

# MongoDB

## Lesson 00

iGATE is now a part of Capgemini.

People matter, results count.



©2016 Capgemini. All rights reserved.  
The information contained in this document is proprietary and confidential. For  
Capgemini only.

## Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
27-Jun-16	1.0		Srividhya	



Copyright © Capgemini 2015. All Rights Reserved. E

## Course Goals and Non Goals

- Course Goals
  - To understand MongoDB concepts.
- Course Non Goals
  - Nothing Specific.



## Intended Audience

- Software Programmers
- Software Analysts



## Day Wise Schedule

- Day 1
  - MongoDB Overview
  - CRUD Operations
  - Basic Operations
- Day 2
  - Aggregations
  - Indexing
  - Replication and Sharding



Copyright © Capgemini 2019. All Rights Reserved. E

## **MongoDB – Overview**

Lesson 01

## MongoDB – Overview

Version Sheet: MANDATORY (hidden in ‘slide show’ mode)

Version	Changes	Author
001	Redesign	Rohan Salvi

 Capgemini  
www.capgemini.com

Copyright © Capgemini 2012. All Rights Reserved 2

## Note to the SME:

Please read before you begin reviewing this module as SME

**Dark Red box with white text**

Note to the SME

The SME must provide missing info.

**Amber box with black text**

Note to the SME

The SME must validate the re-design/ the modification/ addition.

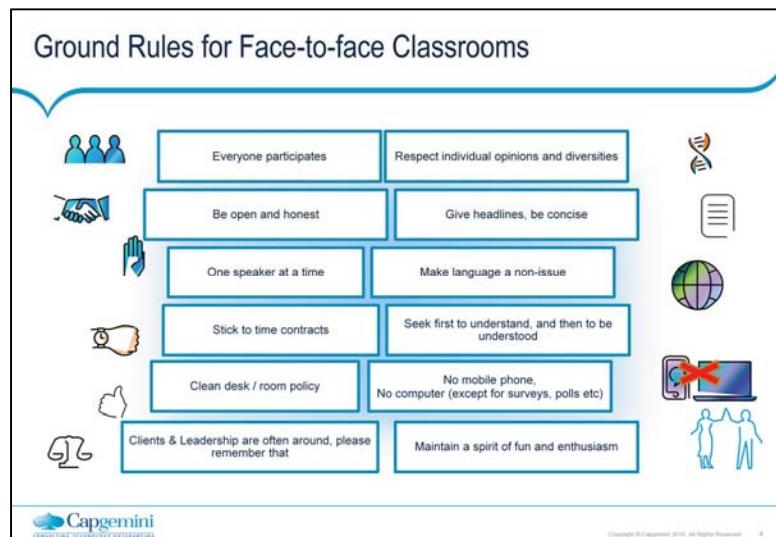
Note to the SME:  
This is a temporary slide and will not be part of the final upload at all. It is to help the SME understand how we will communicate changes to them.

To Review and Navigate the comments in the presentation Click on the "Review" Tab and then Click "Next" to review all the comments . Also you can add your comments using the "New comment" button

The comments in the review section have also been updated in a green textbox with black text.

Copyright © Capgemini 2017. All Rights Reserved

## MongoDB – Overview



Best to print on an A3 sheet and display in class. The idea is to co-create, to connect to learn, to do-learn-do (knowing is not enough, the real purpose is to apply what we learn on the job). It would also be great to have the class share their learning with their teams. You are the facilitator, you will help all share knowledge and learn new concepts or skills, there will be no lectures, everyone is a teacher.

## Ground Rules for Virtual Classrooms

The slide features a title 'Ground Rules for Virtual Classrooms' at the top. Below it are two main columns of rules, each with a list of items. To the right of the columns is a callout box containing an additional rule.

Participate actively in each session	Communicate professionally with others
Share experiences and best practices	Mute when you're not speaking
Bring up challenges, ask questions	Wait for others to finish speaking before you speak
Discuss successes	Each time you speak, state your name
Respond to whiteboards, polls, quizzes, chat boxes	Build on others' ideas and thoughts
Hang up if you need to take an urgent phone call, don't put this call on hold	Disagreeing is OK—with respect and courtesy

**Be on time for each virtual session**

As a best practice...be just a few minutes early!

Copyright © Capgemini 2017. All Rights Reserved.

Show this slide and hide the one for onsite classrooms, if this is a virtual session.

ONLY for Virtual classrooms

## Module at a Glance

SME to provide the details required in the table.

<b>Target Audience:</b>	
<b>Course Level:</b>	<i>Basic</i>
<b>Duration (in hours):</b>	30 mins
<b>Pre-requisites, if any:</b>	NA
<b>Post-requisites, if any:</b>	<i>Submit Session Feedback</i>
<b>Relevant Certifications:</b>	None



Copyright © Capgemini 2017. All Rights Reserved.

## MongoDB – Overview

### Introductions (for Virtual Classrooms)

The template consists of a central title "Introductions (for Virtual Classrooms)" above two large white boxes. Each box contains a placeholder for a "Business Photo". Below each photo is a white box containing the text "Facilitator Name Role" and "Moderator Name Role" respectively. A red callout box with the text "SME to provide the photos and names of the facilitators." is positioned between the two main sections.

Business Photo

SME to provide the photos and names of the facilitators.

Business Photo

Facilitator  
Name  
Role

Moderator  
Name  
Role

 Capgemini  
CONSULTING | TECHNOLOGY | OPERATIONS

Copyright © Capgemini 2017. All Rights Reserved. 7

Add pics and Capgemini roles for the Facilitator and moderator. Establish their credibility, what qualifies them to facilitate this session.

## Agenda

- 1 NOSQL Introduction & Types
- 2 Introduction & Basics
- 3 CRUD
- 4 Aggregation
- 5 Indexes
- 6 Performance
- 7 Replication & Sharding



Copyright © Capgemini 2017. All Rights Reserved.

## Module Objectives



### What you will learn

At the end of this module, you will learn:

- What is NOSQL

Note to the SME : Please provide the module Objectives or validate the partially updated content

### What you will be able to do



At the end of this module, you be able to:

- Understand what is NOSQL
- Describe CRUD
- State the types of NOSQL
- Explain what is Aggregation
- Describe Replication & Sharding

Lets Get Social

Let's get Social

10gen | MongoDB

Capgemini

Copyright © Capgemini 2012. All Rights Reserved.

## More Technologies

### More Technologies and Requirements Than Ever

NoSQL      Document Data Stores  
Analytics

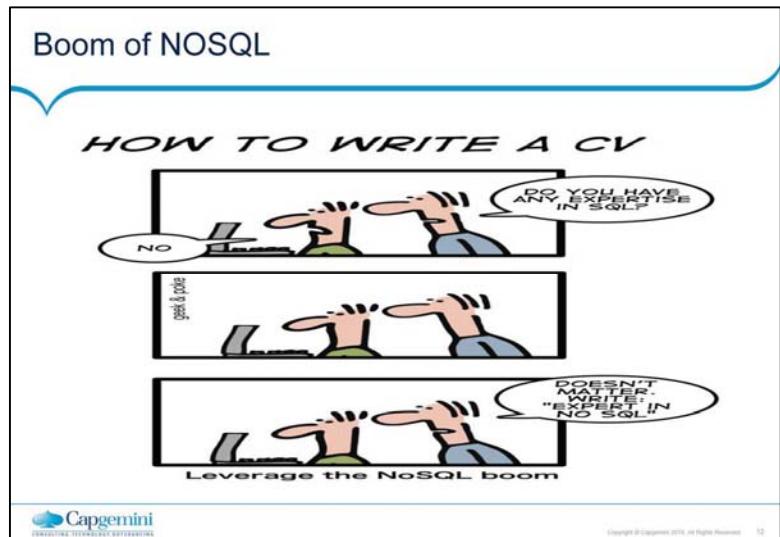
**Big Data**  
Key-value      JSON  
Graph      Datawarehouse  
**MongoDB**  
Hadoop      ODS  
Wide-column

**Cloud Computing**  
Agile Development  
Internet of Things  
**Mobile**  
Social networking  
Consumerization

4

Capgemini

Copyright © Capgemini 2012. All Rights Reserved. 11

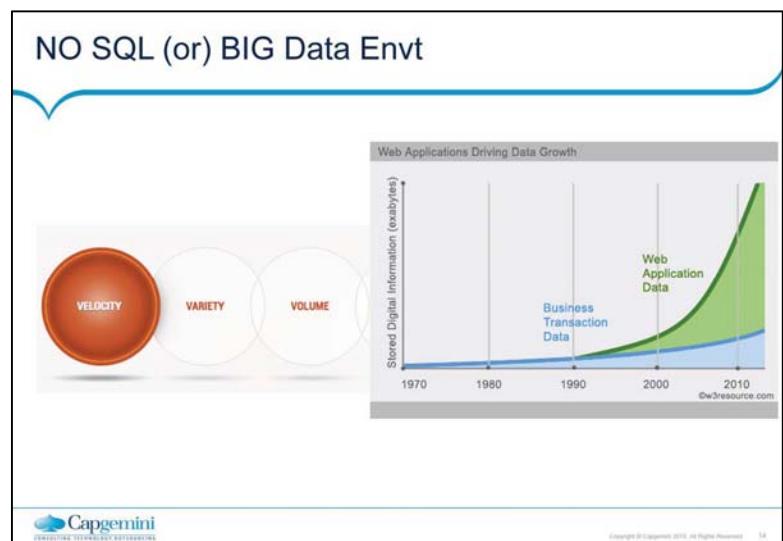


## Why NoSQL

- Handles Schema Changes Well (easy development)
- Solves Impedance Mismatch problem
- Rise of JSON
  - python module: simplejson

Capgemini  
Copyright © Capgemini 2012. All Rights Reserved 12

## MongoDB – Overview



## Example 1

### Social Network Graph

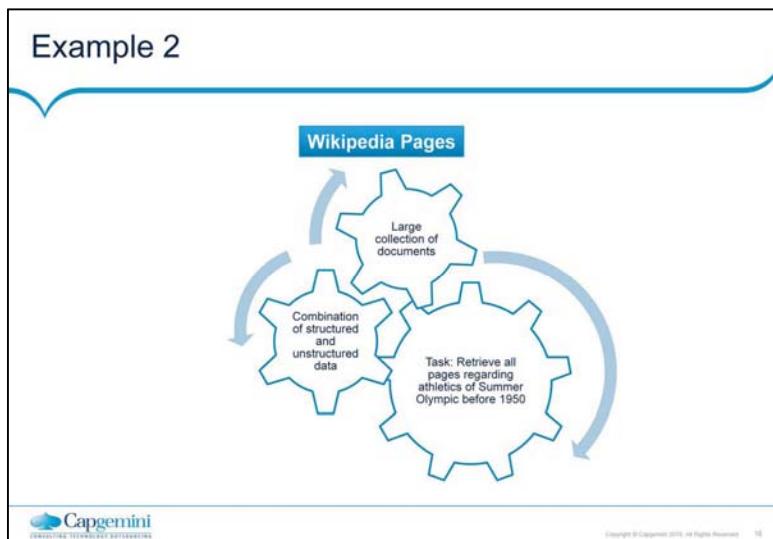
Each record: UserID1, UserID2

Separate records: UserID, first\_name, last\_name, age, gender, ...

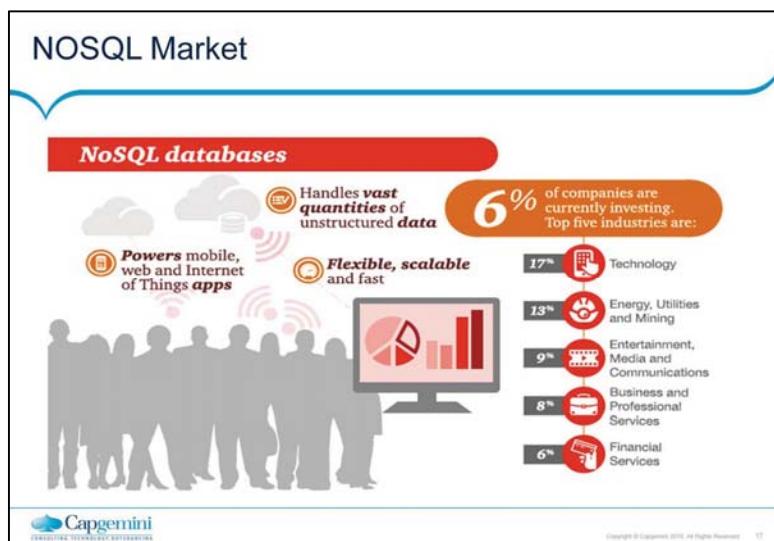
Task: Find all friends of friends of friends of ... friends of a given user



Copyright © Capgemini 2017. All Rights Reserved



## MongoDB – Overview



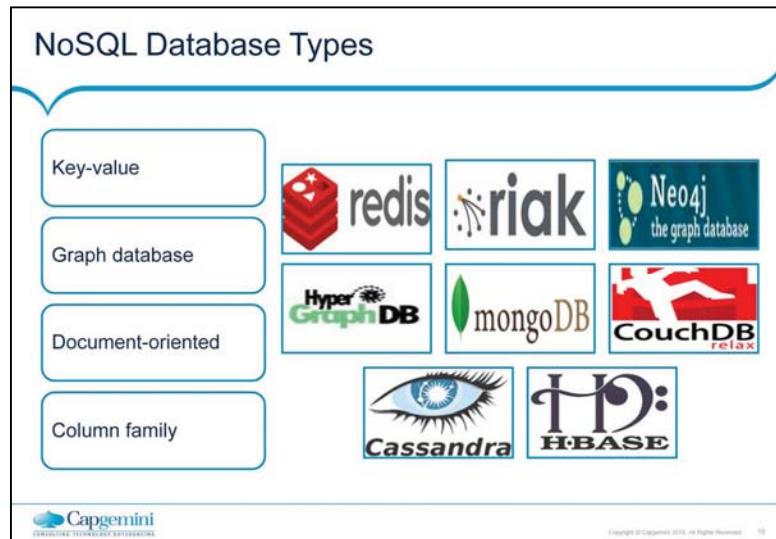
## What is NoSQL?

- Stands for 'Not Only SQL'.
- Originally refers to "non SQL" or "non Relational" database.
- Term coined by Carlo Strozzi in 1998.
- Open Source.
- No Rows-Columns / Tables.
- No Predefined schema.
- Eventually consistency rather than ACID property.
- Distributed computing.
- Unstructured and unpredictable data.
- Prioritizes high scalability ,high availability and scalability.
- Replication support .



Copyright © Capgemini 2017. All Rights Reserved 14

## MongoDB – Overview



- Huge quantity of data => Distributed systems => expensive joins =>
- New fields, new demands (graphs) =>

Different data structures:  
Simpler or more specific

## CAP Theorem

### Consistency

- This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.

### Availability

- This means that the system is always on (service guarantee availability), no downtime.

### Partition Tolerance

- This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

## MongoDB – Overview

# NO SQL Databases Types

### Key Value Stores

**row-store**

ID | Name | City | Country | Order\_No

1 | John | New York | USA | 1000

2 | Jane | London | UK | 1001

3 | Carl | Berlin | Germany | 1002

4 | Paul | Paris | France | 1003

**column-store**

ID | Name | City | Country | Order\_No

1 | John | New York | USA | 1000

2 | Jane | London | UK | 1001

3 | Carl | Berlin | Germany | 1002

4 | Paul | Paris | France | 1003

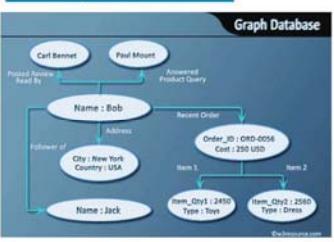
- + easy to add/modify a record
- might read in unnecessary data

- + only need to read in relevant data
- tuple writes require multiple accesses

>> suitable for read-mostly, read-intensive, large data repositories

### Graph Database

**Graph Database**



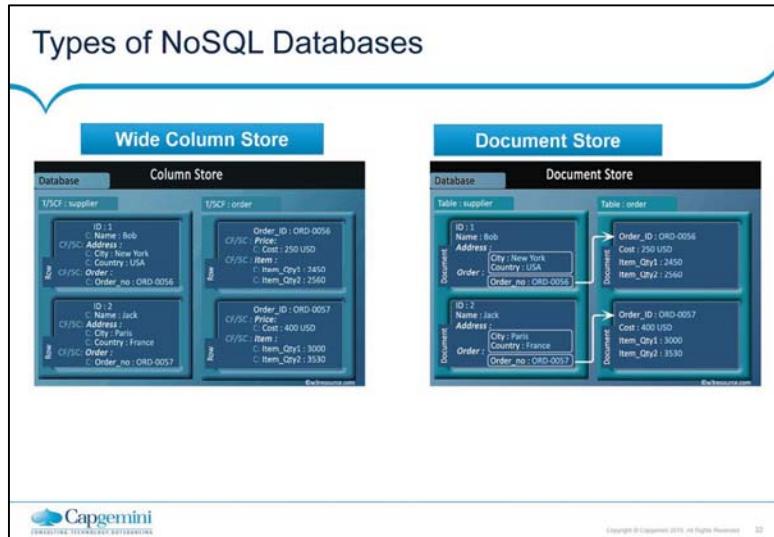
The diagram illustrates a graph database structure with the following components:

- Nodes:** Carl\_Benjet, Paul\_Mount, Name : Bob, Name : Jack, City : New York, Country : USA.
- Relationships:**
  - Carl\_Benjet → Paul\_Mount: "Passed Review - Read By"
  - Paul\_Mount → Answered Product Query: "Answered Product Query"
  - Name : Bob → Name : Jack: "Follower of"
  - Name : Bob → City : New York: "Address"
  - City : New York → Country : USA: "Country"
  - Order\_ID : ORD-0056 → Item\_1: "Item 1"
  - Item\_1 → Item\_2: "Item 2"
  - Item\_2 → Item\_City1 : 2450: "Item\_City1 : 2450"
  - Item\_2 → Item\_City2 : 2560: "Item\_City2 : 2560"
  - Item\_City1 : 2450 → Type : Toy: "Type : Toy"
  - Item\_City2 : 2560 → Type : Dress: "Type : Dress"
- Labels:** The nodes are labeled with their names and properties (e.g., Name : Bob, Order\_ID : ORD-0056).

Capgemini  
www.capgemini.com

Copyright © Capgemini 2017. All Rights Reserved. 31

Page 01-21



## Production Deployment



Google



Facebook



Mozilla



Adobe



Foursquare



LinkedIn



Digg



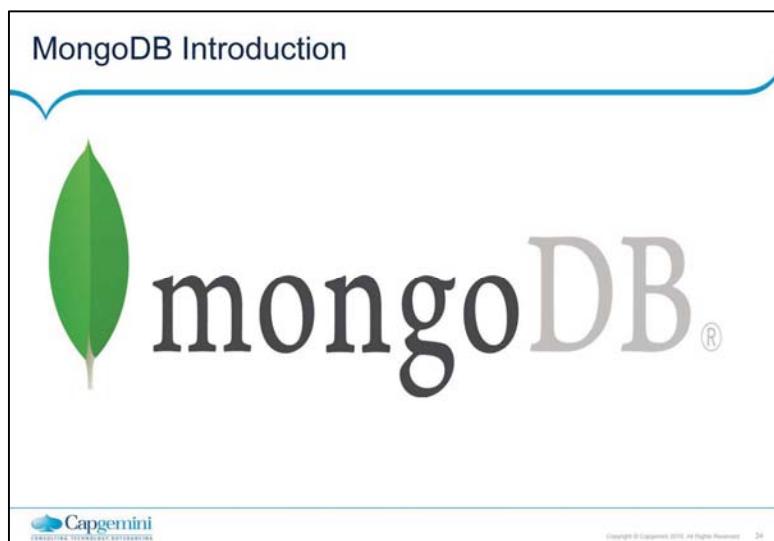
McGraw-Hill  
Education



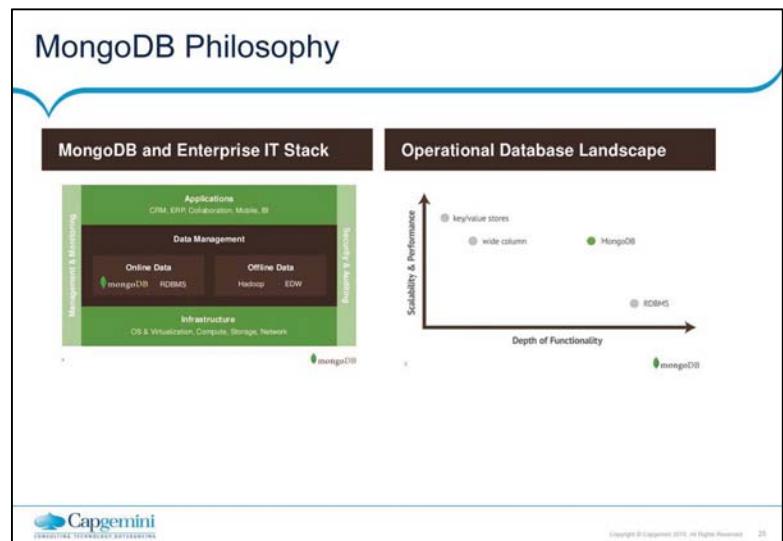
Vermont  
Public Radio



Copyright © Capgemini 2012. All Rights Reserved. 23



## MongoDB – Overview



## MongoDB – Overview

### What is MongoDB

**MongoDB** (from *humongous*) is a cross-platform document oriented database

Has driver to all most every popular language programming

10gen Inc. also offers professional services around MongoDB

Open-source project that mainly driven by 10gen Inc.

Written in C++

Schema-free document database

Nosql database



**Capgemini**  
PERFORMANCE INNOVATION SUSTAINABILITY

Copyright © Capgemini 2012. All Rights Reserved. 04

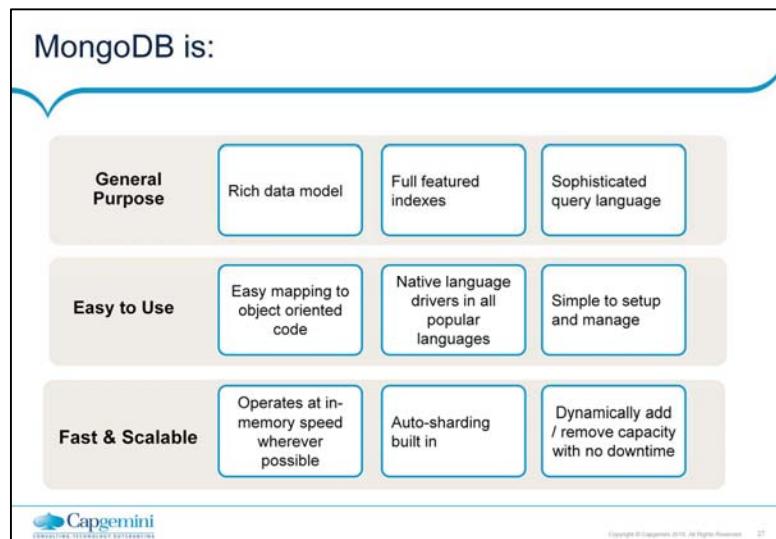
MongoDB (from *humongous*) is a cross-platform document oriented database . Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON -like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open-source software

MongoDB stores data in the form of documents, which are JSON-like field and value pairs.

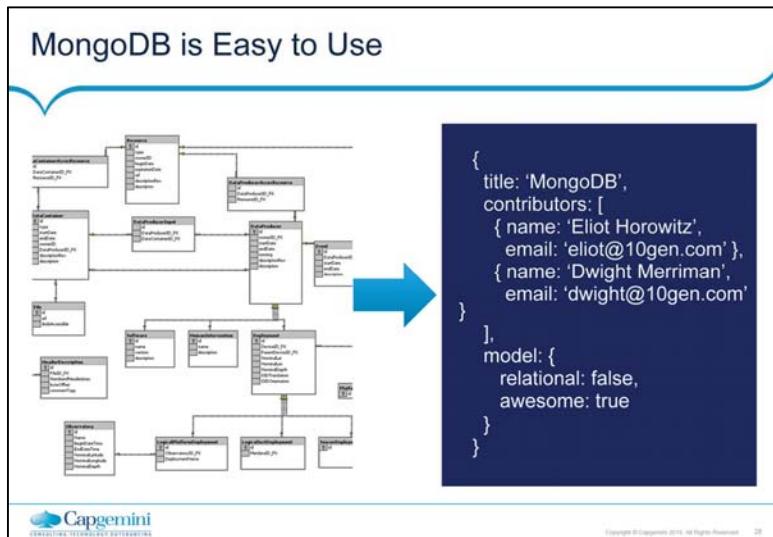
Documents are analogous to structures in programming languages that associate keys with values (e.g. dictionaries, hashes, maps, and associative arrays).

Formally, MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents

## MongoDB – Overview



## MongoDB – Overview



## MongoDB – Overview

### Schema Free

MongoDB does not need any pre-defined data schema.

Every document could have different data!

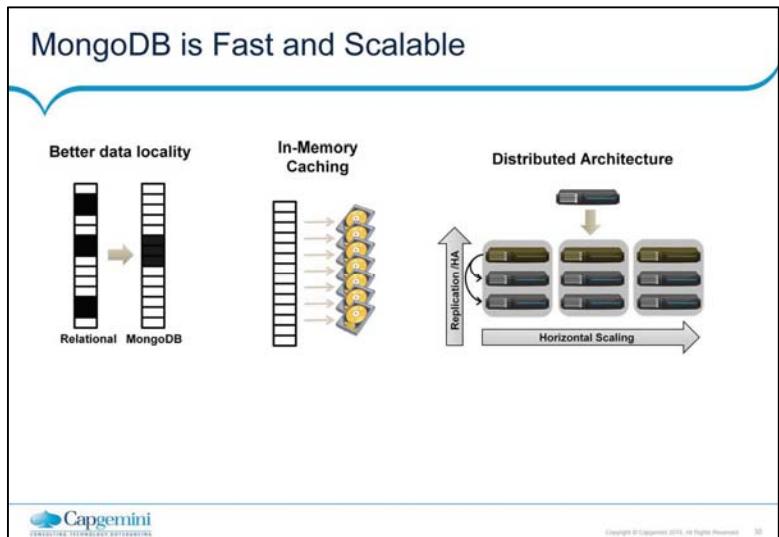
```
{name: "will", eyes: "blue", birthplace: "NY", aliases: ["bill", "la ciacco"], loc: [32.7, 63.4], boss: "ben"}  
{name: "jeff", eyes: "blue", loc: [40.7, 73.4], boss: "ben"}  
{name: "brendan", aliases: ["el diablo"]}  
{name: "ben", hat: "yes"}  
{name: "matt", pizza: "DiGiorno", height: 72, loc: [44.6, 71.3]}
```

Capgemini

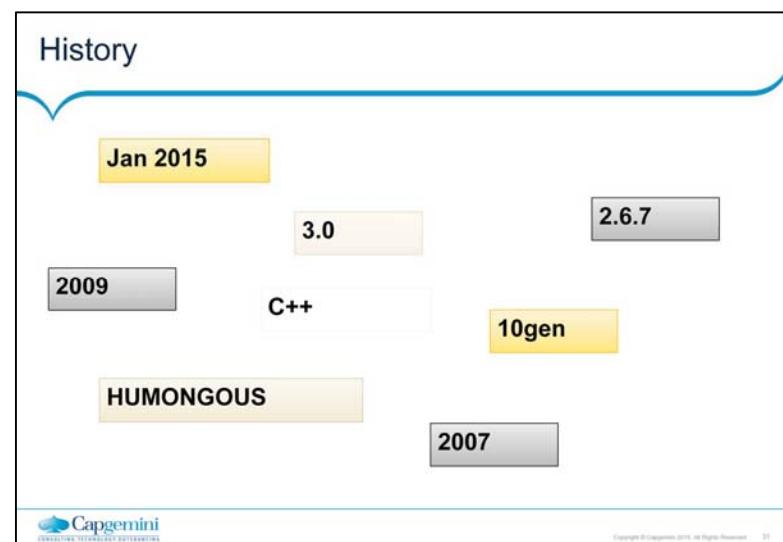
Copyright © Capgemini 2014. All Rights Reserved 29

Applications enforce the data “schema” and integrity, much like MUMPS does in Vista

## MongoDB – Overview



## MongoDB – Overview



## Features of MongoDB

- Document Oriented Database
- Adhoc queries
- Indexing
- High Performance
- High Availability
- Sharding
- Easy Scalability
- File Storage
- Rich Query Language
- Load Balancing
- Replication

 Capgemini  
CONSULTING TECHNOLOGY SYSTEMS

Copyright © Capgemini 2017. All Rights Reserved. 32

Document-oriented

Documents (objects) map nicely to programming language data types

Embedded documents and arrays reduce need for joins

Dynamically-typed (schemaless) for easy schema evolution

No joins and no multi-document transactions for high performance and easy scalability

High performance

No joins and embedding makes reads and writes fast

Indexes including indexing of keys from embedded documents and arrays

Optional streaming writes (no acknowledgements)

High availability

Replicated servers with automatic master failover

Easy scalability

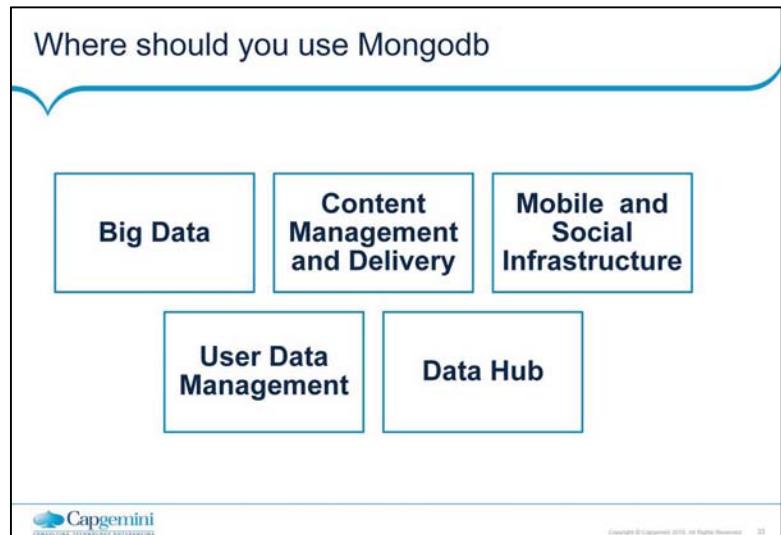
Automatic sharding (auto-partitioning of data across servers)

Reads and writes are distributed over shards

No joins or multi-document transactions make distributed queries easy and fast

Eventually-consistent reads can be distributed over replicated servers

Rich query language



Document-oriented

Documents (objects) map nicely to programming language data types

Embedded documents and arrays reduce need for joins

Dynamically-typed (schemaless) for easy schema evolution

No joins and no multi-document transactions for high performance and easy scalability

High performance

No joins and embedding makes reads and writes fast

Indexes including indexing of keys from embedded documents and arrays

Optional streaming writes (no acknowledgements)

High availability

Replicated servers with automatic master failover

Easy scalability

Automatic sharding (auto-partitioning of data across servers)

Reads and writes are distributed over shards

No joins or multi-document transactions make distributed queries easy and fast

Eventually-consistent reads can be distributed over replicated servers

Rich query language

## MongoDB – Overview

### MongoDB to SQL Terminology

MongoDB	SQL
database	database
collection	table
document	record (row)
field	column
linking/embedded documents	join
primary key (_id field)	primary key (user designated)
index	index



Copyright © Capgemini 2017. All Rights Reserved 34

## Important Terminology

### Database

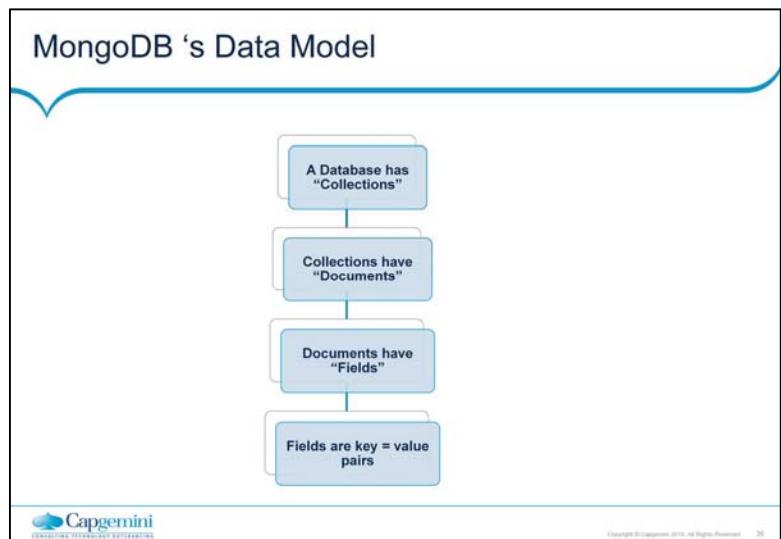
- Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

### Collection

- Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

### Document

- A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.



## Data Model

Document based (max 16 MB).

Documents are in BSON formats consisting of field / value pairs.

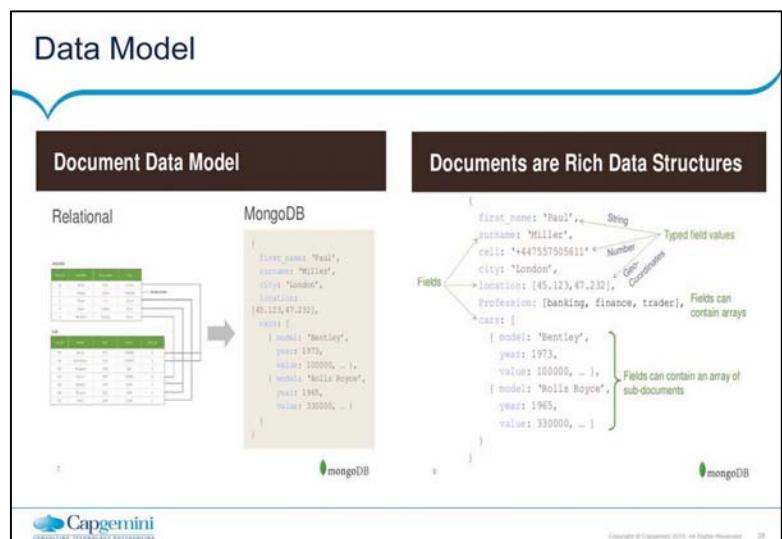
Each document stored in a collection.

Schema less.

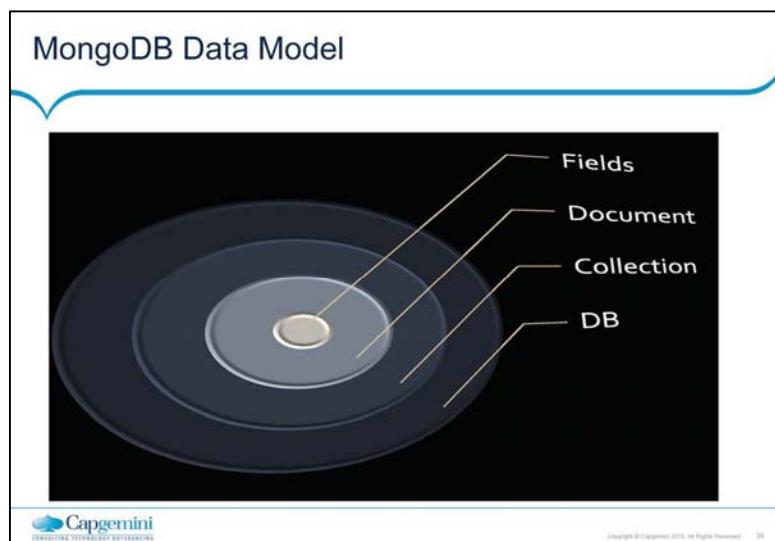


Copyright © Capgemini 2017. All Rights Reserved. 37

## MongoDB – Overview



## MongoDB – Overview



## The Basics of MongoDB

A MongoDB instance may have one or more Databases.

A database may have one or more Collections.

A collection may have zero or more Documents.

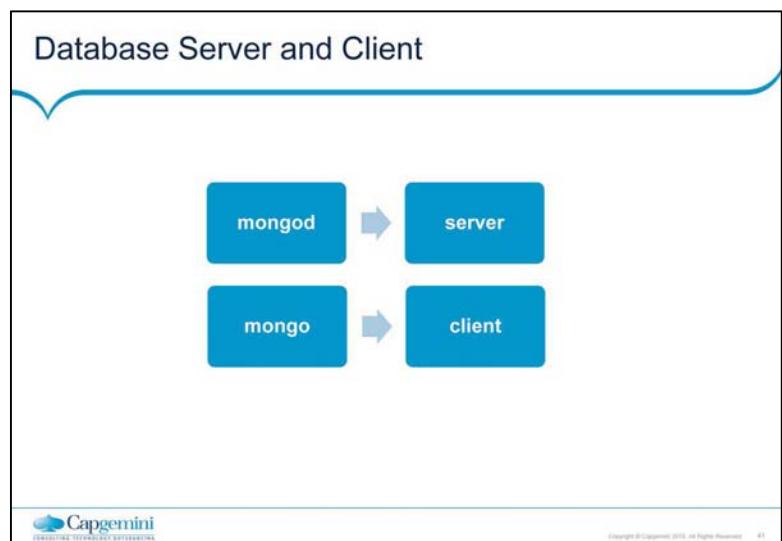
A document may have one or more Fields.

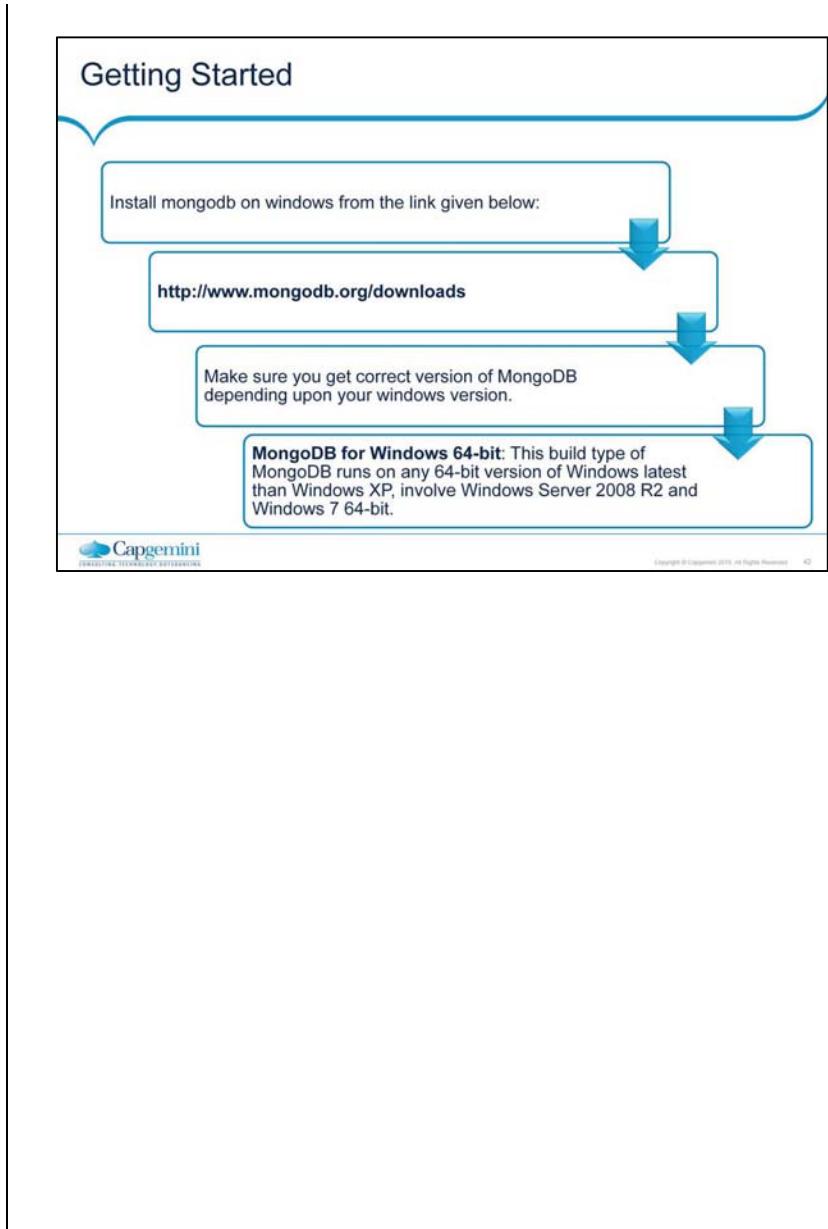
MongoDB indexes function much like their RDBMS counterparts.



Copyright © Capgemini 2017. All Rights Reserved. 40

## MongoDB – Overview





## What MongoDB does, How it works

MongoDB is a server process that runs on Windows/Linux , Os X.

It can be run both as a 32 or 64-bit application. We recommend running in 64-bit mode, since Mongo is limited to a total data size of about 2GB for all databases in 32-bit mode.

Clients connect to the MongoDB process, optionally authenticate themselves if security is turned on, and perform a sequence of actions, such as inserts, queries and updates.



Copyright © Capgemini 2017. All Rights Reserved. 41

## Starting the MongoDB Server

Create a directory where MongoDB stores all its data.

The MongoDB default data directory path is \data\db.

Create the data folder in D:\

Set the Path.

Run mongod.exe

To start MongoDB server, we need to run mongod.exe



Copyright © Capgemini 2017. All Rights Reserved. 44

## Starting the MongoDB Server (contd.)

```
D:\set up\mongodb>mongod.exe --dbpath "d:\set up\mongodb\data"
```

This will show **waiting for connections** message on the console output indicates that the mongod.exe process is running successfully.

Now to run the mongodb you need to open another command prompt and issue the following command.



Copyright © Capgemini 2017. All Rights Reserved. 45

## Starting the MongoDB Server (contd.)

```
D:\set up\mongodb\bin>mongo.exe
```

```
MongoDB shell version: 2.2.0  
connecting to: test  
Welcome to the MongoDB shell
```



Copyright © Capgemini 2017. All Rights Reserved. All

## MongoDB – Overview

The screenshot shows the MongoDB shell interface. On the left, a sidebar lists common commands: D>mongo, >help(), >show dbs, >use <dbname>, >show collections, >db.collectionName.findOne(), >db.collectionName.find(), >db.help(), and >db.collectionName.help(). The main pane displays the results of these commands. The output includes:

```
(8) C:\app\ervers\mongo-1.6.3\bin>mongo.exe
MongoDB shell version: 1.6.3
connecting to test
> show dbs
admin 0.000GB
local 0.000GB
mongorecks
test 0.000GB
> use mongorecks
switched to db mongorecks
> show collections
people
system.indexes
> db.people.findOne()
{
  "_id" : ObjectId("4cb66dae636ac4fa2045ff31"),
  "COUNTER" : NumberLong(1),
  "NAME" : "Marc",
  "DESCRIPTION" : "Crazy",
  "LOVESGIRL" : true,
  "KIDS" : [
    {
      "NAME" : "Alesx",
      "AGE" : NumberLong(2),
      "DESCRIPTION" : "crazy",
      "HAIR" : "blonde"
    },
    {
      "NAME" : "Sidney",
      "AGE" : NumberLong(2),
      "DESCRIPTION" : "crazy",
      "HAIR" : "blonde"
    }
  ],
  "MIXED" : "Heather",
  "TS" : Wed Oct 13 2010 22:40:46 GMT-0400 (Eastern Daylight Time)"
}
```

At the bottom of the interface, there is a Capgemini logo and a copyright notice: Copyright © Capgemini 2011. All Rights Reserved.

## Summary

- What is NOSQL database
- Advantages of NOSQL
- Why MongoDB
- MongoDB Document database
- MongoDB data model
- Mongo Shell
- Establishing Connection
- Understand about Collection, document and fields



Copyright © Capgemini 2017. All Rights Reserved. 48

MongoDB – CRUD

## **MongoDB – CRUD**

Lesson 02

## MongoDB – CRUD

**Version Sheet: MANDATORY (hidden in ‘slide show’ mode)**

Version	Changes	Author
001	Redesign	Rohan Salvi

 Capgemini  
CONSULTING TECHNOLOGY ENTERPRISE

Copyright © Capgemini 2011. All Rights Reserved.

## Note to the SME:

Please read before you begin reviewing this module as SME

Dark Red box with white text

Note to the SME

The SME must provide missing info.

Amber box with black text

Note to the SME

The SME must validate the re-design/ the modification/ addition.

Note to the SME  
This is a temporary slide and will not be part of the final upload at all. It is to help the SME understand how we will communicate changes to them.

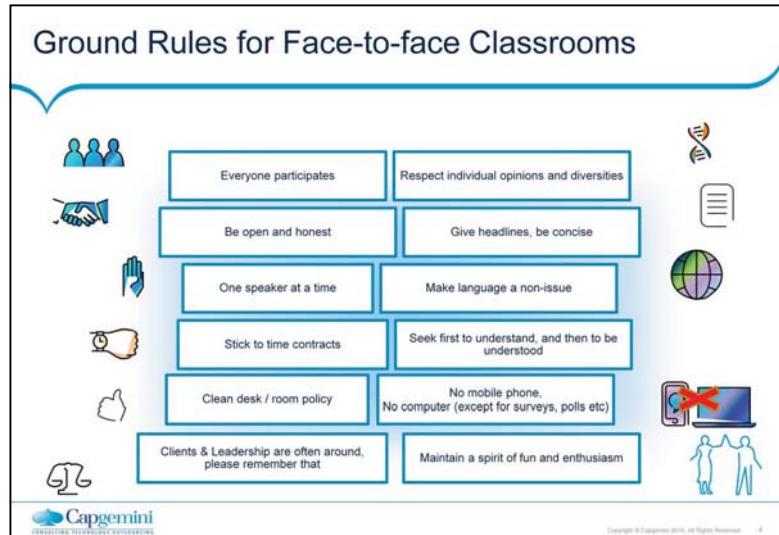
To Review and Navigate the comments in the presentation Click on the "Review" Tab and then Click "Next" to review all the comments . Also you can add your comments using the "New comment" button

The comments in the review section have also been updated in a green textbox with black text.



Copyright © Capgemini 2014 - All Rights Reserved

## MongoDB – CRUD



Best to print on an A3 sheet and display in class. The idea is to co-create, to connect to learn, to do-learn-do (knowing is not enough, the real purpose is to apply what we learn on the job). It would also be great to have the class share their learning with their teams. You are the facilitator, you will help all share knowledge and learn new concepts or skills, there will be no lectures, everyone is a teacher.

## Ground Rules for Virtual Classrooms

**Participate actively in each session**

- Share experiences and best practices
- Bring up challenges, ask questions
- Discuss successes
- Respond to whiteboards, polls, quizzes, chat boxes
- Hang up if you need to take an urgent phone call, don't put this call on hold

**Communicate professionally with others**

- Mute when you're not speaking
- Wait for others to finish speaking before you speak
- Each time you speak, state your name
- Build on others' ideas and thoughts
- Disagreeing is OK –with respect and courtesy

**Be on time for each virtual session**

- As a best practice...be just a few minutes early!

Copyright © Capgemini 2014. All Rights Reserved.

Show this slide and hide the one for onsite classrooms, if this is a virtual session.

ONLY for Virtual classrooms

## Module at a Glance

<b>Target Audience:</b>	SME to provide the details required in the table.
<b>Course Level:</b>	Basic
<b>Duration (in hours):</b>	30 mins
<b>Pre-requisites, if any:</b>	NA
<b>Post-requisites, if any:</b>	<i>Submit Session Feedback</i>
<b>Relevant Certifications:</b>	None



Copyright © Capgemini 2010. All Rights Reserved.

## MongoDB – CRUD

### Introductions (for Virtual Classrooms)

SME to provide the photos and names of the facilitators.

Business Photo

Business Photo

Facilitator  
Name  
Role

Moderator  
Name  
Role

 Capgemini  
CONSULTING IT SERVICES

Copyright © Capgemini 2014 - All Rights Reserved

Add pics and Capgemini roles for the Facilitator and moderator. Establish their credibility, what qualifies them to facilitate this session.

## Agenda

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>1    CRUD</li><li>2    Update with Options</li><li>3    Upsert</li><li>4    Behavior of Upsert</li><li>5    MongoDB- Projection</li><li>6    Query Interface</li></ul> | <ul style="list-style-type: none"><li>7    Queries in MongoDB</li><li>8    Comparison Operators</li><li>9    Logical Operators</li><li>10   Comparison Operators</li><li>11   Wrapped Queries</li><li>12   Query Operators</li></ul> |
|--|--|



Copyright © Capgemini 2014 - All Rights Reserved

## Module Objectives



### What you will learn

At the end of this module, you will learn:

- What are Crud Operations

Note to the SME : Please provide the module Objectives or validate the partially updated content

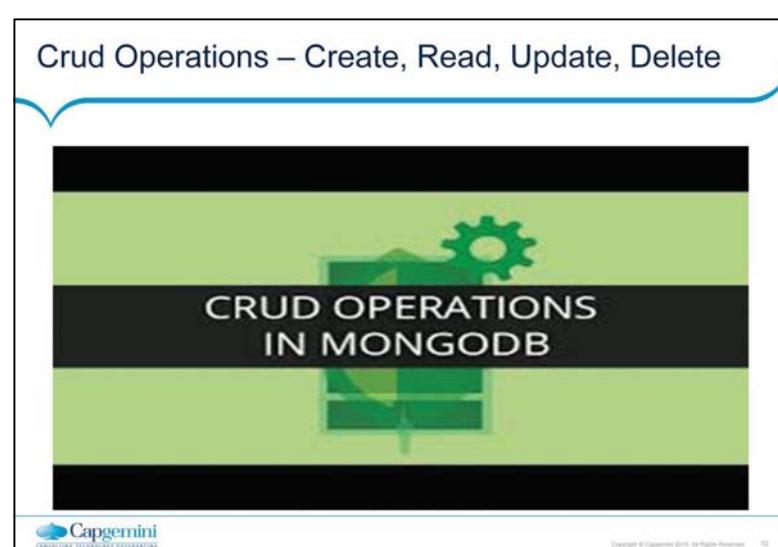


### What you will be able to do

At the end of this module, you will be able to:

- Understand what are Crud Operations
- Explain what is Upsert
- Describe Query Interface
- List the Comparison Operators and Logical Operators
- State what are Wrapped Queries and Query Operators

## MongoDB – CRUD



## MongoDB – CRUD

# CRUD

- Create**
  - db.collection.insert( <document> )
  - db.collection.save( <document> )
  - db.collection.update( <query>, <update>, { upsert: true } )
- Read**
  - db.collection.find( <query>, <projection> )
  - db.collection.findOne( <query>, <projection> )
- Update**
  - db.collection.update( <query>, <update>, <options> )
- Delete**
  - db.collection.remove( <query>, <justOne> )

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2010. All Rights Reserved.

### Create

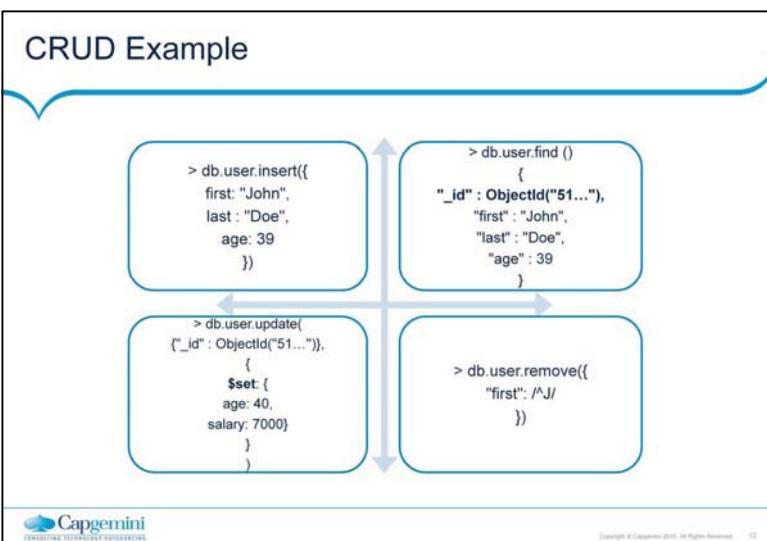
- The field name `_id` is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names **cannot** start with the `$` character.
- The field names **cannot** contain the `.` character.

### Create with save

- If the `<document>` argument does not contain the `_id` field or contains an `_id` field with a value not in the collection, the [`save\(\)`](#) method performs an insert of the document.
- Otherwise, the [`save\(\)`](#) method performs an update.

sds

## MongoDB – CRUD



## Insert Document – insert() method

To insert data into MongoDB collection, we have a method called insert() or save() method.

Basic syntax of **insert()** command is as follows:

- >db.COLLECTION\_NAME.insert(document)

Example:

- db.tutorialspoint.insert({ "name": "tutorialspoint" })
- db.nextTable.save({ column1: "value", column2: "value", ... })'



Copyright © Capgemini 2010. All Rights Reserved.

## Update document – update() method

MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method update values in the existing document while the save() method replaces the existing document with the document passed in save() method.

The update() method updates values in the existing document.

Example:

- >use mydb
- >switched to db mydb >db.dropDatabase()
- >{ "dropped" : "mydb", "ok" : 1 }



Copyright © Capgemini 2010. All Rights Reserved.

## Update with Options

**Update (document)** – Specifies the modifications to apply.

If the **update** parameter contains any update operators expressions such as the \$set operator expression, then:

- The update parameter must contain only update operators expressions.
- The update method updates only the corresponding fields in the document.

If the **update** parameter consists only of field: value expressions, then:

- The update method replaces the document with the updates document. If the updates document is missing the **\_id field**, MongoDB will add the **\_id** field and assign to it a unique object Id .
- The update method updates cannot update multiple documents.

**Options** : (optional) Specifies whether to perform an upsert and/or a multiple update. Use the options parameter instead of the individual upsert and multi parameters.



Copyright © Capgemini 2014 - All Rights Reserved

## Update Example

To update multiple you need to set a parameter 'multi' to true:

- >db.mycol.update({title:'MongoDB Overview'}, {\$set:{title:'New MongoDB Tutorial'}},{multi:true})



Copyright © Capgemini 2014 - All Rights Reserved

## Upsert

**Upsert:** A kind of update that either updates the first document matched in the provided query selector or, if no document matches, inserts a new document having the fields implied by the query selector and the update operation. The default value is false. When true, the update() method will update an existing document that matches the query selection criteria or if no document matches the criteria, insert a new document with the fields and values of the update parameter and if the update included only update operators, the query parameter as well.

**Multi** (optional): Specifies whether to update multiple documents that meet the query criteria.

When not specified, the default value is false and the update() method updates a single document that meet the query criteria.

When true, the update() method updates all documents that meet the query criteria.



Copyright © Capgemini 2010 - All Rights Reserved

## Behavior of Upsert

Upsert will update the field for an already existing document or it would insert the document if it does not exist:

- db.people.update({name : "Maxwell"},{\$set : { age : 30 }},{upsert : true} )



Copyright © Capgemini 2010. All Rights Reserved.

## Remove document – remove() method

MongoDB's **remove()** method is used to remove document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

**Deletion criteria:** (Optional) deletion criteria according to documents will be removed.

**justOne:** (Optional) if set to true or 1, then remove only one document.

- db.collection\_name.remove(DELETION\_CRITERIA)
- db.mycol.remove({'title':'MongoDB Overview'})



Copyright © Capgemini 2014 - All Rights Reserved

## Remove document – remove only one

If there are multiple records and you want to delete only first record, then set **justOne** parameter in **remove()** method

```
db.collection_name.remove(deletion_criteria,1)
```

### Remove All documents

If you don't specify deletion criteria, then mongodb will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

- db.mycol.remove()
- >db.mycol.find()



Copyright © Capgemini 2010. All Rights Reserved.

## MongoDB - Projection

### The **find()** Method

In MongoDB when you execute **find()** method, then it displays all fields of a document. To limit this you need to set list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the field.

- db.COLLECTION\_NAME.find({}, {KEY:1})
- db.mycol.find({}, {"title":1, \_id:0})

if you don't want this field, then you need to set it as 0



Copyright © Capgemini 2014. All Rights Reserved.

## Query document – find() method

To query data from MongoDB collection, you need to use MongoDB's **find()** method

Basic syntax of **find()** method is as follows

>db.COLLECTION\_NAME.find() **find()** method will display all the documents in a non structured way

The **pretty()** Method

To display the results in a formatted way, you can use **pretty()** method

- db.mycol.find().pretty()



Copyright © Capgemini 2014 - All Rights Reserved

## MongoDB – CRUD

### Query Interface

db.users.find (		Collection
{ age : { \$gt : 18} },		Query Criteria
{ name : 1, address :1 }		Projection
) . Limit (5)		Cursor modifier

This query selects the documents in the users collection that match the condition age is greater than 18.

The query returns at most 5 matching documents (or more precisely, a cursor to those documents).

The matching documents will return with only the \_id, name and address fields.

 Capgemini  
CONSULTING TECHNOLOGY SERVICES

Copyright © Capgemini 2010 - All Rights Reserved

## Queries in MongoDB

Query expression objects indicate a pattern to match

- `db.users.find( {last_name: 'Smith'} )`

Several query objects for advanced queries

- `db.users.find( {age: {$gte: 23} } )`
- `db.users.find( {age: {$in: [23,25]} } )`

Exact match an entire embedded object

- `db.users.find( {address: {street: 'Oak Terrace',city: 'Denton'}} )`

Dot-notation for a partial match

- `db.users.find( {"address.city": 'Denton'} )`



Copyright © Capgemini 2014 - All Rights Reserved

## Comparison Operators

Operators	Description
\$eq	Matches values that are equal to a specified value.
\$gt	Matches values that are greater than a specified value.
\$gte	Matches values that are greater than or equal to a specified value.
\$lt	Matches values that are less than a specified value.



Copyright © Capgemini 2010. All Rights Reserved.

## Comparison Operators

Operators	Description
\$lte	Matches values that are less than or equal to a specified value.
\$ne	Matches all values that are not equal to a specified value.
\$in	Matches any of the values specified in an array.
\$nin	Matches none of the values specified in an array.



Copyright © Capgemini 2014 - All Rights Reserved

## Logical Operators

Name	Description
\$lte	Matches values that are less than or equal to a specified value.
\$ne	Matches all values that are not equal to a specified value.
\$in	Matches any of the values specified in an array.
\$nin	Matches none of the values specified in an array.



Copyright © Capgemini 2014 - All Rights Reserved

## MongoDB – CRUD

### Title

Select \* from users where age > 33  
db.users.find({age:{\$gt:33}})

SME to provide  
Slide Title

Select \* from users where age!=33

db.users.find({age:{\$ne:33}})

Select \* from users where name like "%Joe%"

db.users.find({name:/Joe/})

SELECT \* FROM users WHERE a=1 and b='q'

db.users.find({a:1,b:'q'})

SELECT \* FROM users WHERE a=1 or b=2

db.users.find( { \$or : [ { a : 1 } , { b : 2 } ] })



Copyright © Capgemini 2010. All Rights Reserved.

## Title

Select name from emp where sal > 10000

SME to provide  
Slide Title

db.emp.find({sal : {\$gt :10000 }} )

Select name from emp where sal <=50000

db.emp.find({sal : {lte :50000}} )

Select \* from emp where salary = 2000 and age =26

db.emp.find({ \$and : [{salary :2000},{age : 26 }]}))

Select \* from emp where salary =2000 or age =26

db.emp.find({"\$or": [{"salary":2000}, {"age":26}]}))



Copyright © Capgemini 2014 - All Rights Reserved

## Awesome: Query Operators

\$ne:

- db.people.find( { NAME: { \$ne: "Shaggy"} } )

\$nin:

- db.people.find( { NAME: { \$nin: ["Shaggy", "Daphne"] } } )

\$gt and \$lt:

- db.people.find( AGE: { {\$gt: 30, \$lt: 35} } )

\$size (eg, people with exactly 3 kids):

- db.people.find( KIDS: { \$size: 3} )

regex: (eg, names starting with Ma or Mi)

- db.people.find( {NAME: /^M(a|j)/} )



Copyright © Capgemini 2014 - All Rights Reserved

## Wrapped Queries

### Like Query:

- db.Tag.find( {name:/^term/} )

### Sort Query:

- 1 for ascending sort
- -1 for descending sort
- db.Tag.find().sort( {userName:-1, age:1} );

### Limit Query:

- db.Tag.find().limit(10);



Copyright © Capgemini 2014. All Rights Reserved.

## Wrapped Queries (Contd.)

### Count Query:

- db.Tag.find().count();

### Skip Query:

- db.Tag.find().skip(50);

## Query Using Modifiers

### 1. Not Equal Modifier(\$ne):

- db.Tag.find({firstProperty : {\$ne : 3}});

### 2. Greater/Less than Modifier(\$gt, \$lt, \$gte, \$lte):

- db.Tag.find({firstProperty : {\$gt : 2}});
- db.Tag.find({firstProperty : {\$lt : 2}});
- db.Tag.find({firstProperty : {\$gte : 2}});
- db.Tag.find({firstProperty : {\$lte : 2}});



Copyright © Capgemini 2014 - All Rights Reserved

## Query Using Modifiers (Contd.)

### 3. Increment Modifier(\$inc):

- db.Tag.update( {thirdProperty:"Hello19"}, {"\$inc": {firstProperty:2}} )

### 4. Set Modifier(\$set):

- db.Tag.update( {thirdProperty:"Hello19"}, {"\$set": {fourthProperty: "newValue"} } )

NOTE: Key-value will be created if the key doesn't exist yet in the case of both \$set and \$inc. \$inc works for integers only. \$set works for all. \$inc Incrementing is extremely fast, so any performance penalty is negligible.

## Query Using Modifiers (Contd.)

### 5. Unset Modifier(\$unset):

- db.Tag.update( {thirdProperty:"Hello19"}, {"\$unset": {fourthProperty: "anything"} } )

### 6. Push Modifier(\$push):

- db.Blog.update( {title:"1st Blog"}, { \$push: {comment:"1st Comment"} } );

NOTE: "\$push" adds an element to the end of an array if the specified key already exists and creates a new array if it does not.



Copyright © Capgemini 2014 - All Rights Reserved

## Query Using Modifiers (Contd.)

### 7. AddToSet Modifier(\$addToSet):

- db.Blog.update( {title:"1st Blog"}, { \$addToSet: {comment:"2nd Comment"} } );

### 8. Pop Modifier(\$pop):

- comment = -1 remove an element from start.
- comment = 1 remove an element from end
- db.Blog.update( {title:"1st Blog"}, {\$pop:{comment:-1}} );



Copyright © Capgemini 2014. All Rights Reserved.

## Query Using Modifiers (Contd.)

### 9. Pull Modifier(\$pull):

- db.Blog.update({title:"1st Blog"}, {\$pull:{comment:"2st Comment"}})

NOTE: Pulling removes all matching documents, not just a single match.

### 10. Position Operator(\$): The positional operator updates only the first match.

- db.embededDocument.update({"comments.author" : "John"}, {"\$inc" : {"comments.0.votes" : 1}})
- db.embededDocument.update({"comments.votes" : 3}, {"\$set" : {"comments.\$.author" : "Amit Kumar"}})



Copyright © Capgemini 2014. All Rights Reserved.

## OR Queries

There are two ways to do an OR query.

1. "\$in" can be used to query for a variety of values for a singlekey.

- db.Blog.find( {comment: {\$in:["5th Comment", "1st Comment"]}} )

2. "\$or" is more general; it can be used to query for anyof the given values across multiple keys.

- db.Tag.find( {\$or: [ {firstProperty:0}, {secondProperty:/J/} ] } )
- NOTE:  
The opposite of "\$in" is "\$nin".



Copyright © Capgemini 2014 - All Rights Reserved

## AND Queries

There are two ways to do an AND query.

1. "\$all" can be used to query for a variety of values for a singlekey.

- db.Blog.find( {comment:{\$all:["5th Comment", "1st Comment"]}} )

NOTE: Below will do exact match and order too matters, if order will change then it will not match.

- db.Blog({"comment" : ["5th Comment", "comment1st Comment"]})



Copyright © Capgemini 2010. All Rights Reserved.

## AND Queries (Contd.)

Simple queries are and queries.

It can be used to query for anyof the given values across single/multiple keys.  
`db.Tag.find( {firstProperty:0, secondProperty:/J/} )`

- `db.Blog.find({comment:"5th Comment",comment:"1stComment"})`



Copyright © Capgemini 2014 - All Rights Reserved

## Querying on Embedded Document

There are two ways of querying for an embedded document.

1. Querying for an entire embedded document works identically to a normal query.

- db.User.find( {name: { first:"Amit", last:"Kumar" } } )
- This query will do exact match and order too matters, if order will change then it will not find.

2. Querying for its individual key/value pairs.

- db.User.find({ "name.first" : "Amit", "name.last" : "Kumar"})



Copyright © Capgemini 2014 - All Rights Reserved

## Awesome: Atomic Modifiers

### \$inc

- db.pageviews.update( {URL: 'http://myurl.com'}, {\$inc: {N: 1}} )

### \$set

- db.people.update( {NAME: 'Steve'}, {\$set: {Age: 35}} )

### \$push (for atomically adding values to an array)

- db.people.update( {NAME: 'Steve'}, {\$push: {KIDS: {NAME: 'Sylvia', AGE: 3}}})

### findAndModify()

- db.tasks.findAndModify(  
query: {STATUS: 'pending'},  
sort: {PRIORITY: -1},  
update: {\$set: {STATUS: 'running', TS: new Date()}}  
)



Copyright © Capgemini 2010. All Rights Reserved.

## Awesome: Atomic Modifiers (Contd.)

SELECT \* FROM foo WHERE name='bob' and (a=1 or b=2 )

- db.foo.find( { name : "bob" , \$or : [ { a : 1 } , { b : 2 } ] } )

SELECT \* FROM users WHERE age>33 AND age<=40

- db.users.find({age:{\$gt:33,\$lte:40}})



Copyright © Capgemini 2014. All Rights Reserved.

## Query Behavior

All queries in MongoDB address a single Collection

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()

Operations that modify existing documents (i.e. updates) use the same query syntax as queries to select documents to update

In aggregation pipeline , the \$match pipeline stage provides access to MongoDB queries



Copyright © Capgemini 2014 - All Rights Reserved

## The limit() & Skip method()

To limit the records in MongoDB, you need to use **limit()** method. **limit()** method accepts one number type argument, which is number of documents that you want to displayed.

- db.mycol.find({},{ "title":1, "\_id":0 }).limit(2) { "title": "MongoDB Overview" }  
{"title": "NoSQL Overview"}

### SKIP method()

Apart from limit() method there is one more method **skip()** which also accepts number type argument and used to skip number of documents.

- db.mycol.find({},{ "title":1, "\_id":0 }).limit(1).skip(1)

default value in **skip()** method is 0

## Sort Method()

To sort documents in MongoDB, you need to use **sort()** method. **sort()** method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order

- db.COLLECTION\_NAME.find().sort({KEY:1})
- db.mycol.find({}, {"title":1,\_id:0}).sort({"title":-1})

## Cursors

The database returns results from find using a cursor.

If we do not store the results in a variable, the MongoDB shell will automatically iterate through and display the first couple of documents.

When you call find, the shell does not query the database immediately.

It waits until you actually start requesting results to send the query, which allows you to chain additional options onto a query before it is performed.



Copyright © Capgemini 2014 - All Rights Reserved

## Cursors (Contd.)

Almost every method on a cursor object returns the cursor itself so that you can chain them in any order.

For instance, all of the following are equivalent

- > var cursor = db.Tag.find().sort({"x" : 1}).limit(1).skip(10);
- > var cursor = db.Tag.find().limit(1).sort({"x" : 1}).skip(10);
- > var cursor = db.Tag.find().skip(10).limit(1).sort({"x" : 1});



Copyright © Capgemini 2014 - All Rights Reserved

## Developing with MongoDB

### Behind Find() : Cursor

- The database returns results from find using a *cursor*.
- The client-side implementations of cursors generally allow you to control a great deal about the eventual output of a query.



Copyright © Capgemini 2014 - All Rights Reserved

## Developing with MongoDB (Contd.)

```
> for(i=0; i<100; i++) {  
... db.c.insert({x : i});  
... }  
> var cursor =  
db.collection.find();  
- > while  
(cursor.hasNext()) {  
- ... obj = cursor.next();  
- ... // do stuff  
- ... }
```

```
> var cursor =  
db.people.find();  
>  
cursor.forEach(function(x) {  
... print(x.name);  
... });  
adam  
matt  
zak
```



Copyright © Capgemini 2014 - All Rights Reserved

## Developing with MongoDB (Contd.)

### Behind Find() : Cursor (Contd.)

- Getting Consistent Results?
- var cursor = db.myCollection.find({country:'uk'}).snapshot();

A fairly common way of processing data is to pull it out of MongoDB, change it in some way, and then save it again:



Copyright © Capgemini 2014 - All Rights Reserved

## Developing with MongoDB (Contd.)

```
cursor = db.foo.find();
while (cursor.hasNext())
{
    var doc = cursor.next();
    doc = process(doc);
    db.foo.save(doc);
}
```

Copyright © Capgemini 2014. All Rights Reserved.

Capgemini  
EXECUTIVE PRACTICE CONSULTING

<http://www.mongodb.org/display/DOCS/How+to+do+Snapshotted+Queries+in+the+Mongo+Database>

### Snapshot Mode

`snapshot()` mode assures that objects which update during the lifetime of a query are returned once and only once. This is most important when doing a find-and-update loop that changes the size of documents that are returned (`$inc` does not change size).

```
> // mongo shell example
> var cursor = db.myCollection.find({country:'uk'}).snapshot();
```

Even with snapshot mode, items inserted or deleted during the query may or may not be returned; that is, this mode is not a true point-in-time snapshot.

Because snapshot mode traverses the `_id` index, it may not be used with sorting or explicit hints. It also cannot use any other index for the query.

You can get the same effect as snapshot by using any unique index on a field(s) that will not be modified (probably best to use explicit hint() too). If you want to use a non-unique index (such as creation time), you can make it unique by appending `_id` to the index at creation time.

## Summary

- Understand about Create a document
- Insert, Update and delete a document
- Read a document using find()
- Types of Operators
- Comparison and logical Operators
- Sort, limit and skip operators
- Cursors



 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2010. All Rights Reserved.

### Create

- The field name `_id` is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names **cannot** start with the `$` character.
- The field names **cannot** contain the `.` character.

### Create with save

- If the `<document>` argument does not contain the `_id` field or contains an `_id` field with a value not in the collection, the [save\(\)](#) method performs an insert of the document.
- Otherwise, the [save\(\)](#) method performs an update.

sds

## **MongoDB – Basic Operations**

Lesson 03

## MongoDB – Basic Operations

Version Sheet: MANDATORY (hidden in 'slide show' mode)

Version	Changes	Author
001	Redesign	Rohan Salvi

 Capgemini  
Engineering Services

Copyright © Capgemini 2014. All Rights Reserved 2

## Note to the SME:

Please read before you begin reviewing this module as SME

Dark Red box with white text

Amber box with black text

Note to the SME

The SME must provide missing info.

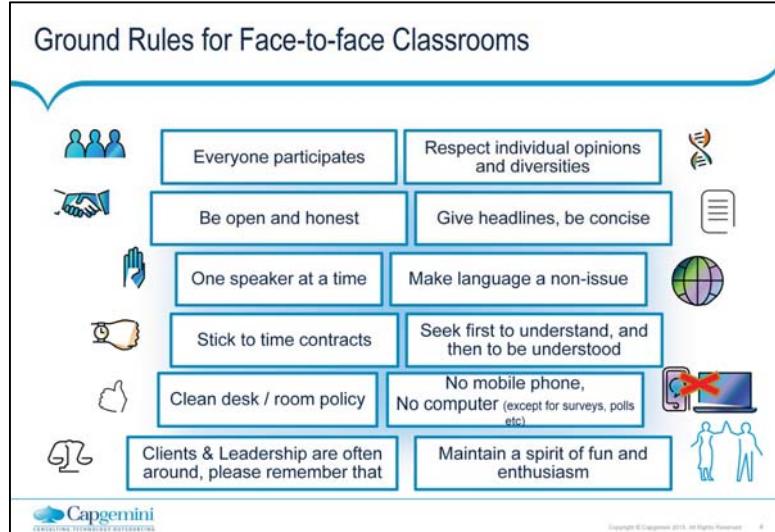
Note to the SME

The SME must validate the re-design/ the modification/ addition.

Note to the SME:  
This is a temporary slide and will not be part of the final upload at all. It is to help the SME understand how we will communicate changes to them.

To Review and Navigate the comments in the presentation Click on the "Review" Tab and then Click "Next" to review all the comments . Also you can add your comments using the "New comment" button

The comments in the review section have also been updated in a green textbox with black text.



Best to print on an A3 sheet and display in class. The idea is to co-create, to connect to learn, to do-learn-do (knowing is not enough, the real purpose is to apply what we learn on the job). It would also be great to have the class share their learning with their teams. You are the facilitator, you will help all share knowledge and learn new concepts or skills, there will be no lectures, everyone is a teacher.

## Ground Rules for Virtual Classrooms

### Participate actively in each session

- Share experiences and best practices
- Bring up challenges, ask questions
- Discuss successes
- Respond to whiteboards, polls, quizzes, chat boxes
- Hang up if you need to take an urgent phone call, don't put this call on hold

### Communicate professionally with others

- Mute when you're not speaking
- Wait for others to finish speaking before you speak
- Each time you speak, state your name
- Build on others' ideas and thoughts
- Disagreeing is OK—with respect and courtesy

### Be on time for each virtual session

As a best practice...be just a few minutes early!



Copyright © Capgemini 2014. All Rights Reserved.

Show this slide and hide the one for onsite classrooms, if this is a virtual session.

ONLY for Virtual classrooms

## Module at a Glance

**Target Audience:**

SME to provide the details required in the table.

**Course Level:** *Basic*

**Duration (in hours):** 30 mins

**Pre-requisites, if any:** NA

**Post-requisites, if any:** *Submit Session Feedback*

**Relevant Certifications:** None



Copyright © Capgemini 2014. All Rights Reserved

## Introductions (for Virtual Classrooms)

Business Photo

SME to provide the  
photos and names of  
the facilitators.

Business Photo

*Facilitator*  
**Name**  
Role

*Moderator*  
**Name**  
Role



Copyright © Capgemini 2014. All Rights Reserved.

Add pics and Capgemini roles for the Facilitator and moderator. Establish their credibility, what qualifies them to facilitate this session.

## Agenda

1 Crud Operations

2 Basic Operations With Mongo Shell

3 Data Model

4 JSON

5 BSON

6 MongoDB – Datatypes

7 BSON Types

8 The \_id Field

9 Document

10 Document Store

11 Blog: A Bad Design

12 Blog: A Better Design



Copyright © Capgemini 2014. All Rights Reserved

## Module Objectives



### What you will learn

At the end of this module, you will learn:

- The Basic Operations of MongoDB

Note to the SME : Please provide the module Objectives or validate the partially updated content



### What you will be able to do

At the end of this module, you be able to:

- Understand the Basic Operations of MongoDB
- Describe the Data Model
- State the features of JSON and BSON
- List the BSON Types
- Explain the features of Document

Crud Operations

CRUD OPERATIONS  
IN MONGODB

Capgemini

Copyright © Capgemini 2014. All Rights Reserved.

## Basic Operations with Mongo Shell

Create Database

Drop Database

Create Collection

Drop Collection

JSON, BSON Document

Datatypes



Copyright © Capgemini 2014. All Rights Reserved. 11

## MongoDB – Commands

To check your currently selected database use the command **db**.

- > db
- mydb

If you want to check your databases list, then use the command.

- > show dbs

To display the currently created database , you need to insert one document into it.

- db.movie.insert({"name":"tutorials point"})
- show dbs
- local 0.78125GB
- mydb 0.23012GB



Copyright © Capgemini 2014. All Rights Reserved

## MongoDB – Create Database

MongoDB **use DATABASE\_NAME** is used to create database on the fly at the time you use it.

The command will create a new database, if it doesn't exist otherwise it will return the existing database.

Basic syntax of **use DATABASE** statement is as follows:  
`use DATABASE_NAME`

- Example:
- > `use mydb`
- switched to db mydb



Copyright © Capgemini 2014. All Rights Reserved. 13

## MongoDB – dropDatabase()

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Basic syntax of **dropDatabase()** command is as follows:

- db.dropDatabase()

To delete new database <mydb>, then **dropDatabase()** command would be as follows:

- >use mydb
- >switched to db mydb >db.dropDatabase()
- >{ "dropped" : "mydb", "ok" : 1 }



Copyright © Capgemini 2014. All Rights Reserved. 14

## MongoDB – `createCollection()` method

MongoDB `db.createCollection(name, options)` is used to create collection. Basic syntax of `createCollection()` command is as follows:

- `db.createCollection(name, options)`

**name** is name of collection to be created. **Options** is a document and used to specify configuration of collection.

`db.createCollection("mycollection")`.

The syntax of `createCollection()` method with few important options:

- `db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 })`



Copyright © Capgemini 2014. All Rights Reserved

## MongoDB – The drop() method

Basic syntax of drop() command is:

- db.COLLECTION\_NAME.drop()

drop() method will return true, if the selected collection is dropped successfully otherwise it will return false.



Copyright © Capgemini 2014. All Rights Reserved. 10

## Data Model

Document-Based (max 16 MB)

Documents are in BSON format, consisting of field-value pairs.

Each document stored in a collection.

Collections

- Have index set in common.
- Like tables of relational db's.
- Documents do not have to have uniform structure.



Copyright © Capgemini 2014. All Rights Reserved

## JSON

"JavaScript Object Notation"

Easy for humans to write / read, easy for computers to parse / generate.

Objects can be nested.

Built on:

- Name / value pairs
- Ordered list of values



Copyright © Capgemini 2014. All Rights Reserved. 10

## BSON

"Binary JSON"

Binary-encoded serialization of JSON-like docs.

Also allows "referencing".

Embedded structure reduces need for joins.

### Goals

- Lightweight
- Traversable
- Efficient (decoding and encoding)



Copyright © Capgemini 2014. All Rights Reserved

### BSON Example

```
{  
  "_id" : "37010"  
  "city" : "ADAMS",  
  "pop" : 2660,  
  "state" : "TN",  
  "councilman" : {  
    "name": "John Smith"  
    "address": "13 Scenic Way"  
  }  
}
```



Copyright © Capgemini 2014. All Rights Reserved. 30

## MongoDB – Datatypes

### String

- This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.

### Integer

- This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

### Boolean

- This type is used to store a boolean (true / false) value.

### Double

- This type is used to store floating point values.

### Min / Max Keys

- This type is used to compare a value against the lowest and highest BSON elements.

## MongoDB – Datatypes (contd.)

### Arrays

- This type is used to store arrays or list or multiple values into one key.

### Timestamp

- **c timestamp**. This can be handy for recording when a document has been modified or added.

### Object

- This datatype is used for embedded documents.

### Null

- This type is used to store a Null value.

### Symbol

- This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type.



Copyright © Capgemini 2014. All Rights Reserved

## MongoDB – Datatypes (contd.)

**Object \_id** • This datatype is used to store the document's ID.

**Binary Data** • This datatype is used to store binary data.

**Code** • This datatype is used to store javascript code into document.

**Regular Expression** • This datatype is used to store regular expression.

**Date** • This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.



Copyright © Capgemini 2014. All Rights Reserved. 21

## MongoDB – Basic Operations

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

The number can be used with the \$type operator to query by type!

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

Copyright © 2014 MongoDB, Inc. All rights reserved.

## The \_id Field

By default, each document contains an \_id field. This field has a number of special characteristics:

- Value serves as primary key for collection.
- Value is unique, immutable, and may be any non-array type.
- Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.



Copyright © Capgemini 2014. All Rights Reserved. 30

### Example: Mongo Collection

```
{ "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
  "Last Name": "DUMONT",
  "First Name": "Jean",
  "Date of Birth": "01-22-1963" },
{ "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
  "Last Name": "PELLERIN",
  "First Name": "Franck",
  "Date of Birth": "09-19-1983",
  "Address": "1 chemin des Loges", "City": "VERSAILLES" }
```



Copyright © Capgemini 2014. All Rights Reserved. 30

### Example: Mongo Document

```
user = {  
    name: "Z",  
    occupation: "A scientist",  
    location: "New York"  
}
```



Copyright © Capgemini 2014. All Rights Reserved. 21

## Document

### Simple Document

A document is roughly equivalent to a row in a relational database, which contain one or multiple key-value pairs.

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

Most documents will be more complex than this simple one and often will contain multiple key/value pairs:

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

Key/value pairs in documents are ordered—the earlier document is distinct from the following document:

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

Values in documents are not just "blobs." They can be one of several different data types.



Copyright © Capgemini 2014. All Rights Reserved. 20

## Document (contd.)

### Documents are Rich Data Structures

```
{  
    first_name: 'Paul', <-- String  
    surname: 'Miller',  
    cell: '+447557505611' <-- Number  
    city: 'London',  
    location: [45.123, 47.232], <-- Geo Coordinates  
    Profession: [banking, finance, trader], Fields can  
    cars: [ <-- contain arrays  
        { model: 'Bentley',  
          year: 1973,  
          value: 100000, ... },  
        { model: 'Rolls Royce',  
          year: 1965,  
          value: 330000, ... }  
    ]  
}
```

Fields

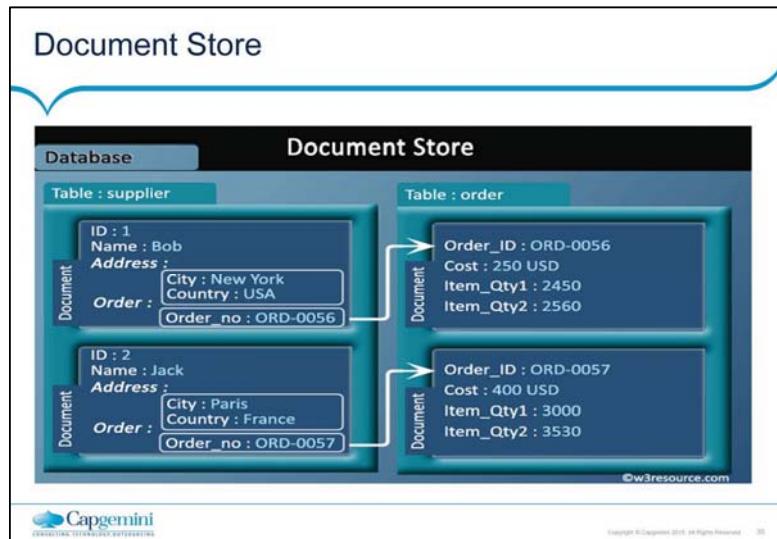
String  
Number  
Geo Coordinates  
Fields can contain arrays  
Fields can contain an array of sub-documents

 mongoDB

 Capgemini  
www.capgemini.com

Copyright © Capgemini 2014. All Rights Reserved.

## MongoDB – Basic Operations



## Document Store (contd.)

### Scenario

- A blog post has an author, some text, and many comments.
- The comments are unique per post, but one author has many posts.
- How would you design this in SQL?



Copyright © Capgemini 2014. All Rights Reserved. 31

## Example: A Blog: Bad Design

Collections for posts, authors, and comments.

References by manually created ID.



Copyright © Capgemini 2014. All Rights Reserved. 30

### Example: A Blog: Bad Design (contd.)

```
post = {  
    id: 150,  
    author: 100,  
    text: 'This is a pretty awesome post.',  
    comments: [100, 105, 112]  
}  
author = {  
    id: 100,  
    name: 'Michael Arrington'  
    posts: [150]  
}  
comment = {  
    id: 105,  
    text: 'Whatever this sux.'  
}
```



Copyright © Capgemini 2014. All Rights Reserved. 33

## Example: Blog – A Better Design

Collection for Posts

Embed comments, author name

```
post = {  
    author: 'Michael Arrington',  
    text: 'This is a pretty awesome post.',  
    comments: [  
        'Whatever this post .',  
        'I agree, lame!'  
    ]  
}
```

Why is this one better?



Copyright © Capgemini 2014. All Rights Reserved. 50

## Benefits

Embedded Objects brought back in the same query as the parent Object.

Only 1 trip to the DB server required.

Objects in the same collection are generally stored contiguously on disk.

Spatial locality = faster

If the document model matches your domain well ,it can be much easier comprehend the nasty joins.



Copyright © Capgemini 2014. All Rights Reserved. 30

Summary

Create Database	Drop Database	Create Collection
Drop Collection	CRUD Operations	JSON, BSON Document
Data Types		

Summary

Capgemini

Copyright © Capgemini 2014. All Rights Reserved.

# **MongoDB – Aggregations**

Lesson 04

## MongoDB – Aggregations

Version Sheet: MANDATORY (hidden in ‘slide show’ mode)

Version	Changes	Author
001	Redesign	Rohan Salvi

 Capgemini  
CONSULTING | TECHNOLOGY | OPERATIONS

2  
Copyright © Capgemini 2014. All Rights Reserved.

## Note to the SME:

Please read before you begin reviewing this module as SME

Dark Red box with white text

Note to the SME

The SME must provide missing info.

Amber box with black text

Note to the SME

The SME must validate the re-design/ the modification/ addition.

Note to the SME:  
This is a temporary slide and will not be part of the final upload at all. It is to help the SME understand how we will communicate changes to them.

To Review and Navigate the comments in the presentation Click on the "Review" Tab and then Click "Next" to review all the comments . Also you can add your comments using the "New comment" button

The comments in the review section have also been updated in a green textbox with black text.



Copyright © Capgemini 2016. All Rights Reserved. 3

## MongoDB – Aggregations

### Ground Rules for Face-to-face Classrooms

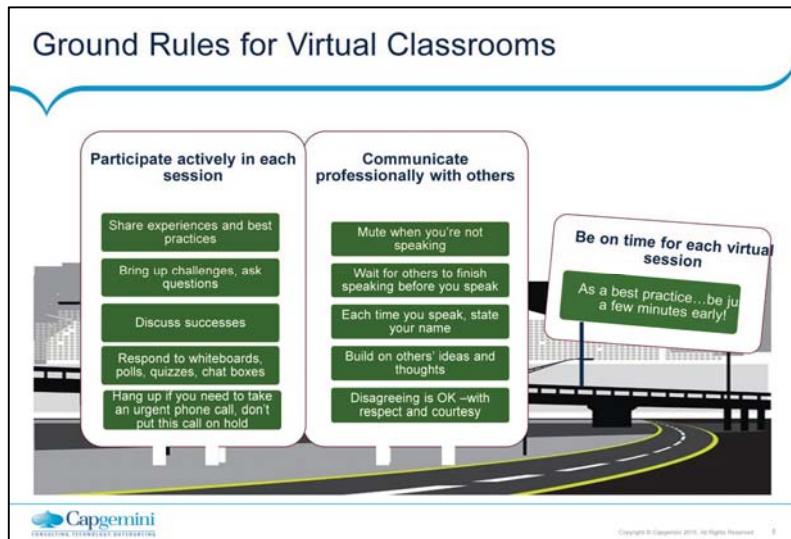
	Everyone participates	
	Be open and honest	
	One speaker at a time	
	Stick to time contracts	
	Clean desk / room policy	
	Clients & Leadership are often around, please remember that	
	Respect individual opinions and diversities	
	Give headlines, be concise	
	Make language a non-issue	
	Seek first to understand, and then to be understood	
	No mobile phone, No computer (except for surveys, polls etc)	
	Maintain a spirit of fun and enthusiasm	

 Capgemini  
TRANSFORMATIVE LEARNING EXPERIENCES

Copyright © Capgemini 2016 - All Rights Reserved

Best to print on an A3 sheet and display in class. The idea is to co-create, to connect to learn, to do-learn-do (knowing is not enough, the real purpose is to apply what we learn on the job). It would also be great to have the class share their learning with their teams. You are the facilitator, you will help all share knowledge and learn new concepts or skills, there will be no lectures, everyone is a teacher.

## MongoDB – Aggregations



Show this slide and hide the one for onsite classrooms, if this is a virtual session.

ONLY for Virtual classrooms

## Module at a Glance

<b>Target Audience:</b>		SME to provide the details required in the table.
<b>Course Level:</b>	<i>Basic</i>	
<b>Duration (in hours):</b>	30 mins	
<b>Pre-requisites, if any:</b>	NA	
<b>Post-requisites, if any:</b>	<i>Submit Session Feedback</i>	
<b>Relevant Certifications:</b>	None	



Copyright © Capgemini 2019. All Rights Reserved.

## MongoDB – Aggregations

### Introductions (for Virtual Classrooms)



Business Photo

SME to provide the photos and names of the facilitators.



Business Photo

*Facilitator*

Name

Role

*Moderator*

Name

Role

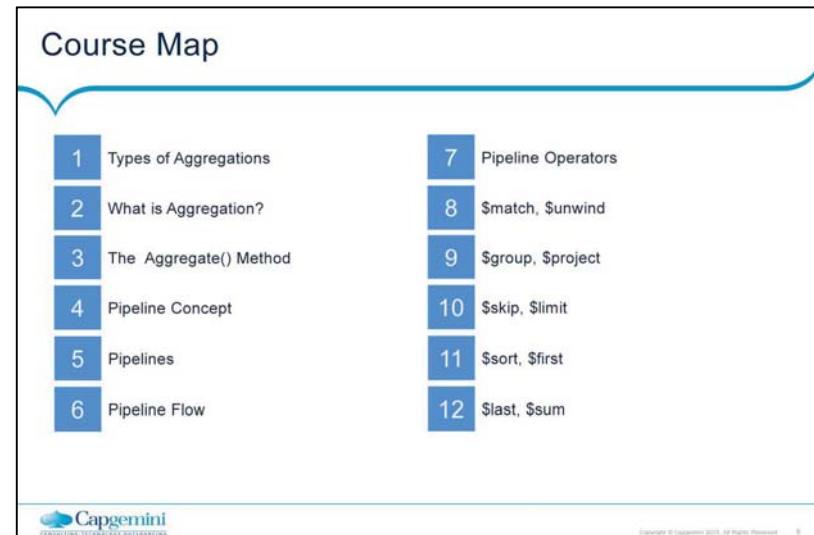


CAPGEMINI  
CONSULTING TECHNOLOGIES SERVICES

Copyright © Capgemini 2019. All Rights Reserved.

Add pics and Capgemini roles for the Facilitator and moderator. Establish their credibility, what qualifies them to facilitate this session.

## MongoDB – Aggregations



## Module Objectives



### What you will learn

At the end of this module, you will learn:

- What is Aggregation

Note to the SME : Please provide the module Objectives or validate the partially updated content



### What you will be able to do

At the end of this module, you will be able to:

- Explain what is Aggregation
- List the various types of Aggregation
- Understand the Pipeline concept
- Describe the various types of Aggregation
- State examples for the various types of Aggregation

## Types of Aggregations

\$match,  
\$unwind

\$group,  
\$project

\$skip,  
\$limit

\$sort,  
\$first

\$last,  
\$sum

\$avg,  
\$min,  
\$max

\$push,  
\$addToSet



Copyright © Capgemini 2017. All Rights Reserved.

## What is Aggregation?

Aggregations operations process data records and return computed results.

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

In sql count(\*) and with group by is an equivalent of mongodb aggregation.



Copyright © Capgemini 2012. All Rights Reserved. 11

### Create

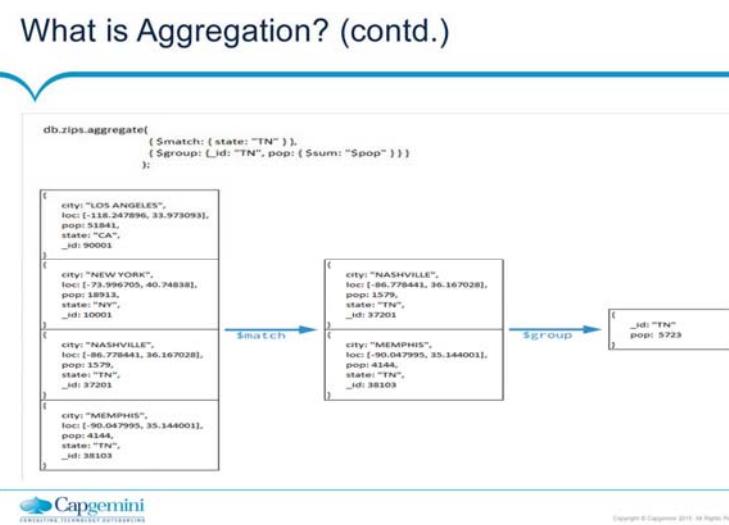
- The field name `_id` is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names **cannot** start with the `$` character.
- The field names **cannot** contain the `.` character.

### Create with save

- If the `<document>` argument does not contain the `_id` field or contains an `_id` field with a value not in the collection, the [`save\(\)`](#) method performs an insert of the document.
- Otherwise, the [`save\(\)`](#) method performs an update.

sds

## MongoDB – Aggregations



## The Aggregate() Method

For the aggregation in Mongodb you should use **aggregate()** method.

### Syntax:

Basic syntax of **aggregate()** method is as follows:

- >db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)



Copyright © Capgemini 2012. All Rights Reserved. 13

## Pipeline Concept

The aggregation framework is based on pipeline concept, just like Unix pipeline.

There can be N number of operators.

Output of first operator will be fed as input to the second operator. Output of second operator will be fed as input to the third operator and so on.



Copyright © Capgemini 2012. All Rights Reserved. 14

## Pipelines

Modeled on the concept of data processing pipelines.

Provides:

- Filters that operate like queries.
- Document transformations that modify the form of the output document.

Provides tools for:

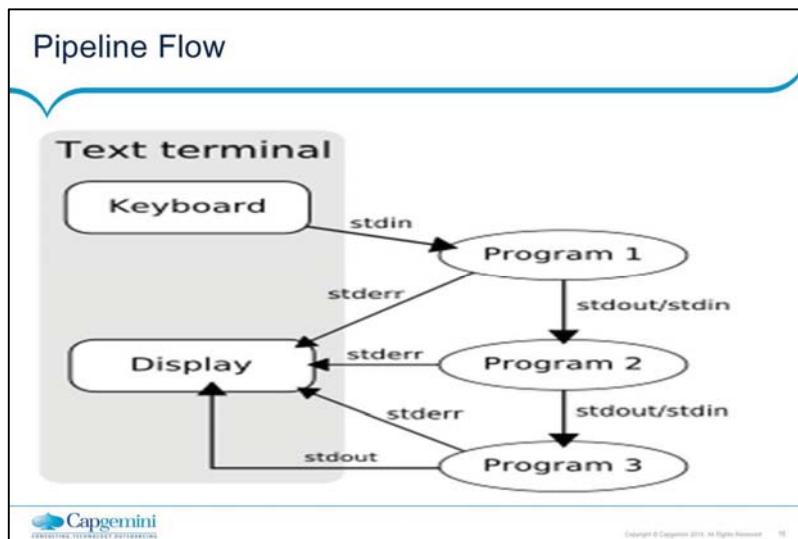
- Grouping and sorting by field.
- Aggregating the contents of arrays, including arrays of documents.

Can use **operators** for tasks such as calculating the average or concatenating a string.



Copyright © Capgemini 2012. All Rights Reserved. 10

## MongoDB – Aggregations



## Pipeline Operators

The basic pipeline operators are:

\$match	\$unwind	\$group
\$project	\$skip	\$limit
\$sort		

 Capgemini  
CONSULTING SERVICES INNOVATION

Copyright © Capgemini 2012. All Rights Reserved. 17

## \$match

This is similar to MongoDB Collection's find method and SQL's WHERE clause.

**Example:** We want to consider only the marks of the students who study in Class "2".

- db.Student.aggregate ([ { "\$match": { "Class": "2" } } ])



Copyright © Capgemini 2017. All Rights Reserved. 14

## \$unwind

This will be very useful when the data is stored as list.

When the unwind operator is applied on a list data field, it will generate a new record for each and every element of the list data field on which unwind is applied.

It basically flattens the data.



Copyright © Capgemini 2017. All Rights Reserved. 10

## Example

```
db.Student.aggregate ([ { "$match": { "Student_Name": "Kalki", } }, {  
    "$unwind": "$Subject" } ])
```

```
{ "result" : [ { "_id" : ObjectId("517ccb98eccb9ee3d000fa5c"),  
    "Student_Name" : "Kalki", "Class" : "2", "Mark_Scored" : 100, "Subject" :  
    "Tamil" }, { "_id" : ObjectId("517ccb98eccb9ee3d000fa5c"),  
    "Student_Name" : "Kalki", "Class" : "2", "Mark_Scored" : 100, "Subject" :  
    "English" }, { "_id" : ObjectId("517ccb98eccb9ee3d000fa5c"),  
    "Student_Name" : "Kalki", "Class" : "2", "Mark_Scored" : 100, "Subject" :  
    "Maths" } ], "ok" : 1 }
```



Copyright © Capgemini 2012. All Rights Reserved.

## \$group

The group pipeline operator is similar to the SQL's GROUP BY clause.

**Example:** Lets try and get the sum of all the marks scored by each and every student, in Class "2".

- db.Student.aggregate ([ { "\$match": { "Class": "2" } }, { "\$unwind": "\$Subject" }, { "\$group": { "\_id": { "Student\_Name": "\$Student\_Name" }, "Total\_Marks": { "\$sum": "\$Mark\_Scored" } } } ])



Copyright © Capgemini 2017. All Rights Reserved. 31

## \$group (contd.)

In this aggregation example, we have specified an `_id` element and `Total_Marks` element.

The `_id` element tells MongoDB to group the documents based on `Student_Name` field.

The `Total_Marks` uses an aggregation function `$sum`, which basically adds up all the marks and returns the sum.



Copyright © Capgemini 2012. All Rights Reserved.

## \$project

The project operator is similar to SELECT in SQL.

We can use this to rename the field names and select / deselect the fields to be returned, out of the grouped fields.

If we specify 0 for a field, it will NOT be sent in the pipeline to the next operator.



Copyright © Capgemini 2017. All Rights Reserved.

## \$Sort

This is similar to SQL's ORDER BY clause. To sort a particular field in descending order specify -1 and specify 1 if that field has to be sorted in ascending order.

- db.Student.aggregate ([ { "\$match": { "Class": "2" } }, { "\$unwind": "\$Subject" }, { "\$group": { "\_id": { "Student\_Name": "\$Student\_Name" }, "Total\_Marks": { "\$sum": "\$Mark\_Scored" } } }, { "\$project": { "\_id":0, "Name": "\$\_id.Student\_Name", "Total": "Total\_Marks" } }, { "\$sort": { "Total":-1, "Name":1 } } ])

## \$limit and \$skip

These two operators can be used to limit the number of documents being returned. They will be more useful when we need pagination support.



Copyright © Capgemini 2012. All Rights Reserved. 30

## \$first

Returns the value that results from applying an expression to the first document in a group of documents that share the same group by key. Only meaningful when documents are in a defined order.

\$first is only available in the **\$group** stage.



Copyright © Capgemini 2012. All Rights Reserved. 30

## Example

Grouping the documents by the item field, the following operation uses the \$first accumulator to compute the first sales date for each item.

```
db.sales.aggregate( [ { $sort: { item: 1, date: 1 } }, { $group: { _id: "$item", firstSalesDate: { $first: "$date" } } } ] )
```



Copyright © Capgemini 2017. All Rights Reserved.

## \$last

\$last returns the value that results from applying an expression to the last document in a group of documents that share the same group by a field.

Only meaningful when documents are in a defined order.

\$last is only available in the **\$group** stage.



Copyright © Capgemini 2012. All Rights Reserved. 30

## \$last (contd.)

The following operation first sorts the documents by item and date, and then in the following **\$group** stage, groups the now sorted documents by the item field and uses the \$last accumulator to compute the last sales date for each item:

- db.sales.aggregate([ { \$sort: { item: 1, date: 1 } }, { \$group: { \_id: "\$item", lastSalesDate: { \$last: "\$date" } } } ])

## \$sum

Calculates and returns the sum of numeric values. \$sum ignores non-numeric values.

When used in the **\$group** stage, \$sum has the following syntax and returns the collective sum of all the numeric values that result from applying a specified expression to each document in a group of documents that share the same group by key.



Copyright © Capgemini 2017. All Rights Reserved. 30

## Example

Grouping the documents by the day and the year of the date field, the following operation uses the \$sum accumulator to compute the total amount and the count for each group of documents.

- db.sales.aggregate( [ { \$group: { \_id: { day: { \$dayOfYear: "\$date"}, year: { \$year: "\$date" } }, totalAmount: { \$sum: { \$multiply: [ "\$price", "\$quantity" ] } }, count: { \$sum: 1 } } } ] )



Copyright © Capgemini 2012. All Rights Reserved. 31

## MongoDB – Aggregations

### \$avg

Returns the average value of the numeric values. \$avg ignores non-numeric values.

\$avg returns the collective average of all the numeric values that result from applying a specified expression to each document in a group of documents that share the same group by key.



Copyright © Capgemini 2017. All Rights Reserved.

## Example

Grouping the documents by the item field, the following operation uses the \$avg accumulator to compute the average amount and average quantity for each grouping.

- db.sales.aggregate([ { \$group: { \_id: "\$item", avgAmount: { \$avg: { \$multiply: ["\$price", "\$quantity"] } }, avgQuantity: { \$avg: "\$quantity" } } } ])

## MongoDB – Aggregations

### \$max

Returns the maximum value. \$max compares both value and type, using the **specified BSON comparison order** for values of different types.

Grouping the documents by the item field, the following operation uses the \$max accumulator to compute the maximum total amount and maximum quantity for each group of documents.

- db.sales.aggregate( [ { \$group: { \_id: "\$item", maxTotalAmount: { \$max: { \$multiply: [ "\$price", "\$quantity" ] } }, maxQuantity: { \$max: "\$quantity" } } } ] )



Copyright © Capgemini 2012. All Rights Reserved. 34

## \$min

Returns the minimum value. \$min compares both value and type, using the **specified BSON comparison order** for values of different types.

Grouping the documents by the item field, the following operation uses the \$min accumulator to compute the minimum amount and minimum quantity for each grouping.

- db.sales.aggregate( [ { \$group: { \_id: "\$item", minQuantity: { \$min: "\$quantity" } } } ] )



Copyright © Capgemini 2012. All Rights Reserved. 31

## \$push

Returns an array of *all* values that result from applying an expression to each document in a group of documents that share the same group by key.

Grouping the documents by the day and the year of the date field, the following operation uses the \$push accumulator to compute the list of items and quantities sold for each group:

- db.sales.aggregate( [ { \$group: { \_id: { day: { \$dayOfYear: "\$date"}, year: { \$year: "\$date" } }, itemsSold: { \$push: { item: "\$item", quantity: "\$quantity" } } } } ] )



Copyright © Capgemini 2012. All Rights Reserved. 30

## \$addToSet

Returns an array of all *unique* values that results from applying an expression to each document in a group of documents that share the same group by key. Order of the elements in the output array is unspecified.

If the value of the expression is an array, \$addToSet appends the whole array as a *single* element.

If the value of the expression is a document, MongoDB determines that the document is a duplicate if another document in the array matches the to-be-added document exactly; i.e. the existing document has the exact same fields and values in the exact same order.



Copyright © Capgemini 2012. All Rights Reserved. 37

## Example

Grouping the documents by the day and the year of the date field, the following operation uses the \$addToSet accumulator to compute the list of unique items sold for each group:

- db.sales.aggregate( [ { \$group: { \_id: { day: { \$dayOfYear: "\$date"}, year: { \$year: "\$date" } }, itemsSold: { \$addToSet: "\$item" } } } ] )

## MongoDB – Aggregations

### Summary

\$match, \$unwind	\$group, \$project	\$skip, \$limit
\$sort, \$first	\$last, \$sum	\$avg, \$min, \$max
\$push, \$addToSet		

 Capgemini  
CONSULTING | TECHNOLOGY | OPERATIONS

Copyright © Capgemini 2012. All Rights Reserved. 30

## **MongoDB – Indexing**

Lesson 05

## MongoDB – Indexing

Version Sheet: MANDATORY (hidden in 'slide show' mode)

Version	Changes	Author
001	Redesign	Rohan Salvi

 Capgemini  
Copyright © Capgemini 2016. All Rights Reserved.

2

## Note to the SME:

Please read before you begin reviewing this module as SME

**Dark Red box with white text**

**Amber box with black text**

**Note to the SME:**  
This is a temporary slide and will not be part of the final upload at all. It is to help the SME understand how we will communicate changes to them.

**Note to the SME**

**The SME must provide missing info.**

**Note to the SME**

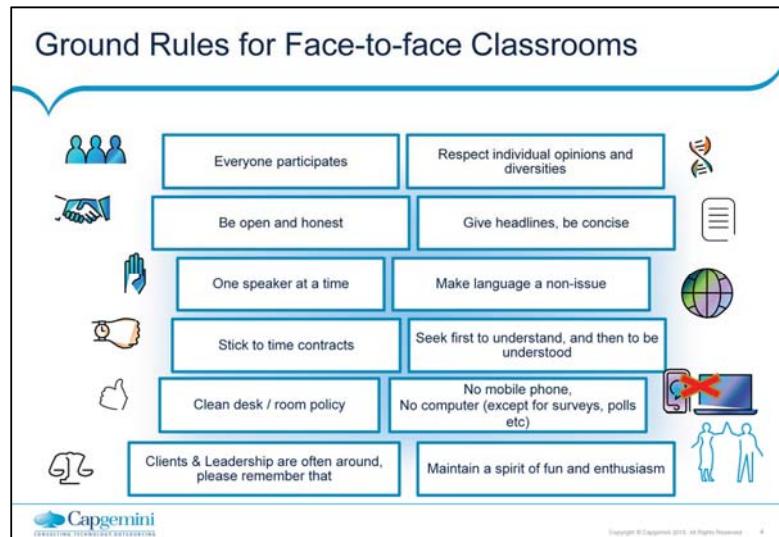
**The SME must validate the re-design/ the modification/ addition.**

To Review and Navigate the comments in the presentation Click on the "Review" Tab and then Click "Next" to review all the comments . Also you can add your comments using the "New comment" button

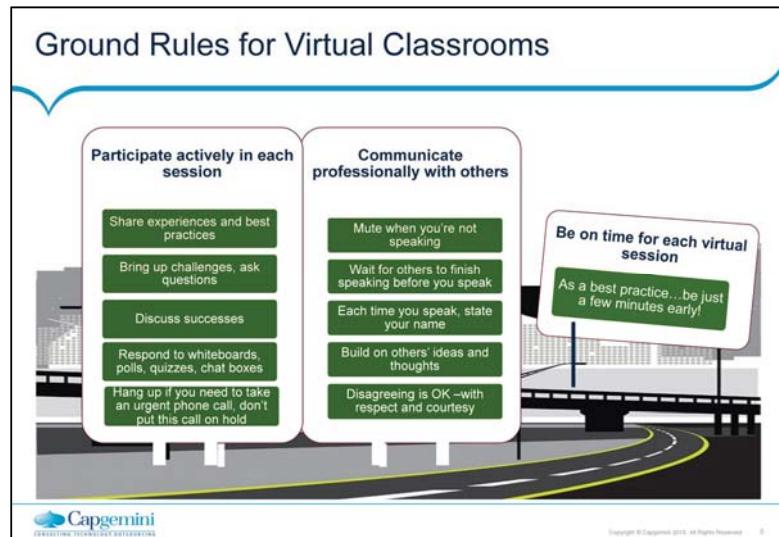
The comments in the review section have also been updated in a green textbox with black text.

 Capgemini  
PERPETUAL INNOVATION

Copyright © Capgemini 2014. All Rights Reserved



Best to print on an A3 sheet and display in class. The idea is to co-create, to connect to learn, to do-learn-do (knowing is not enough, the real purpose is to apply what we learn on the job). It would also be great to have the class share their learning with their teams. You are the facilitator, you will help all share knowledge and learn new concepts or skills, there will be no lectures, everyone is a teacher.



Show this slide and hide the one for onsite classrooms, if this is a virtual session.

ONLY for Virtual classrooms

## Module at a Glance

<b>Target Audience:</b>	
<b>Course Level:</b>	Basic
<b>Duration (in hours):</b>	30 mins
<b>Pre-requisites, if any:</b>	NA
<b>Post-requisites, if any:</b>	<i>Submit Session Feedback</i>
<b>Relevant Certifications:</b>	None



Copyright © Capgemini 2014. All Rights Reserved.

## MongoDB – Indexing

### Introductions (for Virtual Classrooms)

SME to provide the photos and names of the facilitators.

Business Photo

Business Photo

Facilitator  
Name  
Role

Moderator  
Name  
Role

 Capgemini  
PERFORMING TECHNOLOGY REINVENTED

Add pics and Capgemini roles for the Facilitator and moderator. Establish their credibility, what qualifies them to facilitate this session.

## Agenda

- 1 Understand about Indexes
- 2 Understand different types of Indexes
- 3 Understand properties of Indexes
- 4 Explain Plan in MongoDB
- 5 Mongostat
- 6 Mongotop
- 7 Logging Slow queries
- 8 Profiling



Copyright © Capgemini 2016. All Rights Reserved.

## Module Objectives



### What you will learn

At the end of this module, you will learn:

- Indexing in MongoDB

Note to the SME : Please provide the module Objectives or validate the partially updated content



### What you will be able to do

At the end of this module, you will be able to:

- Understand what are Indexes
- List the different types of Indexes and their features
- Explain Plan in MongoDB
- Describe Mongostat and Mongotop
- State the features of Logging Slow queries and Profiling

## Index in MongoDB

### Creation Index

- db.users.ensureIndex( { score: 1 } )

### Show Existing Indexes

- db.users.getIndexes()

### Drop Index

- db.users.dropIndex( {score: 1} )

### Explain—Explain

- db.users.find().explain()
- Returns a document that describes the process and indexes.

### Hint

- db.users.find().hint({score: 1})
- Override MongoDB's default index selection.



Copyright © Capgemini 2016. All Rights Reserved.

### Before Index

What does database normally do when we query?

- MongoDB must scan every document.
- Inefficient because process large volume of data.

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

collection

Database

Collection

Collection

Collection

{ name : "A" }  
{ name : "B" }  
{ name : "C" }  
{ name : "D" }

Capgemini

Copyright © Capgemini 2016. All Rights Reserved.

## What is Index?

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.



Copyright © Capgemini 2016. All Rights Reserved. 12

## Definition of Index

### Definition

- Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.

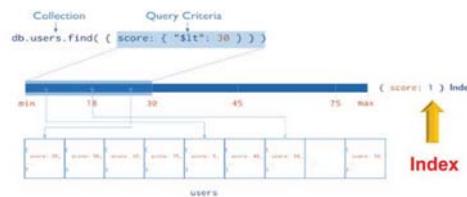


Diagram of a query that uses an index to select.

## Indexes

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

Indexes help efficient execution of queries.

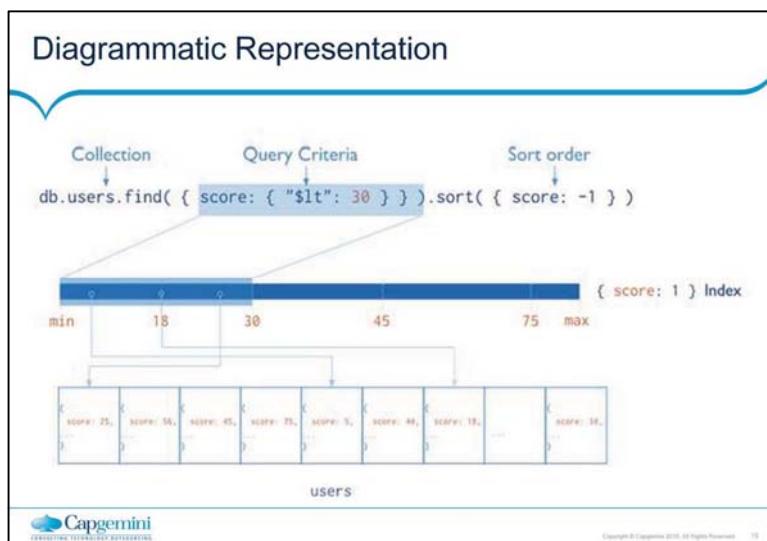
- Without indexes MongoDB must scan every document in a collection to select those documents that match the query statement.

These collection scans are inefficient because they require Mongod to process a larger volume of data than an index for each operation.



Copyright © Capgemini 2016. All Rights Reserved. 14

## MongoDB – Indexing



## Indexes

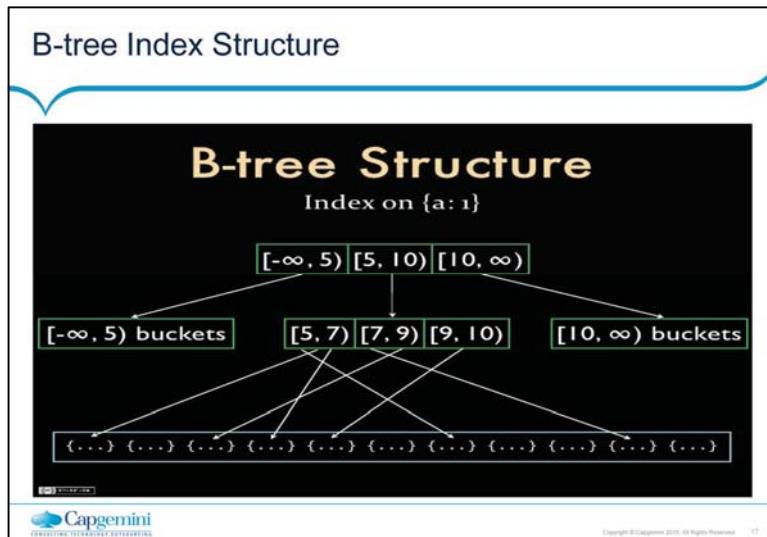
### Indexes Maintain Order

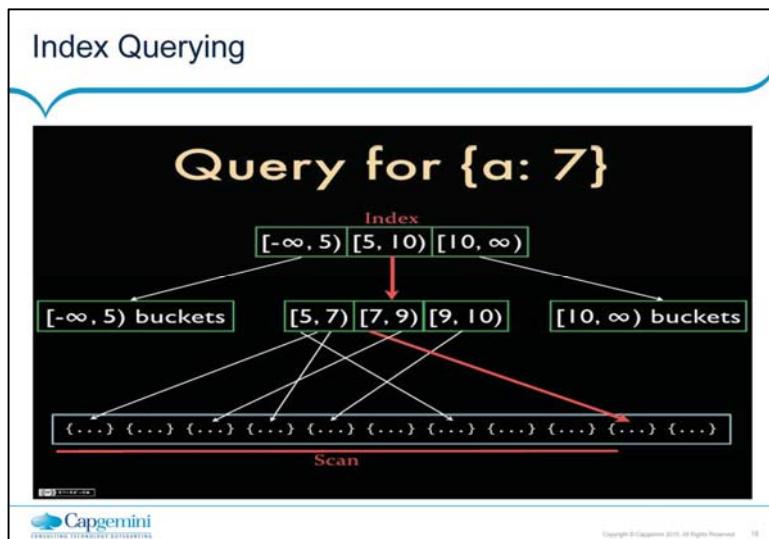
Index on {a: 1, b: -1}

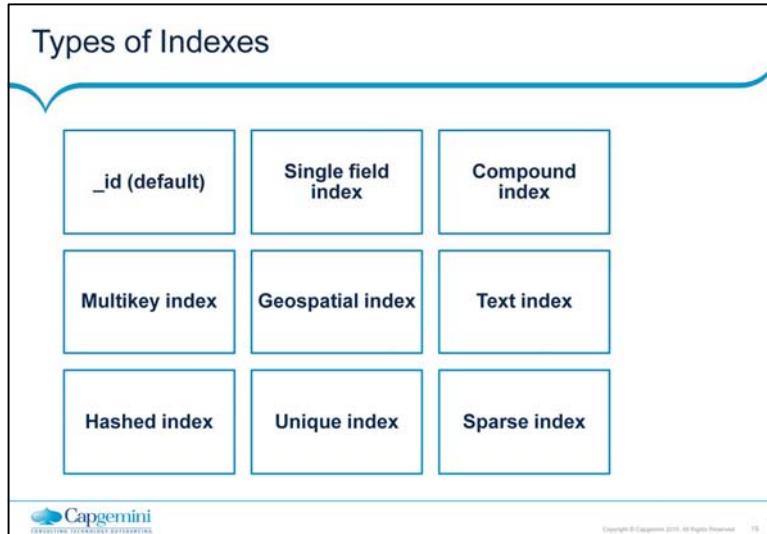
```
{a: 0, b: 9}  
{a: 2, b: 0}  
{a: 3, b: 7}  
{a: 3, b: 5}  
{a: 3, b: 2}  
{a: 7, b: 1}  
{a: 9, b: 1}
```



Copyright © Capgemini 2018. All Rights Reserved. 10







## About Indexes

Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement.

The ordering of the index entries supports efficient equality matches and range-based query operations.

The index stores the value of a specific field or set of fields, ordered by the value of the field.

MongoDB can return sorted results by using the ordering in the index.



Copyright © Capgemini 2016. All Rights Reserved. 30

## Index Types

### Default \_id

- All MongoDB collections have an index on the `_id` field that exists by default. If applications do not specify a value for `_id` the driver or the mongod will create an `_id` field with an objectid value.

### Single Field

- In addition to the MongoDB-defined `_id` index, MongoDB supports the creation of user-defined ascending / descending indexes on a single field of a document.



Copyright © Capgemini 2016. All Rights Reserved. 21

## Example

{ "\_id" : ObjectId(...), "name" : "Alice", "age" : 27 }

The following command creates an index on the name field:

- db.friends.createIndex( { "name" : 1 } )

- Single Field Indexes

- db.users.ensureIndex( { score: 1 } )

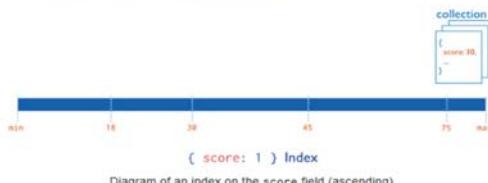


Diagram of an index on the score field (ascending).

## Compound Indexes

MongoDB also supports user-defined indexes on multiple fields, i.e. Compound indexes.

The order of fields listed in a compound index has significance. For instance, if a compound index consists of { userid: 1, score: -1 }, the index sorts first by userid and then, within each user id value, sorts by score.



Copyright © Capgemini 2016. All Rights Reserved. 23

## Compound Indexes (contd.)

### Compound Field Indexes

- db.users.ensureIndex( { userid:1, score:-1 } )

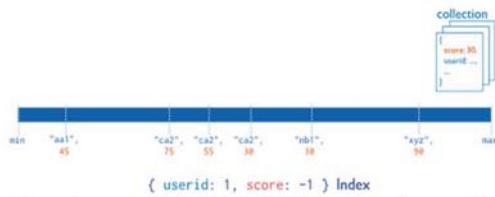


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

## Example

Consider a collection named products that holds documents that resemble the following document:

- { "\_id": ObjectId(...), "item": "Banana", "category": ["food", "produce", "grocery"], "location": "4th Street Store", "stock": 4, "type": "cases", "arrival": Date(...) }

If applications query on the item field as well as query on both the item field and the stock field, you can specify a single compound index to support both of these queries:

- db.products.createIndex( { "item": 1, "stock": 1 } )



Copyright © Capgemini 2016. All Rights Reserved. 21

## Multikey Indexes

When indexing is done on an array field, it is called a multikey index.

To index a field that holds an array value ,MongoDB creates an index key for each element in the array.

These *multikey* indexes support efficient queries against array fields.

Multikey indexes can be constructed over arrays that hold both scalar values (e.g. strings, numbers) *and* nested documents.



Copyright © Capgemini 2016. All Rights Reserved. 20

## Example

- Let's consider a document:

```
{  
  "title": "Superman",  
  "tags": ["comic", "action", "xray"],  
  "issues": [ { "number": 1, "published_on": "June 1938"  
  }  
 ]  
}
```

- Multikey indexes lets us search on the values in the tags array as well as in the issues array. Let's create two indexes to cover both.



Copyright © Capgemini 2016. All Rights Reserved. 21

### Example (contd.)

```
comics.ensureIndex({tags: 1});  
comics.ensureIndex({issues: 1});
```

If the document changes structure it would be better to create a specific compound index on the fields needed in sub element document.

```
comics.ensureIndex({"issues.number":1,  
"issues.published_on":1});
```

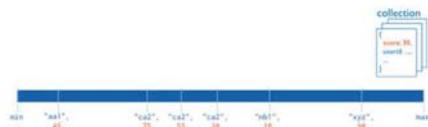


Copyright © Capgemini 2016. All Rights Reserved. 30

## Example (contd.)

### Multikey Indexes

```
• db.users.ensureIndex( { addr.zip:1 } )
```



{ `userid: 1, score: -1` } Index

Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.



## Limitations

Consider a collection that contains the following document:

- { \_id: 1, a: [ 1, 2 ], b: [ 1, 2 ], category: "AB - both arrays" }

You cannot create a compound multikey index { a: 1, b: 1 } on the collection since both the a and b fields are arrays.



Copyright © Capgemini 2016. All Rights Reserved. 30

## Demo of Indexes in MongoDB

Import Data

Create Index

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

Show Existing Index

Hint

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

Explain

Compare with data without indexes

```

> db.zipcodes.find().limit(20)
[{"city": "ACME", "loc": [-96.51657, 33.59432], "pop": 465, "state": "AL", "lat": 33.59432}, {"city": "ADKARVILLE", "loc": [-96.99771, 33.63402], "pop": 3626, "state": "AL", "lat": 33.63402}, {"city": "AGER", "loc": [-97.07405, 33.42477], "pop": 2485, "state": "AL", "lat": 33.42477}, {"city": "AKESONE", "loc": [-96.81281, 33.23898], "pop": 9424, "state": "AL", "lat": 33.23898}, {"city": "NEW ALICE", "loc": [-95.95196, 32.94145], "pop": 1942, "state": "AL", "lat": 32.94145}, {"city": "ALPINE", "loc": [-96.28854, 33.31136], "pop": 182, "state": "AL", "lat": 33.31136}, {"city": "AMAT", "loc": [-96.48693, 34.33139], "pop": 1369, "state": "AL", "lat": 34.33139}, {"city": "BALDITON", "loc": [-96.62129, 34.36236], "pop": 1701, "state": "AL", "lat": 34.36236}, {"city": "BEECHER", "loc": [-96.94750, 33.48001], "pop": 4548, "state": "AL", "lat": 33.48001}, {"city": "BETHEM", "loc": [-96.99897, 33.43485], "pop": 3877, "state": "AL", "lat": 33.43485}, {"city": "BLANTONVILLE", "loc": [-96.50483, 34.85259], "pop": 956, "state": "AL", "lat": 34.85259}, {"city": "BREKIN", "loc": [-97.04931, 33.57934], "pop": 148, "state": "AL", "lat": 33.57934}, {"city": "BRENT", "loc": [-97.21337, 33.45951], "pop": 374, "state": "AL", "lat": 33.45951}, {"city": "BRIEFFIELD", "loc": [-96.95467, 33.44271], "pop": 133, "state": "AL", "lat": 33.44271}, {"city": "CABRA", "loc": [-96.75587, 33.2086], "pop": 405, "state": "AL", "lat": 33.2086}, {"city": "CENTREVILLE", "loc": [-97.11924, 31.95934], "pop": 482, "state": "AL", "lat": 31.95934}, {"city": "CHELSEA", "loc": [-96.84543, 33.37216], "pop": 470, "state": "AL", "lat": 33.37216}, {"city": "COURA PINE", "loc": [-96.35942, 33.26828], "pop": 795, "state": "AL", "lat": 33.26828}, {"city": "CLOWN", "loc": [-96.44247, 32.85523], "pop": 1199, "state": "AL", "lat": 32.85523}, {"city": "GLENCLARE", "loc": [-96.55155, 33.95204], "pop": 138, "state": "AL", "lat": 33.95204}
> db.zipcodes.count()
2907

```

 Capgemini  
www.capgemini.com

Copyright © Capgemini 2016. All Rights Reserved. 31

Page 05-31

## Demo of Indexes in MongoDB (contd.)

Import Data

Create Index

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

Show Existing Index

Hint

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

Explain

Compare with data without indexes

```
db.zips.ensureIndex({pop: -1})  
db.zips.ensureIndex({state: 1, city: 1})  
db.zips.ensureIndex({loc: -1})
```

 Capgemini  
Engineering Services

Copyright © Capgemini 2016. All Rights Reserved.

## Demo of Indexes in MongoDB (contd.)

### Import Data

#### Create Index

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Show Existing Index

#### Hint

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Explain

#### Compare with data without indexes



```
> db.zips.getIndexes()
[{"v": 1, "key": {"_id": 1}, "ns": "blog.zips", "name": "_id_"}, {"v": 1, "key": {"pop": 1}, "ns": "blog.zips", "name": "pop_1"}, {"v": 1, "key": {"state": 1, "city": 1}, "ns": "blog.zips", "name": "state_1_city_1"}, {"v": 1, "key": {"loc": 1}, "ns": "blog.zips", "name": "loc_1"}]
```

Copyright © Capgemini 2016. All Rights Reserved.

## Demo of Indexes in MongoDB (contd.)

**Import Data**

## Create Index

- Single Field Index
  - Compound Field Indexes
  - Multikey Indexes

### Show Existing Index

**Hint**

- Single Field Index
  - Compound Field Indexes
  - Multikey Indexes

## Explain

#### Compare with data without indexes



## Demo of Indexes in MongoDB (contd.)

### Import Data

#### Create Index

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Show Existing Index

#### Hint

- Single Field Index
- **Compound Field Indexes**
- Multikey Indexes

#### Explain

#### Compare with data without indexes

```
> db.zipcodes.find().limit(20).MongoState(state: 1, city: 1)
[{"city": "BOSTON", "loc": [-71.0648, 42.3608], "pop": 5165, "state": "MA", "lat": "42.3608"}, {"city": "AMERSTON", "loc": [-122.388089, 57.795807], "pop": 12099, "state": "WA", "lat": "57.7958"}, {"city": "WACONIA", "loc": [-95.393533, 44.895854], "pop": 481, "state": "MN", "lat": "44.8958"}, {"city": "MORAN", "loc": [-98.199325, 48.896825], "pop": 382, "state": "WA", "lat": "48.8968"}, {"city": "MANAW", "loc": [-105.785588, 54.548283], "pop": 98, "state": "MN", "lat": "54.5482"}, {"city": "ALABAMA", "loc": [-164.8623, 32.748987], "pop": 1206, "state": "WA", "lat": "32.7489"}, {"city": "ALEXANDRA", "loc": [-158.45938, 39.28688], "pop": 185, "state": "WA", "lat": "39.2868"}, {"city": "ALLAGAET", "loc": [-152.7721, 64.545079], "pop": 17, "state": "WA", "lat": "64.5450"}, {"city": "ABLEP", "loc": [-158.45452, 67.4951], "pop": 8, "state": "WA", "lat": "67.4951"}, {"city": "HANCOCK PARK", "loc": [-151.67605, 48.12579], "pop": 284, "state": "WA", "lat": "48.1257"}, {"city": "HOMERDALE", "loc": [-94.87697, -11.2025], "pop": 1408, "state": "WA", "lat": "11.2025"}, {"city": "HOMERDALE", "loc": [-98.85953, 61.8963], "pop": 1899, "state": "WA", "lat": "61.8963"}, {"city": "HOMERDALE", "loc": [-98.85953, 61.8963], "pop": 1854, "state": "WA", "lat": "61.8963"}, {"city": "HOMERDALE", "loc": [-98.74807, 61.35094], "pop": 3293, "state": "WA", "lat": "61.3509"}, {"city": "HOMERDALE", "loc": [-98.82082, 61.11554], "pop": 2028, "state": "WA", "lat": "61.1155"}, {"city": "HOMERDALE", "loc": [-98.82082, 61.11554], "pop": 2027, "state": "WA", "lat": "61.1155"}, {"city": "HOMERDALE", "loc": [-98.8740, 61.11554], "pop": 1764, "state": "WA", "lat": "61.1155"}, {"city": "HOMERDALE", "loc": [-98.77998, 61.1554], "pop": 3556, "state": "WA", "lat": "61.1554"}, {"city": "HOMERDALE", "loc": [-98.59321, 61.29616], "pop": 1550, "state": "WA", "lat": "61.2961"}, {"city": "HOMERDALE", "loc": [-98.88673, 61.15482], "pop": 1110, "state": "WA", "lat": "61.1548"}]
```


Copyright © Capgemini 2018. All Rights Reserved.

## Demo of Indexes in MongoDB (contd.)

### Import Data

#### Create Index

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Show Existing Index

#### Hint

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Explain

#### Compare with data without indexes



```
> db.zipcodes.find().limit(20).hint({loc: -1})
[{"city": "BIRMINGHAM", "loc": [-158.827489, 71.224637], "pop": 3686, "state": "MI", "lat": "43\u00b0 16' 59\"S"}, {"city": "WALNUTWOOD", "loc": [-148.812532, 70.420964], "pop": 402, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "MUSKEGON", "loc": [-138.897119, 71.135717], "pop": 254, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "PROUDIE BAY", "loc": [-148.559366, 70.470761], "pop": 153, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "WATRODOX", "loc": [-148.491024, 70.402689], "pop": 146, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "POINTER LAKE", "loc": [-148.990126, 69.765263], "pop": 139, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "POINT HORN", "loc": [-148.7358, 68.321258], "pop": 140, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "MANISTEE PASS", "loc": [-148.679065, 68.021979], "pop": 130, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "ARCTIC VILLAGE", "loc": [-148.423215, 68.877995], "pop": 107, "state": "MI", "lat": "43\u00b0 07' 59\"S"}, {"city": "KODIAK", "loc": [-163.730701, 61.866369], "pop": 165, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "HOMER", "loc": [-158.495052, 67.409513], "pop": 84, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "KOKAN", "loc": [-158.152232, 67.393824], "pop": 349, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "BETLES FIELD", "loc": [-151.902454, 67.000945], "pop": 150, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "VENERE", "loc": [-146.471273, 67.010484], "pop": 154, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "WIKAKA", "loc": [-148.588453, 66.975533], "pop": 165, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "SHOKOMA", "loc": [-157.41589, 66.952015], "pop": 84, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "KOBAS", "loc": [-157.966684, 66.912233], "pop": 206, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "WOTACHEE", "loc": [-162.124851, 66.948919], "pop": 154, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "WIKONK", "loc": [-161.945231, 66.936553], "pop": 154, "state": "AK", "lat": "41\u00b0 30' 59\"N"}, {"city": "SHALOUTSKA", "loc": [-160.693012, 66.713], "pop": 95, "state": "AK", "lat": "41\u00b0 30' 59\"N"}]
```

Copyright © Capgemini 2016. All Rights Reserved. 30

## Demo of Indexes in MongoDB (contd.)

### Import Data

#### Create Index

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Show Existing Index

#### Hint

- Single Field Index
- Compound Field Indexes
- Multikey Indexes

#### Explain

#### Compare with data without indexes

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
  "cursor" : "BasicCursor",
  "isMultiKey" : false,
  "n" : 19,
  "nscannedObjects" : 29467,
  "nscanned" : 29467,
  "nscannedObjectsAllPlans" : 29467,
  "nscannedAllPlans" : 29467,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nFields" : 0,
  "nChunkSkips" : 0,
  "mills" : 33,
  "IndexBounds" : [
    ],
  "server" : "g:27017"
}
```



Copyright © Capgemini 2016. All Rights Reserved.

## MongoDB Advanced Usage

### Index

- MongoDB's indexes work almost identically to typical relational database indexes.
- Index optimization for MySQL / Oracle / SQLite will apply equally well to MongoDB.
- If an index has N keys, it will make queries on any prefix of those keys fast.

### Example

- db.people.find({"username" : "mark"})
- db.people.ensureIndex({"username" : 1})
- db.people.find({"date" : date1}).sort({"date" : 1, "username" : 1})
- db.ensureIndex({"date" : 1, "username" : 1})
- db.people.find({"username" : "mark"}).explain()



Copyright © Capgemini 2016. All Rights Reserved. 30

Do not index every key. This will make inserts slow, take up lots of space, and probably not speed up your queries very much. Figure out what queries you are running, what the best indexes are for these queries, and make sure that the server is using the indexes you've created using the explain and hint tools described in the next section.

## MongoDB Advanced Usage (contd.)

Indexes can be created on keys in embedded documents in the same way that they are created on normal keys.

Indexing for Sorts: Indexing the sort allows MongoDB to pull the sorted data in order, allowing you to sort any amount of data without running out of memory.

Index Nameing rule: *keyname<sub>1</sub>\_dir<sub>1</sub>\_keyname<sub>2</sub>\_dir<sub>2</sub>...\_keyname<sub>N</sub>\_dir<sub>N</sub>*, where keyname<sub>X</sub> is the index's key and dir<sub>X</sub> is the index's direction (1 or -1).

```
- db.blog.ensureIndex({"comments.date" : 1})  
- db.people.ensureIndex({"username" : 1}, {"unique" : true})  
- db.people.ensureIndex({"username" : 1}, {"unique" : true, "dropDups" : true})  
- autocomplete:PRIMARY> db.system.indexes.find()  
{ "name" : "_id", "ns" : "test.fs.files", "key" : { "_id" : 1 }, "v" : 0 }  
{ "ns" : "test.fs.files", "key" : { "filename" : 1 }, "name" : "filename_1", "v" : 0 }  
{ "name" : "_id", "ns" : "test.fs.chunks", "key" : { "_id" : 1 }, "v" : 0 }  
{ "ns" : "test.fs.chunks", "key" : { "files_id" : 1, "n" : 1 }, "name" : "files_id_1_n_1", "v" : 0 }
```



Copyright © Capgemini 2016. All Rights Reserved.

Do not index every key. This will make inserts slow, take up lots of space, and probably not speed up your queries very much. Figure out what queries you are running, what the best indexes are for these queries, and make sure that the server is using the indexes you've created using the explain and hint tools described in the next section.

## MongoDB Advanced Usage (contd.)

### Index continue: explain()

- Explain will return information about the indexes used for the query (if any) and stats about timing and the number of documents scanned.

```
{  
  "_id": 1,  
  "cursor": "BasicCursor name_1",  
  "nscanned": 1,  
  "nscannedObject": 1,  
  "n": 1,  
  "millis": 0,  
  "traversed": 0,  
  "throttleSleep": 0,  
  "totalMemory": 0,  
  "indexOnly": false,  
  "indexBounds": [  
    {"name": "  
      $gt": 0,  
      "$lt": 1  
    }  
  ]  
}
```



Copyright © Capgemini 2016. All Rights Reserved. 40

The important parts of this result are as follows:

"cursor" : "BasicCursor"

This means that the query did not use an index (unsurprisingly, because there was no query criteria). We'll see what this value looks like for an indexed query in a moment.

"nscanned" :

This is the number of documents that the database looked through. You want to make sure this is as close to the number returned as possible.

"n" :

This is the number of documents returned. We're doing pretty well here, because the number of documents scanned exactly matches the number returned. Of course, given that we're returning the entire collection, it would be difficult to do otherwise.

"millis" :

The number of milliseconds it took the database to execute the query. 0 is a good time to shoot for.

## MongoDB Advanced Usage (contd.)

### Index continue: hint()

- If you find that Mongo is using different indexes than you want it to for a query, you can force it to use a certain index by using hint.
  - `db.c.find({"age" : 14, "username" : /.*/}).hint({"username" : 1, "age" : 1})`

### Index continue: change index

- `db.runCommand({"dropIndexes" : "foo", "index" : "alphabet"})`
- `db.people.ensureIndex({"username" : 1}, {"background" : true})`
- Using the {"background" : true} option builds the index in the background, while handling incoming requests. If you do not include the background option, the database will block all other requests while the index is being built.



Copyright © Capgemini 2016. All Rights Reserved. #1

Hinting is usually unnecessary. MongoDB has a query optimizer and is very clever about

choosing which index to use. When you first do a query, the query optimizer tries out a number of query plans concurrently. The first one to finish will be used, and the rest of the query executions are terminated. That query plan will be remembered for future queries on the same keys. The query optimizer periodically retries other plans, in case you've added new data and the previously chosen plan is no longer best. The only part you should need to worry about is giving the query optimizer useful indexes to choose from.

## MongoDB Advanced Usage (contd.)

### Advanced Usage

- Index
- Aggregation

```
- db foo count()
- db foo count("x", 1)
- db runCommand("distinct", "people", "key", "Age")
- Group
-   { "day": "2010/10/03", "time": "10/3/2010 03:57:01 GMT+000", "price": 4.23}
-   { "day": "2010/10/04", "time": "10/4/2010 11:28:39 GMT+000", "price": 4.27}
-   { "day": "2010/10/03", "time": "10/3/2010 09:00:23 GMT+000", "price": 4.10}
-   { "day": "2010/10/06", "time": "10/6/2010 05:27:58 GMT+000", "price": 4.30}
-   { "day": "2010/10/04", "time": "10/4/2010 08:34:50 GMT+000", "price": 4.01}

- db runCommand("group", {
  -   "ref": "books",
  -   "key": "day",
  -   "reduce": "(function(doc, prev) {
    -     if (doc.time > prev.time) {
    -       prev.price = doc.price;
    -       prev.time = doc.time;
    -     }
  })()
```



## Geospatial Indexes

Finding the nearest N things to a current location.

MongoDB provides a special type of index for coordinate plane queries, called a geospatial index.

The indexes makes it possible to perform efficient Geospatial queries.

The 2d Geospatial Sphere index allows to perform queries on a earth-like sphere making for better accuracy in matching locations.

MongoDB's geospatial indexes assumes that:

- Whatever you're indexing is a flat plane.
- This means that results aren't perfect for spherical shapes, like the earth, especially near the poles.



Copyright © Capgemini 2016. All Rights Reserved. 43

## Geospatial Indexes (contd.)

A geospatial index can be created using the ensureIndex function, but by passing "2d" as a value instead of 1 or -1:

- > db.map.ensureIndex({"gps" : "2d"})
- "gps" : [ 0, 100 ] }
- { "gps" : { "x" : -30, "y" : 30 } }
- { "gps" : { "latitude" : -180, "longitude" : 180 } }
- > db.star.trek.ensureIndex({"light-years" : "2d"}, {"min" : -1000, "max" : 1000})
- > db.map.find({"gps" : {"\$near" : [40, -73]}})
- > db.map.find({"gps" : {"\$near" : [40, -73]}}).limit(10)



Copyright © Capgemini 2016. All Rights Reserved. 40

## Example

- Take the following example document.

```
{ loc: {  
    type: "Point",  
    coordinates: [60, 79] },  
    type: "house" }
```

```
var locations = db.getSiblingDB("geo").locations;  
locations.ensureIndex({loc: "2dsphere", house: 1});
```



Copyright © Capgemini 2016. All Rights Reserved. 41

## Text Indexes

MongoDB provides text indexes to support query operations that perform a text search of string content. text indexes can include any field whose value is a string or an array of string elements.

To index a field that contains a string or an array of string elements, include the field and specify the string literal "text" in the index document, as in the following example:

- db.reviews.createIndex( { comments: "text" } )



Copyright © Capgemini 2016. All Rights Reserved. 40

## Properties of Indexes

Hashes Indexes

Unique Indexes

Sparse Indexes

## Hashed Indexes

Hashed indexes maintain entries with hashes of the values of the indexed field. The hashing function collapses embedded documents and computes the hash for the entire value but does not support multi-key (i.e. arrays) indexes.

Create a hashed index using an operation that resembles the following:

- db.active.createIndex( { a: "hashed" } )
- This operation creates a hashed index for the active collection on the a field.



Copyright © Capgemini 2016. All Rights Reserved. 48

## Hashed Indexes (contd.)

### Unique Indexes

- The unique property for an index causes MongoDB to reject duplicate values for the indexed field. To create a unique index on a field that already has duplicate values.

```
db.collection.ensureIndex( { "a.b": 1 }, { unique: true } )
```

```
db.accounts.ensureIndex( { username: 1 }, { unique: true, dropDups: true } )
```



Copyright © Capgemini 2016. All Rights Reserved. 40

## Unique Index

Unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.

By default, unique is false on MongoDB indexes.

To solve this issue MongoDB allows to add the *unique* option to the *ensureIndex* statement when created, making the target field unique across the entire collection.

```
db.users.ensureIndex( { "userId": 1 }, { unique: true } )
```



Copyright © Capgemini 2016. All Rights Reserved. 30

## Sparse Index

Sparse indexes only contain entries for documents that have the indexed field, even if the index field contains a null value.

The index skips over any document that is missing the indexed field. The index is "sparse" because it does not include all documents of a collection.

By contrast, non-sparse indexes contain all documents in a collection, storing null values for those documents that do not contain the indexed field.



Copyright © Capgemini 2016. All Rights Reserved. 11

## Sparse Index (contd.)

The sparse property of an index ensures that the index only contain entries for documents that have the indexed field.

The index skips documents that do not have the indexed field.

You can combine the sparse index option with the unique index option to reject documents that have duplicate values for a field but ignore documents that do not have the indexed key.

If we create unique index on column of documents and it don't exists in another documents then it will give error which will be taken care by **sparse : true**.

```
db.scores.ensureIndex( { score: 1 } , { sparse: true } )
```



Copyright © Capgemini 2016. All Rights Reserved. 10

## TTL Indexes

TTL indexes are special indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time. This is ideal for certain types of information like machine generated event data, logs, and session information that only need to persist in a database for a finite amount of time.

**Note :** By default, creating an index blocks all other operations on a database. When building an index on a collection, the database that holds the collection is unavailable for read or write operations until the index build completes. Any operation that requires a read or write lock databases and will wait for the foreground index build to complete.

```
db.people.ensureIndex( { zipcode: 1}, {background: true} )
```



Copyright © Capgemini 2016. All Rights Reserved. 11

## Explain Plan in MongoDB

Like in other RDBMS database, MongoDB also provide Explain Plan option to find query is doing full table scan or Index scan.

```
db.inventory.find( { quantity: { $gte: 100, $lte: 200 } } ).explain()
```

**n**

- No of document matches query selection criteria or query results.

**nscanned**

- Indicate no of document Mongodb scanned index entry.

**nscannedObjects**

- Total number of documents scanned. when this is equal to nscanned then collection scan.

**indexOnly**

- True means that Mongodb used only the index to match and return result of query.

```
{
  "cursor": "BtreeCursor",
  "location": 1,
  "isMultiKey": false,
  "n": 1,
  "nscanned": 1,
  "nscannedObjects": 1,
  "nscannedAllPlans": 1,
  "nscannedAllPlans": 1,
  "scanAndOrder": false,
  "indexOnly": false,
  "nYields": 0,
  "nChunkSkips": 0,
  "millis": 0,
  "indexBounds": [
    {
      "location": [
        null,
        null
      ]
    }
  ],
  "server": "MongoDB_BIM",
  "filterSet": false
}
```



Copyright © Capgemini 2016. All Rights Reserved. 50

## Why MongoDB: Performance

### No Joins + No multi-row transactions

- = Fast Reads
- = Fast Writes (b/c you write to fewer tables, no trans. log)

### Async writes

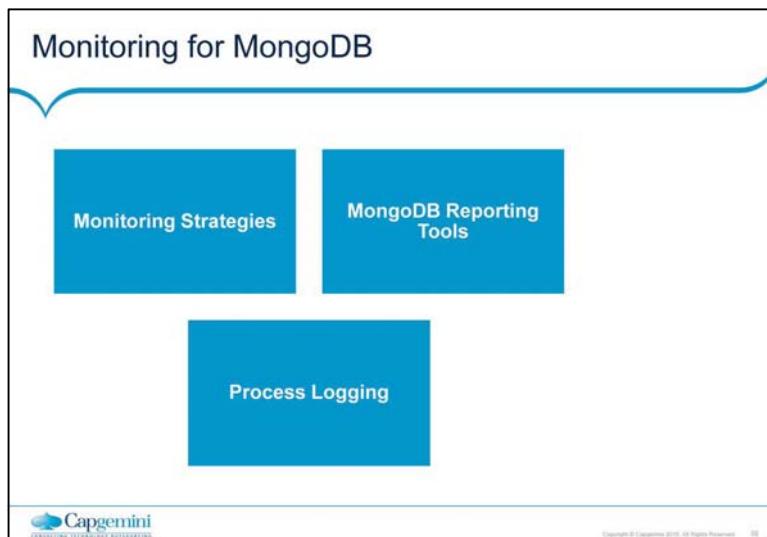
- = you don't wait for inserts to complete.
- (optional, though)

### Secondary Indexes

- = Index on embedded document fields for superfast ad-hoc queries.
- Indexes live in RAM.



Copyright © Capgemini 2016. All Rights Reserved.



## MongoDB Strategies

There are three methods for collecting data about the state of a running MongoDB instance:

First, there is a set of utilities distributed with MongoDB that provides real-time reporting of database activities.

Second, **database commands** return statistics regarding the current database state with greater fidelity.

Third, **MongoDB Cloud Manager**, a hosted service, and **Ops Manager**, an on-premise solution available in **MongoDB Enterprise Advanced**, provide monitoring to collect data from running MongoDB deployments as well as providing visualization and alerts based on that data.



Copyright © Capgemini 2016. All Rights Reserved. 10

## MongoDB Reporting Tools

### Utilities

- The MongoDB distribution includes a number of utilities that quickly return statistics about instances' performance and activity. Typically, these are most useful for diagnosing issues and assessing normal operation.

## Mongostat

**Mongostat** captures and returns the counts of database operations by type (e.g. insert, query, update, delete, etc.).

These counts report on the load distribution on the server.

Use **mongostat** to understand the distribution of operation types and to inform capacity planning.

Mongostat is functionally similar to the UNIX/Linux file system utility **vmstat**, but provides data regarding **mongod** and **mongos** instances.



Copyright © Capgemini 2016. All Rights Reserved. 30

## Mongostat (contd.)

Command also shows when you're hitting page faults, and showcase your lock percentage. This means that you're running low on memory, hitting write capacity or have some performance issue.

To run the command start your mongod instance. In another command prompt go to **bin directory of your mongodb** installation and type **mongostat**.

**Example:** D:\set up\mongodb\bin>mongostat



Copyright © Capgemini 2016. All Rights Reserved. 30

## MongoDB – Indexing

## Output of the Command



## Mongotop Command

This command track and report the read and write activity of MongoDB instance on a collection basis.

By default mongotop returns information in each second, by you can change it accordingly.

We can check that this read and write activity matches your application intention, and you're not firing too many writes to the database at a time.

Reading too frequently from disk, or are exceeding your working set size.



Copyright © Capgemini 2016. All Rights Reserved. 43

### Mongotop Command (contd.)

To run the command start your mongod instance.

In another command prompt go to **bin** directory of your **mongodb** installation and type **mongotop**.

D:\set up\mongodb\bin>mongotop 30

The above example will return values every 30 seconds.



Copyright © Capgemini 2016. All Rights Reserved. 43

## ServerStatus Command

The **serverStatus** command, or **db.serverStatus()** from the shell, returns a general overview of the status of the database, detailing disk usage, memory use, connection, journaling, and index access.

The command returns quickly and does not impact MongoDB performance.

**serverStatus** outputs an account of the state of a MongoDB instance.



Copyright © Capgemini 2016. All Rights Reserved. 60

## Dbstats Command

The **dbStats** command, or **db.stats()** from the shell, returns a document that addresses storage use and data volumes.

The **dbStats** reflect the amount of storage used, the quantity of data contained in the database, and object, collection, and index counters.



Copyright © Capgemini 2016. All Rights Reserved. 01

## Collstats Command

The **collStats** or **db.collection.stats()** from the shell that provides statistics that resemble **dbStats** on the collection level, including a count of the objects in the collection, the size of the collection, the amount of disk space used by the collection, and information about its indexes.



Copyright © Capgemini 2018. All Rights Reserved. 30

## ReplSetGetStatus Command

The **replSetGetStatus** command (`rs.status()` from the shell) returns an overview of your replica set's status. The **replSetGetStatus** document details the state and configuration of the replica set and statistics about its members.



Copyright © Capgemini 2016. All Rights Reserved. 87

## Process Logging

During normal operation, **mongod** and **mongo** instances report a live account of all server activity and operations to either standard output or a log file. The following runtime settings control these options.

**Quiet:**

Limits the amount of information written to the log or output.

**Verbosity:**

Increases the amount of information written to the log or output. You can also modify the logging verbosity during runtime with the **logLevel** parameter or the **db.setLogLevel()** method in the shell.

**Path:**

Enables logging to a file, rather than the standard output. You must specify the full path to the log file when adjusting this setting.

**LogAppend:**

Adds information to a log file instead of overwriting the file.



Copyright © Capgemini 2016. All Rights Reserved. 10

## Profiler

The database profiler collects fine grained data about MongoDB write operations, cursors, database commands on a running **mongod** instance.

You can enable profiling on a per-database or per-instance basis.

The ***profiling level*** is also configurable when enabling profiling. The profiler is *off* by default.



Copyright © Capgemini 2016. All Rights Reserved. 30

## Profiling Levels

The database profiler writes all the data it collects to the `system.profile` collection, which is a **capped collection**.

### Profiling Levels - The following profiling levels are available:

- **0:** The profiler is off, does not collect any data. `Mongod` always writes operations longer than the `slowOpThresholdMs` threshold to its log. This is the default profiler level.
- **1:** Collects profiling data for slow operations only. By default slow operations are those slower than 100 milliseconds.
- You can modify the threshold for “slow” operations with the `slowOpThresholdMs` runtime option or the `setParameter` command. See the **Specify the Threshold for Slow Operations** section for more information.
- **2:** Collects profiling data for all database operations.



Copyright © Capgemini 2016. All Rights Reserved.

## Enable Database Profiling and Set the Profiling Level

You can enable database profiling from the **mongo** shell or through a driver using the **profile** command.

When you enable profiling, you also set the **profiling level**. The profiler records data in the **system.profile** collection.

MongoDB creates the **system.profile** collection in a database after you enable profiling for that database.

To enable profiling and set the profiling level, use the **db.setProfilingLevel()** helper in the **mongo** shell, passing the profiling level as a parameter.



Copyright © Capgemini 2016. All Rights Reserved. 71

## Enable Database Profiling and Set the Profiling Level (contd.)

To enable profiling for all database operations, consider the following operation in the **mongo** shell:

- `db.setProfilingLevel(2)`

The shell returns a document showing the *previous* level of profiling.  
The "ok" : 1 key-value pair indicates the operation succeeded:

- `{ "was" : 0, "slowms" : 100, "ok" : 1 }`



Copyright © Capgemini 2016. All Rights Reserved. 72

## Thresholds for Slow Operations

The threshold for slow operations applies to the entire **mongod** instance. When you change the threshold, you change it for all databases on the instance.

By default the slow operation threshold is 100 milliseconds. Databases with a profiling level of 1 will log operations slower than 100 milliseconds.

To change the threshold, pass two parameters to the **db.setProfilingLevel()** helper in the **mongo** shell. The first parameter sets the profiling level for the current database, and the second sets the default slow operation threshold *for the entire mongod instance*.



Copyright © Capgemini 2016. All Rights Reserved.

## Setting Profiling Level

The following command sets the profiling level for the current database to 0, which disables profiling, and sets the slow-operation threshold for the **mongod** instance to 20 milliseconds.

Any database on the instance with a profiling level of 1 will use this threshold:

- `db.setProfilingLevel(0,20)`



Copyright © Capgemini 2016. All Rights Reserved.

## Checking Profiling Level

To view the *profiling level*, issue the following from the **mongo** shell:

- `db.getProfilingStatus()`

The shell returns a document similar to the following:

- `{ "was" : 0, "slowms" : 100 }`

The `was` field indicates the current level of profiling.

The `slowms` field indicates how long an operation must exist in milliseconds for an operation to pass the “slow” threshold. MongoDB will log operations that take longer than the threshold if the profiling level is 1.



Copyright © Capgemini 2016. All Rights Reserved.

## Get Profiling Level

To return only the profiling level, use the `db.getProfilingLevel()` helper in the `mongo` as in the following:

- `db.getProfilingLevel()`

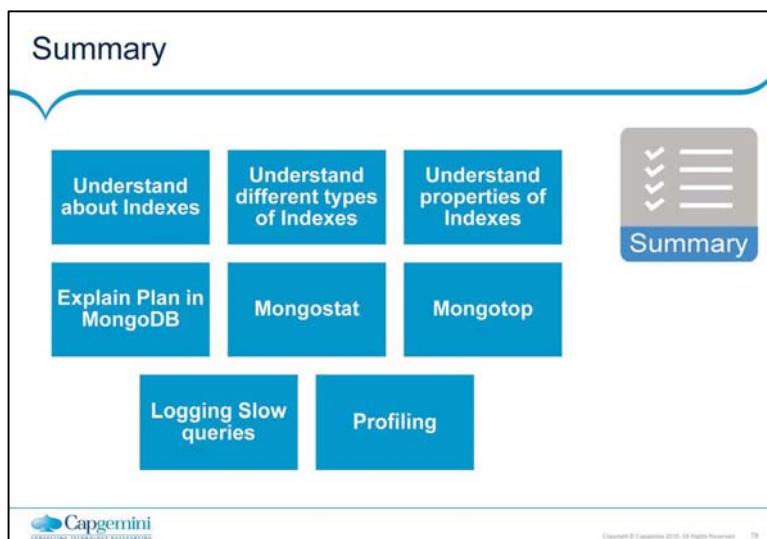
## Disable Profiling

To disable profiling, use the following helper in the **mongo** shell:

- db.setProfilingLevel(0)



Copyright © Capgemini 2018. All Rights Reserved. 77



## **MongoDB – Replication & Sharding**

Lesson 06

## MongoDB – Replication & Sharding

Version Sheet: MANDATORY (hidden in ‘slide show’ mode)

Version	Changes	Author
001	Redesign	Rohan Salvi

 Capgemini  
Engineering Services

Copyright © Capgemini 2016. All Rights Reserved. 2

## Note to the SME:

Please read before you begin reviewing this module as SME

Dark Red box with white text

Note to the SME

The SME must provide missing info.

Amber box with black text

Note to the SME

The SME must validate the re-design/ the modification/ addition.

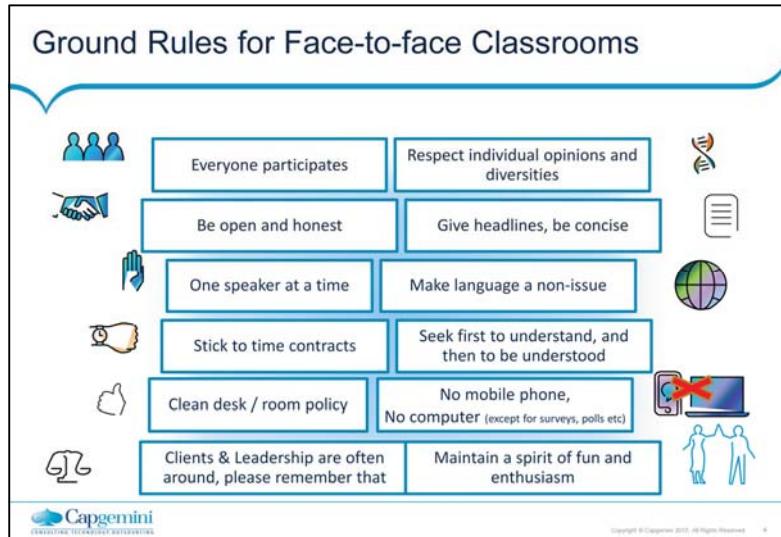
Note to the SME:  
This is a temporary slide and will not be part of the final upload at all. It is to help the SME understand how we will communicate changes to them.

To Review and Navigate the comments in the presentation Click on the "Review" Tab and then Click "Next" to review all the comments . Also you can add your comments using the "New comment" button

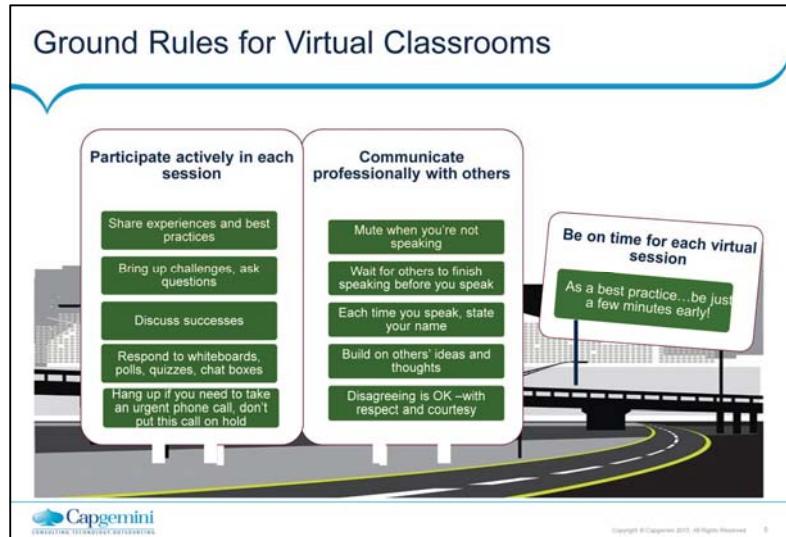
The comments in the review section have also been updated in a green textbox with black text.



Copyright © Capgemini 2016. All Rights Reserved



Best to print on an A3 sheet and display in class. The idea is to co-create, to connect to learn, to do-learn-do (knowing is not enough, the real purpose is to apply what we learn on the job). It would also be great to have the class share their learning with their teams. You are the facilitator, you will help all share knowledge and learn new concepts or skills, there will be no lectures, everyone is a teacher.



Show this slide and hide the one for onsite classrooms, if this is a virtual session.

ONLY for Virtual classrooms

## Module at a Glance

<b>Target Audience:</b>		SME to provide the details required in the table.
<b>Course Level:</b>	Basic	
<b>Duration (in hours):</b>	30 mins	
<b>Pre-requisites, if any:</b>	NA	
<b>Post-requisites, if any:</b>	<i>Submit Session Feedback</i>	
<b>Relevant Certifications:</b>	None	



Copyright © Capgemini 2016. All Rights Reserved

## Introductions (for Virtual Classrooms)

The template consists of a central title 'Introductions (for Virtual Classrooms)' with a blue wavy underline. Below the title are two large rectangular boxes, each labeled 'Business Photo'. Between these boxes is a red rounded rectangle containing the text 'SME to provide the photos and names of the facilitators.' Below each 'Business Photo' box is a smaller box labeled 'Facilitator Name Role' and 'Moderator Name Role' respectively. At the bottom left is the Capgemini logo, and at the bottom right is a small copyright notice: 'Copyright © Capgemini 2016. All Rights Reserved'.

Add pics and Capgemini roles for the Facilitator and moderator. Establish their credibility, what qualifies them to facilitate this session.

## Course Map

- 1 Understand about Replication
- 2 Purpose of Replication
- 3 Understand Replica Set
- 4 Sharding
- 5 Sharding Mechanics
- 6 GridFS



Copyright © Capgemini 2012. All Rights Reserved.

## Module Objectives



### What you will learn

At the end of this module, you will learn:

- Replication and Sharding in MongoDB

Note to the SME : Please provide the module Objectives or validate the partially updated content

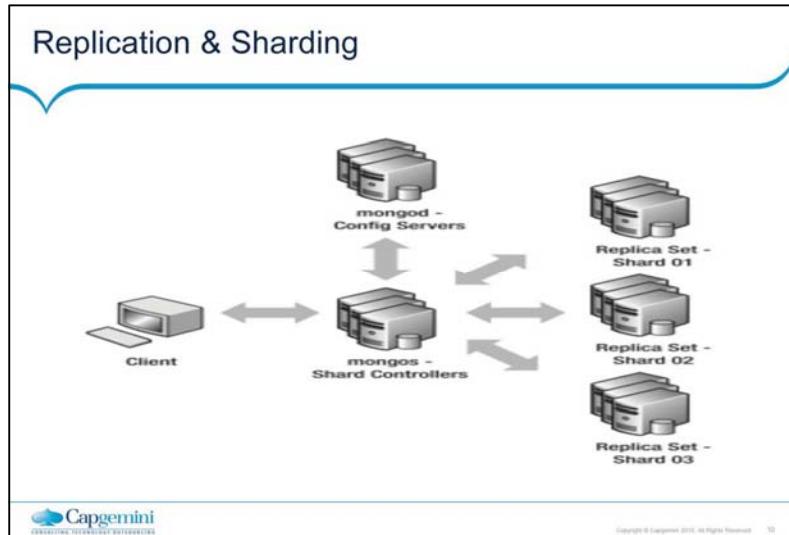


### What you will be able to do

At the end of this module, you be able to:

- Understand what is Replication and Sharding
- Explain the purpose of Replication
- Describe what is a Replica Set
- State the features of Sharding mechanics
- Explain GridFS

## MongoDB – Replication & Sharding



## Replication

What is replication?

Purpose of replication / Redundancy

- Fault tolerance
- Availability
- Increase read capacity



Copyright © Capgemini 2014. All Rights Reserved.

## Why Replication?

How many have faced node failures?

How many have been woken up from sleep to do a fail-over(s)?

How many have experienced issues due to network latency?

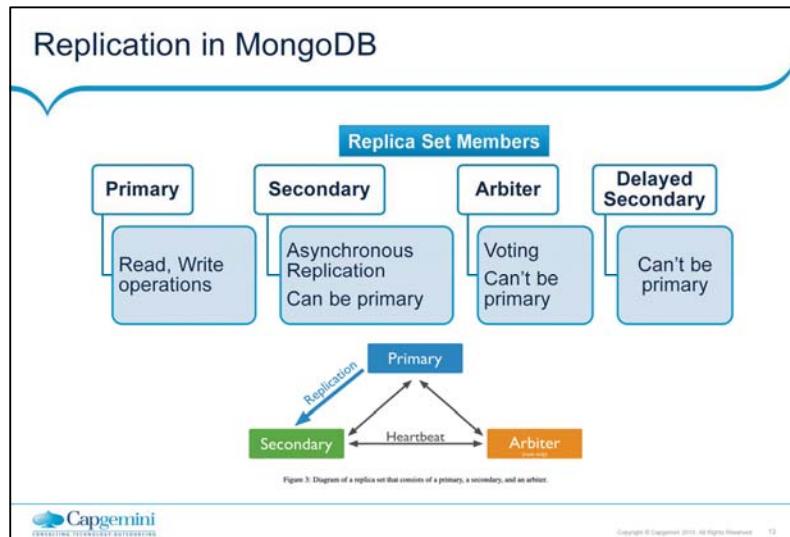
Different uses for data:

- Normal processing
- Simple analytics



Copyright © Capgemini 2012. All Rights Reserved.

## MongoDB – Replication & Sharding



## Replication in MongoDB

- Automatic Failover
  - Heartbeats
  - Elections
- The Standard Replica Set Deployment
- Deploy an Odd Number of Members
- Rollback
- Security
  - SSL/TLS

The diagram illustrates the election of a new primary in a three-member replica set. It shows two secondary nodes and one primary node. The primary node has a red 'X' over it, indicating it is no longer the primary. An arrow labeled 'Heartbeat' points from one secondary to the other. A dashed box labeled 'Election for New Primary' encloses both secondaries. An arrow labeled 'New Primary Elected' points to a final state where one secondary has become the primary, indicated by a blue box. Arrows labeled 'Replication' and 'Heartbeat' show data flow between the primary and the new secondary.

Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unavailable. The loss of a primary triggers an election where one of the secondaries becomes the new primary.

## Replication

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication protects a database from the loss of a single server. Replication also allows to recover from hardware failure and service interruptions. With additional copies of the data, can dedicate one to disaster recovery, reporting, or backup.

In some cases, we can use replication to increase read capacity. Clients have the ability to send read and write operations to different servers. We can also maintain copies in different data centers to increase the locality and availability of data for distributed applications.

A replica set in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments.



Copyright © Capgemini 2016. All Rights Reserved.

## Replication in MongoDB



A replica set is a group of mongod instances that host the same data set. One mongod, the primary, receives all write operations. All other instances, secondaries, apply operations from the primary so that they have the same data set.

The primary accepts all write operations from clients. Replica set can have only one primary. Because only one member can accept write operations, replica sets provide strict consistency for all reads from the primary. To support replication, the primary logs all changes to its data sets in its oplog.

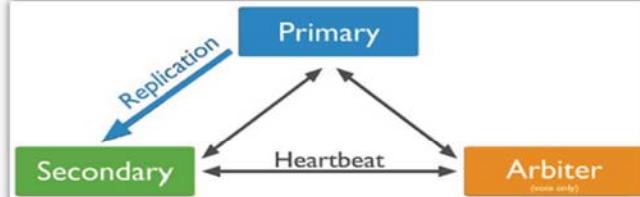
## Replication in MongoDB (contd.)



The secondaries replicate the primary's oplog and apply the operations to their data sets. Secondary's data sets reflect the primary's data set. If the primary is unavailable, the replica set will elect a secondary to be primary.

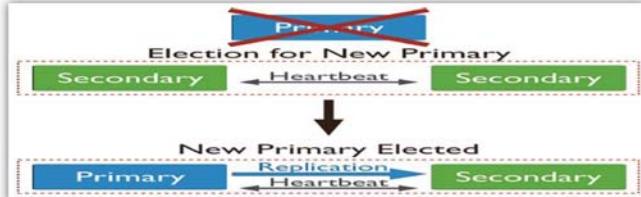
By default, clients read from the primary, however, clients can specify a read preferences to send read operations to secondaries.

### Replication in MongoDB (contd.)



We can add an extra mongod instance to a replica set as an arbiter. Arbiters do not maintain a data set. Arbiters only exist to vote in elections. If your replica set has an even number of members, add an arbiter to obtain a majority of votes in an election for primary. Arbiters do not require dedicated hardware.

## Replication in MongoDB (contd.)



### Automatic Failover

When a primary does not communicate with the other members of the set for more than 10 seconds, the replica set will attempt to select another member to become the new primary. The first secondary that receives a majority of the votes becomes primary.

## Replica Set Members

- Primary:**
  - Receives all write operations.
- Secondaries:**
  - Replicate operations from the primary to maintain an identical data set. Secondaries may have additional configurations for special usage profiles.
- Arbiters:**
  - Do not keep a copy of the data. However, arbiters play a role in the elections that select a primary if the current primary is unavailable.
- Priority 0 Replica Set Members:**
  - A priority 0 member is a secondary that cannot become primary. Priority 0 members cannot trigger elections. Otherwise these members function as normal secondaries. A priority 0 member maintains a copy of the data set, accepts read operations, and votes in elections. Configure a priority 0 member to prevent secondaries from becoming primary, which is particularly useful in multi-data center deployments. It can be used as standby.
- Hidden Replica Set Members:**
  - A hidden member maintains a copy of the primary's data set but is invisible to client applications. Hidden members are good for workloads with different usage patterns from the other members in the replica set. Hidden members must always be priority 0 members and so cannot become primary.
- Delayed Replica Set Members:**
  - Delayed members contain copies of a replica set's data set. However, a delayed member's data set reflects an earlier, or delayed, state of the set. For example, if the current time is 09:52 and a member has a delay of an hour, the delayed member has no operation more recent than 08:52. It helps to recover from various kinds of human error. For example, a delayed member can make it possible to recover from unsuccessful application upgrades and operator errors including dropped databases and collections.



Copyright © Capgemini 2016. All Rights Reserved

## Replica Set Members (contd.)

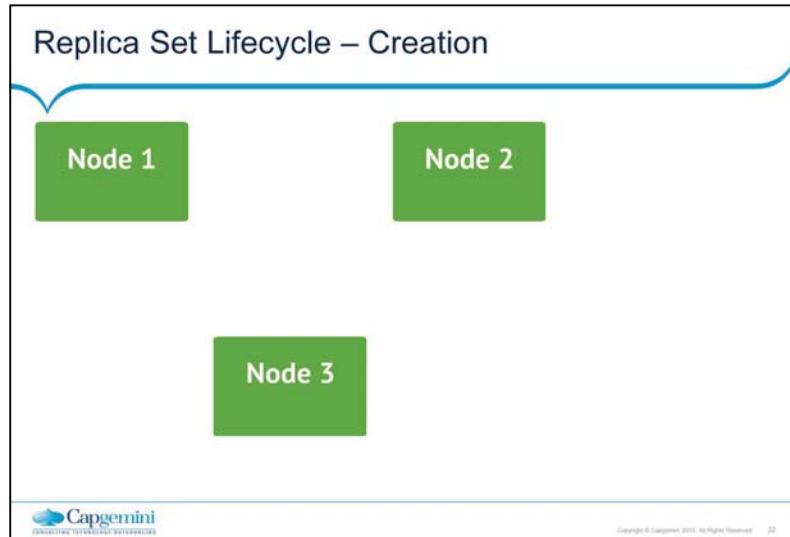
A replica set can have up to 12 members. However, only 7 members can vote at a time.

The minimum requirements for a replica set are:

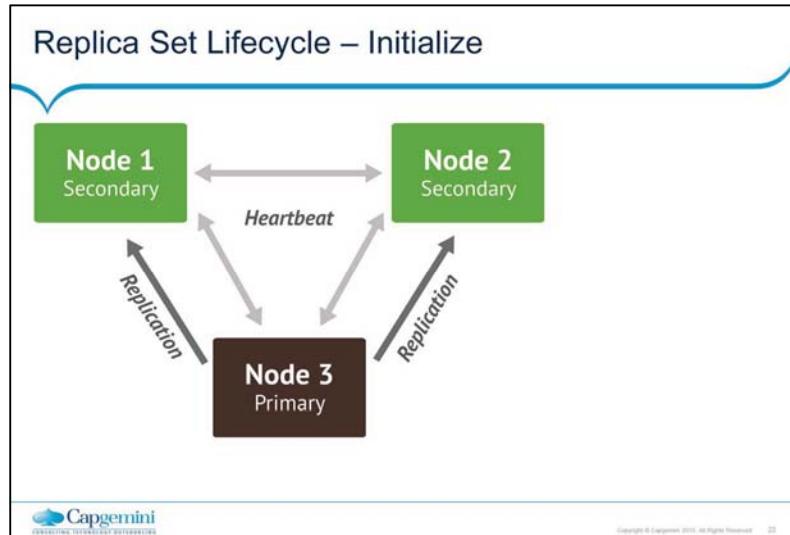
- Primary
- Secondary
- Arbiter.



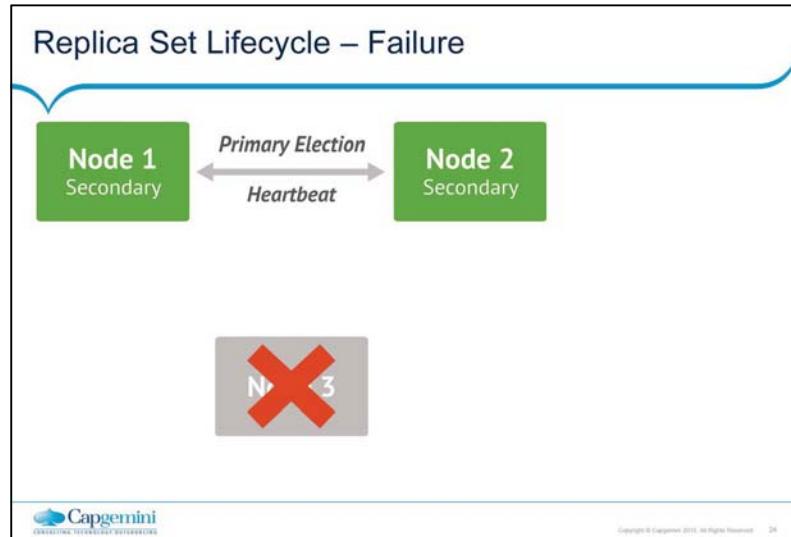
Copyright © Capgemini 2012. All Rights Reserved. 31



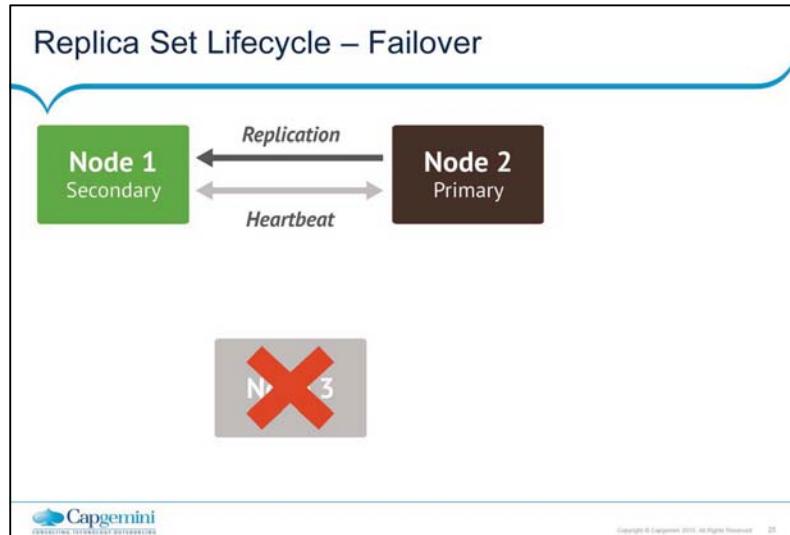
Basic explanation  
2 or more nodes form the set  
Quorum



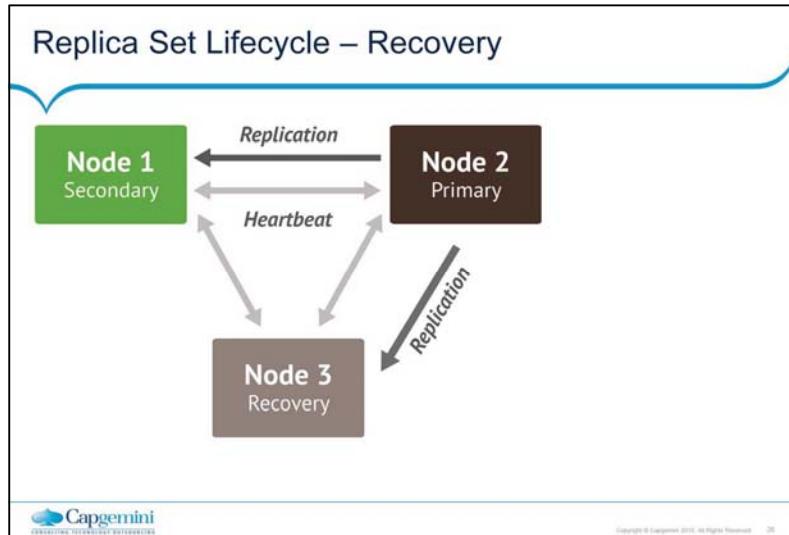
Initialize -> Election  
Primary + data replication from primary to secondary

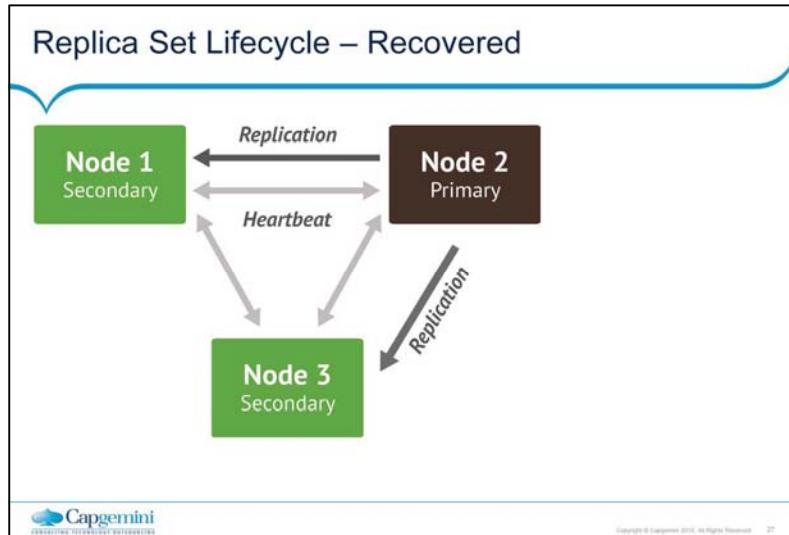


Primary down/network failure  
Automatic election of new primary if majority exists



New primary elected  
Replication established from new primary





## Setting up Local Replica Sets

Make 3 directories to set up 3 replicas on local node.

```
mkdir -p /data/mongodb/rs0-0 /data/mongodb/rs0-1 /data/mongodb/rs0-2
```

Start 3 mongo daemons on local node with 3 different available ports.

- mongod --port 27017 --dbpath /data/mongodb/rs0-0 --replSet rs0 --smallfiles --oplogSize 128
- mongod --port 27018 --dbpath /data/mongodb/rs0-1 --replSet rs0 --smallfiles --oplogSize 128
- mongod --port 27019 --dbpath /data/mongodb/rs0-2 --replSet rs0 --smallfiles --oplogSize 128

Check the mongo daemons running with "ps -ef|grep mongo" command.

```
ps -ef | grep mongo
root 11613 1 0 Dec19 ? 00:14:17 /home/badrap/mongodb/bin/mongod --port 27017 --dbpath /var/mongodb/rs0-0 --replSet rs0 --smallfiles --oplogSize 128
root 11614 1 0 Dec19 ? 00:12:45 /home/badrap/mongodb/bin/mongod --port 27018 --dbpath /var/mongodb/rs0-1 --replSet rs0 --smallfiles --oplogSize 128
root 11615 1 0 Dec19 ? 00:12:45 /home/badrap/mongodb/bin/mongod --port 27019 --dbpath /var/mongodb/rs0-2 --replSet rs0 --smallfiles --oplogSize 128
```



Copyright © Capgemini 2016. All Rights Reserved.

## Failover and Primary Election

This election process will be initiated by any node that cannot reach the primary.

- The new primary must be elected by a *majority of the nodes in the set. The new primary will be the node with the highest priority, using freshness of data to break ties between nodes with the same priority.*

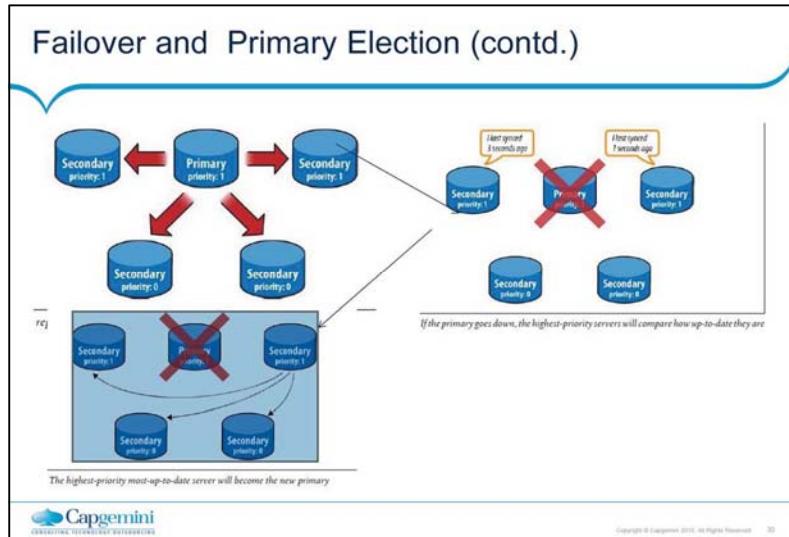
The primary node uses a heartbeat to track how many nodes in the cluster are visible to it.

- If this falls below a majority, the primary will automatically fall back to secondary status.

Whenever the primary changes, the data on the new primary is assumed to be the most up-to-date data in the system.



Copyright © Capgemini 2012. All Rights Reserved. 29



## Sharding

### What is Sharding?

#### Purpose of Sharding

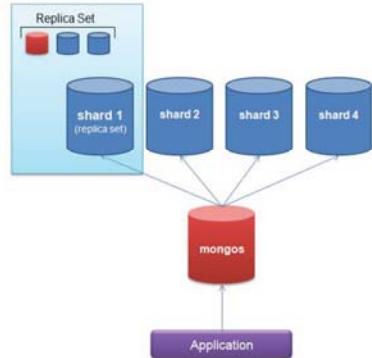
- Horizontal scaling out

#### Query Routers

- mongos

#### Shard keys

- Range based Sharding
- Cardinality
- Avoid hot spotting



Copyright © Capgemini 2012. All Rights Reserved. 31

## Main Components

### Shard

- A Shard is a node of the cluster.
- Each Shard can be a single mongod or a replica set.

### Config Server (meta data storage)

- Stores cluster chunk ranges and locations.
- Can be only 1 or 3 (production must have 3).
- Not a replica set.

### Mongos

- Acts as a router / balancer.
- No local data (persists to config database).
- Can be 1 or many.



Copyright © Capgemini 2012. All Rights Reserved.

## Sharding

Sharding is MongoDB's approach to scaling out.

Sharding allows you to add more machines to handle increasing load and data size without affecting your application.

Manual sharding will work well but become difficult to maintain when adding or removing nodes from the cluster or in face of changing data distributions or load patterns.

MongoDB supports autosharding, which eliminates some of the administrative headaches of manual sharding.



Copyright © Capgemini 2010. All Rights Reserved. 33

## Auto Sharding in MongoDB

The basic concept behind MongoDB's sharding is to break up collections into small chunks.

- We don't need to know what shard has what data, so we run a *router in front of the application, it knows where the data located, so applications can connect to it and issue requests normally.*
- An application will connect to a normal mongod, the router, knowing what data is on which shard, is able to forward the requests to the appropriate shard(s).



Copyright © Capgemini 2012. All Rights Reserved.

## When to Shard

In general, you should start with a non sharded setup and convert it to a sharded one, if and when you need.

When the situations like this, you should probably start to shard:

- You've run out of disk space on your current machine.
- You want to write data faster than a single mongod can handle.
- You want to keep a larger proportion of data in memory to improve performance.



Copyright © Capgemini 2012. All Rights Reserved.

## Shard Keys

When you set up sharding, you choose a key from a collection and use that key's values to split up the data. This key is call a *shard key*.

For example, If we chose "name" as our shard key, one shard could hold documents where the "name" started with A–F, the next shard could hold names from G–P, and the final shard would hold names from Q–Z.

As you added (or removed) shards, MongoDB would rebalance this data so that each shard was getting a balanced amount of traffic and a sensible amount of data.



Copyright © Capgemini 2012. All Rights Reserved

## Chunks

Suppose we add a new shard. Once this shard is up and running, MongoDB will break up the collection into two pieces, called chunks.

A chunk contains all of the documents for a range of values for the shard key.

- For example, if we use "timestamp" as the shard key, so one chunk would have documents with a timestamp value between  $-\infty$  and, say, June 26, 2003, and the other chunk would have timestamps between June 27, 2003 and  $\infty$ .



Copyright © Capgemini 2012. All Rights Reserved. 37

## Pre Sharding a Table

### Determine a shard key

- Define how we distribute data.
- MongoDB's sharding is order-preserving; adjacent data by shard key tends to be on the same server.
- The config database stores all the metadata indicating the location of data by range:
- It should be granular enough to ensure an even distribution of data.

### Chunks

- A contiguous range of data from a particular collection.
- Once a chunk has reached about 200M size, the chunk splits into two new chunks. When a particular shard has excess data, chunks will then migrate to other shards in the system.
- The addition of a new shard will also influence the migration of chunks.

collection	minkey	maxkey	location
users	{ name : 'Miller' }	{ name : 'Nessman' }	shard_2
users	{ name : 'Nessman' }	{ name : 'Ogden' }	shard_4
...			



Copyright © Capgemini 2010. All Rights Reserved. 10

<http://www.mongodb.org/display/DOCS/Choosing+a+Shard+Key>

## Sharding a Table

```
Enable sharding on a database db.runCommand({ "enablesharding" : "foo" })
Enable sharding on collection.
db.runCommand({ "shardcollection" : "foo.bar", "key" : { "_id" : 1 } })

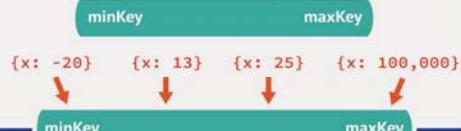
Show autosharding status
> db.printShardingStatus
--- Sharding Status ---
sharding version: { "_id" : 1, "version" : 3 }
shards:
[ "_id" : "shard0", "host" : "localhost:10000" ]
[ "_id" : "shard1", "host" : "localhost:10001" ]
databases:
[ "_id" : "admin", "partitioned" : false, "primary" : "config" ]
[ "_id" : "foo", "partitioned" : false, "primary" : "shard1" ]
[ "_id" : "x", "partitioned" : false, "primary" : "shard0" ]
[ "_id" : "test", "partitioned" : true, "primary" : "shard0" ]
sharded : { "test.foo" : { "key" : { "x" : 1 }, "unique" : false } }
test.foo chunks:
{ "x" : { $minKey : 1 } } -->> { "x" : { $maxKey : 1 } } on : shard0
{ "t" : 1276636243000, "i" : 1 }
```



Copyright © Capgemini 2012. All Rights Reserved.

## Chunk Partitioning

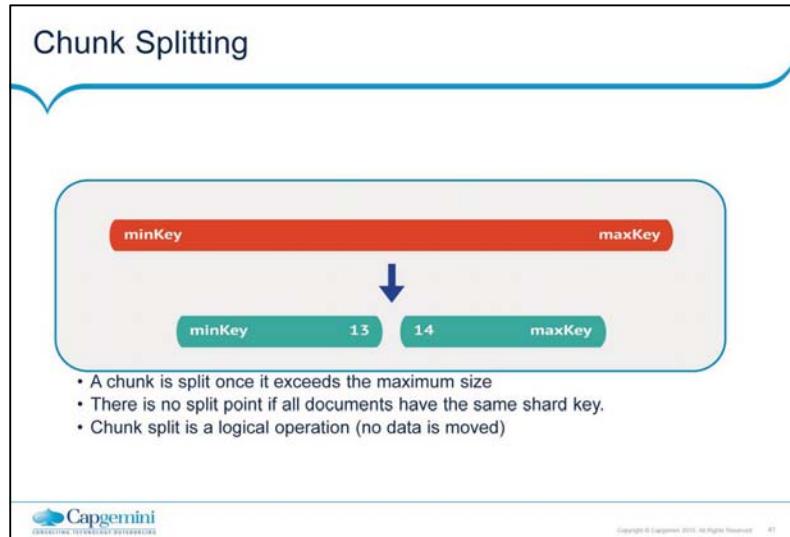
**Chunk** is a section of the entire range.



**minKey**                            **64MB**                            **maxKey**

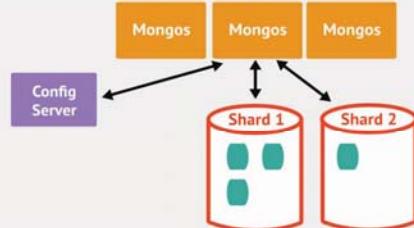


MongoDB – Replication & Sharding



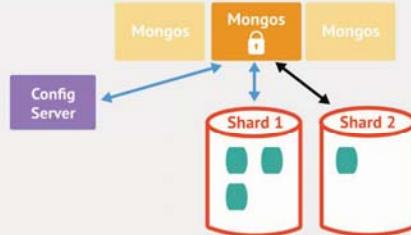
Once chunk size is reached, mongos asks mongod to split a chunk  
+ internal function called `splitVector()`  
mongod counts number of documents on each side of split  
+ based on avg. document size `db.stats()`  
Chunk split is a \*\*logical\*\* operation (no data has moved)

## Balancing



- Balancer is running on mongos.
- Once the difference in chunks between the most dense shard and the least dense shard is above the migration threshold, a balancing round starts.

## Acquiring the Balancer Lock



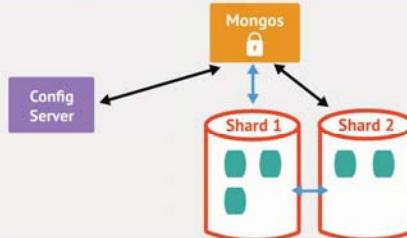
- The balancer on mongos takes out a "balancer lock"
- To see the status of these locks:  
  | use config  
  | db.locks.find({ \_id: 1balancer1 })



Copyright © Capgemini 2012. All Rights Reserved. 47

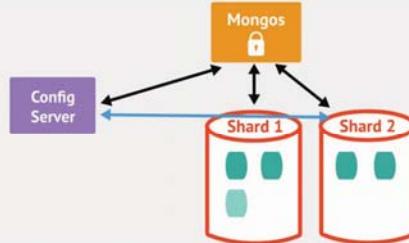
Balancer lock actually held on config server.

## Moving the Chunk



- The mongos sends a moveChunk command to source shard.
- The source shard then notifies destination shard.
- Destination shard starts pulling documents from source shard.

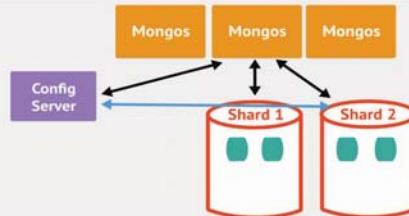
### Committing Migration



- When complete, destination shard updates config server.
  - Provides new locations of the chunks.

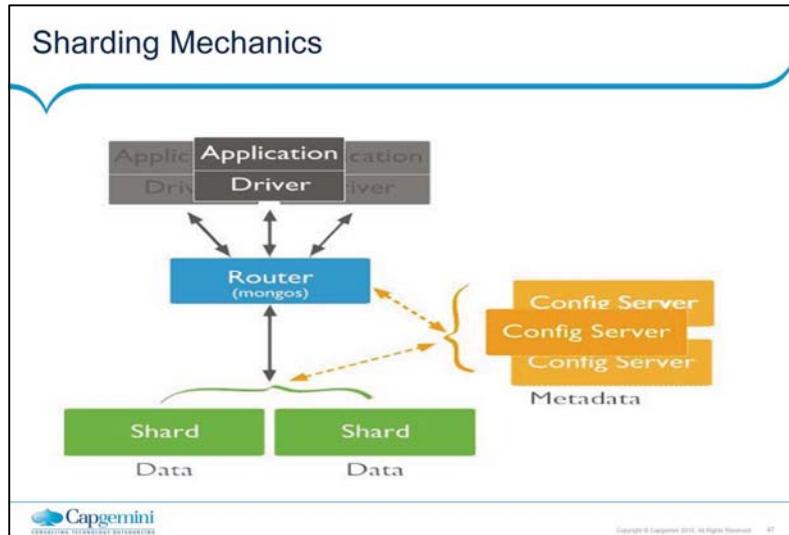
## MongoDB – Replication & Sharding

### Cleanup



- Source shard deletes moved data.
- Must wait for open cursors to either close or time out.
- The `mongos` releases the balancer lock after old chunks are deleted.

## MongoDB – Replication & Sharding



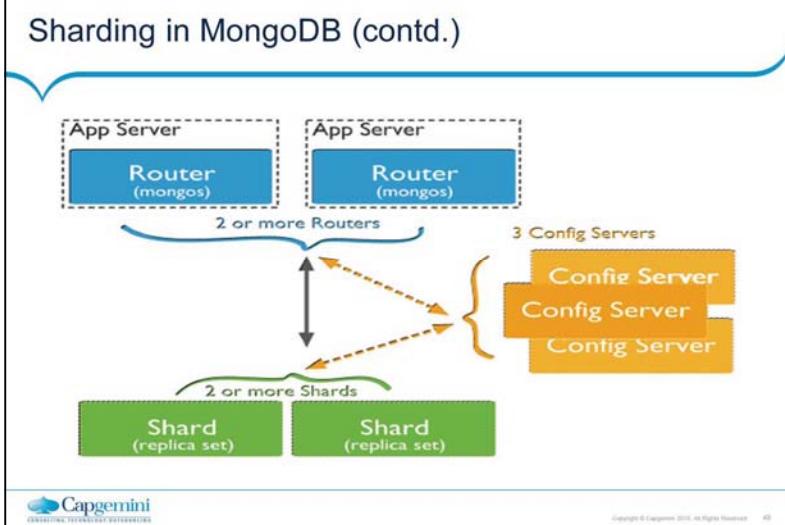
## Sharding in MongoDB

### Sharded cluster has the following components:

**Shards** store the data. To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set

**Query Routers**, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. The query router processes and targets operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Most sharded clusters have many query routers.

**Config servers** store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. Production sharded clusters have exactly 3 config servers.



## Data Partitioning

MongoDB distributes data, or shards, at the collection level. Sharding partitions a collection's data by the shard key. A shard key is either an indexed field or an indexed compound field that exists in every document in the collection.

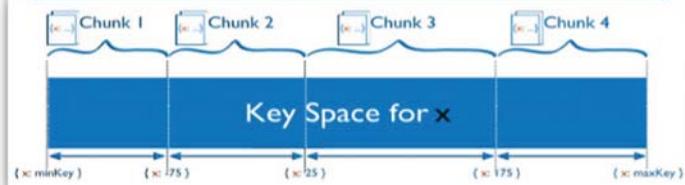
MongoDB divides the shard key values into chunks and distributes the chunks evenly across the shards. To divide the shard key values into chunks, MongoDB uses either range based partitioning or hash based partitioning.



Copyright © Capgemini 2014. All Rights Reserved

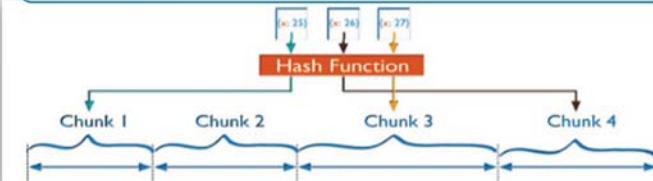
## Data Partitioning (contd.)

**Range Based Sharding:** MongoDB divides the data set into ranges determined by the shard key values to provide range based partitioning.



## Data Partitioning (contd.)

**Hash Based Sharding:** MongoDB computes a hash of a field's value, and then uses these hashes to create chunks.



## GridFS

MongoDB provides GridFS specification for storing and retrieving files such as images, audio files, video files that exceed the BSON-document size limit of 16MB.

GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.

By default GridFS limits chunk size to 255k.

GridFS uses two collections to store files:

- fs.files
- fs.chunks

When we query a GridFS store for a file, the driver or client will reassemble the chunks as needed.

We can also access information from arbitrary sections of files, which allows you to "skip" into the middle of a video or audio file.

GridFS can store any types of file.



Copyright © Capgemini 2012. All Rights Reserved.

## Adding Files in GridFS

Run mongofiles.exe from command prompt as shown below:

```
mongofiles.exe -d <database> -c <collection> -prefix <string> put <filename>
mongofiles.exe -d <database> get <filename>
mongofiles -d records list [to list lists in GridFS]
```

```
C:\Users>mongofiles -d test put c:\MongoDB\mongodb.pdf
2015-03-05T19:33:24.376+0000    connected to: localhost
added file: c:\MongoDB\mongodb.pdf
```

```
C:\Users>mongofiles -d test list
2015-03-05T19:34:21.834+0000    connected to: localhost
c:\MongoDB\mongodb.pdf      182116
```



Copyright © Capgemini 2010. All Rights Reserved.

## GridFS Collections

```
> show collections
emp
foo
fs.chunks
fs.files
system.indexes
> -
```

```
> db.fs.files.find().pretty()
{
    "_id" : ObjectId("54f8af84f019ec1010000001"),
    "chunkSize" : 261120,
    "uploadDate" : ISODate("2015-03-05T19:33:24.470Z"),
    "length" : 182116,
    "md5" : "dc1e86d56dc11bac84cf5ed6ca2bb76a",
    "filename" : "c:\\MongoDB\\mongodb.pdf"
}
```

Note: fs.chunks store binary data and querying this collection will return binary data of file.

Structure is shown right side.

```
{
    "_id" : <ObjectId>,
    "files_id" : <ObjectId>,
    "n" : <num>,
    "data" : <binary>
}
```



Copyright © Capgemini 2014. All Rights Reserved

## Retrieving Files from GridFS

Run **mongofiles.exe** from command prompt as shown below:

- mongofiles.exe -d <database> get <filename>

**Note:** MongoDB copy file to same location from where it has copied. To copy to different location change the file location by renaming filename in `fs.files` collection.

```
C:\Users>mongofiles -d test get c:\MongoDB\mongodb.pdf  
2015-03-05T19:40:47.456+0000 connected to: localhost  
finished writing to: c:\MongoDB\mongodb.pdf
```



Copyright © Capgemini 2014. All Rights Reserved

## Summary

Understand  
about  
Replication

Purpose of  
Replication

Understand  
Replica Set

Sharding

Sharding  
Mechanics

GridFS



Copyright © Capgemini 2017. All Rights Reserved.

# Mongo DB Essentials Lab Book

## MongoDB Exercise in mongo shell

Connect to a running mongo instance, use a database named mongo\_practice.

Document all your queries in a javascript file to use as a reference.

### Insert Documents

Insert the following documents into a movies collection.

```
title : Fight Club
writer : Chuck Palahniuk
year : 1999
actors : [
    Brad Pitt
    Edward Norton
]
title : Pulp Fiction
writer : Quentin Tarantino
year : 1994
actors : [
    John Travolta
    Uma Thurman
]
title : Inglourious Basterds
writer : Quentin Tarantino
year : 2009
actors : [
    Brad Pitt
    Diane Kruger
    Eli Roth
]
title : The Hobbit: An Unexpected Journey
writer : J.R.R. Tolkein
year : 2012
franchise : The Hobbit
title : The Hobbit: The Desolation of Smaug
writer : J.R.R. Tolkein
year : 2013
franchise : The Hobbit
title : The Hobbit: The Battle of the Five Armies
```

writer : J.R.R. Tolkein

year : 2012

franchise : The Hobbit

synopsis : Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness.

title : Pee Wee Herman's Big Adventure

title : Avatar

### Query / Find Documents

query the movies collection to

1. get all documents
2. get all documents with writer set to "Quentin Tarantino"
3. get all documents where actors include "Brad Pitt"
4. get all documents with franchise set to "The Hobbit"
5. get all movies released in the 90s
6. get all movies released before the year 2000 or after 2010

### Update Documents

1. add a synopsis to "The Hobbit: An Unexpected Journey" : "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."
2. add a synopsis to "The Hobbit: The Desolation of Smaug" : "The dwarves, along with Bilbo Baggins and Gandalf the Grey, continue their quest to reclaim Erebor, their homeland, from Smaug. Bilbo Baggins is in possession of a mysterious and magical ring."
3. add an actor named "Samuel L. Jackson" to the movie "Pulp Fiction"

### Text Search

1. find all movies that have a synopsis that contains the word "Bilbo"
2. find all movies that have a synopsis that contains the word "Gandalf"
3. find all movies that have a synopsis that contains the word "Bilbo" and not the word "Gandalf"
4. find all movies that have a synopsis that contains the word "dwarves" or "hobbit"
5. find all movies that have a synopsis that contains the word "gold" and "dragon"

### Delete Documents

1. delete the movie "Pee Wee Herman's Big Adventure"
2. delete the movie "Avatar"

**Relationships****Insert the following documents into a users collection**

```
username : GoodGuyGreg  
first_name : "Good Guy"  
last_name : "Greg"  
username : ScumbagSteve  
full_name :  
    first : "Scumbag"  
    last : "Steve"
```

**Insert the following documents into a posts collection**

```
username : GoodGuyGreg  
title : Passes out at party  
body : Wakes up early and cleans house  
username : GoodGuyGreg  
title : Steals your identity  
body : Raises your credit score  
username : GoodGuyGreg  
title : Reports a bug in your code  
body : Sends you a Pull Request  
username : ScumbagSteve  
title : Borrows something  
body : Sells it  
username : ScumbagSteve  
title : Borrows everything  
body : The end  
username : ScumbagSteve  
title : Forks your repo on github  
body : Sets to private
```

**Insert the following documents into a comments collection**

```
username : GoodGuyGreg  
comment : Hope you got a good deal!  
post : [post_obj_id]  
where [post_obj_id] is the ObjectId of the posts document: "Borrows something"  
  
username : GoodGuyGreg  
comment : What's mine is yours!  
post : [post_obj_id]  
where [post_obj_id] is the ObjectId of the posts document: "Borrows everything"  
  
username : GoodGuyGreg  
comment : Don't violate the licensing agreement!  
post : [post_obj_id]
```

where [post\_obj\_id] is the ObjectId of the posts document: "Forks your repo on github

username : ScumbagSteve

comment : It still isn't clean

post : [post\_obj\_id]

where [post\_obj\_id] is the ObjectId of the posts document: "Passes out at party"

username : ScumbagSteve

comment : Denied your PR cause I found a hack

post : [post\_obj\_id]

where [post\_obj\_id] is the ObjectId of the posts document: "Reports a bug in your code"

## Querying related collections

1. find all users
2. find all postsG
3. find all posts that was authored by "Good Guy Greg"
4. find all posts that was authored by "ScumbagSteve"
5. find all comments
6. find all comments that was authored by "GoodGuyGreg"
7. find all comments that was authored by "ScumbagSteve"
8. find all comments belonging to the post "Reports a bug in your code"

## Exercises on Crud Operations

---

1. Find scores not greater than 78

```
db.scores.find({score:{$not : {$gt:78}}},{student:true,score:true})
```

2. Find the scores greater than 79 and less than 92 with the type essay .

```
db.scores.find({score:{$gte :79,$lte : 92},type:"essay"})
```

3. Find all documents with type: essay and score: 50 and only retrieve the *student* field?

3. To find the documents that have the profession fied

```
Db..people.find({profession :{$exists:true}})
```

3. Display the document where the profession field does not exist

```
db.people.find({profession : { $exists:false }})
```

4. Find all documents where the name field is a string

```
db.people.find({name:{$type :2}})
```

5. To find the documents with the name field that has the letter 'a'

```
db.people.find({name:{$regex : "a"}})
```

6. Find the documents with name ending in e

```
Db.people.find({name:{$regex : "e$"}})
```

7. Find all those people who name ends with an e, or who had an age we can do so using \$ operator

```
db.people.find({$or :[{ name: {$regex :"e$"}}, {age : {$exists : true}}]})
```

8. Find all documents in the *scores* collection where the *score* is less than 50 or greater than 90?

```
db.scores.find({$or :[{score : { $lt:50}},{ score:{ $gt :90 }}] })
```

9. Find the documents with the name greater than c and that contains the letter 'a' in it

```
db.people.find({$and :[{name :{$gt :"C"}}, {name :{$regex :"a" }}]})
```

### Documents can nest

Lets see an example

```
Db.users.insert ({ name : "richard", email :{ work :" richard@10gen.com", personal :"kreuter@example.com "}})
```

Here email field is a nested document

```
> db.users.findOne()  
{  
  "_id" : ObjectId("56cc08c875913d0a9c0b0614"),  
  "name" : "Richard Keauter",  
  "city of birth" : "chicago",  
  "favorite_color" : "red"  
}  
  
> db.users.find({email :{work :"richard@10gen.com",personal :  
"kreuter@gmail.com"}});  
{ "_id" : ObjectId("56d82de70ef02d0aec0aacb5"), "name" : "richard", "email" : {  
"work" : "richard@10gen.com", "personal" : "kreuter@gmail.com" } }  
  
db.users.find({email :{personal : "kreuter@gmail.com", work :  
"richard@10gen.com"});  
  
db.users.find({email :{work :"richard@10gen.com"})}
```

The above query does not give any result becos the query will match only a document whose email field is exactly work:"Richard@10gen.com"

To resolve the above issue , Mongodb has special syntax that allows to reach the inside of the email field for a specific embedded field regardless what are fields are present around . We call that as dot notation .

```
> db.users.find({"email.work" :"richard@10gen.com"})
```

```
{ "_id" : ObjectId("56d82de70ef02d0aec0aacb5"), "name" : "richard", "email" : {  
"work" : "richard@10gen.com", "personal" : "kreuter@gmail.com" } }
```

The purpose of dot notation is to reach the inside of nested documents looking for a specific embedded piece of information

## Bulk mode of Insert

---

```
> db.foo.bulk.insert([{"_id" : 0}, {"_id" : 1}, {"_id" : 2}])
```

```
BulkWriteResult({  
  
    "writeErrors" : [ ],  
  
    "writeConcernErrors" : [ ],  
  
    "nInserted" : 3,  
  
    "nUpserted" : 0,  
  
    "nMatched" : 0,  
  
    "nModified" : 0,  
  
    "nRemoved" : 0,  
  
    "upserted" : [ ]  
  
})  
  
> db.foo.find()  
  
{ "_id" : 1, "price" : 1.99 }
```

```
{ "_id" : 2 }
{ "_id" : 2.9 }
{ "_id" : "hello" }
{ "_id" : ISODate("2016-04-29T06:24:45.625Z") }
{ "_id" : { "a" : "x", "b" : 2 } }
{ "_id" : ObjectId("5722fef262daae10e560d1a2"), "name" : "Bob" }
{ "_id" : ObjectId("57371b949bbb29d3e0ca8229"), "a" : 1, "b" : 2 }
```

## Cursor

=====

```
cur=db.scores.find();null;
```

```
null
```

cur is a variable that holds on the cursor .cursor objects have variety of methods

.

cur.hasNext() returns true as long as there is any document .

for example the HasNext method returns true so long as theres another document to visit on the cursor .

```
> cur=db.scores.find();null;
```

```
null
```

```
> cur.size()  
3000  
  
> cur.hasNext()  
true  
  
> cur.forEach(function(d){print(d.student)})  
  
> cur.next()
```

```
while (cur.hasNext())printjson(cur.next())
```

```
> cur=db.scores.find();null;  
null  
  
> cur.limit(5) --- prints the 5 docs in the cursor
```

## Sort Operations

---

```
> db.scores.find({}, {student:true}).sort({student:1})  
  
> db.scores.find({}, {student:true}).sort({student:-1})  
  
> db.scores.find({}, {_id:true}).sort({student:1})  
  
> db.scores.find({}, {student:true}).sort({student:-1})
```

```
> db.scores.find({},{score:true,student:true}).sort({student:1})
```

## Limit and skip

---

```
> db.scores.find({},{student:true}).sort({student:1}).limit(3)
```

```
> db.scores.find({},{student:true}).sort({student:1}).skip(2).limit(3)
```

## With Cursors

---

```
=====
```

```
> cur=db.scores.find();null;
```

```
null
```

```
> cur.sort({score:-1}).limit(3);null;
```

```
null
```

```
> while (cur.hasNext())printjson(cur.next());
```

```
> cur=db.scores.find();null;
```

```
null
```

```
> cur.sort({student:-1}).limit(10).skip(2);null;
```

```
null
```

```
> cur.size()
```

```
10
```

```
> while (cur.hasNext())printjson(cur.next());
```

```
> cur=db.scores.find();null;  
null  
> cur.size()  
3000  
> cur.sort({student:-1}).limit(3).skip(4);null;  
null  
> cur.size()  
3  
> while (cur.hasNext())printjson(cur.next());
```

---

### Count Method

---

Find the number of documents that have the type “essay”

```
db.scores.count({type :"exam"})
```

---

### Update Method:

---

```
db.people.find()
```

```
{ "_id" : ObjectId("562887a16000ded4b84f279f"), "age" : 30, "name" : "Smith",  
"profession" : "hacker", "title" : "Dr" }  
  
{ "_id" : ObjectId("5628b51c6000ded4b84f335b"), "name" : "Bob", "title" : "Dr"  
}  
  
{ "_id" : ObjectId("5628b5236000ded4b84f335c"), "name" : "Charlie", "title" :  
"Dr" }  
  
{ "_id" : ObjectId("5628b52b6000ded4b84f335d"), "name" : "Dave", "title" :  
"Dr" }  
  
{ "_id" : ObjectId("5628b5346000ded4b84f335e"), "name" : "Edgar", "title" :  
"Dr" }  
  
{ "_id" : ObjectId("5628b53b6000ded4b84f335f"), "name" : "Fred", "title" : "Dr"  
}  
  
{ "_id" : ObjectId("5628b8ae6000ded4b84f3360"), "name" : 43, "title" : "Dr" }  
  
{ "_id" : ObjectId("5631ed9cfdad1bc4cff7b582"), "age" : 40, "name" : "George",  
"title" : "Dr" }  
  
{ "_id" : ObjectId("5628b5146000ded4b84f335a"), "age" : 61, "name" :  
"William", "title" : "Dr" }  
  
{ "ACAD" : "MD", "_id" : ObjectId("5628892f6000ded4b84f27a0"), "age" : 35,  
"name" : "jones", "title" : "Dr" }  
  
{ "_id" : "Smith", "age" : 30, "title" : "Dr" }  
  
{ "_id" : "James", "age" : 30, "title" : "Dr" }  
  
{ "_id" : ObjectId("5633390706a449cca998fd9b"), "name" : "Thompson", "title"  
: "Dr" }
```

```
{ "_id" : ObjectId("56d67e7671aa3d5811a0bb95"), "name" : "Smith", "age" : 30,  
"profession" : "hacker" }  
  
{ "_id" : ObjectId("56d6850371aa3d5811a0bb96"), "name" : "Jones", "age" : 35,  
"Profession" : "baker" }  
  
{ "_id" : ObjectId("56d6d9b071aa3d5811a0c74f"), "name" : "Alice" }  
  
{ "_id" : ObjectId("56d6d9dd71aa3d5811a0c750"), "name" : "charlie" }  
  
{ "_id" : ObjectId("56d6d9e571aa3d5811a0c751"), "name" : "Bob" }  
  
{ "_id" : ObjectId("56d6d9f271aa3d5811a0c752"), "name" : "Dave" }  
  
{ "_id" : ObjectId("56d6d9fa71aa3d5811a0c753"), "name" : "Edgar" }
```

Type "it" for more

Modify the name and salary of the person

=====

```
> db.people.update({name :"Smith" },{name : "Thompson",salary : 50000})  
db.people.find()
```

Using the \$set command

```
db.people.find()
```

```
db.people.update({name :"Alice"},{name :"Alice",age :30 })
```

To update the age field for the name alice ,e need to kno other fields .. This kind of update is not really useful . Instead we can go for \$set operator and update the corresponding field which we want to update ..

**Ex .Find the document whose name field is Alice and set the age to 30 . If there is no age field , one such will be created .**

```
> db.people.update({name:"Alice"},{$set : { age: 30 } })
```

```
> db.people.find({name:"Alice"})
```

```
> db.people.update({name:"Alice"},{$set : { age: 31 } })
```

```
> db.people.find({name:"Alice"})
```

**Another method called \$inc → increment is present .**

**\$set will add or modify a field in a document .**

**\$inc will modify a filed in a document**

```
db.people.update({name:"Alice" },{ $inc : { age : 1 } })
```

```
> db.people.find({name:"Alice"})
```

```
> db.people.update({name:"Bob"},{$inc : { age: 1 } })
```

```
> db.people.find({name:"Bob"})
```

**\$unset operator**

**To remove a particular field from the document , that is to remove Jone's profession we do in using the below method.**

```
> db.people.update({ name: "Jones"},{$unset : { Profession : 1 } })
```

```
> db.people.find({name:"Jones"})
```

This operator is useful for

Schema change manipulations

The application requirement changes

To model certain kinds of transformations of data.

Instead of adding or updating a null value , we can remove the field using  
\$unset operator

Rename Operator -->changes the name of the field

```
=====
```

```
db.a.save({_id:1,naem:"bob"})
```

```
db.a.update({_id:1},{$rename:{'naem':'Name'}})
```

Array based operations

```
=====
```

```
db.a.update({_id:1},{$push:{things:'one'}})
```

```
db.a.find()
```

```
{"_id":1,"things":["one"]}
```

```
db.a.update({_id:1},{$push:{things:'two'}})
```

```
db.a.update({_id:1},{$push:{things:'three'}})
```

```
db.a.find()
```

```
{"_id":1,"things":["one","two","three"]}  
  
db.a.update({_id:1},{$push:{things:'three'}})  
  
db.a.find()  
  
{"_id":1,"things":["one","two","three","three"]}
```

If we do not want three to be duplicated , we use

**\$addToSet operator**

=====

```
db.a.update({_id:1},{$addToSet:{things:'four'}})
```

**db.a.find()**

```
{"_id":1,"things":["one","two","three","three","four"]}
```

```
db.a.update({_id:1},{$addToSet:{things:'four'}})
```

**db.a.find()**

```
{"_id":1,"things":["one","two","three","three","four"]} ( it did not add any element)
```

**\$pull**

=====

If we have to remove an element from the array ,we can use \$pull operator

```
db.a.update({_id:1},{$pull:{things:'three'}})
```

```
db.a.find()
```

```
{"_id":1,"things":["one","two","four"]}
```

```
db.a.update({_id:1},{$addToSet:{things:'three'}})
```

```
db.a.find()
```

```
{"_id":1,"things":["one","two","four","three"]}
```

If we are not sure of the elements to pull out .. lets say we want to pull out the last element from the array , we use pop operator .

```
db.a.find()
```

```
{"_id":1,"things":["one","two","four","three"]}
```

```
db.a.update({_id:1},{$pop:{things:1}})
```

**db.a.find()**

```
{"_id":1,"things":["one","two","four"]}
```

**db.a.update({ \_id:1 },{\$pop:{things:-1}}) ( to remove the first element )**

**db.a.find()**

```
{"_id":1,"things":["two","four"]}
```

**all these operators work on arrays and not on strings(non arrays)**

**Multiupdate :**

=====

**db.a.find()**

```
{"_id":1,"things":[1,2,3]}
```

```
{"_id":1,"things":[2,3]}
```

```
{"_id":1,"things":[3]}
```

```
{"_id":1,"things":[1,3]}
```

**db.a.update({},{\$push:{things:4}})**

**db.a.find()**

```
{"_id":1,"things":[1,2,3,4]}
```

```
{"_id":1,"things":[2,3]}
```

```
{"_id":1,"things":[3]}
```

```
{"_id":1,"things":[1,3]}
```

**Only one document is affected**

**if multiple documents have to be affected then we give the option multi**

**db.a.update({},{\$push:{things:4}},{"multi":true})**

**db.a.find()**

```
{"_id":1,"things":[1,2,3,4,4]}
```

```
{"_id":1,"things":[2,3,4]}
```

```
{"_id":1,"things":[3,4]}
```

```
{"_id":1,"things":[1,3,4]} ( all updated)
```

**If we have to update the elements for things array which has element 2**

**then**

**db.a.update({things:2},{\$push:{things:42}},{"multi":true})**

**db.a.find()**

```
{"_id":3,"things":[3,4]}

{"_id":4,"things":[1,3,4]}

{"_id":1,"things":[1,2,3,4,4,42]} --> updated

{"_id":2,"things":[1,3,4,42]} ----> updated
```

---

## Find and modify

---

### Upserts

In Mongo Shell the update operator does four different things :

**Whole Cell replacement document**

**Manipulation of fields inside a document**

**The third one is upsert**

**It can update multiple documents**

---

```
> db.people.update({name : "srilekha"},{$set : { age : 30 }},{upsert : true})
```

```
WriteResult{
```

```
  "nMatched" : 0,  
  "nUpserted" : 1,  
  "nModified" : 0,  
  "_id" : ObjectId("573ab9f18cc7c61fb3defdf2")  
}
```

```
> db.people.find()
```

```
{ "_id" : ObjectId("573999bcb8bedb60cb42ff3f"), "name" : "akash" }  
{ "_id" : 1, "name" : "Hemanth" }  
{ "_id" : 2, "name" : "mani" }  
{ "_id" : 3, "name" : "vish", "depno" : 30 }  
{ "_id" : 5, "values" : [ 1, 2, 3 ] }  
{ "_id" : ObjectId("573ab9f18cc7c61fb3defdf2"), "name" : "srilekha", "age" : 30 }
```

```
> db.people.update({name : "george"},{$set : { age : 30 }})
```

```
WriteResult{ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 }
```

```
> db.people.find()
```

```
{ "_id" : ObjectId("573999bcb8bedb60cb42ff3f"), "name" : "akash" }  
{ "_id" : 1, "name" : "Hemanth" }
```

```
{ "_id" : 2, "name" : "mani" }

{ "_id" : 3, "name" : "vish", "depno" : 30 }

{ "_id" : 5, "values" : [ 1, 2, 3 ] }

{ "_id" : ObjectId("573ab9f18cc7c61fb3defdf2"), "name" : "srilekha", "age" : 30
}
```

```
db.people.update ({ age : {$gt : 50 }},{$set : { name : "srisuman" }},{upsert : true
})
```

The above document gets created if not already existing

Upset is used rarely . For example if you are merging data in from a data vendor and you don't if the record exists and so we straight away use the upsert operator to either update the existing document or create a new document .

To update multiple documents

```
=====
```

```
> db.people.update ({} ,{ $set : {title :"Dr"}}, {multi : true } )
```

```
WriteResult({ "nMatched" : 7, "nUpserted" : 0, "nModified" : 7 })
```

```
> db.people.find()
```

```
{ "_id" : ObjectId("573999bcb8bedb60cb42ff3f"), "name" : "akash", "title" :
"Dr" }
```

```
{ "_id" : 1, "name" : "Hemanth", "title" : "Dr" }
```

```
{ "_id" : 2, "name" : "mani", "title" : "Dr" }
```

```
{ "_id" : 3, "name" : "vish", "depno" : 30, "title" : "Dr" }
```

```
{ "_id" : 5, "values" : [ 1, 2, 3 ], "title" : "Dr" }

{ "_id" : ObjectId("573ab9f18cc7c61fb3defdf2"), "name" : "srilekha", "age" : 30,
"title" : "Dr" }

{ "_id" : ObjectId("573abaae8cc7c61fb3defdf3"), "name" : "srisuman", "title" :
"Dr" }

> db.people.update ({} , { $set : {title :"Sr"}}, {multi : true } )

> db.people.find()

{ "_id" : ObjectId("573999bcb8bedb60cb42ff3f"), "name" : "akash", "title" : "Sr"
}

{ "_id" : 1, "name" : "Hemanth", "title" : "Sr" }

{ "_id" : 2, "name" : "mani", "title" : "Sr" }

{ "_id" : 3, "name" : "vish", "depno" : 30, "title" : "Sr" }

{ "_id" : 5, "values" : [ 1, 2, 3 ], "title" : "Sr" }

{ "_id" : ObjectId("573ab9f18cc7c61fb3defdf2"), "name" : "srilekha", "age" : 30,
"title" : "Sr" }

{ "_id" : ObjectId("573abaae8cc7c61fb3defdf3"), "name" : "srisuman", "title" :
"Sr" }
```

### Remove :

Remove is a method in a collection to remove a document . It works like find which takes the argument to specify what documents to remove ..

We must pass a document to remove . If we specify an empty document then it will remove all documents in a collection one by one . If we specify one argument as given below then only that document gets removed .

```
> db.people.remove ({ name : "Alice" })  
  
db.people.remove({ name : {$gt : "M"} })  
  
> db.people.find()  
  
> db.people.remove() – no argument will remove all documents from the collection  
  
> db.people.find()
```

Delete every document with a score of less than 60 .

```
db.scores.remove({ score : { $lt : 60 } })
```

---

## Find and modify

Create an array and an object

```
=====
```

```
> db.b.insert({"_id":1,"elements":[1,2,3]})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.b.insert({"_id":2,"elements":[2,3]})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.b.insert({"_id":3,"elements":[1,3]})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.b.insert({"_id":4,"elements":[3]})  
  
WriteResult({ "nInserted" : 1 })  
  
> db.b.find()  
  
{ "_id" : 1, "elements" : [ 1, 2, 3 ] }  
  
{ "_id" : 2, "elements" : [ 2, 3 ] }  
  
{ "_id" : 3, "elements" : [ 1, 3 ] }  
  
{ "_id" : 4, "elements" : [ 3 ] }  
  
➤ > mod  
  =({"query":{"elements":1}, "update":{"$set":{"touched":true}}, "sort":{"_id":-1}})  
➤ {  
➤   "query": {  
➤     "elements" : 1  
➤   },  
➤   "update": {  
➤     "$set": {  
➤       "touched" : true  
➤     }  
➤   },  
➤   "sort": {  
➤     "_id" : -1  
➤   }  
➤ }  
➤ > db.b.findAndModify(mod)  
➤ { "_id" : 3, "elements" : [ 1, 3 ] }  
  
> db.b.find()
```

```
{ "_id" : 1, "elements" : [ 1, 2, 3 ] }  
{ "_id" : 2, "elements" : [ 2, 3 ] }  
{ "_id" : 3, "elements" : [ 1, 3 ], "touched" : true }  
{ "_id" : 4, "elements" : [ 3 ] }
```

Returns the documents after it is modified

---

```
> mod.new=true  
true  
> mod  
{  
  "query" : {  
    "elements" : 1  
  },  
  "update" : {  
    "$set" : {  
      "touched" : true  
    }  
  },  
  "sort" : {
```

```
"_id" : -1
},
"new" : true
}
> mod.new=true
true
> mod
{
  "query" : {
    "elements" : 1
  },
  "update" : {
    "$set" : {
      "touched" : true
    }
  },
  "sort" : {
    "_id" : -1
  },
  "new" : true
}
```

```
}
```

```
> mod.update.$set.touched=false
```

```
false
```

```
> mod
```

```
{
```

```
  "query" : {
```

```
    "elements" : 1
```

```
  },
```

```
  "update" : {
```

```
    "$set" : {
```

```
      "touched" : false
```

```
    }
```

```
  },
```

```
  "sort" : {
```

```
    "_id" : -1
```

```
  },
```

```
  "new" : true
```

```
}
```

```
> db.b.findAndModify(mod)
```

```
{ "_id" : 3, "elements" : [ 1, 3 ], "touched" : false }
```

The above command returns the document after modification

**Db.b.find()**

```
{ "_id" : 1, "elements" : [ 1, 2, 3 ] }
```

```
{ "_id" : 2, "elements" : [ 2, 3 ] }
```

```
{ "_id" : 3, "elements" : [ 1, 3 ], "touched" : false }
```

```
{ "_id" : 4, "elements" : [ 3 ] }
```

To count the number of elements in the array

```
=====
```

```
> db.b.find({"elements":{$size:2}})
```

```
{ "_id" : 2, "elements" : [ 2, 3 ] }
```

```
{ "_id" : 3, "elements" : [ 1, 3 ], "touched" : false }
```

```
> db.b.find({"elements":{$size:3}})
```

```
{ "_id" : 1, "elements" : [ 1, 2, 3 ] }
```

```
> db.b.find({"elements":{$size:1}})
```

```
{ "_id" : 4, "elements" : [ 3 ] }
```

## Few Cursor Operations

```
=====
```

```
> cur=db.people.find();null;
null

> cur.hasNext()

true

> cur.next()

{
    "_id" : ObjectId("573999bcb8bedb60cb42ff3f"),
    "name" : "akash",
    "title" : "Sr"
}

> cur.next()

{ "_id" : 1, "name" : "Hemanth", "title" : "Sr" }

> cur.next()

{ "_id" : 2, "name" : "mani", "title" : "Sr" }

> cur.next()

{ "_id" : 3, "name" : "vish", "depno" : 30, "title" : "Sr" }

> cur=db.people.find();null;
null

> cur.limit(3);null;

null
```

```
> while(cur.hasNext())printjson(cur.next())
{
  "_id" : ObjectId("573999bcb8bedb60cb42ff3f"),
  "name" : "akash",
  "title" : "Sr"
}
{ "_id" : 1, "name" : "Hemanth", "title" : "Sr" }
{ "_id" : 2, "name" : "mani", "title" : "Sr" }
```

## MongoDB Aggregation Framework Basics Explained

To understand the MongoDB's aggregation framework, lets start with inserting the following data.

```
db.Student.insert ({Student_Name:"Kalki",Class:"2",Mark_Scored:100,Subject:["Tamil","English","Maths"]})  
  
db.Student.insert ({Student_Name:"Matsya",Class:"1",Mark_Scored:10,Subject:["Tamil","English"]})  
  
  
db.Student.insert ({Student_Name:"Krishna",Class:"1",Mark_Scored:50,Subject:["Tamil"]})  
  
db.Student.insert ({Student_Name:"Buddha",Class:"2",Mark_Scored:60,Subject:["Tamil"]})  
db.Student.insert ({Student_Name:"Rama",Class:"2",Mark_Scored:80,Subject:["Tamil"]})  
  
  
db.Student.insert ({Student_Name:"Krishna",Class:"1",Mark_Scored:50,Subject:["English"]})  
  
db.Student.insert ({Student_Name:"Buddha",Class:"2",Mark_Scored:60,Subject:["English"]})  
  
db.Student.insert ({Student_Name:"Rama",Class:"2",Mark_Scored:80,Subject:["English"]})  
  
  
db.Student.insert ({Student_Name:"Matsya",Class:"1",Mark_Scored:67,Subject:["Maths"]})  
db.Student.insert ({Student_Name:"Krishna",Class:"1",Mark_Scored:95,Subject:["Maths"]})  
  
db.Student.insert ({Student_Name:"Buddha",Class:"2",Mark_Scored:88,Subject:["Maths"]})  
db.Student.insert ({Student_Name:"Rama",Class:"2",Mark_Scored:40,Subject:["Maths"]})
```

## Pipeline

The aggregation framework is based on pipeline concept, just like [unix\\_pipeline](#). There can be N number of operators. Output of first operator will be fed as input to the second operator. Output of second operator will be fed as input to the third operator and so on.

## Pipeline Operators

Following are the basic pipeline operators and let us make use of these operators over the sample data which we created.

### \$match

1. [\\$unwind](#)
2. [\\$group](#)
3. [\\$project](#)
4. [\\$skip](#)
5. [\\$limit](#)
6. [\\$sort](#)

### **\$match**

This is similar to MongoDB Collection's find method and SQL's WHERE clause. Basically this filters the data which is passed on to the next operator. There can be multiple \$match operators in the pipeline.

**Note:**The data what we pass to the aggregate function should be a list of Javascript objects (Python dictionaries). Each and every operator should be in a separate javascript object like shown in all the examples below.

**Example:**We want to consider only the marks of the students who study in Class "2"

```
db.Student.aggregate ([  
  {  
    "$match":  
    {  
      "Class": "2"  
    }  
  }  
])
```

and the result is

```
{  
  "result": [  
    {  
      "_id": ObjectId("517ccb98eccb9ee3d000fa5c"),  
      "Student_Name": "Kalki",  
      "Marks": 85  
    }  
  ]  
}
```

```
"Class":"2",
"Mark_Scored":100,
"Subject": [
    "Tamil",
    "English",
    "Maths"
],
{
    "_id":ObjectId("517cbb98eccb9ee3d000fa5f"),
    "Student_Name":"Buddha",
    "Class":"2",
    "Mark_Scored":60,
    "Subject": [
        "Tamil"
    ],
    {
        "_id":ObjectId("517cbb98eccb9ee3d000fa60"),
        "Student_Name":"Rama",
        "Class":"2",
        "Mark_Scored":80,
        "Subject": [
            "Tamil"
        ],
        {
            "_id":ObjectId("517cbb98eccb9ee3d000fa61"),
            "Student_Name":"Lakshmi",
            "Class":"2",
            "Mark_Scored":90,
            "Subject": [
                "Tamil"
            ],
            {
                "_id":ObjectId("517cbb98eccb9ee3d000fa62"),
                "Student_Name":"Ganesh",
                "Class":"2",
                "Mark_Scored":70,
                "Subject": [
                    "Tamil"
                ],
                {
                    "_id":ObjectId("517cbb98eccb9ee3d000fa63"),
                    "Student_Name":"Sita",
                    "Class":"2",
                    "Mark_Scored":50,
                    "Subject": [
                        "Tamil"
                    ],
                    {
                        "_id":ObjectId("517cbb98eccb9ee3d000fa64"),
                        "Student_Name":"Krishna",
                        "Class":"2",
                        "Mark_Scored":40,
                        "Subject": [
                            "Tamil"
                        ],
                        {
                            "_id":ObjectId("517cbb98eccb9ee3d000fa65"),
                            "Student_Name":"Rama",
                            "Class":"2",
                            "Mark_Scored":80,
                            "Subject": [
                                "Tamil"
                            ],
                            {
                                "_id":ObjectId("517cbb98eccb9ee3d000fa66"),
                                "Student_Name":"Lakshmi",
                                "Class":"2",
                                "Mark_Scored":90,
                                "Subject": [
                                    "Tamil"
                                ],
                                {
                                    "_id":ObjectId("517cbb98eccb9ee3d000fa67"),
                                    "Student_Name":"Ganesh",
                                    "Class":"2",
                                    "Mark_Scored":70,
                                    "Subject": [
                                        "Tamil"
                                    ],
                                    {
                                        "_id":ObjectId("517cbb98eccb9ee3d000fa68"),
                                        "Student_Name":"Sita",
                                        "Class":"2",
                                        "Mark_Scored":50,
                                        "Subject": [
                                            "Tamil"
                                        ],
                                        {
                                            "_id":ObjectId("517cbb98eccb9ee3d000fa69"),
                                            "Student_Name":"Krishna",
                                            "Class":"2",
                                            "Mark_Scored":40,
                                            "Subject": [
                                                "Tamil"
                                            ],
                                            {
                                                "_id":ObjectId("517cbb98eccb9ee3d000fa6a"),
                                                "Student_Name":"Rama",
                                                "Class":"2",
                                                "Mark_Scored":80,
                                                "Subject": [
                                                    "Tamil"
                                                ],
                                                {
                                                    "_id":ObjectId("517cbb98eccb9ee3d000fa6b"),
                                                    "Student_Name":"Lakshmi",
                                                    "Class":"2",
                                                    "Mark_Scored":90,
                                                    "Subject": [
                                                        "Tamil"
                                                    ],
                                                    {
                                                        "_id":ObjectId("517cbb98eccb9ee3d000fa6c"),
                                                        "Student_Name":"Ganesh",
                                                        "Class":"2",
                                                        "Mark_Scored":70,
                                                        "Subject": [
                                                            "Tamil"
                                                        ],
                                                        {
                                                            "_id":ObjectId("517cbb98eccb9ee3d000fa6d"),
                                                            "Student_Name":"Sita",
                                                            "Class":"2",
                                                            "Mark_Scored":50,
                                                            "Subject": [
                                                                "Tamil"
                                                            ],
                                                            {
                                                                "_id":ObjectId("517cbb98eccb9ee3d000fa6e"),
                                                                "Student_Name":"Krishna",
                                                                "Class":"2",
                                                                "Mark_Scored":40,
                                                                "Subject": [
                                                                    "Tamil"
                                                                ],
                                                                {
                                                                    "_id":ObjectId("517cbb98eccb9ee3d000fa6f"),
                                                                    "Student_Name":"Rama",
                                                                    "Class":"2",
                                                                    "Mark_Scored":80,
                                                                    "Subject": [
                                                                        "Tamil"
                                                                    ],
                                                                    {
                                                                        "_id":ObjectId("517cbb98eccb9ee3d000fa70"),
                                                                        "Student_Name":"Lakshmi",
                                                                        "Class":"2",
                                                                        "Mark_Scored":90,
                                                                        "Subject": [
                                                                            "Tamil"
                                                                        ],
                                                                        {
                                                                            "_id":ObjectId("517cbb98eccb9ee3d000fa71"),
                                                                            "Student_Name":"Ganesh",
                                                                            "Class":"2",
                                                                            "Mark_Scored":70,
                                                                            "Subject": [
                                                                                "Tamil"
                                                                            ],
                                                                            {
                                                                                "_id":ObjectId("517cbb98eccb9ee3d000fa72"),
                                                                                "Student_Name":"Sita",
                                                                                "Class":"2",
                                                                                "Mark_Scored":50,
                                                                                "Subject": [
                                                                                    "Tamil"
                                                                                ],
                                                                                {
                                                                                    "_id":ObjectId("517cbb98eccb9ee3d000fa73"),
                                                                                    "Student_Name":"Krishna",
                                                                                    "Class":"2",
                                                                                    "Mark_Scored":40,
                                                                                    "Subject": [
                                                                                        "Tamil"
                                                                                    ],
                                                                                    {
                                                                                        "_id":ObjectId("517cbb98eccb9ee3d000fa74"),
                                                                                        "Student_Name":"Rama",
                                                                                        "Class":"2",
                                                                                        "Mark_Scored":80,
                                                                                        "Subject": [
                                                                                            "Tamil"
                                                                                        ],
                                                                                        {
                                                                                            "_id":ObjectId("517cbb98eccb9ee3d000fa75"),
                                                                                            "Student_Name":"Lakshmi",
                                                                                            "Class":"2",
                                                                                            "Mark_Scored":90,
                                                                                            "Subject": [
                                                                                                "Tamil"
                                                                                            ],
                                                                                            {
                                                                                                "_id":ObjectId("517cbb98eccb9ee3d000fa76"),
                                                                                                "Student_Name":"Ganesh",
                                                                                                "Class":"2",
                                                                                                "Mark_Scored":70,
                                                                                                "Subject": [
                                                                                                    "Tamil"
                                                                                                ],
                                                                                                {
                                                                                                    "_id":ObjectId("517cbb98eccb9ee3d000fa77"),
                                                                                                    "Student_Name":"Sita",
                                                                                                    "Class":"2",
                                                                                                    "Mark_Scored":50,
                                                                                                    "Subject": [
                                                                                                        "Tamil"
                                                                                                    ],
                                                                                                    {
                                                                                                        "_id":ObjectId("517cbb98eccb9ee3d000fa78"),
................................................................
```

```
{  
    "_id":ObjectId("517cbb98eccb9ee3d000fa62"),  
    "Student_Name":"Buddha",  
    "Class":"2",  
    "Mark_Scored":60,  
    "Subject": [  
        "English"  
    ]  
},  
{  
    "_id":ObjectId("517cbb98eccb9ee3d000fa63"),  
    "Student_Name":"Rama",  
    "Class":"2",  
    "Mark_Scored":80,  
    "Subject": [  
        "English"  
    ]  
},  
{  
    "_id":ObjectId("517cbb98eccb9ee3d000fa66"),  
    "Student_Name":"Buddha",  
    "Class":"2",  
    "Mark_Scored":88,  
    "Subject": [  
        "Maths"  
    ]  
}
```

```
        ]  
  
    },  
  
    {  
  
        "_id":ObjectId("517ccb98eccb9ee3d000fa67"),  
  
        "Student_Name":"Rama",  
  
        "Class":"2",  
  
        "Mark_Scored":40,  
  
        "Subject": [  
  
            "Maths"  
  
        ]  
  
    },  
  
    "ok":1  
}
```

Let us say, we want to consider only the marks of the students who study in Class "2" and whose marks are more than or equal to 80

```
db.Student.aggregate ([  
  
{  
  
    "$match":  
  
    {  
  
        "Class":"2",  
  
        "Mark_Scored":
```

```
{  
  "$gte": 80  
}  
}  
}  
]  
)
```

Or we can use \$match operator twice to achieve the same result

```
db.Student.aggregate ([  
  {  
    "$match":  
    {  
      "Class": "2",  
    }  
  },  
  {  
    "$match":  
    {  
      "Mark_Scored":  
      {  
        "$gte": 80  
      }  
    }  
  }  
)
```

])

and the result would be

```
{  
  "result": [  
    {  
      "_id": ObjectId("517cbb98eccb9ee3d000fa5c"),  
      "Student_Name": "Kalki",  
      "Class": "2",  
      "Mark_Scored": 100,  
      "Subject": [  
        "Tamil",  
        "English",  
        "Maths"  
      ]  
    },  
    {  
      "_id": ObjectId("517cbb98eccb9ee3d000fa60"),  
      "Student_Name": "Rama",  
      "Class": "2",  
      "Mark_Scored": 80,  
      "Subject": [  
        "Tamil"  
      ]  
    }  
  ]  
}
```

```
        ]  
  
    },  
  
    {  
  
        "_id":ObjectId("517ccb98eccb9ee3d000fa63"),  
  
        "Student_Name":"Rama",  
  
        "Class":"2",  
  
        "Mark_Scored":80,  
  
        "Subject": [  
  
            "English"  
  
        ]  
  
    },  
  
    {  
  
        "_id":ObjectId("517ccb98eccb9ee3d000fa66"),  
  
        "Student_Name":"Buddha",  
  
        "Class":"2",  
  
        "Mark_Scored":88,  
  
        "Subject": [  
  
            "Maths"  
  
        ]  
  
    }  
  
],  
  
"ok":1  
}
```

## \$unwind

This will be very useful when the data is stored as list. When the unwind operator is applied on a list data field, it will generate a new record for each and every element of the list data field on which unwind is applied. It basically flattens the data. Lets see an example to understand this better

**Note:** The field name, on which unwind is applied, should be prefixed with \$ (dollar sign)

**Example:**Lets apply unwind over "Kalki"'s data.

```
db.Student.aggregate ([  
  {  
    "$match":  
      {  
        "Student_Name": "Kalki",  
      }  
  }  
])
```

This generates the following output

```
{  
  "result": [  
    {  
      "_id": ObjectId("517cbb98eccb9ee3d000fa5c"),  
      "Student_Name": "Kalki",  
    }]
```

```
"Class":"2",
"Mark_Scored":100,
"Subject":[
    "Tamil",
    "English",
    "Maths"
]
},
"ok":1
}
```

Whereas

```
db.Student.aggregate ([
{
"$match":
{
"Student_Name":"Kalki",
}
},
{
"$unwind":"$Subject"
}
])
```

1)

will generate the following output

```
{  
  "result": [  
    {  
      "_id": ObjectId("517cbb98eccb9ee3d000fa5c"),  
      "Student_Name": "Kalki",  
      "Class": "2",  
      "Mark_Scored": 100,  
      "Subject": "Tamil"  
    },  
    {  
      "_id": ObjectId("517cbb98eccb9ee3d000fa5c"),  
      "Student_Name": "Kalki",  
      "Class": "2",  
      "Mark_Scored": 100,  
      "Subject": "English"  
    },  
    {  
      "_id": ObjectId("517cbb98eccb9ee3d000fa5c"),  
      "Student_Name": "Kalki",  
      "Class": "2",  
      "Mark_Scored": 100,  
      "Subject": "Maths"  
    }  
  ]  
}
```

```
        "Mark_Scored":100,  
        "Subject":"Maths"  
    },  
],  
"ok":1  
}
```

## \$group

Now that we have flatten the data to be processed, lets try and group the data to process them. The group pipeline operator is similar to the SQL's GROUP BY clause. In SQL, we can't use GROUP BY unless we use any of the aggregation functions. The same way, we have to use an aggregation function in MongoDB as well. You can read more about the aggregation functions [here](#). As most of them are like in SQL, I don't think much explanation would be needed.

**Note:** The \_id element in group operator is a must. We cannot change it to some other name.

MongoDB identifies the grouping expression with the \_id field only.

**Example:** Lets try and get the sum of all the marks scored by each and every student, in Class "2"

```
db.Student.aggregate ([  
{  
    "$match":  
    {
```

```
"Class":"2"
}
},
{
"$unwind":"$Subject"
},
{
"$group":
{
"_id":
{
"Student_Name":"$Student_Name"
},
"Total_Marks":
{
"$sum":"$Mark_Scored"
}
}
}
]
)
```

If we look at this aggregation example, we have specified an \_id element and Total\_Marks element.

The \_id element tells MongoDB to group the documents based on Student\_Name field. The

Total\_Marks uses an aggregation function **\$sum**, which basically adds up all the marks and returns

the sum. This will produce this Output

```
{  
  "result": [  
    {  
      "_id": {  
        "Student_Name": "Rama"  
      },  
      "Total_Marks": 200  
    },  
    {  
      "_id": {  
        "Student_Name": "Buddha"  
      },  
      "Total_Marks": 208  
    },  
    {  
      "_id": {  
        "Student_Name": "Kalki"  
      },  
      "Total_Marks": 300  
    }  
  ],  
  "ok": 1  
}
```

We can use the sum function to count the number of records match the grouped data. Instead of "\$sum": "\$Mark\_Scored", "\$sum": 1 will count the number of records. "\$sum": 2 will add 2 for each and every grouped data.

```
db.Student.aggregate ([  
  {  
    "$match":  
      {  
        "Class": "2"  
      }  
  },  
  {  
    "$unwind": "$Subject"  
  },  
  {  
    "$group":  
      {  
        "_id":  
          {  
            "Student_Name": "$Student_Name"  
          },  
        "Total_Marks":  
          {  
            "$sum": 1  
          }  
      }  
  }])
```

```
}
```

```
}
```

```
}
```

```
])
```

This will produce this Output

```
{
```

```
    "result": [
```

```
        {
```

```
            "_id": {
```

```
                "Student_Name": "Rama"
```

```
            },
```

```
            "Total_Marks": 3
```

```
        },
```

```
        {
```

```
            "_id": {
```

```
                "Student_Name": "Buddha"
```

```
            },
```

```
            "Total_Marks": 3
```

```
        },
```

```
        {
```

```
            "_id": {
```

```
                "Student_Name": "Kalki"
```

```
            },
```

```
        "Total_Marks":3
    }
],
"ok":1
}
```

This is because each and every student has marks for three subjects.

## \$project

The project operator is similar to SELECT in SQL. We can use this to rename the field names and select/deselect the fields to be returned, out of the grouped fields. If we specify 0 for a field, it will NOT be sent in the pipeline to the next operator. We can even flatten the data using project as shown in the example below

### Example:

```
db.Student.aggregate ([
{
"$match":
{
"Class":"2"
}
},
{
}
```

```
"$unwind": "$Subject"
},
{
"$group":
{
"_id":
{
"Student_Name": "$Student_Name"
},
"Total_Marks":
{
"$sum": "$Mark_Scored"
}
}
},
{
"$project":
{
"_id": 0,
"Name": "$_id.Student_Name",
"Total": "$Total_Marks"
}
}
])
)
```

will result in

```
{  
  "result": [  
    {  
      "Name": "Rama",  
      "Total": 200  
    },  
    {  
      "Name": "Buddha",  
      "Total": 208  
    },  
    {  
      "Name": "Kalki",  
      "Total": 300  
    }  
  "ok": 1  
}
```

Lets say we try to retrieve Subject field by specifying project like shown below. MongoDB will simply ignore the Subject field, since it is not used in the group operator's \_id field.

```
  "$project":
```

```
{  
  "_id":0,  
  "Subject":1,  
  "Name":"$_id.Student_Name",  
  "Total":"$Total_Marks"  
}
```

## \$sort

This is similar to SQL's ORDER BY clause. To sort a particular field in descending order specify -1 and specify 1 if that field has to be sorted in ascending order. I don't think this section needs more explanation. Lets straight away look at an example

### Example:

```
db.Student.aggregate ([  
  {  
    "$match":  
    {  
      "Class": "2"  
    }  
  },  
  {  
    "$unwind": "$Subject"  
  },  
  {
```

```
 "$group":  
 {  
   "_id":  
   {  
     "Student_Name": "$Student_Name"  
   },  
   "Total_Marks":  
   {  
     "$sum": "$Mark_Scored"  
   }  
 },  
 {  
   "$project":  
   {  
     "_id": 0,  
     "Name": "$_id.Student_Name",  
     "Total": "$Total_Marks"  
   }  
 },  
 {  
   "$sort":  
   {  
     "Total": -1,  
     "Name": 1
```

```
}
```

```
}
```

```
])
```

Will Sort the data based on Marks in descending Order first and then by Name in Ascending Order.

```
{
```

```
    "result": [
        {
            "Name": "Kalki",
            "Total": 300
        },
        {
            "Name": "Buddha",
            "Total": 208
        },
        {
            "Name": "Rama",
            "Total": 200
        }
    ],
    "ok": 1
}
```

## \$limit and \$skip

These two operators can be used to limit the number of documents being returned. They will be more useful when we need pagination support.

**Example:**

```
db.Student.aggregate ([  
  {  
    "$match":  
      {  
        "Class": "2"  
      }  
  },  
  {  
    "$unwind": "$Subject"  
  },  
  {  
    "$group":  
      {  
        "_id":  
          {  
            "Student_Name": "$Student_Name"  
          },  
        "Total_Marks":  
          {  
            "$sum": "$Mark_Scored"
```

```
}

}

} , 

{

"$project": 

{ 

"_id":0, 

"Name":"$_id.Student_Name", 

"Total":"$Total_Marks"

}

} , 

{

"$sort": 

{ 

"Total": -1, 

"Name":1

}

} , 

{

"$limit":2, 

}

{

"$skip":1, 

}

]

)
```

will result in

```
{"result": [{"Name": "Buddha", "Total": 208}], "ok": 1}
```

Because the limit operator receives 3 documents from the sort operator and allows only the first two documents to pass through it, thereby dropping Rama's record. The skip operator skips one document (that means the first document (Kalki's document) is dropped) and allows only the Buddha's document to pass through.

All the examples shown here are readily usable with pyMongo (Just replace db.Student with your Collection object name)

---

### Aggregation Framework Example

---

1. Import the json documents into a database called test and create a collection by name zips

```
mongoimport --db test --collection zips --file D:\zips.json
```

Each document in this collection has the following form:

```
{  
  "_id" : "35004",  
  "city" : "Acmar",  
  "state" : "AL",  
  "pop" : 6055,  
  "loc" : [-86.51557, 33.584132]  
}
```

- The \_id field holds the zipcode as a string.
- The city field holds the city name.
- The state field holds the two letter state abbreviation.
- The pop field holds the population.
- The loc field holds the location as a [latitude, longitude] array.

2. Find the states with Population over 10 Million

```
db.zips.aggregate([  
  {"$group" => {"_id": "$state", total_pop: {"$sum" => "$pop"}},  
  {"$match" => {"total_pop: {"$gte" => 10_000_000}}}  
])
```

3. Find the average city population by state

```
db.zips.aggregate([
  {"$group" => {_id: {state: "$state", city: "$city"}, pop: {"$sum" => "$pop"}}},
  {"$group" => {_id: "$_id.state", avg_city_pop: {"$avg" => "$pop"}}},
  {"$sort" => {avg_city_pop: -1}},
  {"$limit" => 3}
])
```

#### 4. Find the largest and smalles cities by state

```
db.zips.aggregate([
  {"$group" => {_id: {state: "$state", city: "$city"}, pop: {"$sum" => "$pop"}}},
  {"$sort" => {pop: 1}},
  {"$group" => {
    _id: "$_id.state",
    smallest_city: {"$first" => "$_id.city"},
    smallest_pop: {"$first" => "$pop"},
    biggest_city: { "$last" => "$_id.city"},
    biggest_pop: { "$last" => "$pop"}
  }}
])
```

## Creating Indexes

**See below the steps to configure the WiredTiger Storage Engine**

---

C:\Users\sr senthi>mkdir WT

A subdirectory or file WT already exists.

C:\Users\sr senthi>mongod --dbpath WT --storageEngine  
wiredTiger

---

### Index Creation Examples

---

1.

```
> for (i=0;i<1000000;i++) { db.users.insert({ "i" : i, "username" : "user" + i, "age":Math.floor(Math.random()*120), "created" : new Date()});}
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.users.find().count()
```

```
1000000
```

```
> db.users.find({username:"user01"}).explain()
```

```
{
```

```
    "queryPlanner" : {
```

```
"plannerVersion" : 1,  
"namespace" : "test.users",  
"indexFilterSet" : false,  
"parsedQuery" : {  
    "username" : {  
        "$eq" : "user01"  
    }  
},  
"winningPlan" : {  
    "stage" : "COLLSCAN",  
    "filter" : {  
        "username" : {  
            "$eq" : "user01"  
        }  
    },  
    "direction" : "forward"  
},  
"rejectedPlans" : [ ]  
},  
"serverInfo" : {
```

```
"host" : "LIN73000604",
"port" : 27017,
"version" : "3.2.4",
"gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}

3. > db.users.find({username: "user101"}).limit(1).explain()

{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "test.users",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "username" : {
        "$eq" : "user101"
      }
    },
    "winningPlan" : {
      "stage" : "LIMIT",
    }
  }
}
```

```
"limitAmount" : 1,  
  
"inputStage" : {  
    "stage" : "COLLSCAN",  
    "filter" : {  
        "username" : {  
            "$eq" : "user101"  
        }  
    },  
    "direction" : "forward"  
},  
},  
"rejectedPlans" : [ ]  
},  
  
"serverInfo" : {  
    "host" : "LIN73000604",  
    "port" : 27017,  
    "version" : "3.2.4",  
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"  
},  
"ok" : 1
```

```
}
```

```
4. > db.users.ensureIndex({"username" : 1}) ( creating Single key index )
```

```
{
```

```
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 1,  
    "numIndexesAfter" : 2,  
    "ok" : 1
```

```
}
```

Test for Index usage :

```
=====
```

```
5. > db.users.find({"username" : "user101"}).explain()
```

```
{
```

```
    "queryPlanner" : {  
        "plannerVersion" : 1,  
        "namespace" : "test.users",  
        "indexFilterSet" : false,  
        "parsedQuery" : {  
            "username" : {  
                "$eq" : "user101"  
            }  
        }  
    }
```

```
},  
"winningPlan" : {  
    "stage" : "FETCH",  
    "inputStage" : {  
        "stage" : "IXSCAN",  
        "keyPattern" : {  
            "username" : 1  
        },  
        "indexName" : "username_1",  
        "isMultiKey" : false,  
        "isUnique" : false,  
        "isSparse" : false,  
        "isPartial" : false,  
        "indexVersion" : 1,  
        "direction" : "forward",  
        "indexBounds" : {  
            "username" : [  
                "[\"user101\", \"user101\"]"  
            ]  
        }  
    }
```

```
    },  
    },  
    "rejectedPlans" : [ ]  
,  
    "serverInfo" : {  
        "host" : "LIN73000604",  
        "port" : 27017,  
        "version" : "3.2.4",  
        "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"  
,  
        "ok" : 1  
    }  

```

## 6. Sorting based on age and username

---

```
> db.users.find().sort({ "age" : 1, "username" : 1 })  
  
Error: error: {  
    "waitedMS" : NumberLong(0),  
    "ok" : 0,
```

"errmsg" : "Executor error during find command: OperationFailed: Sort operation used more than the maximum 33554432 bytes of RAM. Add an index, or specify a smaller limit."

,

    "code" : 96

}

To optimize the sort ,we could make an index on username and age called as compound index

```
> db.users.ensureIndex({"age" : 1, "username" : 1})
```

{

    "createdCollectionAutomatically" : false,

    "numIndexesBefore" : 2,

    "numIndexesAfter" : 3,

    "ok" : 1

}

---

This is called a **compound index** and is useful if your query has multiple sort directions or multiple keys in the criteria. A compound index is an index on more than one field

Each index entry contains an age and a username and points to the location of a document on disk (represented by the hexadecimal numbers, which can be ignored)

```
> db.users.find({"age" : 21}).sort({"username" : -1})  
  
{ "_id" : ObjectId("5736a5849bbb29d3e0ca8207"), "i" : 999978, "username" :  
"user999978", "age" : 21, "created" : ISODate("2016-05-14T04:11:48.550Z") }  
  
{ "_id" : ObjectId("5736a5849bbb29d3e0ca81d0"), "i" : 999923, "username" :  
"user999923", "age" : 21, "created" : ISODate("2016-05-14T04:11:48.436Z") }  
  
{ "_id" : ObjectId("5736a5849bbb29d3e0ca8137"), "i" : 999770, "username" :  
"user999770", "age" : 21, "created" : ISODate("2016-05-14T04:11:48.112Z") }  
  
{ "_id" : ObjectId("5736a5839bbb29d3e0ca80f2"), "i" : 999701, "username" :  
"user999701", "age" : 21, "created" : ISODate("2016-05-14T04:11:47.968Z") }  
  
{ "_id" : ObjectId("5736a5839bbb29d3e0ca8026"), "i" : 999497, "username" :  
"user999497", "age" : 21, "created" : ISODate("2016-05-14T04:11:47.539Z") }  
  
{ "_id" : ObjectId("5736a5839bbb29d3e0ca7f28"), "i" : 999243, "username" :  
"user999243", "age" : 21, "created" : ISODate("2016-05-14T04:11:47.001Z") }  
  
{ "_id" : ObjectId("5736a5829bbb29d3e0ca7ec1"), "i" : 999140, "username" :  
"user999140", "age" : 21, "created" : ISODate("2016-05-14T04:11:46.784Z") }  
  
{ "_id" : ObjectId("5736a5829bbb29d3e0ca7eb8"), "i" : 999131, "username" :  
"user999131", "age" : 21, "created" : ISODate("2016-05-14T04:11:46.758Z") }  
  
{ "_id" : ObjectId("5736a5829bbb29d3e0ca7ea0"), "i" : 999107, "username" :  
"user999107", "age" : 21, "created" : ISODate("2016-05-14T04:11:46.697Z") }  
  
{ "_id" : ObjectId("5736a5829bbb29d3e0ca7da8"), "i" : 998859, "username" :  
"user998859", "age" : 21, "created" : ISODate("2016-05-14T04:11:46.164Z") }  
  
{ "_id" : ObjectId("5736a5829bbb29d3e0ca7da0"), "i" : 998851, "username" :  
"user998851", "age" : 21, "created" : ISODate("2016-05-14T04:11:46.147Z") }
```

```
{ "_id" : ObjectId("5736a5809bbb29d3e0ca7b36"), "i" : 998233, "username" :  
"user998233", "age" : 21, "created" : ISODate("2016-05-14T04:11:44.840Z") }  
  
{ "_id" : ObjectId("5736a5809bbb29d3e0ca7a83"), "i" : 998054, "username" :  
"user998054", "age" : 21, "created" : ISODate("2016-05-14T04:11:44.467Z") }  
  
{ "_id" : ObjectId("5736a5809bbb29d3e0ca7a65"), "i" : 998024, "username" :  
"user998024", "age" : 21, "created" : ISODate("2016-05-14T04:11:44.407Z") }  
  
{ "_id" : ObjectId("5736a5809bbb29d3e0ca7a4c"), "i" : 997999, "username" :  
"user997999", "age" : 21, "created" : ISODate("2016-05-14T04:11:44.354Z") }  
  
{ "_id" : ObjectId("5736a5809bbb29d3e0ca7a2d"), "i" : 997968, "username" :  
"user997968", "age" : 21, "created" : ISODate("2016-05-14T04:11:44.285Z") }  
  
{ "_id" : ObjectId("5736a57e9bbb29d3e0ca7752"), "i" : 997237, "username" :  
"user997237", "age" : 21, "created" : ISODate("2016-05-14T04:11:42.719Z") }  
  
{ "_id" : ObjectId("57369d8d9bbb29d3e0bcc54e"), "i" : 99697, "username" :  
"user99697", "age" : 21, "created" : ISODate("2016-05-14T03:37:49.482Z") }  
  
{ "_id" : ObjectId("5736a57d9bbb29d3e0ca7608"), "i" : 996907, "username" :  
"user996907", "age" : 21, "created" : ISODate("2016-05-14T04:11:41.938Z") }  
  
{ "_id" : ObjectId("5736a57c9bbb29d3e0ca73cb"), "i" : 996334, "username" :  
"user996334", "age" : 21, "created" : ISODate("2016-05-14T04:11:40.684Z") }
```

**MongoDB can jump directly to the correct age  
and doesn't need to sort the results as traversing the index returns the data in  
the  
correct order.**

```
db.users.find({"age" : {"$gte" : 21, "$lte" : 30}})
```

This is a multi-value query, which looks for documents matching multiple values (in this case, all ages between 21 and 30).

```
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc68a"), "i" : 100013, "username" :  
"user100013", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.292Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc691"), "i" : 100020, "username" :  
"user100020", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.307Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6bc"), "i" : 100063, "username" :  
"user100063", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.414Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc744"), "i" : 100199, "username" :  
"user100199", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.706Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc79f"), "i" : 100290, "username" :  
"user100290", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.912Z") }  
  
{ "_id" : ObjectId("57369d8f9bbb29d3e0bcc816"), "i" : 100409, "username" :  
"user100409", "age" : 21, "created" : ISODate("2016-05-14T03:37:51.181Z") }  
  
{ "_id" : ObjectId("57369d8f9bbb29d3e0bcc8a2"), "i" : 100549, "username" :  
"user100549", "age" : 21, "created" : ISODate("2016-05-14T03:37:51.474Z") }  
  
{ "_id" : ObjectId("57369cc69bbb29d3e0bb6748"), "i" : 10091, "username" :  
"user10091", "age" : 21, "created" : ISODate("2016-05-14T03:34:30.909Z") }  
  
{ "_id" : ObjectId("57369d909bbb29d3e0bcc5b"), "i" : 100990, "username" :  
"user100990", "age" : 21, "created" : ISODate("2016-05-14T03:37:52.493Z") }  
  
{ "_id" : ObjectId("57369d909bbb29d3e0bcc86"), "i" : 101033, "username" :  
"user101033", "age" : 21, "created" : ISODate("2016-05-14T03:37:52.592Z") }
```

```
{ "_id" : ObjectId("57369d909bbb29d3e0bccaf9"), "i" : 101148, "username" :  
"user101148", "age" : 21, "created" : ISODate("2016-05-14T03:37:52.868Z") }  
  
{ "_id" : ObjectId("57369d919bbb29d3e0bccbeb"), "i" : 101390, "username" :  
"user101390", "age" : 21, "created" : ISODate("2016-05-14T03:37:53.430Z") }  
  
{ "_id" : ObjectId("57369d919bbb29d3e0bcc37"), "i" : 101466, "username" :  
"user101466", "age" : 21, "created" : ISODate("2016-05-14T03:37:53.593Z") }  
  
{ "_id" : ObjectId("57369cc79bbb29d3e0bb6793"), "i" : 10166, "username" :  
"user10166", "age" : 21, "created" : ISODate("2016-05-14T03:34:31.067Z") }  
  
{ "_id" : ObjectId("57369d929bbb29d3e0bccdc0"), "i" : 101859, "username" :  
"user101859", "age" : 21, "created" : ISODate("2016-05-14T03:37:54.489Z") }  
  
{ "_id" : ObjectId("57369d929bbb29d3e0bccdd1"), "i" : 101876, "username" :  
"user101876", "age" : 21, "created" : ISODate("2016-05-14T03:37:54.535Z") }  
  
{ "_id" : ObjectId("57369d929bbb29d3e0bccdf2"), "i" : 101909, "username" :  
"user101909", "age" : 21, "created" : ISODate("2016-05-14T03:37:54.604Z") }  
  
{ "_id" : ObjectId("57369d929bbb29d3e0bcce17"), "i" : 101946, "username" :  
"user101946", "age" : 21, "created" : ISODate("2016-05-14T03:37:54.680Z") }  
  
{ "_id" : ObjectId("57369d929bbb29d3e0bcce62"), "i" : 102021, "username" :  
"user102021", "age" : 21, "created" : ISODate("2016-05-14T03:37:54.839Z") }  
  
{ "_id" : ObjectId("57369d929bbb29d3e0bcce75"), "i" : 102040, "username" :  
"user102040", "age" : 21, "created" : ISODate("2016-05-14T03:37:54.879Z") }  
  
Type "it" for more  
  
> db.users.find({ "age" : { "$gte" : 21, "$lte" : 30 }}).sort({ "username" :  
... 1})
```

```
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc68a"), "i" : 100013, "username" :  
"user100013", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.292Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc691"), "i" : 100020, "username" :  
"user100020", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.307Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6a9"), "i" : 100044, "username" :  
"user100044", "age" : 26, "created" : ISODate("2016-05-14T03:37:50.365Z") }  
  
{ "_id" : ObjectId("57369cc69bbb29d3e0bcc6f3"), "i" : 10006, "username" :  
"user10006", "age" : 26, "created" : ISODate("2016-05-14T03:34:30.728Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6bc"), "i" : 100063, "username" :  
"user100063", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.414Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6ce"), "i" : 100081, "username" :  
"user100081", "age" : 22, "created" : ISODate("2016-05-14T03:37:50.454Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6d1"), "i" : 100084, "username" :  
"user100084", "age" : 22, "created" : ISODate("2016-05-14T03:37:50.461Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6da"), "i" : 100093, "username" :  
"user100093", "age" : 28, "created" : ISODate("2016-05-14T03:37:50.479Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6db"), "i" : 100094, "username" :  
"user100094", "age" : 28, "created" : ISODate("2016-05-14T03:37:50.481Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6df"), "i" : 100098, "username" :  
"user100098", "age" : 23, "created" : ISODate("2016-05-14T03:37:50.490Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6e0"), "i" : 100099, "username" :  
"user100099", "age" : 23, "created" : ISODate("2016-05-14T03:37:50.492Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6e2"), "i" : 100101, "username" :  
"user100101", "age" : 22, "created" : ISODate("2016-05-14T03:37:50.497Z") }
```

```
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc6e6"), "i" : 100105, "username" :  
"user100105", "age" : 26, "created" : ISODate("2016-05-14T03:37:50.506Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc719"), "i" : 100156, "username" :  
"user100156", "age" : 23, "created" : ISODate("2016-05-14T03:37:50.611Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc71b"), "i" : 100158, "username" :  
"user100158", "age" : 30, "created" : ISODate("2016-05-14T03:37:50.615Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc73d"), "i" : 100192, "username" :  
"user100192", "age" : 25, "created" : ISODate("2016-05-14T03:37:50.687Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc744"), "i" : 100199, "username" :  
"user100199", "age" : 21, "created" : ISODate("2016-05-14T03:37:50.706Z") }  
  
{ "_id" : ObjectId("57369cc69bbb29d3e0bb6701"), "i" : 10020, "username" :  
"user10020", "age" : 29, "created" : ISODate("2016-05-14T03:34:30.758Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc74b"), "i" : 100206, "username" :  
"user100206", "age" : 27, "created" : ISODate("2016-05-14T03:37:50.722Z") }  
  
{ "_id" : ObjectId("57369d8e9bbb29d3e0bcc74c"), "i" : 100207, "username" :  
"user100207", "age" : 29, "created" : ISODate("2016-05-14T03:37:50.723Z") }
```

```
db.users.find({{"age" : {"$gte" : 21, "$lte" : 30}}).sort({"username" :  
1})
```

However, the index doesn't return the usernames in sorted order and the query requested that the results be sorted by username, so MongoDB has to sort the results

in memory before returning them. Thus, this query is usually less efficient than the queries above.

One other index you can use in the last example is the same keys in reverse order:

`{"username" : 1, "age" : 1}`. MongoDB will then traverse all the index entries, but in the order you want them back in. It will pick out the matching documents using the "age" part of the index:

This is good in that it does not require any giant in-memory sorts. However, it does

have to scan the entire index to find all matches. Thus, putting the sort key first is

generally a good strategy when you're using a limit so MongoDB can stop scanning the index after a couple of matches.

```
> db.users.find({"age" : {"$gte" : 21, "$lte" : 30}}).sort({"username" : 1}).explain()  
{  
  "queryPlanner" : {  
    "plannerVersion" : 1,  
    "namespace" : "test.users",  
    "indexFilterSet" : false,
```

```
"parsedQuery" : {  
    "$and" : [  
        {  
            "age" : {  
                "$lte" : 30  
            }  
        },  
        {  
            "age" : {  
                "$gte" : 21  
            }  
        }  
    ]  
    "winningPlan" : {  
        "stage" : "FETCH",  
        "filter" : {  
            "$and" : [  
                {  
                    "age" : {  
                        "$gt" : 21  
                    }  
                }  
            ]  
        }  
    }  
}
```

```
    "$lte" : 30
  }
},
{
  "age" : {
    "$gte" : 21
  }
}
],
{
  "inputStage" : {
    "stage" : "IXSCAN",
    "keyPattern" : {
      "username" : 1
    },
    "indexName" : "username_1",
    "isMultiKey" : false,
    "isUnique" : false,
    "isSparse" : false,
    "isPartial" : false,
  }
}
```

```
"indexVersion" : 1,  
"direction" : "forward",  
"indexBounds" : {  
    "username" : [  
        "[MinKey, MaxKey]"  
    ]  
}  
}  
},  
"rejectedPlans" : [  
    {  
        "stage" : "SORT",  
        "sortPattern" : {  
            "username" : 1  
        },  
        "inputStage" : {  
            "stage" : "SORT_KEY_GENERATOR",  
            "inputStage" : {  
                "stage" : "FETCH",  
                "inputStage" : {  
                    "stage" : "COLLSCAN"  
                }  
            }  
        }  
    }  
]
```

```
"stage" : "IXSCAN",
"keyPattern" : {
    "age" : 1,
    "username" : 1
},
"indexName" : "age_1_username_1",
"isMultiKey" : false,
"isUnique" : false,
"isSparse" : false,
"isPartial" : false,
"indexVersion" : 1,
"direction" : "forward",
"indexBounds" : {
    "age" : [
        "[21.0, 30.0]"
    ],
    "username" : [
        "[MinKey, MaxKey]"
    ]
}
```

```
        }
    }
}
]
},
"serverInfo" : {
    "host" : "LIN73000604",
    "port" : 27017,
    "version" : "3.2.4",
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}

> db.users.dropIndex({"age" : 1, "username" : 1})
{
    "nIndexesWas" : 3,
    "ok" : 1
}

> db.users.ensureIndex({"username" : 1, "age" : 1}) ( Compound index created
with the sort key as the prefix)

This avoids memory level sorts ... but scans the entire index documents
```

```
{  
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 2,  
    "numIndexesAfter" : 3,  
    "ok" : 1  
}  
  
> db.users.find({"age" : {"$gte" : 21, "$lte" : 30}}).sort({"username" : 1}).explain()  
  
{  
    "queryPlanner" : {  
        "plannerVersion" : 1,  
        "namespace" : "test.users",  
        "indexFilterSet" : false,  
        "parsedQuery" : {  
            "$and" : [  
                {  
                    "age" : {  
                        "$lte" : 30  
                    }  
                },  
                {  
                    "username" : {  
                        "$gt" : "abc"  
                    }  
                }  
            ]  
        }  
    }  
}
```



```
        "age" : {  
            "$gte" : 21  
        }  
    }  
}  
},  
"winningPlan" : {  
    "stage" : "FETCH",  
    "filter" : {  
        "$and" : [  
            {  
                "age" : {  
                    "$lte" : 30  
                }  
            },  
            {  
                "age" : {  
                    "$gte" : 21  
                }  
            }  
        ]  
    }  
},  
"cursor" : {}  
}
```

```
        }  
    ]  
},  
"inputStage": {  
    "stage": "IXSCAN",  
    "keyPattern": {  
        "username": 1  
    },  
    "indexName": "username_1",  
    "isMultiKey": false,  
    "isUnique": false,  
    "isSparse": false,  
    "isPartial": false,  
    "indexVersion": 1,  
    "direction": "forward",  
    "indexBounds": {  
        "username": [  
            "[MinKey, MaxKey]"  
        ]  
    }  
}
```

```
    },  
    },  
    "rejectedPlans" : [ ]  
,  
    "serverInfo" : {  
        "host" : "LIN73000604",  
        "port" : 27017,  
        "version" : "3.2.4",  
        "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"  
,  
        "ok" : 1  
    }  

```

**Note :** The index pattern of {"sortKey" : 1, "queryCriteria" : 1} often works well in applications, as most application do not want all possible results for a query but only the first few.

**Putting the sort key first is generally a good strategy when you're using a limit so MongoDB can stop scanning the index after a couple of matches.**

Indexes are basically trees, with the smallest value on the leftmost leaf and the greatest on the rightmost. If you have a "sortKey" that is a date (or any value that increases over time) then as you traverse the tree from left to right, you're basically travelling forward in

time. Thus, for applications that tend to use recent data more than older data, MongoDB only has to keep the rightmost (most recent) branches of the tree in memory, not the whole thing. An index like this is called right-balanced and, whenever possible, you should make your indexes right-balanced. The "\_id" index is an example of a rightbalanced index.

To optimize compound sorts in different directions, use an index with matching directions.

In this example, we could use {"age" : 1, "username" : -1}

If our application also needed to optimize sorting by {"age" : 1, "username" : 1}, we would have to create a second index with those directions. To figure out which

directions to use for an index, simply match the directions your sort is using.

Note that

inverse indexes (multiplying each direction by -1) are equivalent: {"age" : 1, "user

name" : -1} suits the same queries that {"age" : -1, "username" : 1} does.

Index direction only really matters when you're sorting based on multiple criteria. If

you're only sorting by a single key, MongoDB can just as easily read the index in the

opposite order. For example, if you had a sort on {"age" : -1} and an index on {"age" : 1}, MongoDB could optimize it just as well as if you had an index on {"age" :

-1} (so don't create both!). The direction only matters for multikey sorts.

## Types of Indexes

---

### Unique Indexes :

```
> db.users.dropIndexes() ( to drop the existing indexes )
{
```

```
"nIndexesWas" : 2,
"msg" : "non-_id indexes dropped for collection",
"ok" : 1
}
> db.users.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "test.users"
  }
]
> db.users.ensureIndex({"username" : 1}, {"unique" : true})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.users.insert({username: "bob"})
WriteResult({ "nInserted" : 1 })
> db.users.insert({username: "bob"})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: test.users index:
username_1 dup key: { : \"bob\" }"
  }
})
```

If a key does not exist, the index stores its value as null for that document. This means that if you create a unique index and try to insert more than one document that is missing the indexed field, the inserts will fail because you already have a document with a value of null

```
> db.users.insert({age:23})
WriteResult({ "nInserted" : 1 })
> db.users.insert({"age":44})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: test.users index:
username_1 dup key: { : null }"
  }
})
```

**Note:**

All fields must be smaller than 1024 bytes to be included in an index. MongoDB does not return any sort of error or warning if a document's fields cannot be indexed due to size. This means that keys longer than 8 KB will not be subject to the unique index constraints: you can insert identical 8 KB strings,

---

**Compound Unique Indexes :**

---

```
> db.users.ensureIndex({"username" : 1, "age" : 1}, {"unique" : true})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
```

```
}

> db.users.insert({"username" : "bob"})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "E11000 duplicate key error collection: test.users index:
username_1_age_1 dup key: { : \"bob\\\", : null }"
    }
})
>> db.users.insert({"username" : "bob", "age" : 23})
2016-05-14T13:56:24.904+0530 E QUERY  [thread1] SyntaxError: expected
expression, got '>' @(shell):1:0

>> db.users.insert({"username" : "fred", "age" : 23})
2016-05-14T13:56:37.909+0530 E QUERY  [thread1] SyntaxError: expected
expression, got '>' @(shell):1:0

> db.users.insert({"username" : "fred", "age" : 23})
WriteResult({ "nInserted" : 1 })
> db.users.insert({"username" : "fred", "age" : 23})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "E11000 duplicate key error collection: test.users index:
username_1_age_1 dup key: { : \"fred\\\", : 23.0 }"
    }
})
```

## Dropping Duplicates.

---

### To delete documents with duplicate values

"**dropDups**" option will save the first document found and remove any subsequent

documents with duplicate values:

```
> db.people.ensureIndex({"username" : 1}, {"unique" : true, "dropDups" : true})
```

you have no control over which documents are dropped and which are kept  
(and MongoDB gives

you no indication of which documents were dropped, if any). If your data is of  
any

importance, do not use "dropDups".

```
> db.users.insert({"salary":1300,"eno":102})
WriteResult({ "nInserted" : 1 })
> db.users.insert({"salary":1400,"eno":104})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "E11000 duplicate key error collection: test.users index:
username_1_age_1 dup key: { : null, : null }"
    }
})
> db.users.dropIndexes()
{
    "nIndexesWas" : 2,
    "msg" : "non-_id indexes dropped for collection",
    "ok" : 1
}
```

In the above case , not more than one document can be entered with the index key as null. To avoid this we go for Sparse Indexes

### Creating Sparse Indexes

unique indexes count null as a value, so you cannot have a unique index with more than one document missing the key. However, there are lots of cases where you may want the unique index to be enforced only if the key exists. If you have a field that may or may not exist but must be unique when it does, you can combine the unique option with the sparse option.

```
> db.users.ensureIndex({"username" : 1, "age" : 1}, {"unique" : true,"sparse":true})  
{  
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 1,  
    "numIndexesAfter" : 2,  
    "ok" : 1  
}  
> db.users.insert({ "salary":1400,"eno":104})  
WriteResult({ "nInserted" : 1 })  
> db.users.insert({ "salary":1300,"eno":103})  
WriteResult({ "nInserted" : 1 })
```

Sparse indexes do not necessarily have to be unique. To make a non-unique sparse index,  
simply do not include the unique option

### Multikey indexes

**db.foo.insert({a:1,b:2}) --- a document been created**

```
> db.foo.find()  
{ "_id" : ObjectId("56f0d67ee8436b9274fa4e80"), "a" : 1, "b" : 2 }  
➤ Lets say we have to create index on a,b in the ascending  
order  
➤ Db.foo.createIndex({a:1,b:1})  
➤ To get more info on the usage of index ,we can go for explain  
method
```

**Db.foo.explain().find({a:1,b:1})** in the query planner it tells about the winning plan and in the winning plan it says about the index scan it says that it is not a multikey index

Now in the next example lets create a document as the case given below

```
Db.foo.insert({a:3,b:[3,5,7]})  
  
> db.multi.insert({a:1,b:2})  
WriteResult({ "nInserted" : 1 })  
> db.multi.find()  
{ "_id" : ObjectId("57371bbe9bbb29d3e0ca822a"), "a" : 1, "b" : 2 }  
> db.multi.find().explain()  
{  
  "queryPlanner": {  
    "plannerVersion": 1,  
    "namespace": "test.multi",  
    "indexFilterSet": false,  
    "parsedQuery": {  
      "$and": [ ]  
    },  
    "winningPlan": {  
      "stage": "COLLSCAN",
```

```
"filter" : {
    "$and" : [ ]
},
"direction" : "forward"
},
"rejectedPlans" : [ ]
},
"serverInfo" : {
    "host" : "LIN73000604",
    "port" : 27017,
    "version" : "3.2.4",
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}
> db.multi.createIndex({a:1,b:1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
> db.multi.find().explain()
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.multi",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "$and" : [ ]
        },
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
```

```
        "$and" : [ ]
    },
    "direction" : "forward"
},
"rejectedPlans" : [ ]
},
"serverInfo" : {
    "host" : "LIN73000604",
    "port" : 27017,
    "version" : "3.2.4",
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}
> db.multi.insert({a:3,b:4})
WriteResult({ "nInserted" : 1 })
> db.multi.find({a:1}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.multi",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "a" : {
                "$eq" : 1
            }
        },
        "winningPlan" : {
            "stage" : "FETCH",
            "inputStage" : {
                "stage" : "IXSCAN",
                "keyPattern" : {
                    "a" : 1,
                    "b" : 1
                }
            }
        }
    }
}
```

```
        },
        "indexName" : "a_1_b_1",
        "isMultiKey" : false,
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 1,
        "direction" : "forward",
        "indexBounds" : {
            "a" : [
                "[1.0, 1.0]"
            ],
            "b" : [
                "[MinKey, MaxKey]"
            ]
        }
    },
    "rejectedPlans" : [ ]
},
"serverInfo" : {
    "host" : "LIN73000604",
    "port" : 27017,
    "version" : "3.2.4",
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}
> db.multi.insert({a:5,b:[1,2,3]})
WriteResult({ "nInserted" : 1 })
> db.multi.find({a:5}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 1,
```

```
"namespace" : "test.multi",
"indexFilterSet" : false,
"parsedQuery" : {
    "a" : {
        "$eq" : 5
    }
},
"winningPlan" : {
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
            "a" : 1,
            "b" : 1
        },
        "indexName" : "a_1_b_1",
        "isMultiKey" : true, ( becos a is scalar and b is array) it uses multikey array
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 1,
        "direction" : "forward",
        "indexBounds" : {
            "a" : [
                "[5.0, 5.0]"
            ],
            "b" : [
                "[MinKey, MaxKey]"
            ]
        }
    }
},
"rejectedPlans" : [ ]
},
```

```
"serverInfo" : {
    "host" : "LIN73000604",
    "port" : 27017,
    "version" : "3.2.4",
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}
> db.multi.insert({a:[2,3,4],b:[1,2,3]}) In a multi key index we cannot have
both a and b as arrays
WriteResult{
    "nInserted" : 0,
    "writeError" : {
        "code" : 10088,
        "errmsg" : "cannot index parallel arrays [b] [a]"
    }
}
```

### **Multいけ with dot notation**

---

We shall see how to use dot notation to reach deep into a document and add index that is something in a subdocument of the main document .

#### **Examples**

`Db.students.findOne()`

We have `student_id` and `scores` array that has a bunch of documents as elements in the array where each document has a type `exame` and a score .

And it s also a class id .

Let say we have to index on the score itself .

`Db.students.createIndex(['scores.score':1]);` → use square bracket for index creation as shown

It would take 15 or 20 mins to create this index

There are 10 million documents

If we use `getIndexes`, we can see that there are two indexes one is on id and the other is on

`scores.score` which is the multi key index

To search of the records where any score is above the given value ,

The belo ex find s everything where `scores.score` is > 99

```
Db.students.explain().find({'scores.score':{'gt':99}})
```

In the explain plan we can see the index scan ,Winning plan included `scores.score` index with the `scores.score` between 99.0 and infinity ..

To find people that had exam score that was above 99 .

```
Db.students.explain().find({'scores':{$elemMatch:{type:'exam',score:{'gt':99.8}}}});
```

Trying to inspect scores array inside the document .

Then we want ti find a document which has an element of array that is of type exam and a score .

**LM match** matches the document that contains an array field with at least one element that matches all specific criteria .

In other words , there might be more than one element , there might be more than one exam in this array that matches this criteria but . we ensure that we match atleast one with all this criteria .

So we are looking for element of type exam and score greater than 99.8

So in the result we could see that there is an exam score more than 99.8

```
Db.students.explain().find({{"scores":{$elemMatch:{type : 'exam',score:{'gt':99.8}}}}).count();
```

---

## Covered Queries

---

```
> for (i=0;i<1000000;i++) { db.numbers.insert({"i" :i,"j" :"j" + i, "k":Math.floor(Math.random()*120),"created" : new Date()});}
```

```
➤ db.numbers.ensureIndex({i:1,j:1,k:1})
```

```
>var exp=db.numbers.find().explain(executionStats)
> exp.find({i:6,j:"j6",k:77})
{
  "queryPlanner" : {
    "plannerVersion" : 1,
```

```
"namespace" : "test.numbers",
"indexFilterSet" : false,
"parsedQuery" : {
    "$and" : [
        {
            "i" : {
                "$eq" : 6
            }
        },
        {
            "j" : {
                "$eq" : "j6"
            }
        },
        {
            "k" : {
                "$eq" : 77
            }
        }
    ]
},
"winningPlan" : {
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
            "i" : 1,
            "j" : 1,
            "k" : 1
        },
        "indexName" : "i_1_j_1_k_1",
        "isMultiKey" : false,
        "isUnique" : false,
        "isSparse" : false,
    }
}
```

```
"isPartial" : false,
"indexVersion" : 1,
"direction" : "forward",
"indexBounds" : {
    "i" : [
        "[6.0, 6.0]"
    ],
    "j" : [
        "[\"j6\", \"j6\"]"
    ],
    "k" : [
        "[77.0, 77.0]"
    ]
}
},
"rejectedPlans" : [ ]
},
"executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 1,
    "executionTimeMillis" : 4,
"totalKeysExamined" : 1,
"totalDocsExamined" : 1, ( it scans the documents) ( this is not a covered query
becos _id is projected )
"executionStages" : {
    "stage" : "FETCH",
    "nReturned" : 1,
    "executionTimeMillisEstimate" : 0,
    "works" : 2,
    "advanced" : 1,
    "needTime" : 0,
    "needYield" : 0,
    "saveState" : 0,
```

```
"restoreState" : 0,  
"isEOF" : 1,  
"invalidates" : 0,  
"docsExamined" : 1,  
"alreadyHasObj" : 0,  
"inputStage" : {  
    "stage" : "IXSCAN",  
    "nReturned" : 1,  
    "executionTimeMillisEstimate" : 0,  
    "works" : 2,  
    "advanced" : 1,  
    "needTime" : 0,  
    "needYield" : 0,  
    "saveState" : 0,  
    "restoreState" : 0,  
    "isEOF" : 1,  
    "invalidates" : 0,  
    "keyPattern" : {  
        "i" : 1,  
        "j" : 1,  
        "k" : 1  
    },  
    "indexName" : "i_1_j_1_k_1",  
    "isMultiKey" : false,  
    "isUnique" : false,  
    "isSparse" : false,  
    "isPartial" : false,  
    "indexVersion" : 1,  
    "direction" : "forward",  
    "indexBounds" : {  
        "i" : [  
            "[6.0, 6.0]"  
        ],  
        "j" : [  
    }
```

```
"[\"j6\", \"j6\"]"
],
"k": [
    "[77.0, 77.0]"
]
},
"keysExamined": 1,
"dupsTested": 0,
"dupsDropped": 0,
"seenInvalidated": 0
}
}
},
"serverInfo": {
    "host": "LIN73000604",
    "port": 27017,
    "version": "3.2.4",
    "gitVersion": "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok": 1
}
> exp.find({i:6,j:"j6",k:77},{_id:0,i:1,j:1,k:1}) (we need to project id:0 to make
mongodb use only index scan which is called covered index)
{
    "queryPlanner": {
        "plannerVersion": 1,
        "namespace": "test.numbers",
        "indexFilterSet": false,
        "parsedQuery": {
            "$and": [
                {
                    "i": {
                        "$eq": 6
                    }
                }
            ]
        }
    }
}
```

```
        },
        {
          "j" : {
            "$eq" : "j6"
          }
        },
        {
          "k" : {
            "$eq" : 77
          }
        }
      ],
    },
    "winningPlan" : {
      "stage" : "PROJECTION",
      "transformBy" : {
        "_id" : 0,
        "i" : 1,
        "j" : 1,
        "k" : 1
      },
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "i" : 1,
          "j" : 1,
          "k" : 1
        },
        "indexName" : "i_1_j_1_k_1",
        "isMultiKey" : false,
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 1,
      }
    }
  }
}
```

```
"direction" : "forward",
"indexBounds" : {
    "i" : [
        "[6.0, 6.0]"
    ],
    "j" : [
        "[\"j6\", \"j6\"]"
    ],
    "k" : [
        "[77.0, 77.0]"
    ]
},
"rejectedPlans" : [ ],
},
"executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 1,
    "executionTimeMillis" : 7,
"totalKeysExamined" : 1,
"totalDocsExamined" : 0, ( It scans only the index )
"executionStages" : {
        "stage" : "PROJECTION",
        "nReturned" : 1,
        "executionTimeMillisEstimate" : 0,
        "works" : 2,
        "advanced" : 1,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "invalidates" : 0,
```

```
"transformBy" : {
    "_id" : 0,
    "i" : 1,
    "j" : 1,
    "k" : 1
},
"inputStage" : {
    "stage" : "IXSCAN",
    "nReturned" : 1,
    "executionTimeMillisEstimate" : 0,
    "works" : 2,
    "advanced" : 1,
    "needTime" : 0,
    "needYield" : 0,
    "saveState" : 0,
    "restoreState" : 0,
    "isEOF" : 1,
    "invalidates" : 0,
    "keyPattern" : {
        "i" : 1,
        "j" : 1,
        "k" : 1
    },
    "indexName" : "i_1_j_1_k_1",
    "isMultiKey" : false,
    "isUnique" : false,
    "isSparse" : false,
    "isPartial" : false,
    "indexVersion" : 1,
    "direction" : "forward",
    "indexBounds" : {
        "i" : [
            "[6.0, 6.0]"
        ],
        "j" : [
            "[6.0, 6.0]"
        ],
        "k" : [
            "[6.0, 6.0]"
        ]
    }
}
```

```
"j" : [
    "[\"j6\", \"j6\"]"
],
"k" : [
    "[77.0, 77.0]"
]
},
"keysExamined" : 1,
"dupsTested" : 0,
"dupsDropped" : 0,
"seenInvalidated" : 0
}
}
},
"serverInfo" : {
    "host" : "LIN73000604",
    "port" : 27017,
    "version" : "3.2.4",
    "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"
},
"ok" : 1
}
```

**Geospatial Indexes :**

```
=====
```

Allows us to find things based on location .

First we shall discuss about 2 dimensional model and then 3 d model .

```
C:\Users\srsenthil>mongo
2016-04-07T09:41:31.760+0530 I CONTROL [main] Hotfix KB2731284 or later
update is not installed, will zero-out data files
MongoDB shell version: 3.2.4
```

connecting to: test

```
> db.stores.insert({"name":"rubys","type":"Barber","Location":[40,74]})  
WriteResult({ "nInserted" : 1 })  
> db.stores.insert({"name":"ACE  
Hardware","type":"Hardware","Location":[40.232,-74.343]})  
WriteResult({ "nInserted" : 1 })  
> db.stores.insert({"name":"Tickel Candy","type":"food","Location":[41.232,-  
75.343]})  
WriteResult({ "nInserted" : 1 })  
  
> db.stores.find()  
{ "_id" : ObjectId("5705ef6912c458824b4eebb5"), "name" : "rubys", "type" :  
"Barber", "Location" : [ 40, 74 ] }  
{ "_id" : ObjectId("5705f08812c458824b4eebb6"), "name" : "ACE Hardware",  
"type" : "Hardware", "Location" : [ 40.232, -74.343 ] }  
{ "_id" : ObjectId("5705f0b012c458824b4eebb7"), "name" : "Tickel Candy",  
"type" : "food", "Location" : [ 41.232, -75.343 ] }  
> db.stores.find({Location:{$near:[66,74]}})  
  
> db.stores.ensureIndex({Location:"2d",type:1})  
{  
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 1,  
    "numIndexesAfter" : 2,  
    "ok" : 1  
}  
> db.stores.getIndexes()  
[  
    {  
        "v" : 1,  
        "key" : {  
            "_id" : 1  
        },  
        "name" : "_id_" ,
```

```
"ns" : "test.stores"
},
{
  "v" : 1,
  "key" : {
    "Location" : "2d",
    "type" : 1
  },
  "name" : "Location_2d_type_1",
  "ns" : "test.stores"
}
]
> db.stores.find({Location:{$near:[50,50]}})
{ "_id" : ObjectId("5705ef6912c458824b4eebb5"), "name" : "rubys", "type" :
"Barber", "Location" : [ 40, 74 ] }
{ "_id" : ObjectId("5705f08812c458824b4eebb6"), "name" : "ACE Hardware",
"type" : "Hardware", "Location" : [ 40.232, -74.343 ] }
{ "_id" : ObjectId("5705f0b012c458824b4eebb7"), "name" : "Ticket Candy",
"type" : "food", "Location" : [ 41.232, -75.343 ] }
>
>db.places.insert({"name":"Apple Store","city":"Palo
alto","location":{"type":"Point","coordinates":[-
122.169129,37.4434854]}, "type":"Retail"})

WriteResult({ "nInserted" : 1 })

> db.places.insert({"name":"Peninsula Ceremery","city":"Palo
alto","location":{"type":"Point","coordinates":[-
122.137044,37.423556]}, "type":"Retail"})

WriteResult({ "nInserted" : 1 })
```

```
> db.places.insert({"name":"Tamalpais State","city":"Mill  
Valley","location":{"type":"Point","coordinates":[-  
122.5995522,37.895943]},{"type":"Park"})  
  
WriteResult({ "nInserted" : 1 })  
  
> db.places.insert({"name":"Fry's Electroincs","city":"Palo  
alto","location":{"type":"Point","coordinates":[-  
122.137044,37.423556]},{"type":"Retail"})  
  
WriteResult({ "nInserted" : 1 })  
  
  
> db.places.ensureIndex({location:"2dsphere"})  
  
{  
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 1,  
    "numIndexesAfter" : 2,  
    "ok" : 1  
}  
  
> db.places.getIndexes()  
[  
    {  
        "v" : 1,  
        "key" : {
```

```
"_id": 1
},
"name": "_id_",
"ns": "test.places"
},
{
  "v": 1,
  "key": {
    "location": "2dsphere"
  },
  "name": "location_2dsphere",
  "ns": "test.places",
  "2dsphereIndexVersion": 3
}
]

> db.places.find({location:{$near : {$geometry : { type : "Point",coordinates :[-122.166641,37.4278925]},$maxDistance :2000}}}).pretty()
{
  "_id": ObjectId("5706057e12c458824b4eebb8"),

```

```
"name" : "Apple Store",
"city" : "Palo alto",
"location" : {
    "type" : "Point",
    "coordinates" : [
        -122.169129,
        37.4434854
    ]
},
"type" : "Retail"
}
```

---

**Creating Index in the background:**

---

The students collection already has index on student\_id and scores.score array field .

If not available , create it as given below

```
> db.students.createIndex(['scores.score:1'])
```

This will take nearly 15 mins time , meanwhile , open another session and type the steps given below

In another terminal ,

- Use school
- Db.students.findOne()

The request is still in process and the it blocks .

It means that when we create the index in the foreground and try to do some querying on the same database from the other session , it blocks the reads .

This can be avoided by creating the index in the background .

In first Terminal

```
> db.students.getIndexes()  
[  
 {  
   "v" : 1,  
   "key" : {  
     "_id" : 1  
   },  
   "ns" : "school.students",
```

```
"name" : "_id_"
},
{
  "v" : 1,
  "key" : [
    "scores.score:1"
  ],
  "ns" : "school.students",
  "name" : "0_scores.score:1"
}
]
```

In second terminal,

- Use school
- Db.students.findOne()
- ..(response takes a lot of time ... it blocks )
- Go back to terminal one and press ctl c ,it prompts if the index creation has to be killed .type yes

**Terminal One :**

- Create the index in the background ,
- > db.students.createIndex(['scores.score:1'],{background:true})

**This would take some time , meanwhile do the seps given below in terminal two**

**MongoDB shell version: 2.2.0**

**connecting to: test**

**> db.students.findOne()**

**null**

**> db.employees.findOne()**

**{**

**"\_id" : ObjectId("5700bf5c071d2db146924623"),**

**"employee\_id" : 1,**

**"name" : "andrew",**

**"cell" : 986055433**

**}**

**> use school**

**switched to db school**

**> db.students.findOne()**

**{**

**"\_id" : ObjectId("56e29884b5493e90ac7ec4d7"),**

**"student\_id" : 0,**

**"scores" : [**

**{**

```
"type" : "exam",
"score" : 1.5629349419872596
},
{
  "type" : "quiz",
  "score" : 69.23197999806101
},
{
  "type" : "homework",
  "score" : 78.82490318401389
},
{
  "type" : "homework",
  "score" : 97.93998624112517
}
],
"class_id" : 386
}
```

**What we find is that when we query the students database from the second session , it gives the results immediately without blocking unlike the first case ... so when indexes are created in the background , still read operations are permitted in the second session**



---

## Index Examples

---

### 1 . Creating a 1 million documents

---

```
for (i=0; i<1000000; i++) {  
... db.users.insert(  
... {  
... "i" : i,  
  
"username" : "user"+i,  
... "age" : Math.floor(Math.random()*120),  
... "created" : new Date()  
... }  
... );  
... }
```

2. db.users.find({username: "user101"}).explain()

3. db.users.find({username: "user101"}).limit(1).explain()

4. db.users.ensureIndex({"username" : 1}) ( Single Key Index)

5. db.users.find({username" : "user101"}).explain()

6. db.users.find().sort({"age" : 1, "username" : 1})

7. db.users.ensureIndex({"age" : 1, "username" : 1}) ( Compound Indexes )

8. db.users.find({}, {"\_id" : 0, "i" : 0, "created" : 0})

9. db.users.find({"age" : 21}).sort({"username" : -1})

10. db.users.find({"age" : {"\$gte" : 21, "\$lte" : 30}}).sort({"username" : 1}).  
hint({"username" : 1, "age" : 1}).

... explain()

11. db.users.ensureIndex({"username" : 1}, {"unique" : true}) (unique Index)

12. db.users.insert({username: "bob"})

E11000 duplicate key error index: test.users.\$username\_1 dup key: { : "bob" }

## Profiling

---

---

### Creating Students collection

---

Loading 10 million documents in students collection in school database using the script given below

```
db=db.getSiblingDB("school");
db.students.drop();

types = ['exam', 'quiz', 'homework', 'homework'];
// 1 million students
for (i = 0; i < 1000000; i++) {

    // take 10 classes
    for (class_counter = 0; class_counter < 10; class_counter
++) {
        scores = []
        // and each class has 4 grades
        for (j = 0; j < 4; j++) {

            scores.push({'type':types[j], 'score':Math.random()*100});
        }

        // there are 500 different classes that they can take
        class_id = Math.floor(Math.random()*501); // get a class
id between 0 and 500

        record = {'student_id':i, 'scores':scores,
'class_id':class_id};
        db.students.insert(record);

    }
}

MongoDb by default logs queries which take more 100ms in a log
file which is read at the time of start up
```

Profiler writes queries ,documents to system.profile

Three levels of profiler

Level 0 => profiling is off  
level 1 => log slow queries  
level 2 => log all queries ---general debugging feature - to see database traffic

Enable the profiler from the command line

Create a directory called proflog in d:\data\db

```
mongod --dbpath D:\data\db\proflog --profile=1 --slowms=2
```

log in to mongo shell

Remove the index from students collection

```
db.students.find({student_id:10000})  
  
db.system.profile.find().pretty()  
> db.getProfilingLevel()  
1  
> db.getProfilingStatus()  
{ "was" : 1, "slowms" : 2 }  
> db.setProfilingLevel(1,4)  
{ "was" : 1, "slowms" : 2, "ok" : 1 }  
> db.setProfilingLevel(0)  
{ "was" : 1, "slowms" : 4, "ok" : 1 }  
> db.getProfilingStatus()  
{ "was" : 0, "slowms" : 4 }  
> db.getProfilingLevel()  
0
```

Mongotop

Gives the info about time taken for reads and writes

Mongotop 3 (every 3 seconds it gives the details)

Mongostat → similar to iostat

Simulate the population of students collection with two storage engines configured on 27017 and 271018

Monitor the infor in both the ports using mongostat

From command line → 27017

➤ Mongostat

From 2<sup>nd</sup> CMD line → 27018

➤ Mongostat -port 27018

---

