method for storing data across multiple machines.**
- **MongoDB uses sharding to support deployments with very large data sets and high throughput operations.**

Advantages of Sharding

- **Each shard processes fewer operations as the cluster grows.**
- **As a result, shared clusters can increase capacity and throughput *horizontally*.**

- **For example, to insert data, the application only needs to access the shard responsible for that record**
- **Sharding reduces the amount of data that each server needs to store**

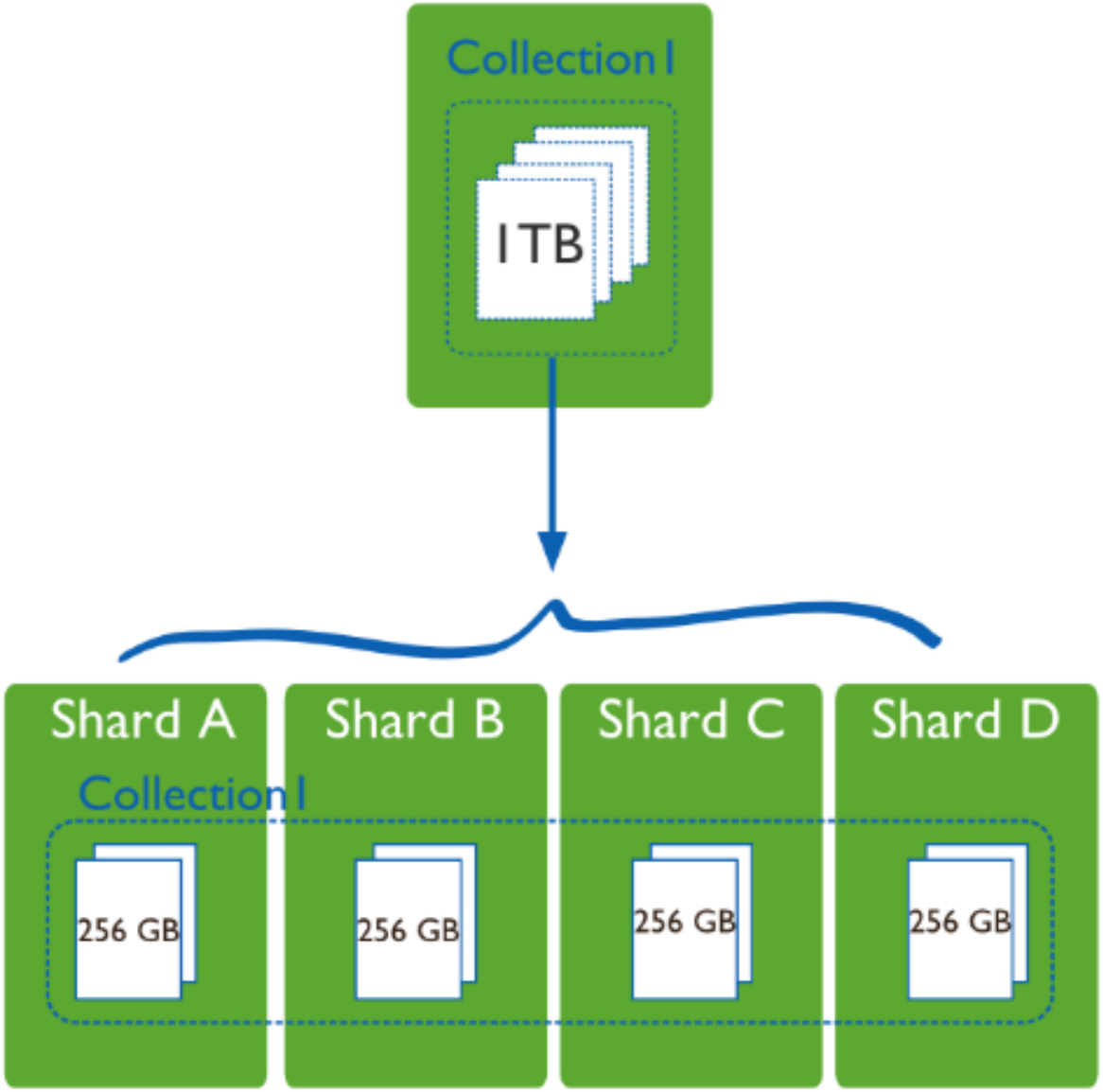


Diagram of a large collection with data distributed across 4 shards.

Sharding addresses the challenge of scaling to support high throughput and large data sets:

http://docs.mongodb.org/

Replication

- **Replication is the process of synchronizing data across multiple servers**
- **This protects the database from the loss of a single server.**
- **It also allows recovery from hardware failure and service interruption**
- **A Replica Set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability**

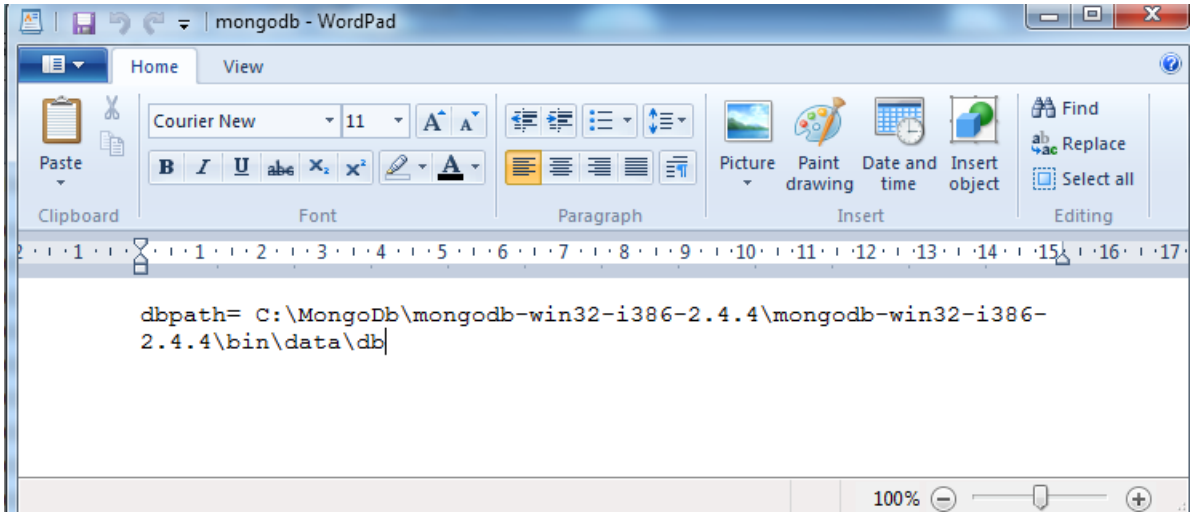
MongoDB – Interactive Shell

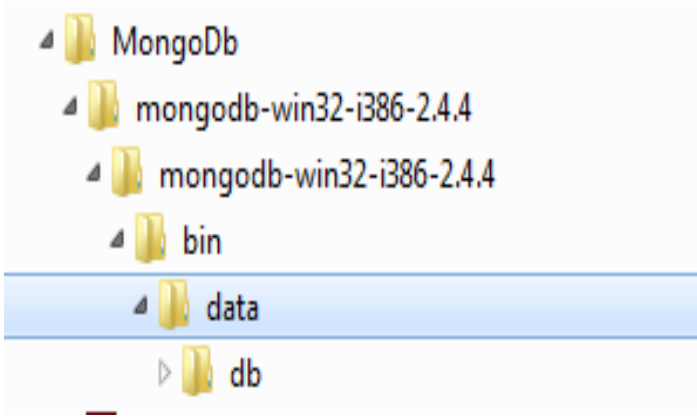
Download the zip file from Moodle

mongodb.config

- **Create data/db folder in the bin folder- this determines where databases are created.**

- **Create a file called mongodb.config in the bin folder.**





Run the Server

- **mongod --config .\mongodb.config**

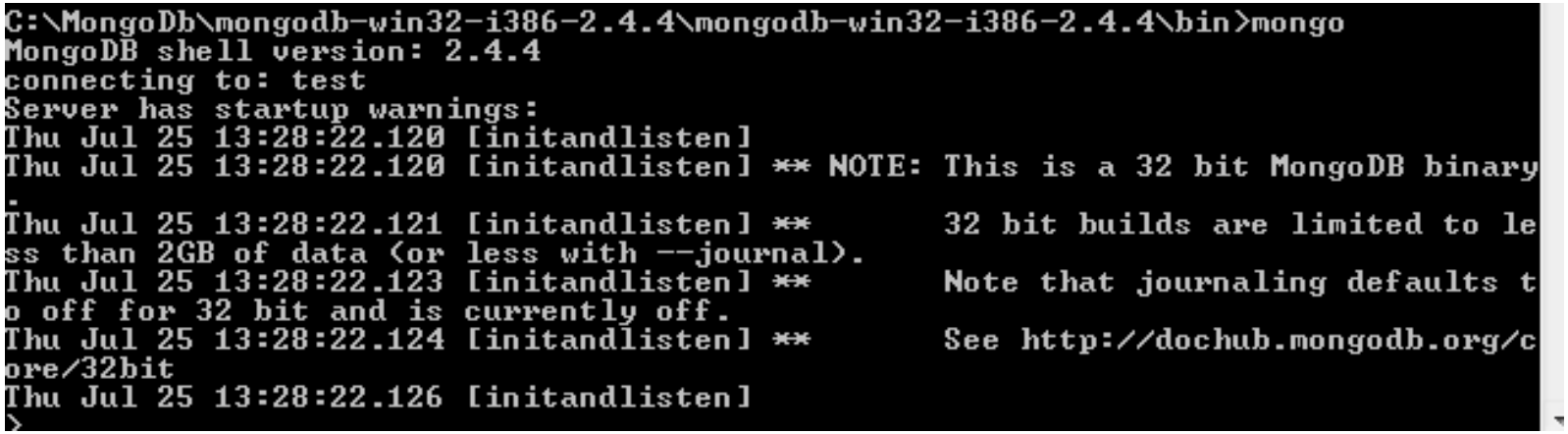
- **[You might need to delete a lock file (mongod.lock) in your example folder if mongod was not shut down correctly].**

```
C:\MongoDb\mongodb-win32-i386-2.4.4\mongodb-win32-i386-2.4.4\bin>mongod --config
 .\mongodb.config
Thu Jul 25 13:28:22.082
Thu Jul 25 13:28:22.085 warning: 32-bit servers don't have journaling enabled by
 default. Please use --journal if you want durability.
Thu Jul 25 13:28:22.085
Thu Jul 25 13:28:22.119 [initandlisten] MongoDB starting : pid=7032 port=27017 d
bpath=C:\MongoDb\mongodb-win32-i386-2.4.4\mongodb-win32-i386-2.4.4\bin\data\db 3
2-bit host=W305-EF-LTOP
Thu Jul 25 13:28:22.120 [initandlisten]
Thu Jul 25 13:28:22.120 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Thu Jul 25 13:28:22.121 [initandlisten] **       32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Thu Jul 25 13:28:22.123 [initandlisten] **       Note that journaling defaults t
o off for 32 bit and is currently off.
Thu Jul 25 13:28:22.124 [initandlisten] **       See http://dochub.mongodb.org/c
ore/32bit
Thu Jul 25 13:28:22.126 [initandlisten]
Thu Jul 25 13:28:22.129 [initandlisten] db version v2.4.4
Thu Jul 25 13:28:22.129 [initandlisten] git version: 4ec1fb96702c9d4c57b1e06dd34
eb73a16e407d2
Thu Jul 25 13:28:22.129 [initandlisten] build info: windows sys.getwindowsversio
n(major=6, minor=0, build=6002, platform=2, service_pack='Service Pack 2') BOOST
_LIB_VERSION=1_49
Thu Jul 25 13:28:22.130 [initandlisten] allocator: system
Thu Jul 25 13:28:22.130 [initandlisten] options: { config: ".\mongodb.config", d
bpath: "C:\MongoDb\mongodb-win32-i386-2.4.4\mongodb-win32-i386-2.4.4\bin\data\db
" }
Thu Jul 25 13:28:22.384 [initandlisten] waiting for connections on port 27017
Thu Jul 25 13:28:22.484 [websvr] admin web console waiting for connections on po
rt 28017
```

Run the Client

- In a separate cmd, run the following command

  mongo



MongoDB Interactive Shell

- **Allows you to issue commands to MongoDB**
- **Is a Javascript Shell.**
- **Allows you to insert, query, update, delete**

JSON

- **JSON provides an easy way to create and store data structures for use with JavaScript**

- **Data encoded with JSON can be read into a variable which creates an object.**

- ***www.json.org/js.htm***

JSON - Example

```
{
"book": [
{
"id":"01",
"language": "Java",
"edition": "third",
"author": "Herbert Schildt"
},
{
"id":"07",
"language": "C++",
"edition": "second"
"author": "E.Balagurusamy"
}]
}
```

JSON - Syntax

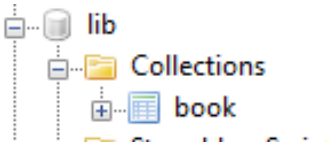
- **Data is represented in name/value pairs**

- **Curly braces hold objects and each name is followed by ':‘ (colon), the name/value pairs are separated by , (comma).**

JSON - Datatypes

- **Number**
- **String**
- **Boolean**

```
{“student": {
  "name":“James",
  “marks":97,
  “distinction": false
  }
}
```



Querying data in Mongodb

- **db – to display the database you are using**
- **show dbs – to list the databases**
- **use Lib – to switch to Lib database**
- **db.getCollectionNames() or show collections**

- **insert**

  db.book.insert({name:'Harry Potter', price:19.99})
  (db refers to the current database)

  If the book collection doesn’t exist, the above command will create it

Insert (using interactive shell)

db.book.insert({name:'Introduction to MongoDB', price:'39.99'})

- **db.book.find() (lists all the documents in a collection)**
- **db.book.find().pretty() makes the output a more easy-to-read format**

Insert – Schema-less

- **db.book.insert({name:'Introduction to Java', price:49.99})**
- **db.book.insert({name:‘C#’})**
- **db.book.insert({name:‘C++’})**

- **Schema-less database.**

- **db.book.find()**

Query Selectors

- **{field: value} is used to find any documents where field is equal to value.**

- **db.book.find({name:‘C++'})**

More Data

- **db.book.insert({name:'MySQL', price:20.99})**
- **db.book.insert({name:‘Android', price:20.99})**

Range Queries & Operators

- **db.book.find({price:{$lt:25}})**
- **db.book.find({price: {$lt: 25, $gt: 20}})**

- **$lt, $lte, $gt, $gte and $ne are used for less than, less than or equal, greater than, greater than or equal and not equal operations**

Updating / Set

- **Set is used to change fields.**
- **db.book.update({name:'Databases'}, {$set: {price: 24.99}})**
- **db.book.find({name:'Databases'})**

```
>
> db.book.update({name:'Databases'}, {$set: {price:24.99}})
> db.book.find({name:'Databases'})
{ "_id" : ObjectId("52f4cb564b80c94b84070b74"), "name" : "Databases", "price" :
24.99 }
>
```

Adding a new field to all documents in a collection

- To add a new_field to all your collection, use the update command

db.collection.update({},{$set:{"new_field":1}},false,true)

- In the above example last 2 fields “false”, “true” specifies the upsert and multi flags.

- {} implies the update should be made to all documents

Upsert

- Upsert: If set to false, ensures a new document is not created when no document matches the query criteria. Example

db.students.update({ "_id": 1 }, { $set: { "grade": "B" } },{ upsert: true })

- If a document with "_id": 1 exists, it will be updated to set the grade to "B.“
- If no document with "_id": 1 exists, a new document will be inserted with "_id": 1 and "grade": "B"

Multi Flags

the multi option in the update command specifies whether to update a single document or multiple documents that match the specified query criteria.

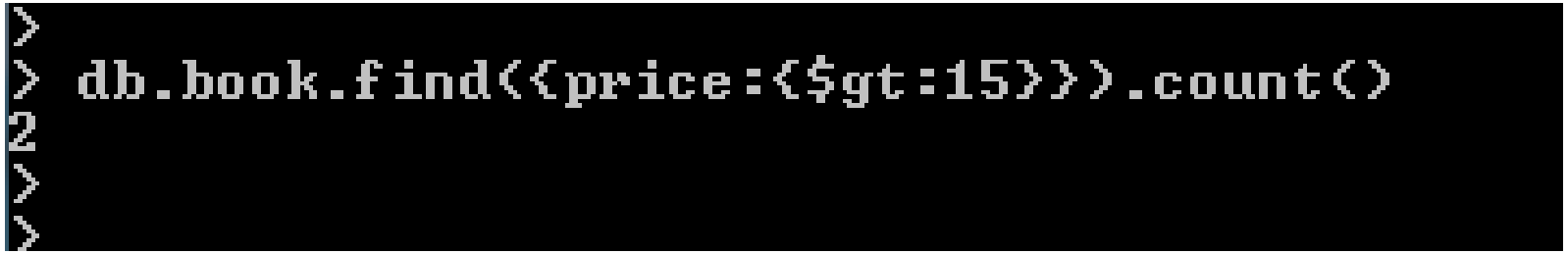
When multi is set to true, the update operation affects all documents that match the query; when set to false (or omitted), only the first matching document is updated

db.students.update({ "name": "Bob" }, { $set: { "grade": "A+" } }, { multi: true })

In the above example, all students named Bob will get grade A+

Counting

- db.book.find({price: {$gt: 25}}).count()



Return specific values

- Employees collection, lets say you only want to retrieve all name and salary values
- { "_id": 1, "name": "Alice", "salary": 60000, "department": "HR" }
- { "_id": 2, "name": "Bob", "salary": 70000, "department": "IT" }
- { "_id": 3, "name": "Charlie", "salary": 55000, "department": "Finance" }

- **db.employees.find({}, { "_id": 0, "name": 1, "salary": 1 })**

- Here, only the name and salary fields are returned
- (“_id”:0 is not necessary here)

Remove Documents

- In MongoDB, the db.collection.remove() method removes all documents from a collection
- db.book.remove({ name : “Databases" })

```
> db.book.find()
{ "_id" : ObjectId("52f4b8c80e101cd3383fbf19"), "name" : "C++" }
{ "_id" : ObjectId("52f4b51e2fad157ddc602ce7"), "name" : "Introduction to Java",
 "price" : "49.99" }
{ "_id" : ObjectId("52f4bd6dd0f3c30e2c7f30bf"), "name" : "C#" }
{ "_id" : ObjectId("52f4cac14b80c94b84070b73"), "price" : 18.99 }
{ "_id" : ObjectId("52f4cb564b80c94b84070b74"), "name" : "Databases", "price" :
24.99 }
{ "_id" : ObjectId("52f4db27f67cf32c1070df70"), "name" : "Math 1", "price" : 21.
99 }
{ "_id" : ObjectId("52f4db27f67cf32c1070df71"), "name" : "Math 2", "" : 30.99 }
{ "_id" : ObjectId("52f4db27f67cf32c1070df72"), "name" : "Math 3", "age" : 35.99
 }
>
>
>
>
>
>
>
> db.book.remove({name:'Databases'})
> db.book.find()
{ "_id" : ObjectId("52f4b8c80e101cd3383fbf19"), "name" : "C++" }
{ "_id" : ObjectId("52f4b51e2fad157ddc602ce7"), "name" : "Introduction to Java",
 "price" : "49.99" }
{ "_id" : ObjectId("52f4bd6dd0f3c30e2c7f30bf"), "name" : "C#" }
{ "_id" : ObjectId("52f4cac14b80c94b84070b73"), "price" : 18.99 }
{ "_id" : ObjectId("52f4db27f67cf32c1070df70"), "name" : "Math 1", "price" : 21.
99 }
{ "_id" : ObjectId("52f4db27f67cf32c1070df71"), "name" : "Math 2", "" : 30.99 }
{ "_id" : ObjectId("52f4db27f67cf32c1070df72"), "name" : "Math 3", "age" : 35.99
 }
>
```

Remove a Database

- The best way to do it is from the **mongodb** console: > use mydb; > **db**.dropDatabase();

- Alternatively, you can stop mongod and **delete** the data files from your data directory, then restart.

Excercise

- Download the stocks.json file from Moodle. Place it in the Mongo Bin Folder.
- Use the following command to import the file.
- Enter the following into a **terminal**. Don't enter this into the Mongo console or it won't work. If already in Mongo, press control and c to exit.
- **mongoimport –db stocks –collection stocks –file stocks.json**

- Find all the stocks where the profit margin is over 0.5
- Find all the stocks with negative growth this year
- Find all stocks where the 52 week low is less than 2
- Find all stocks where average volume is between 160 and 200

Validation

Benefit of flexible schemas

- Relational
  - Up-front schema definition phase
  - Adding a new column takes time to develop and “lots” of time to roll out in production
  - Existing rows must be reformatted
- MongoDB
  - Start hacking your app right away
  - Want to store new type on information?
  - Just start adding it
  - If it doesn’t apply to all instances – just leave it out

Why validate documents?

- Many people writing to the database
  - Many developers
  - Many teams
  - Many companies
  - Many development languages

- Multiple application want to exploit the same data, need to agree on what's there
- Usually a core set of keys and attributes you want there.

Why validate documents? ctd

- For any key you may care about
  - Existence
  - Type
  - Format
  - Value
  - Existence in combination with other keys (e.g. need a phone number or an email address)
  - Some core checks
    - E.g. price should always be there and it should be a number and e.g. > 0

MongoDB 3.2 validation

- Using MongoDB query language
- Automatically tests each insert/update and delivers a warning or error if a rule is broke.
- You choose which attributes to validate and which you don’t care about

Book collection fails if you don't include a name attribute

```
> db.runCommand ({collMod: "book", validator: { name: {$exists:true} } } );
{ "ok" : 1 }
>
>
>
> db.book.insert({title: "Databases made simple"})
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 121,
                "errmsg" : "Document failed validation"
        }
})
>
```

To see what rules have been added to a collection

```
> db.getCollectionInfos({name:"book"})
[
        {
                "name" : "book",
                "options" : {
                        "validator" : {
                                "name" : {
                                        "$exists" : true
                                }
                        },
                        "validationLevel" : "strict",
                        "validationAction" : "error"
                }
        }
]
>
```

Restricting the values for a field

```
> db.runCommand({collMod:"book", validator: {category: {$in:["Children", "Educat
ion", "Other"]}}})
{ "ok" : 1 }
```

```
> db.getCollectionInfos({name:"book"})
[
        {
                "name" : "book",
                "options" : {
                        "validator" : {
                                "category" : {
                                        "$in" : [
                                                "Children",
                                                "Education",
                                                "Other"
                                        ]
                                }
                        },
                        "validationLevel" : "strict",
                        "validationAction" : "error"
                }
        }
]
>
```

The category field in the book Collection has a validation rule added limiting it's value to Children, Education or other

```
> db.book.insert({name:"Sooty", category:"Children"});
WriteResult({ "nInserted" : 1 })
>
>
>
> db.book.insert({name:"Databases", category:"IT"});
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 121,
                "errmsg" : "Document failed validation"
        }
})
>
```

Validation - String

```
> db.runCommand({collMod: "book", validator: {name: {$type: "string"}}});
{ "ok" : 1 }
>
```

```
> db.getCollectionInfos({name:"book"})
[
        {
                "name" : "book",
                "options" : {
                        "validator" : {
                                "name" : {
                                        "$type" : "string"
                                }
                        },
                        "validationLevel" : "strict",
                        "validationAction" : "error"
                }
        }
]
> db.book.insert({name: 1245, category:"IT"});
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 121,
                "errmsg" : "Document failed validation"
        }
})
>
```

Validation : Number

```
> db.runCommand({collMod: "book", validator: {price: {$type: 1}}});
{ "ok" : 1 }
>
```

Type	Number	Alias
Double	1	“double”
String	2	“string”

Validation – combining a number of rules

```
> db.runCommand({collMod: "book", validator:{$and: [ {price: {$type: 1}}, {name:
 {$type:2}} ]}});
{ "ok" : 1 }
```

```
> db.getCollectionInfos({name:"book"})
[
        {
                "name" : "book",
                "options" : {
                        "validator" : {
                                "$and" : [
                                        {
                                                "price" : {
                                                        "$type" : 1
                                                }
                                        },
                                        {
                                                "name" : {
                                                        "$type" : 2
                                                }
                                        }
                                ]
                        },
                        "validationLevel" : "strict",
                        "validationAction" : "error"
                }
        }
]
>
```

Where MongoDB validation excels

- Simple, flexible – only enforced on parts of the schema
- You can start adding data at any point and then add validation later if needed
- Practical to deploy
  - Simple to role out new rules across thousands of production servers
- Lightweight
  - Negligible impact to performance

Cleaning up legacy data

- The validator does not check if existing documents in the collection meet the new validation rules
- User/app can execute a query to identify & update any document which doesn’t meet the new rules

Controlling Validation

- validationLevel option - determines how strictly MongoDB applies validation rules to existing documents during an update
- validationAction option -determines whether MongoDB should error and reject documents that violate the validation rules or warn about the violations in the log but allow invalid documents.

		validationLevel		
		off	moderate	strict
validationAction	warn	No checks	Warn on validation failure for inserts & updates to existing valid documents. Updates to existing invalid docs OK.	Warn on any validation failure for any insert or update.
	error	No checks	Reject invalid inserts & updates to existing valid documents. Updates to existing invalid docs OK.	Reject any violation of validation rules for any insert or update. DEFAULT

Existing Documents

- You can control how MongoDB handles existing documents using the validationLevel option.

- By default, validationLevel is **strict** and MongoDB applies validation rules to all inserts and updates.
- Setting validationLevel to **moderate** applies validation rules to inserts and to updates to existing documents that fulfil the validation criteria.
- With the moderate level, updates to existing documents that do not fulfil the validation criteria are not checked for validity.

Accept or Reject Invalid Documents

- The validationAction option determines how MongoDB handles documents that violate the validation rules.

- By default, validationAction is error and MongoDB rejects any insertion or update that violates the validation criteria. When validationAction is set to warn, MongoDB logs any violations but allows the insertion or update to proceed.

Controlling Validation

Controlling validation

- Set behavior:

```
db.bleh.runCommand("collMod",
                   {validationLevel: "moderate",
                    validationAction: "warn"})
```

  - Note that the warnings are written to the log