https://www.mongodb.com/docs/manual/tutorial/getting-started/

https://www.jdoodle.com/online-mongodb-terminal/

https://cloud.mongodb.com/v2#/clusters

https://www.mycompiler.io/new/mongodb

Types Of Big Data

Following are the types of Big Data:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

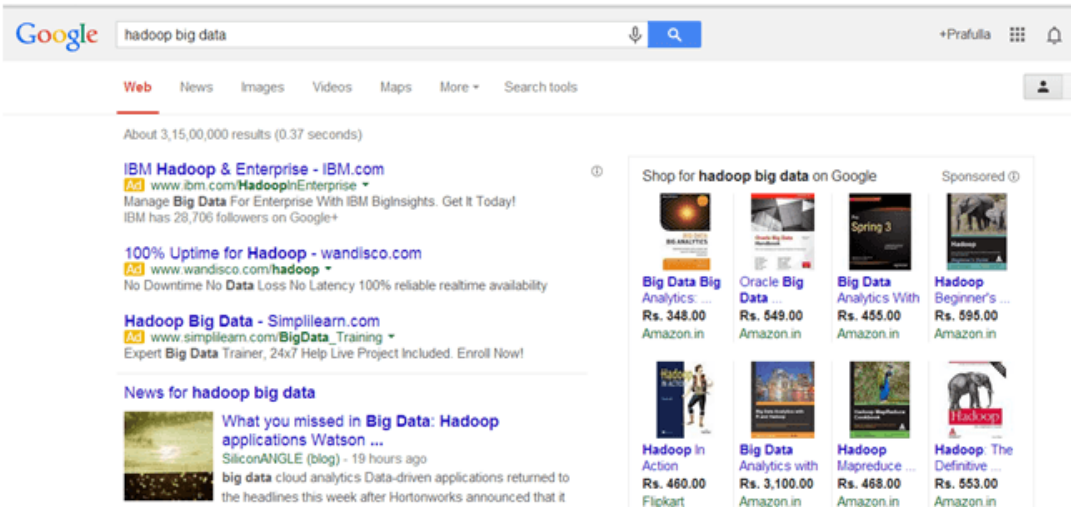
Examples:

An ‘Employee’ table in a database is an example of Structured Data

Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
2365	Rajesh Kulkarni	Male	Finance	650000
3398	Pratibha Joshi	Female	Admin	650000
7465	Shushil Roy	Male	Admin	500000
7500	Shubhojit Das	Male	Finance	500000
7699	Priya Sane	Female	Finance	550000

Examples Of Un-structured Data

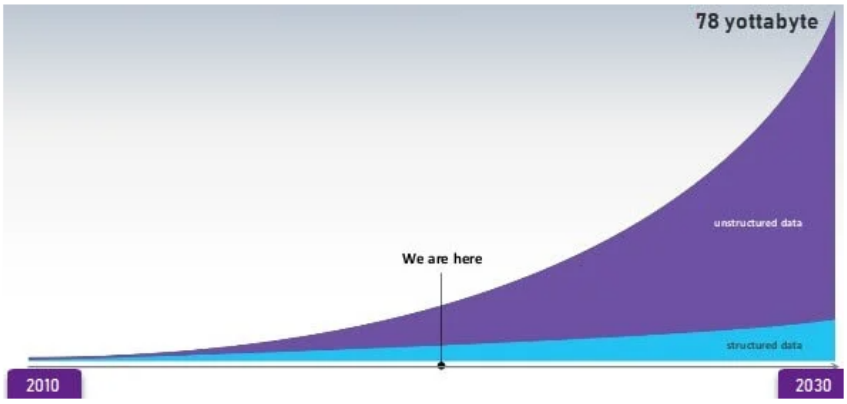
The output returned by ‘Google Search’



Personal data stored in an XML file-

```
<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>
<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>
<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>
```

Data Growth over the years



Data Growth over the years

Characteristics Of Big Data

Big data can be described by the following characteristics:

- Volume
- Variety
- Velocity
- Variability

Advantages Of Big Data Processing

Ability to process Big Data in DBMS brings in multiple benefits, such as-

- Businesses can utilize outside intelligence while taking decisions

Access to social data from search engines and sites like facebook, twitter are enabling organizations to fine tune their business strategies.

- Improved customer service

Traditional customer feedback systems are getting replaced by new systems designed with Big Data technologies. In these new systems, Big Data and natural language processing technologies are being used to read and evaluate consumer responses.

- Early identification of risk to the product/services, if any
- Better operational efficiency

Problems with big data:

1. **Lack of knowledge Professionals**

   These professionals will include data scientists, data analysts, and data engineers to work with the tools and make sense of giant data sets. One of the challenges that any Company faces is a lack of massive Data professionals. This is often because data handling tools have evolved rapidly, but in most cases, the professionals haven't
2. **Lack of proper understanding of Massive Data**
3. **Data Growth Issues**

   One of the foremost pressing challenges of massive Data is storing these huge sets of knowledge properly. The quantity of knowledge being stored in data centers and databases of companies is increasing rapidly. As these data sets grow exponentially with time, it gets challenging to handle. Most of the info is unstructured and comes from documents, videos, audio, text files, and other sources

   Sol

Compression is employed for reducing the number of bits within the data, thus reducing its overall size. Deduplication is the process of removing duplicate and unwanted data from a knowledge set. Data tiering allows companies to store data in several storage tiers. It ensures that the info is residing within the most appropriate space for storing. Data tiers are often public cloud, private cloud, and flash storage, counting on the info size and importance. Companies also are choosing its tools, like Hadoop, NoSQL, and other technologies.

4. **Confusion while Big Data Tool selection**

5. **Securing Data**

   Sol:
   Companies are recruiting more cybersecurity professionals to guard their data. Other steps taken for Securing it include: Data encryption Data segregation Identity and access control Implementation of endpoint security Real-time security monitoring Use its security tools, like IBM Guardian.

MONGO DB:

MongoDB is a document database. It stores data in a type of JSON format
A record in MongoDB is a document, which is a data structure composed of key value pairs similar to the structure of JSON objects.
MongoDB database are called documents, and the field values may include numbers, strings, booleans, arrays, or even nested documents.

```
{
    title: "Post Title 1",
    body: "Body of post.",
```

```
    category: "News",
    likes: 1,
    tags: ["news", "events"],
    date: Date()
}
```

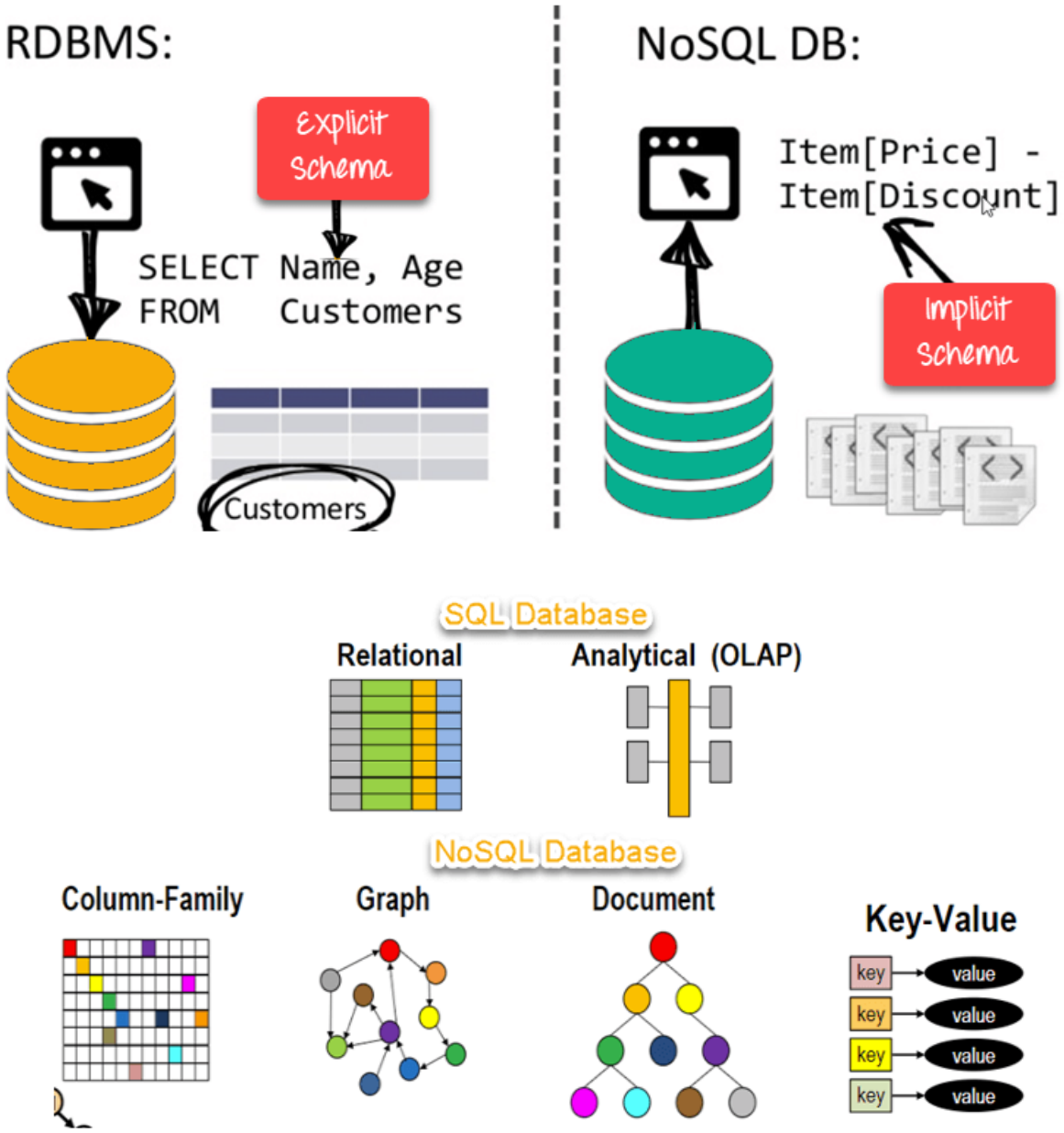
db.posts.find({category: "News"})

```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    body: 'Body of post.',
    category: 'News',
    likes: 1,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  }
```

NOSQL:

- Cloud computing also rose in popularity, and developers began using public clouds to host their applications and data.
- They wanted the ability to distribute data across multiple servers and regions to make their applications resilient, to scale out instead of scale up, and to intelligently geo-place their data.
- Some NoSQL databases like MongoDB provide these capabilities.

- **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema.
- It avoids joins, and is easy to scale.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps.
- For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.
- **NoSQL database** stands for “Not Only SQL” or “Not SQL.” Though a better term would be “NoREL”, NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.



Types of NoSQL databases

Over time, four major types of NoSQL databases emerged: document databases, key-value databases, wide-column stores, and graph databases.

- Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of

fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.

- Key-value databases are a simpler type of database where each item contains keys and values.
- Wide-column stores store data in tables, rows, and dynamic columns.
- Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

RDBMS vs NoSQL: Data Modeling Example

Let's consider an example of storing information about a user and their hobbies. We need to store a user's first name, last name, cell phone number, city, and hobbies.

In a relational database, we'd likely create two tables: one for Users and one for Hobbies.

Users

ID	first_name	last_name	cell	city
1	Leslie	Yepp	8125552344	Pawnee

Hobbies

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working

In order to retrieve all of the information about a user and their hobbies, information from the Users table and Hobbies table will need to be joined together.

The data model we design for a NoSQL database will depend on the type of NoSQL database we choose. Let's consider how to store the same information about a user and their hobbies in a document database like MongoDB.

In order to retrieve all of the information about a user and their hobbies, a single document can be retrieved from the database. No joins are required, resulting in faster queries.

```
{
   "_id": 1,
   "first_name": "Leslie",
   "last_name": "Yepp",
   "cell": "8125552344",
   "city": "Pawnee",
   "hobbies": ["scrapbooking", "eating waffles", "working"]
}
```


When should NoSQL be used?

- Storage of structured and semi-structured data
- Huge volumes of data
- Requirements for scale-out architecture
- Modern application paradigms like microservices and real-time streaming

Mongo Collection:

{

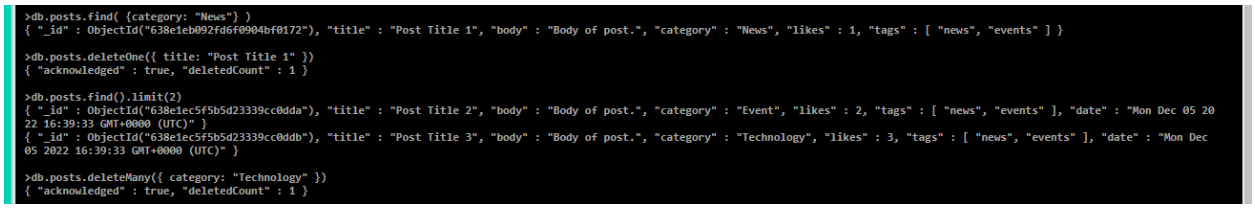
```
    title: "Post Title 1",
    body: "Body of post.",
    category: "News",
    likes: 1,
    tags: ["news", "events"],
    date: Date()
}
```

```
Welcome to JDoodle - online mongo Terminal, Starting mongo Terminal, Please wait...
New mongoDB session started...
>use bhavans
switched to db bhavans

>db.createCollection("posts")
{ "ok" : 1 }

>db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"]
})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("638e1eb092fd6f0904bf0172")
}
```

```
>db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 4",
    body: "Body of post.",
    category: "Event",
    likes: 4,
    tags: ["news", "events"],
    date: Date()
  }
])
{
```

```
{
   "acknowledged" : true,
   "insertedIds" : [
      ObjectId("638e1ec5f5b5d23339cc0dda"),
      ObjectId("638e1ec5f5b5d23339cc0ddb"),
      ObjectId("638e1ec5f5b5d23339cc0ddc")
   ]
}

>db.posts.findOne()
{
   "_id" : ObjectId("638e1eb092fd6f0904bf0172"),
   "title" : "Post Title 1",
   "body" : "Body of post.",
   "category" : "News",
   "likes" : 1,
   "tags" : [
      "news",
      "events"
   ]
}

>db.posts.find( {category: "News"} )
{ "_id" : ObjectId("638e1eb092fd6f0904bf0172"), "title" : "Post Title 1", "body" : "Body of post.", "category" : "News", "likes" : 1, "tags" : [ "news", "events" ] }

>db.posts.deleteOne({ title: "Post Title 1" })
{ "acknowledged" : true, "deletedCount" : 1 }

>db.posts.find().limit(2)
{ "_id" : ObjectId("638e1ec5f5b5d23339cc0dda"), "title" : "Post Title 2", "body" : "Body of post.", "category" : "Event", "likes" : 2, "tags" : [ "news", "events" ], "date" : "Mon Dec 05 20
22 16:39:33 GMT+0000 (UTC)" }
{ "_id" : ObjectId("638e1ec5f5b5d23339cc0ddb"), "title" : "Post Title 3", "body" : "Body of post.", "category" : "Technology", "likes" : 3, "tags" : [ "news", "events" ], "date" : "Mon Dec
05 2022 16:39:33 GMT+0000 (UTC)" }
```

```
> db.student.update({name:"avi"}, {$set:{name:"helloword"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find().pretty()
{
        "_id" : ObjectId("5f8dd0790cf217478ba9355d"),
        "name" : "helloword",
        "age" : 12
}
{
        "_id" : ObjectId("5f8dd08a0cf217478ba9355e"),
        "name" : "payal",
        "age" : 15
}
{
        "_id" : ObjectId("5f8dd0940cf217478ba9355f"),
        "name" : "prachi",
        "age" : 17
}
>
```



Feature	MySQL	MongoDB

Data Structure	It stores each individual record as a table cell with rows and columns	It stores unrelated data in JSON like documents
Schema	MySQL requires a schema definition for the tables in the database	MongoDB doesn't require any prior schema
Languages	Supports Structured Query Language (SQL)	Supports JSON Query Language to work with data
Foreign Key	Supports the usage of Foreign keys	Doesn't support the usage of Foreign keys
Scalability	SQL Database can be scaled vertically	MongoDB database can be scaled both vertically and horizontally

Join Operation	Supports Join operation	Doesn't support Join operation
Performance	Optimized for high performance joins across multiple tables	Optimized for write performance

```
> db.student.update({name:"prachi"}, {$set:{age:20}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find().pretty()
{
        "_id" : ObjectId("5f8dd0790cf217478ba9355d"),
        "name" : "helloword",
        "age" : 12
}
{
        "_id" : ObjectId("5f8dd08a0cf217478ba9355e"),
        "name" : "payal",
        "age" : 15
}
{
        "_id" : ObjectId("5f8dd0940cf217478ba9355f"),
        "name" : "prachi",
        "age" : 20
}
```

MongoDb

```
db.inventory.insertMany
([
  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

```
db.inventory.find( {} )
```

```
SELECT * FROM inventory
```

```
db.inventory.find( { status: "D" } )
```

```
SELECT * FROM inventory WHERE status = "D"
```

The

$or

operator performs a logical OR operation on an array of one or more <expressions> and

selects the documents that satisfy at least one of the <expressions>.

The

$or

has the following syntax:

{ $or: [{ <expression1> }, { <expression2> }, ... , { <expressionN> }] }

Consider the following example:

db.inventory.find({ $or: [{ quantity: { $lt: 20 } }, { price: 10 }] })

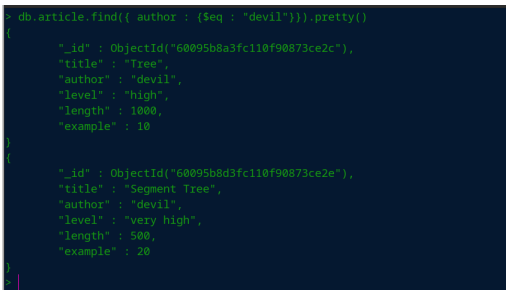
This query will select all documents in the inventory collection where either the quantity field value is less than 20 or the price field value equals 10.

db.article.find().pretty()

db.restaurants.find().sort( )

db.article.find({author:{$eq:"devil"}}).pretty()

Here, we are going to display the documents that matches the filter query(i.e., {author : {$eq : “devil”}}) from the article collection



To get the specific numeric data using conditions like greater than equal or less than equal use the $gte or $lte operator in the find() method.

Syntax:

db.collection_name.find({< key > : {$gte : < value >}})

or

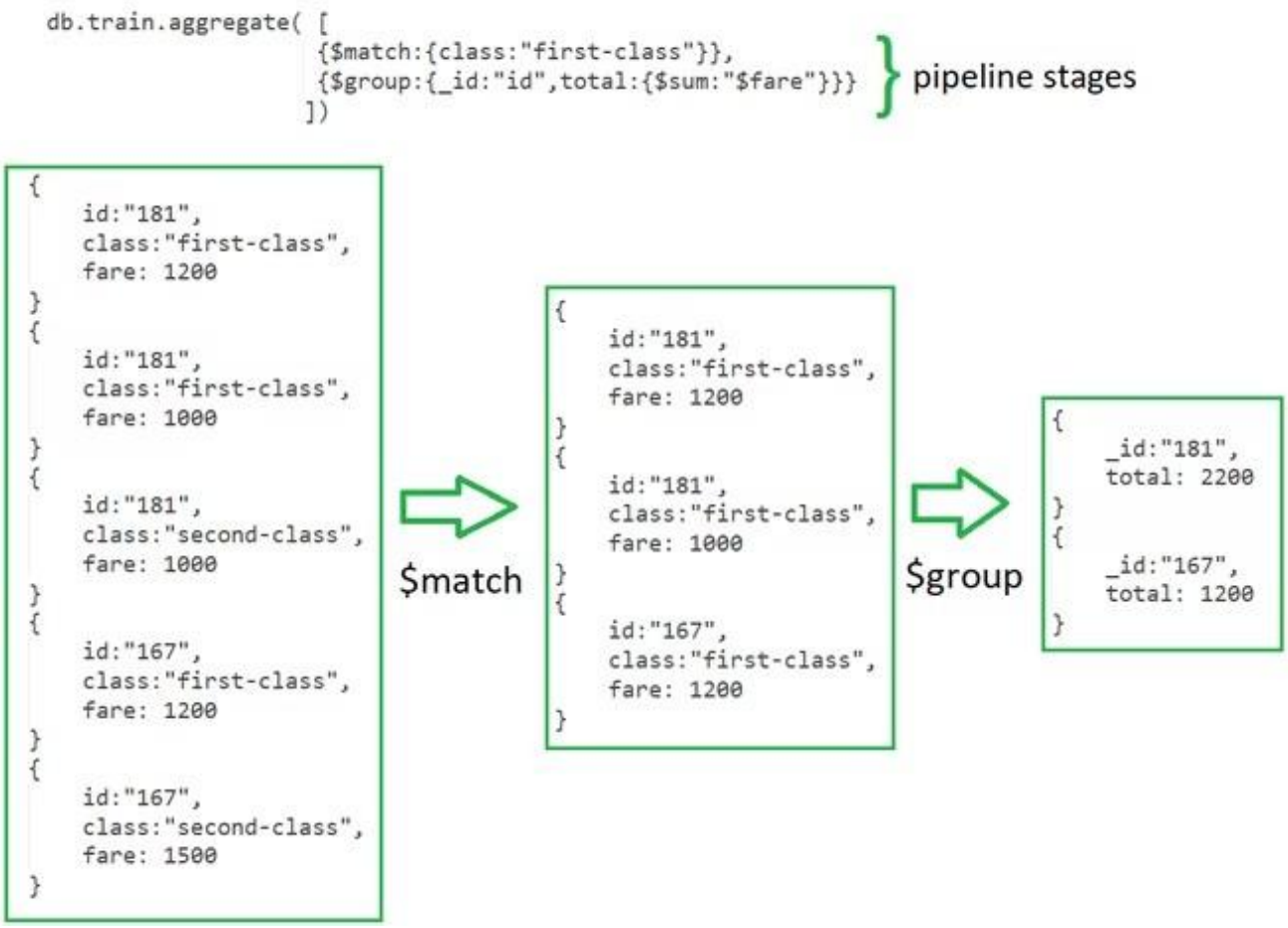
db.collection_name.find({< key > : {$lte : < value >}})

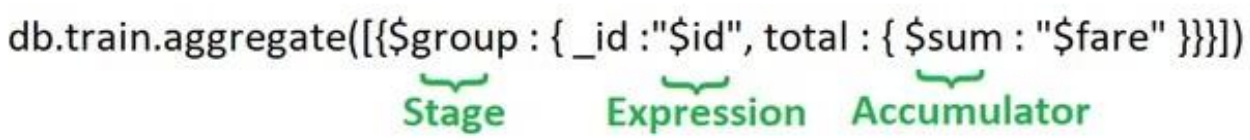
db.article.find({length:{$gte:510}}).pretty()

$exists operator shows all the collection documents if they exist on a given key.

Syntax:

db.collection_name.find({< key > : {$exists : < boolean >}})





$match:

It is used for filtering the documents and can reduce the amount of documents that are given as input to the next stage.

$project: It is used to select some specific fields from a collection.

$group: It is used to group documents based on some value.

$sort: It is used to sort the document that is rearranging them

$skip: It is used to skip n number of documents and passes the remaining documents

$limit: It is used to pass first n number of documents thus limiting them.

$unwind: It is used to unwind documents that are using arrays i.e. it deconstructs an array field in the documents to return documents for each element.

$out: It is used to write resulting documents to a new collection

Expressions:

It refers to the name of the field in input documents for e.g. { $group : { _id : “$id“, total:{$sum:”$fare“}}}

Here $id and $fare are expressions.

Accumulators: These are basically used in the group stage

sum: It sums numeric values for the documents in each group

count: It counts total numbers of documents

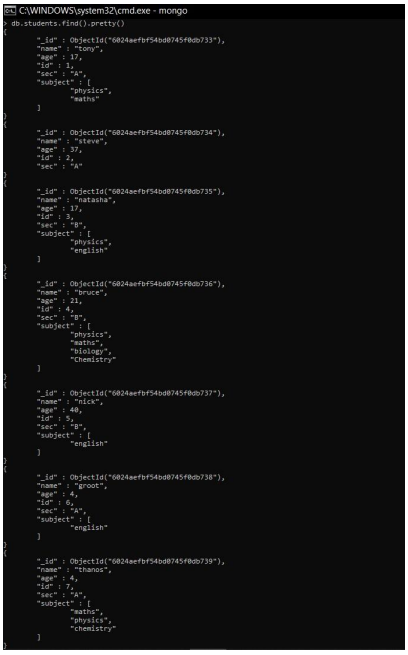
avg: It calculates the average of all given values from all documents

min: It gets the minimum value from all the documents

max: It gets the maximum value from all the documents

first: It gets the first document from the grouping

last: It gets the last document from the grouping



Displaying the total number of students in one section only

db.students.aggregate([{$match:{sec:"B"}},{$count:"Total student in sec:B"}])

Displaying details of students whose age is greater than 30 using match stage

db.students.aggregate([{$match:{age:{$gt:30}}}])

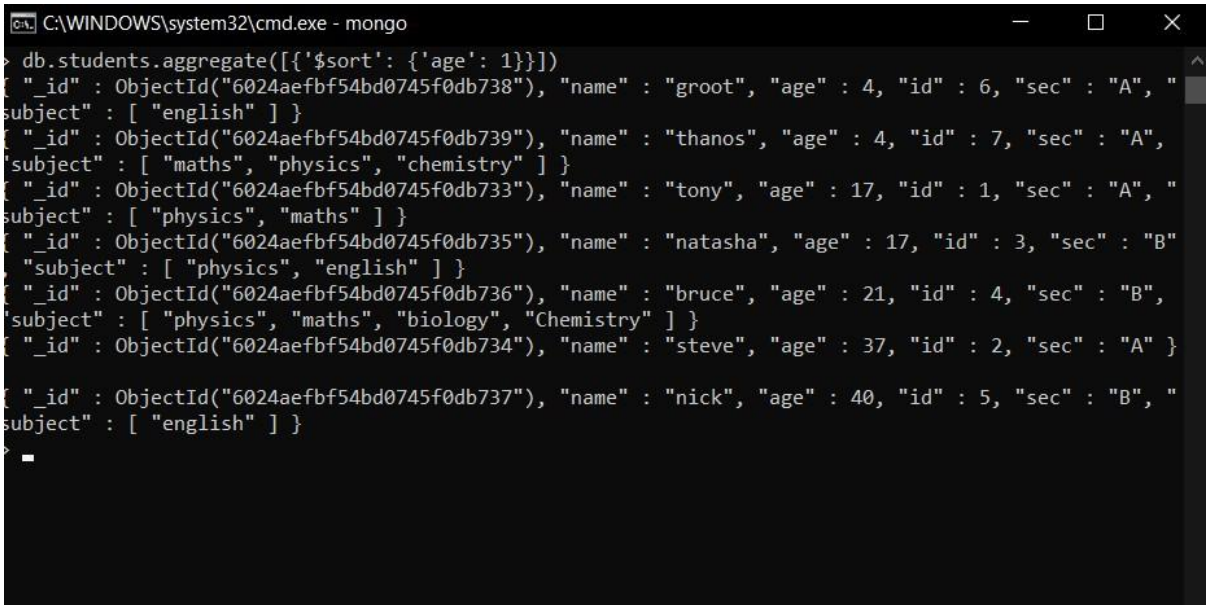
In this example, we display students whose age is greater than 30. So we use the $match operator to filter out the documents.

Sorting the students on the basis of age

db.students.aggregate([{'$sort': {'age': 1}}])

In this example, we are using the $sort operator to sort in ascending order we provide ‘age’:1

if we want to sort in descending order we can simply change 1 to -1 i.e. ‘age’:-1.



Displaying details of a student having the largest age in the section – B

db.students.aggregate([{$match:{sec:"B"}},{'$sort': {'age': -1}},{$limit:1}])

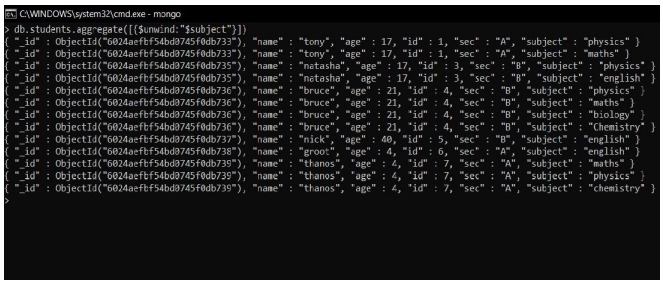
In this example, first, we only select those documents that have section B, so for that, we use the $match operator then we sort the documents in

descending order using $sort by setting ‘age’:-1 and then to only show the topmost result we use $limit

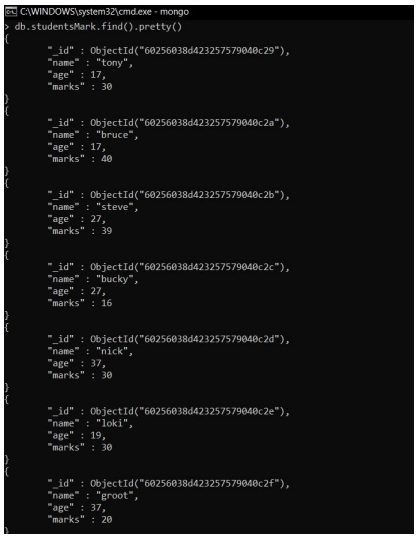
Unwinding students on the basis of subject

Unwinding works on array here in our collection we have array of subjects (which consists of different subjects inside it like math, physics, English, etc) so unwinding will be done on that i.e. the array will be deconstructed and the output will have only one subject not an array of subjects which were there earlier.

db.students.aggregate([{$unwind:"$subject"}])



Displaying distinct names and ages (non-repeating)



db.studentsMarks.distinct("name")

Here, we use a distinct() method that finds distinct values of the specified field(i.e., name).

https://www.google.com/amp/s/www.geeksforgeeks.org/aggregation-in-mongodb/amp/

Counting the total numbers of documents

db.studentsMarks.count()

Here, we use count() to find the total number of the document, unlike find() method it does not find all the document rather it counts them and return a number.

ACID stands for:

Atomic – Each transaction is either properly carried out or the process halts and the database reverts back to the state before the transaction started. This ensures that all data in the database is valid.

This property states transaction must be treated as an atomic unit, that is, either all of its operations are executed or none, and there must be no state in a database where a transaction is left partially completed also the states should be defined either before the execution of the transaction or after the execution of the transaction

Consistent – A processed transaction will never endanger the structural integrity of the database.

The database must remain in a consistent state after any transaction. Also no transaction should have any adverse effect on the data residing in the database and if the database was in a consistent state before the execution of a transaction then it must remain consistent after the execution of the transaction as well.

Isolated – Transactions cannot compromise the integrity of other transactions by interacting with them while they are still in progress.

In a database system where more than one transaction is being executed simultaneously and in parallel, the property of isolation states that each one of the transactions is going to be administered and executed as it is the only transaction in the system. Also no transaction will affect the existence of any other transactions.

Durable – The data related to the completed transaction will persist even in the cases of network or power outages. If a transaction fails, it will not impact the manipulated data.

The database should be durable enough to hold all its latest updates even if the system fails or restarts so, In a practical way of saying that if a transaction updates a chunk of data in a database and commit is performed then the database will hold the modified data but if the commit is not performed then no data is modified and it can only be done when the system start.

SQL

NoSQL

RDBMS is a row-oriented database

NoSQL is column-oriented databases

RDBMS works with structured and related data

NoSQL works on both unstructured and unrelated data

RDBMSs use schema which means the structure of the data should be predefined.

No need of schema for storing data

SQL databases can be scaled only using enhancing hardware

NoSQL databases can store unlimited data

SQL databases are a costly affair

NoSQL databases are cheaper

SQL database maintains data integrity

NoSQL database sometimes compromises data integrity to handle the large set of data.

opensourc

RDBMS databases are license based

NoSQL databases are opensource.

ACID Use Case Example

Financial institutions will almost exclusively use ACID databases. Money transfers depend on the atomic nature of ACID.

An interrupted transaction which is not immediately removed from the database can cause a lot of issues. Money could be debited from one account and, due to an error, never credited to another.

Which Databases are ACID compliant?

One safe way to make sure your database is ACID compliant is to choose a relational database management system. These include MySQL, PostgreSQL, Oracle, SQLite, and Microsoft SQL Server.

Some NoSQL DBMSs, such as Apache's CouchDB or IBM's Db2, also possess a certain degree of ACID compliance. However, the philosophy behind the NoSQL approach to database management goes against the strict ACID rules. Hence, NoSQL databases are not the recommended choice for those who need strict environments.

NoSQL is a Better Fit for Big Data Applications

We can consider big data from two perspectives.

Operational data – It mostly deals with online live data which are stored in operational databases. For example – flight booking data. This holds large sets of data.

Analytical data – It is a large amount of data to collect insights from it. For example – social media data for market analysis.

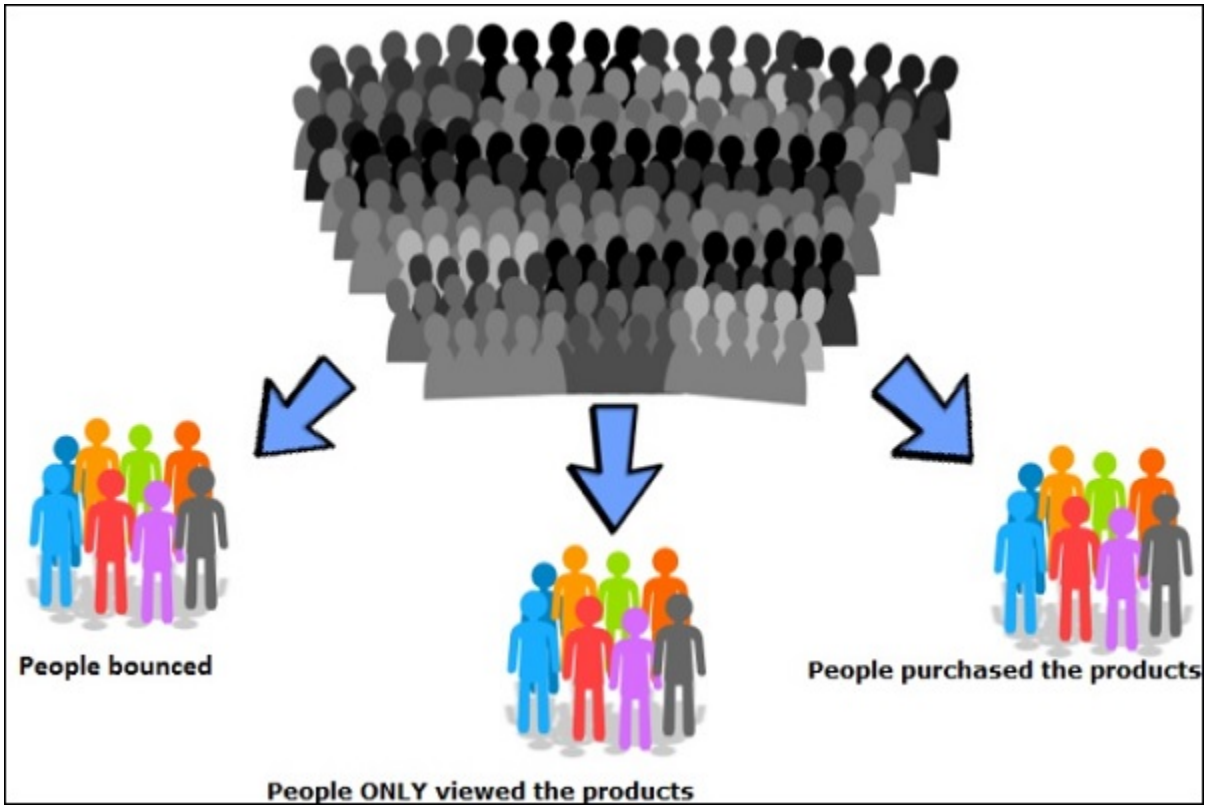
https://www.geeksforgeeks.org/sql-vs-nosql-which-one-is-better-to-use/

https://www.google.com/amp/s/www.geeksforgeeks.org/mongodb-tutorial/amp/

12/12/22

Segmentation:

Segmentation is the process that segregates the data to find the actionable items.



Data segmentation is the process of grouping your data into at least two subsets, although more separations may be necessary on a large network with sensitive data. Data should be grouped based on use cases and types of information, but also based on the sensitivity of that data and the level of authority needed to access that type of information. Once data is segmented, different security parameters and authentication rules should be established depending on the data segment at hand.

If a hacker gets past a network's traditional perimeter firewall and that network has skipped the data segmentation process, the hacker now has access to everything, rather than just a small portion of data within a segment. This lack of data segmentation leaves more data vulnerable to security breaches, and also makes it more difficult to find and stop the source of the breach across the wider network landscape.

The idea behind data segmentation, then, is to categorize your data, separate out the most sensitive data from the rest and define that as your protected surface, and then apply additional security measures around any protected surfaces that you have identified. Even if a breach occurs, your most sensitive data is now protected by extra layers of security measures.

https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/

NoSQL databases usually fall under any one of these four categories:

1. Key-value stores: is the most straightforward type where every item of your database gets stored in the form of an attribute name (i.e., "key") along with the value.

2. Wide-column stores: accumulate data collectively as a column rather than rows which are optimized for querying big datasets.

3. Document databases: couple every key with a composite data structure termed as a document. These documents hold a lot of different key-value pairs, as well as key-array pairs or sometimes nested documents.

4. Graph databases: are used for storing information about networks, like social connections.

Here is the list of comparisons between both the DBMS:

- SQL databases are mainly coming under Relational Databases (RDBMS) whereas NoSQL databases mostly come under non-relational or distributed database.

- SQL databases are table-oriented databases, whereas NoSQL databases document-oriented have key-value pairs or wide-column stores or graph databases.

- SQL databases have a predefined or static schema that is rigid, whereas NoSQL databases have dynamic or flexible schema to handle unstructured data.

- SQL is used to store structured data, whereas NoSQL is used to store structured as well as unstructured data.

- SQL databases can be considered as vertically scalable, but NoSQL databases are considered horizontally scalable.

- Scaling of SQL databases is done by mounting the horse-power of your hardware. But, scaling of NoSQL databases is calculated by mounting the databases servers for reducing the load.

- Examples of SQL databases: MySql, Sqlite, Oracle, Postgres SQL, and MS-SQL. Examples of NoSQL databases: BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB and Neo4j

- When your queries are complex SQL databases are a good fit for the intensive environment, and NoSQL databases are not an excellent fit for complex queries. Queries of NoSQL are not that powerful as compared to SQL query language.

- SQL databases need vertical scalability, i.e., excess of load can be managed by increasing the CPU, SSD, RAM, GPU, etc., on your server. In the case of NoSQL databases, they are horizontally scalable, i.e., the addition of more servers will ease out the load management thing to handle.

Semi- structure:

I*t's* a type of **big data** that doesn't conform with a formal structure of data models. But it comes with some kinds of organizational tags or other markers that help to separate semantic elements, as well as, enforce hierarchies of fields and records within that data. You can think of JSON documents or XML files as this type of **big data**. The reason behind the existence of this category is that semi-structured data is significantly easier to analyze than unstructured data. A significant number of **big data** solutions and tools come with the ability of reading and processing XML files or JASON documents, reducing the complexity of the analyzing process.

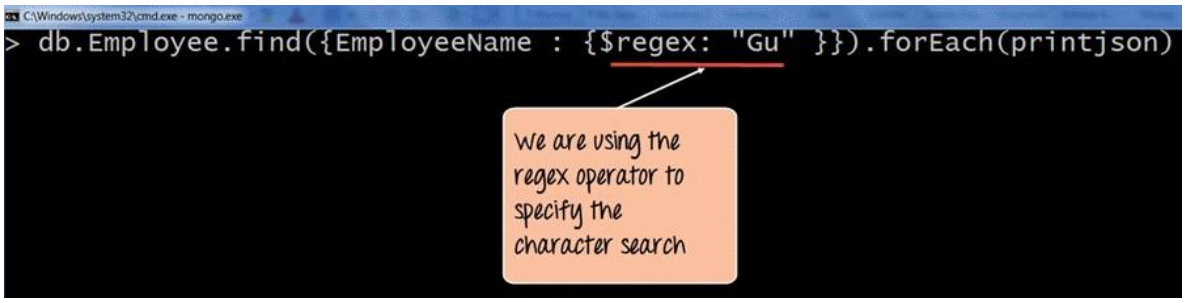
https://www.educba.com/mongodb-vs-sql/

db.collection.count()	Returns a count of the number of documents in a collection or a view.
db.collection.distinct()	Returns an array of documents that have distinct values for the specified field.

```
db.cities.find().sort({ "population": -1 })
```

Regular expression

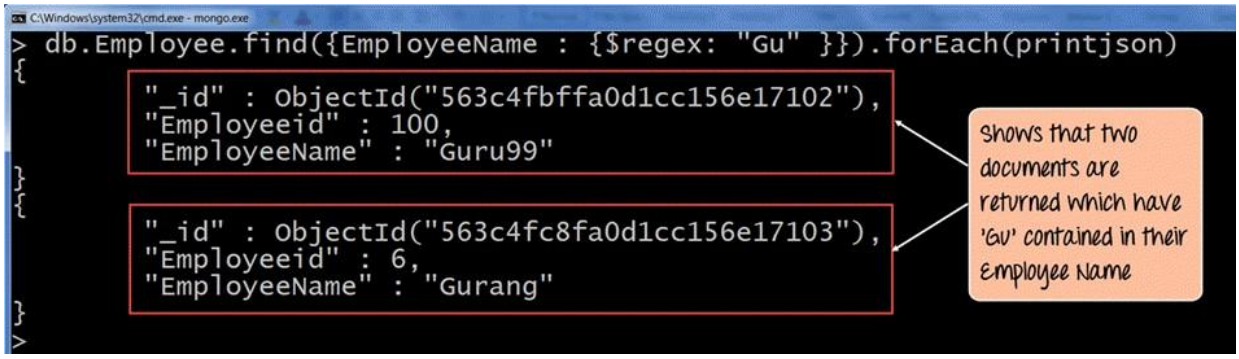
- Regular expressions are used for pattern matching, which is basically for finding strings within documents
- Sometimes when retrieving documents in a collection, you may not know exactly what the exact Field value to search for. Hence, one can use regular expressions to assist in retrieving data based on pattern matching search values.

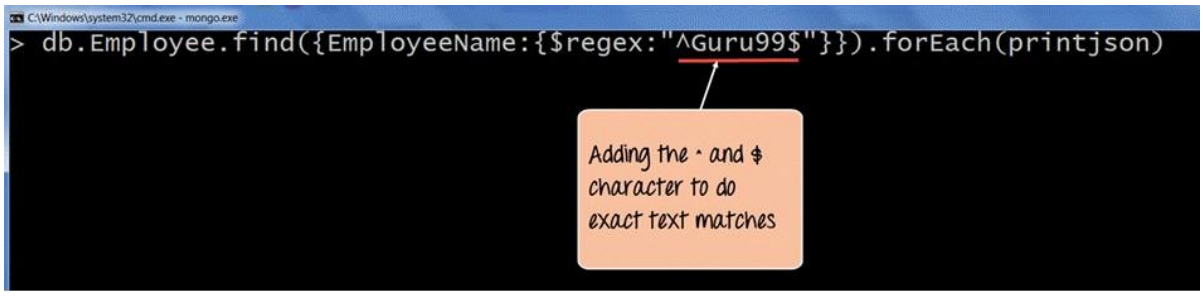


Code Explanation:

Here we want to find all Employee Names which have the characters ‘Gu’ in it. Hence, we specify the $regex operator to define the search criteria of ‘Gu’

The printjson is being used to print each document which is returned by the query in a better way.





Code Explanation:

Here in the search criteria, we are using the ^ and $ character.

The ^ is used to make sure that the string starts with a certain character, and $ is used to ensure that the string ends with a certain character. So when the code executes it will fetch only the string with the name “Guru99”.

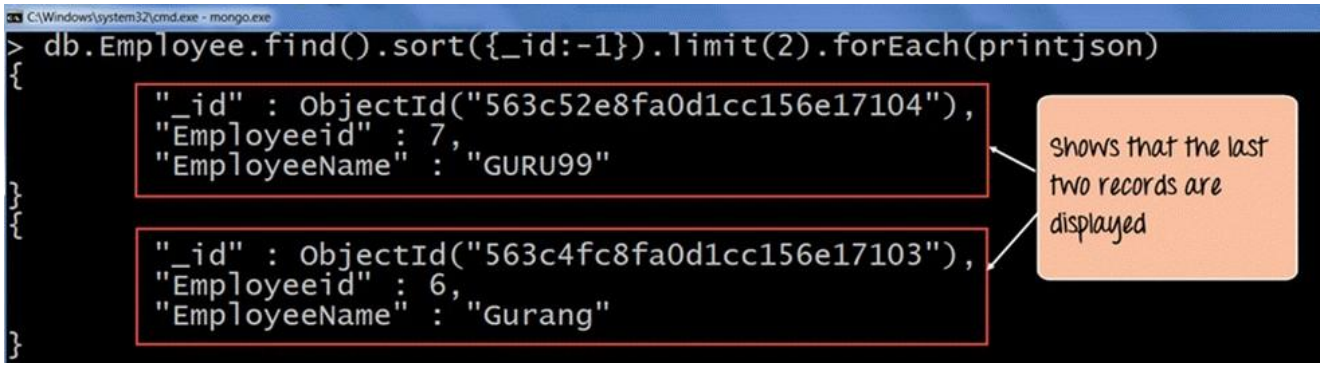
Pattern matching:

db.Employee.find({EmployeeName:{$regex:"Aj",$options:'i'}}).forEach(printjson)

The **$options** with **'I'** parameter (which means case insensitivity) specifies that we want to carry out the search no matter if we find the letters **'Aj' in lower or upper case.**

db.Employee.find({EmployeeName: /Aj/'}).forEach(printjson)

Limit and reverse order



INDEXING

db.mycol.createIndex({"age":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

“ok” : 1

}

DROP ONE AND MANY :

db.NAME_OF_COLLECTION.dropIndex({KEY:1})

db.NAME_OF_COLLECTION.dropIndexes({KEY1:1, KEY2, 1})

db.NAME_OF_COLLECTION.getIndexes()

CREATE INDEX

```
> db.studentgrades.createIndex(
...     {name: 1},
...     {name: "student name index"}
... )
{
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

FIND INDEX

```
> db.studentgrades.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1
                },
                "name" : "student name index"
        }
]
>
```

DROP INDEX AND FIND

```
> db.studentgrades.dropIndex("student name index")
{ "nIndexesWas" : 2, "ok" : 1 }
>
> db.studentgrades.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
>
```

```
> db.studentgrades.dropIndex({name:1})
{ "nIndexesWas" : 2, "ok" : 1 }
>
>
```

Single field index

These user-defined indexes use a single field in a document to create an index in an ascending or descending sort order (1 or -1). In a single field index, the sort order of the index key does not have an impact because MongoDB can traverse the index in either direction.

```
> db.studentgrades.createIndex({name: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

CONDITIONAL OPERATORS

In MongoDB the conditional operators are :

(>) greater than - $gt

(<) less than - $lt

(>=) greater than equal to - $gte

(<=) less than equal to - $lte

>db.testtable.find({age : {$gt : 22}})

>db.testtable.find({age : {$gte : 22}})

Select * from testtable where age >=22;

>db.testtable.find({age : {$lt : 19}})

>db.testtable.find({age : {$lte : 19}})

>db.testtable.find({age : {$lt :24, $gt : 17}})

LEGACY SYSTEM

A legacy system is outdated computing software and/or hardware that is still in use.

The system still meets the needs it was originally designed for, but doesn't allow for growth. What a legacy system does now for the company is all it will

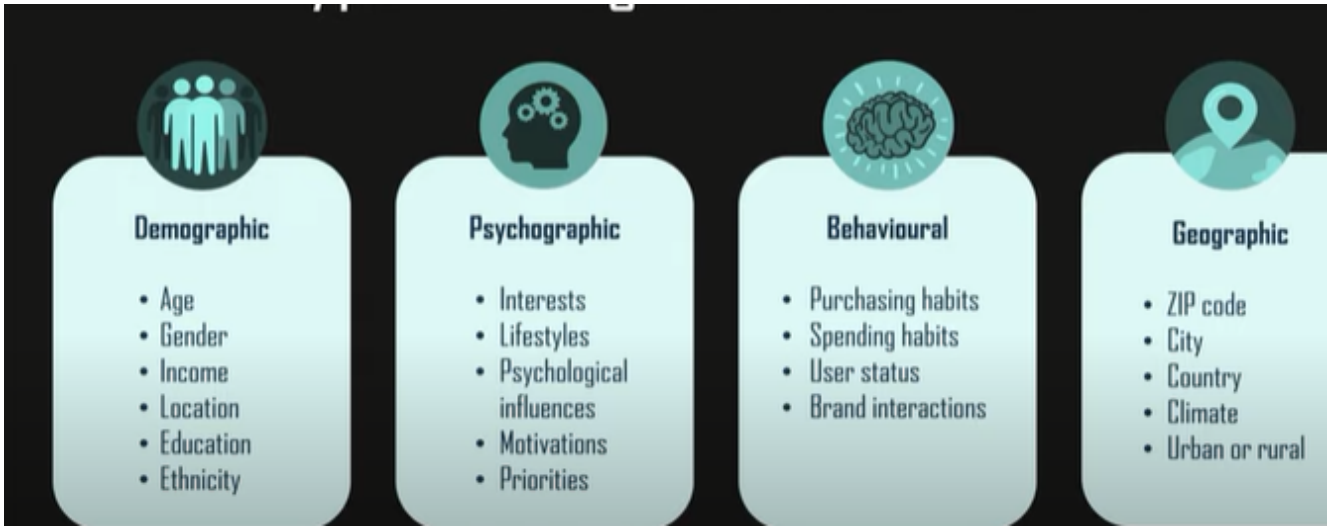
ever do. A legacy system's older technology won't allow it to interact with newer systems.

1. Maintenance is costly
2. Security gets weaker by the day
3. New systems don't integrate

Segmentation

- Insurance company
- Purchase their channel of choice
- Buying behavior when and how the customers were researching their channel of choice
- Identifying network of friends, influencers, opinion leaders.

1. Real time intelligence
2. Better return of investment on promotions
3. 12 to 14 % of uplift
4. Penetration of product increased



Data processing and stored

Data generated from organizations

Financial institute such as banks or stock market , hospitals whicn turns to be an input to ETL system and this would extract data and tranform this data taht it would convert to a proper format and then finally load this data to

database , now the end users can generate reports and perform analytics by acquiring this data.

ETL, which stands for extract, transform and load

ETL provides the foundation for data analytics and machine learning workstreams. Through a series of business rules, ETL cleanses and organizes data in a way which addresses specific business intelligence needs, like monthly reporting, but it can also tackle more advanced analytics, which can improve back-end processes or end user experiences. ETL is often used by an organization to:

- Extract data from legacy systems
- Cleanse the data to improve data quality and establish consistency
- Load data into a target database

Drawbacks to Traditional approach

1. Expensive -requires to invest a lot and some companies
2. Scalability -expanding the system
3. Time consumption system- takes time to process data and extract valuable information.

Big data technologies

can be categorized into four main types: data storage, data mining, data analytics, and data visualization

Data storage

- Apache Hadoop:

  Apache is the most widely used big data tool. It is an open-source software platform that stores and processes big data in a distributed computing environment across hardware clusters. This distribution allows for faster data processing. The framework is designed to reduce bugs or faults, be scalable, and process all data formats.

- MongoDB: MongoDB is a NoSQL database that can be used to store large volumes of data. Using key-value pairs (a basic unit of data), MongoDB categorizes documents into collections. It is written in C, C++, and JavaScript, and is one of the most popular big data databases because it can manage and store unstructured data with ease.

Data mining

- Rapidminer: Rapidminer is a data mining tool that can be used to build predictive models. It draws on these two roles as strengths, of

processing and preparing data, and building machine and deep learning models. The end-to-end model allows for both functions to drive impact across the organization [3].

- Presto: Presto is an open-source query engine that was originally developed by Facebook to run analytic queries against their large datasets. Now, it is available widely. One query on Presto can combine data from multiple sources within an organization and perform analytics on them in a matter of minutes.

Data analytics

- Apache Spark: Spark is a popular big data tool for data analysis because it is fast and efficient at running applications. It is faster than Hadoop because it uses random access memory (RAM) instead of being stored and processed in batches via MapReduce [4]. Spark supports a wide variety of data analytics tasks and queries.
- Splunk: Splunk is another popular big data analytics tool for deriving insights from large datasets. It has the ability to generate graphs, charts, reports, and dashboards. Splunk also enables users to incorporate artificial intelligence (AI) into data outcomes.

Data visualization

- Tableau: Tableau is a very popular tool in data visualization because its drag-and-drop interface makes it easy to create pie charts, bar charts,

box plots, Gantt charts, and more. It is a secure platform that allows users to share visualizations and dashboards in real time.

- Looker: Looker is a business intelligence (BI) tool used to make sense of big data analytics and then share those insights with other teams. Charts, graphs, and dashboards can be configured with a query, such as monitoring weekly brand engagement through social media analytics.

Mongo practicals

Example	RDBMS Equivalent
db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
db.mycol.find({"likes": {$lt:50}}).pretty()	where likes < 50
db.mycol.find({"likes": {$lte:50}}).pretty()	where likes <= 50
db.mycol.find({"likes": {$gt:50}}).pretty()	where likes > 50
db.mycol.find({"likes": {$gte:50}}).pretty()	where likes >= 50
db.mycol.find({"likes": {$ne:50}}).pretty()	where likes != 50
db.mycol.find({"name":{$in: ["Raj", "Ram", "Raghu"]}}).pretty()	Where name matches any of the value in :["Raj", "Ram", "Raghu"]

AND:

db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

OR:

```
db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB
Overview"}]}).pretty()
```

AND & OR TOGETHER

```
db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},
  {"title": "MongoDB Overview"}]}).pretty()
```

NOT :

```
db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )
```

UPDATE :

```
db.mycol.update({'title':'MongoDB Overview'},
{$set:{'title':'New MongoDB Tutorial'}})
```

UPDATE MULTIPLE :

To update multiple documents, you need to set a parameter 'multi' to true.

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

UPDATE MANY

```
db.empDetails.updateMany(
        {Age:{ $gt: "25" }},
        { $set: { Age: '00'}}
```

REMOVE :

```
db.mycol.remove({'title':'MongoDB Overview'})
```

REMOVE ALL :

db.mycol.remove({}) # LIKE TRUNCATE IN SQL

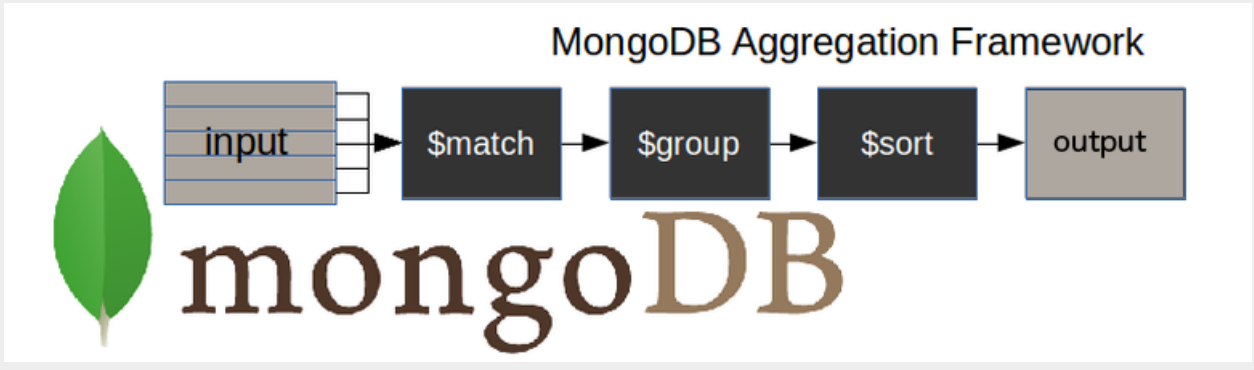
Regular expression:

db.posts.find({tags:{$regex:"tutorial"}})

if the regular expression is a prefix expression, all the matches are meant to start with a certain string character.
For e.g., if the regex expression is ^jan,
then the query has to search for only those strings that begin with jan.

DISTINCT :

```
> db.studentsMark.distinct("name")
[ "bruce", "bucky", "groot", "loki", "nick", "steve", "tony" ]
> db.studentsMark.distinct("age")
[ 17, 19, 27, 37 ]
```



- `$match` stage – filters those documents we need to work with, those that fit our needs
- `$group` stage – does the aggregation job
- `$sort` stage – sorts the resulting documents the way we require (ascending or descending)

EMPTY QUERY

```
db.getCollection('persons').find({})
```

persons 0.616 sec.

```
/* 1 */
{
    "_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
    "index" : 0,
    "name" : "Aurelia Gonzales",
    "isActive" : false,
    "registered" : ISODate("2015-02-11T04:22:39.000Z"),
    "age" : 20,
```

NOT IN & IN OPERATOR INCLUDING COUNT()

```
db.getCollection('persons')
  .find({"age": {"$in": [21 , 22]}})
```

```
db.getCollection('persons')
  .find({"age": {"$nin": [21 , 22]}})
  .count()
```

EMBEDDED DOC

```
"company" : {
   "title" : "SHEPARD",
   "email" : "carmellamorse@shepard.com",
   "phone" : "+1 (829) 478-3744",
   "location" : {
      "country" : "USA",
      "address" : "379 Tabor Court"
   }
}
```

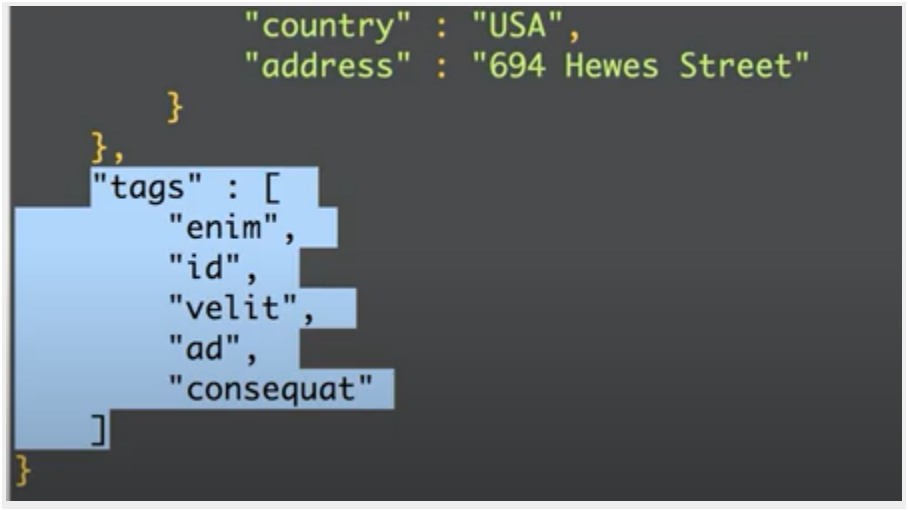
```
{
        "_id" : ObjectId("5f7c0b90b44f6e405e4e61c7"),
        "name" : "Siya",
        "address" : {
                "city" : "Pune",
                "state" : "MH",
                "country" : "India"
        }
}
{
        "_id" : ObjectId("5f7e96b4dfeaabf21b6746c7"),
        "name" : "Raj",
        "cname" : "SE"
}
```

Note

Fields accessed with dot notation (.) MUST be inside quotation marks

```
{ "company.title": "SHEPARD" }
{ "company.location.address: "379 Tabor Court"}
```

ARRAY


"country" : "USA",
"address" : "694 Hewes Street"
}
},
"tags" : [
"enim",
"id",
"velit",
"ad",
"consequat"
]
}


db.getCollection('persons')
.find({tags: {$all: ["id" , "ad"]}})
persons 0.206 sec.
/* 1 */
{
"_id" : ObjectId("5ad4cbde2edbf6ddeec717
"index" : 0,
"name" : "Aurelia Gonzales",
"isActive" : false,
"registered" : ISODate("2015-02-11T04:22
"age" : 20,
"gender" : "female",
"eyeColor" : "green",

REGEX :

Filter using Regular Expression

```
{<fieldName>: {"$regex": /pattern/<options> }}
{<fieldName>: {"$regex": /pattern/ , $options: "<options>"}}
```

Examples

```
{ city: {"$regex": /ton/i}}
{ tags: {"$regex": /^ad$/ , $options: "i"}}
{ state: {"$regex": "ca"}}
```

```
db.getCollection('persons')
  .find({name: {$regex: /rel/i}})
```

FILTERING QUERY

```
db.getCollection('persons')
  .find({} , {name: 1, age: 1, eyeColor: 1})
```

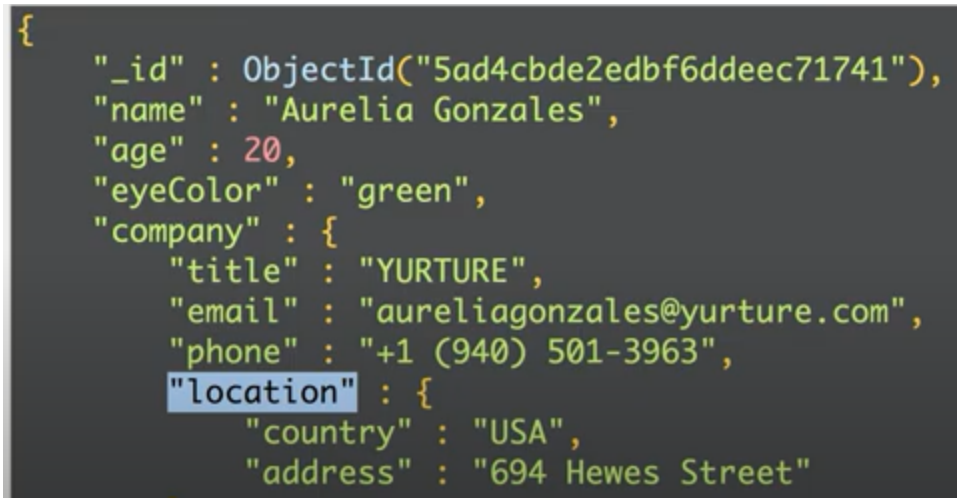
persons 0.216 sec.

```
/* 1 */
{
    "_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
    "name" : "Aurelia Gonzales",
    "age" : 20,
    "eyeColor" : "green"
}
```

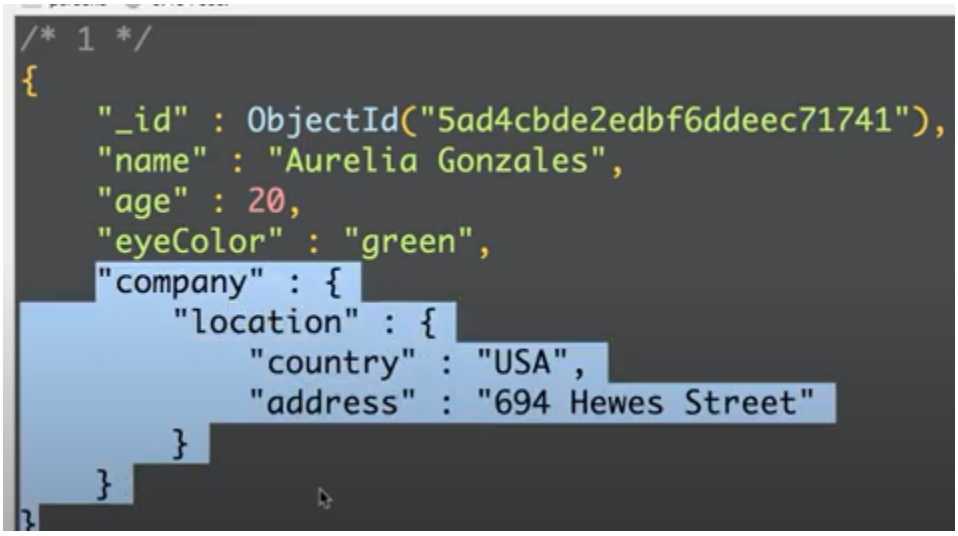
```
db.getCollection('persons')
  .find({} , {name: 1, age: 1, eyeColor: 1, company: 1})
```

```
/* 1 */
{
    "_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
    "name" : "Aurelia Gonzales",
    "age" : 20,
    "eyeColor" : "green",
    "company" : {
        "title" : "YURTURE",
        "email" : "aureliagonzales@yurture.com",
        "phone" : "+1 (940) 501-3963",
        "location" : {
            "country" : "USA",
            "address" : "694 Hewes Street"
```

```
db.getCollection('persons')
  .find({} , {name: 1, age: 1, eyeColor: 1, "company.location": 1})
```


{
"_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
"name" : "Aurelia Gonzales",
"age" : 20,
"eyeColor" : "green",
"company" : {
"title" : "YURTURE",
"email" : "aureliagonzales@yurture.com",
"phone" : "+1 (940) 501-3963",
"location" : {
"country" : "USA",
"address" : "694 Hewes Street"

OUTPUT


/* 1 */
{
"_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
"name" : "Aurelia Gonzales",
"age" : 20,
"eyeColor" : "green",
"company" : {
"location" : {
"country" : "USA",
"address" : "694 Hewes Street"
}
}
}

```
db.getCollection('persons')
  .find({} , {name: 1, age: 1, eyeColor: 1, _id: 0})
```

persons 0.208 sec.

```
/* 1 */
{
    "name" : "Aurelia Gonzales",
    "age" : 20,
    "eyeColor" : "green"
}

/* 2 */
{
    "name" : "Kitty Snow",
    "age" : 38,
    "eyeColor" : "blue"
}
```

```
db.getCollection('persons')
  .find({} , {name: 0, age: 0})
```

persons 0.596 sec.

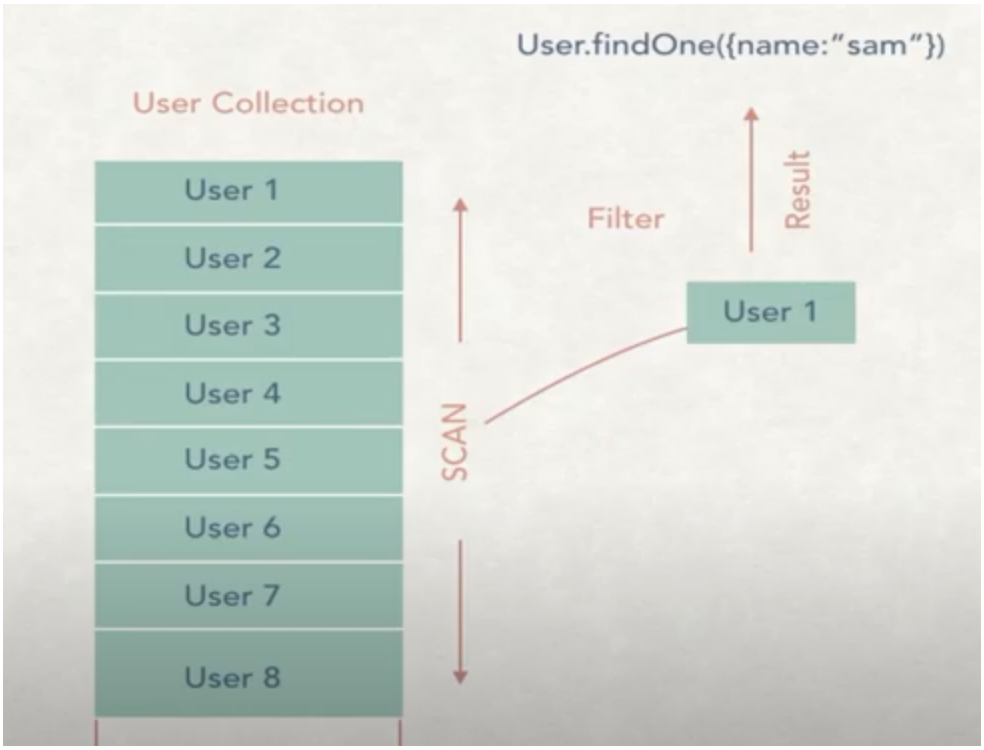
```
/* 1 */
{
    "_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
    "index" : 0,
    "isActive" : false,
    "registered" : ISODate("2015-02-11T04:22:39.00
    "gender" : "female",
    "eyeColor" : "green",
    "favoriteFruit" : "banana",
    "company" : {
        "title" : "YURTURE",
        "email" : "aureliagonzales@yurture.com",
        "phone" : "+1 (940) 501-3963",
```

```
db.getCollection('persons')
  .findOne({} , {name: 1, age: 1})
```

0.253 sec.

```
/* 1 */
{
    "_id" : ObjectId("5ad4cbde2edbf6ddeec71741"),
    "name" : "Aurelia Gonzales",
    "age" : 20
}
```

INDEXING



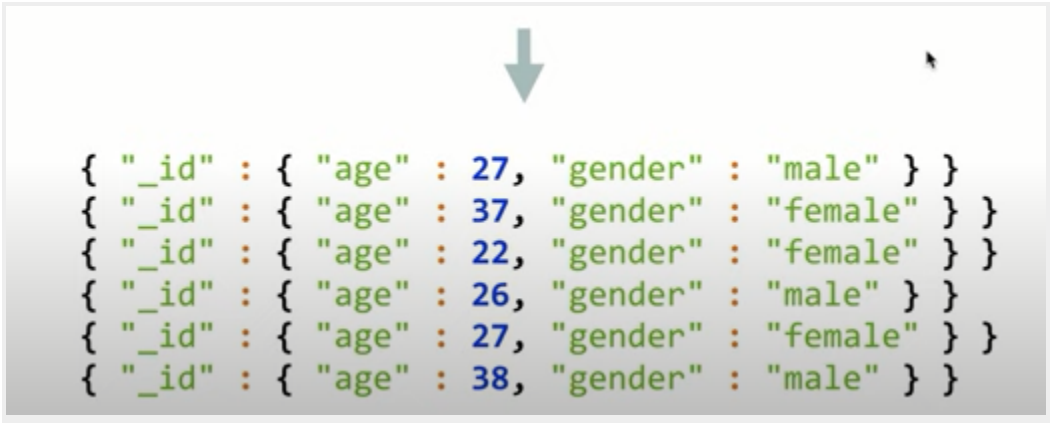

```
> db.student.createIndex({rollno:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.student.ensureIndex({age:-1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

```
> db.student.getIndexes()
```


```
> //compound_key_index
> db.student.createIndex({fname:1,lname:1})
```

MULTIPLE AGGREGATION

```
db.persons.aggregate([
  { $group: { _id: {age: "$age", gender: "$gender"} } }
])
```



```
{ "_id" : { "age" : 27, "gender" : "male" } }
{ "_id" : { "age" : 37, "gender" : "female" } }
{ "_id" : { "age" : 22, "gender" : "female" } }
{ "_id" : { "age" : 26, "gender" : "male" } }
{ "_id" : { "age" : 27, "gender" : "female" } }
{ "_id" : { "age" : 38, "gender" : "male" } }
```

```
db.persons.aggregate([
  { $match: { favoriteFruit: "banana" } },
  { $group: { _id: {age: "$age", eyeColor: "$eyeColor"} } }
])
```

MONGO DB + PYTHON

PYTHON FILE- Connection of python and mongo db (pymongo used)

```
import pymongo
from pymongo import MongoClient
client = MongoClient()
client = MongoClient('mongodb://localhost:27017')
db = client['test-database']
collection = db['test-collection']
record=[{
        'empno':123,
        'name':'abc',
        'sal':20000
        }]
collection.insert_many(record)
```

https://pymongo.readthedocs.io/en/stable/tutorial.html

pymongo.readthedocs.io/en/stable/

.3 documentation » PyMongo 4.3.3 Documentation

ontents
3.3
on
lelp
Requests /
ing
is
ntation
nd tables
ngo with
las

PyMongo 4.3.3 Documentation

Overview ¶

PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python. This documentation attempts to explain everything you need to know to use **PyMongo**.

Installing / Upgrading
: Instructions on how to get the distribution.

Tutorial
: Start here for a quick overview.

Examples
: Examples of how to perform specific tasks.

Using PyMongo with MongoDB Atlas
: Using PyMongo with MongoDB Atlas.

```
>>> import pymongo
```

This tutorial also assumes that a MongoDB instance is running on the default host and port. Assuming you have downloaded and installed MongoDB, you can start it like so:

```
$ mongod
```

Making a Connection with MongoClient

The first step when working with **PyMongo** is to create a `MongoClient` to the running **mongod** instance. Doing so is easy:

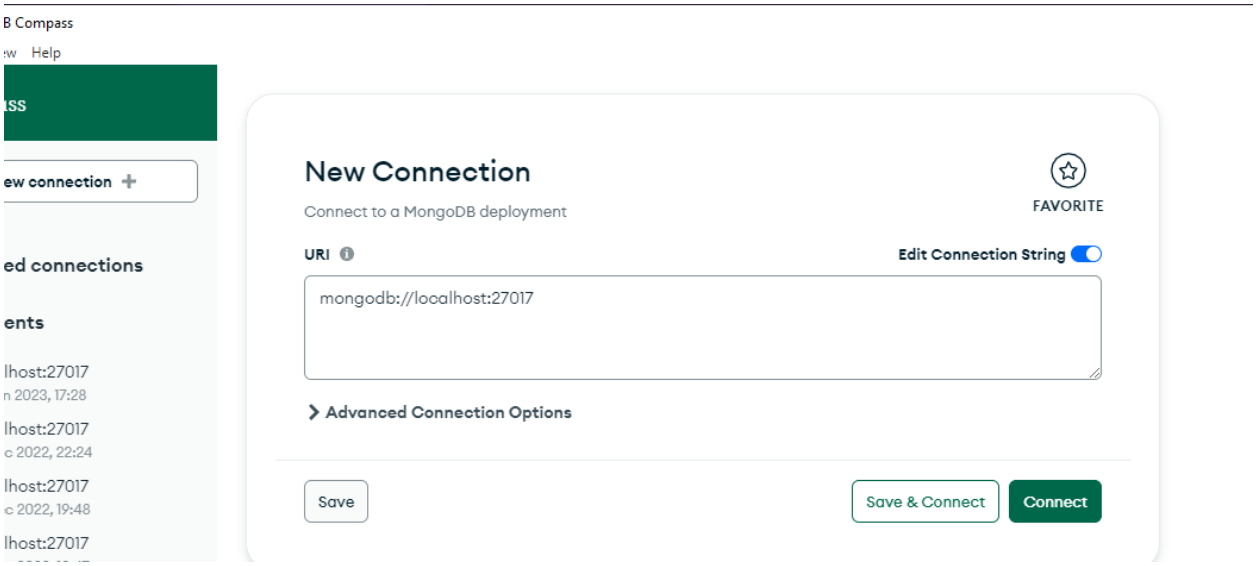
```
>>> from pymongo import MongoClient
>>> client = MongoClient()
```

The above code will connect on the default host and port. We can also specify the host and port explicitly, as follows:

```
>>> client = MongoClient('localhost', 27017)
```

Or use the MongoDB URI format:

```
>>> client = MongoClient('mongodb://localhost:27017/')
```



https://www.thecodebuzz.com/mongo-is-not-recognized-as-an-internal-or-external-command/

Retrieval

```
import pymongo

if __name__=="__main__":

    print("pymongo Example")

    client=pymongo.MongoClient("mongodb://localhost:27017/")

    print(client)

    db=client.college

    collection=db.student

#retrieving all files from mongodb
```

```
#alldocs=collection.find({'name':'shabana'})
#for item in alldocs:
 #   print(item)
 #exclude some fields using 0 and 1 to display
alldocs=collection.find({'name':'shabana'},{'name':1,'_id':0})
for item in alldocs:
    print(item)
```

Code 2# creating collection and insert doc

```
import pymongo
if __name__=="__main__":
    print("pymongo Example")
    client=pymongo.MongoClient("mongodb://localhost:27017/")
    print(client)
    db=client.college
    collection=db.student
    # insertion of multiple data
```

```
dictionary=[{'name':'shabana','marks':80},
   {'name':'nancy','marks':80,'result':'first class'},
   {'name':'shamaila','result':'first class'}]
collection.insert_many(dictionary)
#Retrieving data from mongodb
#one=collection.find_one({'name':'shabana'})        working
#print(one)
# Retrieving
```

Delete

```
import pymongo
if __name__=="__main__":
```

```
    print("pymongo Example")
    client=pymongo.MongoClient("mongodb://localhost:27017/")

    db=client.college
    collection=db.student
    #deleting one record
    rec={'name':'shamaila'}
    #collection.delete_one(rec)          # working for single deletion
    #deleting multiple record
    collection.delete_many(rec)
```

UPDATE

```
import pymongo
if __name__=="__main__":
    print("pymongo Example")
```

```
client=pymongo.MongoClient("mongodb://localhost:27017/")

print(client)



db=client.college

collection=db.student

prev={'name':'sana'}

neww={'$set':{'name':'shabana'}}

#collection.update_one(prev,neww) # for updation in one data only

modi=collection.update_many(prev,neww)

print(modi.modified_count)
```

Theory

Visibility

Consider a manufacturing company that has R&D, engineering, and manufacturing departments dispersed geographically. If the data is accessible across all these departments and can be readily integrated, it can not only reduce the search and processing time but will also help in improving the product quality according to the present needs.

Discover and Analyze Information

Organizations are capturing detailed data on inventories, employees, and customers. Using all of this data, they can discover and analyze new

information and patterns; as a result. This information and knowledge can be used to improve processes and performance.

Segmentation and Customizations

used in the social sector to accurately segment populations and target benefit schemes for specific needs.

Segmentation of customers based on various parameters can aid in targeted marketing campaigns and tailoring of products to suit the needs of customers.

Aiding Decision Making

Big data can substantially minimize risks, improve decision making , and uncover valuable insights.

Automated fraud alert systems in credit card processing and

automatic fine-tuning of inventory are examples of systems

that aid or automate decision-making based on big data analytics.

Innovation

Example, real-time data from machines and vehicles can be analyzed to provide insight into maintenance schedules; wear and tear on machines can be monitored to make more resilient machines; fuel consumption monitoring can lead to higher efficiency engines.

Real-time traffic information is already making life easier for commuters by providing them options to take alternate routes.

It not only provides the opportunity for organizations to strengthen existing business by making informed decisions, it also helps in identifying new opportunities.

- Consistency means that the data remains consistent

  after any operation is performed that changes the data, and that all users or clients accessing the application see the same updated data.

- Availability means that the system is always available.
- Partition Tolerance means that the system will continue to function even if it is partitioned into groups of servers that are not able to communicate with one another.

Big Data Challenges

-Policies and Procedures

Data privacy, security, intellectual property, and protection are of immense importance to organizations.

-Access to Data

Data about a product or service is available on Facebook, Twitter feeds, reviews, and blogs, so how does the product owner access this data from various sources owned by various providers?

Technology and Techniques

Lack of experienced resources in newer technologies is a challenge that any big data project has to manage.

Data Storage

Legacy systems use big servers and NAS and SAN systems to store the data. As the data increases, the server size and the backend storage size has to be increased. Traditional legacy systems typically work in a scale- up model where more and more compute, memory, and storage needs to be added to a server to meet the increased data needs. Hence the processing time increases exponentially, which defeats the other important requirement of big data, which is velocity.

Data Processing

The algorithms in legacy systems are designed to work with **structured data such as strings and integers**. They are also **limited by the size** of data. Thus, legacy systems are **not capable of handling the processing of unstructured data,** huge volumes of such data, and the speed at which the processing needs to be performed.

As a result, to capture value from big data, we need to deploy newer technologies in the field of storing, computing, and retrieving, and we need new techniques for analyzing the data.

ACID

Atomic implies either all changes of a transaction are applied completely or not applied at all.

• **Consistent** means the data is in a consistent state after the transaction is applied. This means after a transaction is committed, the queries fetching a particular data will see the same result.

• **Isolated** means the transactions that are applied to the same set of data are independent of each other. Thus, one transaction will not interfere with another transaction.

• **Durable** means the changes are permanent in the system and will not be lost in case of any failures.

Some examples of big data use cases that are a good fit for NoSQL databases are the following:

• Social Network Graph : Whose identity is connected to whom? Whose post should be visible on the user's wall or homepage on a social network site?

• Search and Retrieve : Search all relevant pages with a particular keyword ranked by the number of times a keyword appears on a page.

The BASE

Basically Available means the system will be available in terms of the CAP theorem.

• Soft state indicates that even if no input is provided to the system, the state will change over time. This is in accordance with eventual consistency.

• Eventual consistency means the system will attain consistency in the long run, provided no input is sent to the system during that time.

Disadvantages of NoSQL

In addition to the above mentioned advantages, there are many impediments that you need to be aware of before you start developing applications using these platforms.

• Maturity : Most NoSQL databases are pre-production versions with key features that are still to be implemented. Thus, when deciding on a NoSQL database, you should analyze the product properly to ensure the features are fully implemented and not still on the To-do list .

• Support : Support is one limitation that you need to consider. Most NoSQL databases are from start-ups which were open sourced. As a result, support is

very minimal as compared to the enterprise software companies and may not have global reach or support resources.

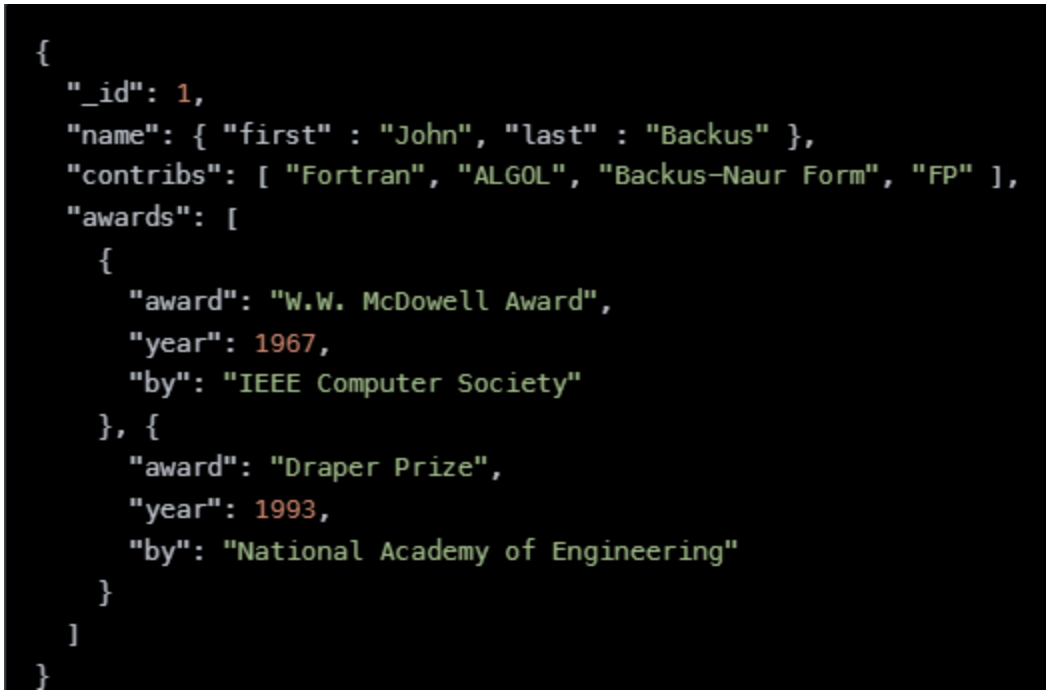
• Limited Query Capabilities : Since NoSQL databases are generally developed to meet the scaling requirement of the web-scale applications, they provide limited querying capabilities. A simple querying requirement may involve significant programming expertise.

• Administration : Although NoSQL is designed to provide a no-admin solution, it still requires skill and effort for installing and maintaining the solution.

• Expertise : Since NoSQL is an evolving area, expertise on the technology is limited in the developer and administrator community.

JSON and BSON

All the basic data types (such as strings, numbers, Boolean values, and arrays) are supported by JSON.



Binary JSON (BSON)

MongoDB stores the JSON document in a binary-encoded format. This is termed as BSON. The BSON data model is an extended form of the JSON data model.

```
{"hello": "world"} →
\x16\x00\x00\x00           // total document size
\x02                       // 0x02 = type String
hello\x00                  // field name
\x06\x00\x00\x00world\x00  // field value
\x00                       // 0x00 = type EOO ('end of object')

{"BSON": ["awesome", 5.05, 1986]} →
\x31\x00\x00\x00
 \x04BSON\x00
 \x26\x00\x00\x00
 \x02\x30\x00\x08\x00\x00\x00awesome\x00
 \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
 \x10\x32\x00\xc2\x07\x00\x00
 \x00
 \x00
```

It supports embedding of arrays and objects within other arrays, and also enables MongoDB to reach inside the objects to build indexes and match objects against queried expressions, both on top-level and nested BSON keys.

Identifier (_id)

You need to have a key that uniquely identifies each document within a collection.

This is referred to as **_id** in MongoDB.

The _id field is a 12-byte Field of BSON type made up of several 2-4 byte chains and is the unique identifier/naming convention MongoDB uses

If you have not explicitly specified any value for a key, a unique value will be automatically generated and assigned to it by MongoDB. This key value is immutable.

```
// "Document collections" - "HTMLPage" document
{
_id: 1,
title: "Hello",
type: "HTMLpage",
text: "<html>Hi..Welcome to my world</html>"
}
...
// Document collection also has a "Picture" documen
{
_id: 3,
title: "Family Photo",
type: "JPEG",
sizeInMB: 10,........
}
```

```
> show dbs
```

At any point, help can be accesses using the `help()` command.

```
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        sh.help()                    sharding helpers
        rs.help()                    replica set helpers
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce
        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
.............
        exit                         quit the mongo shell
```

Table 6-1. *SQL and MongoDB Terminology*

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index
Joins within table	Embedding and referencing
Primary Key: A column or set of columns can be specified	Primary Key: Automatically set to _id field

```
> use mydbpoc
switched to db mydbpoc
>
```

This switches the context from `test` to `MYDBPOC`. The same can be confirmed using the `db` command.

```
> db
mydbpoc
```

Projector

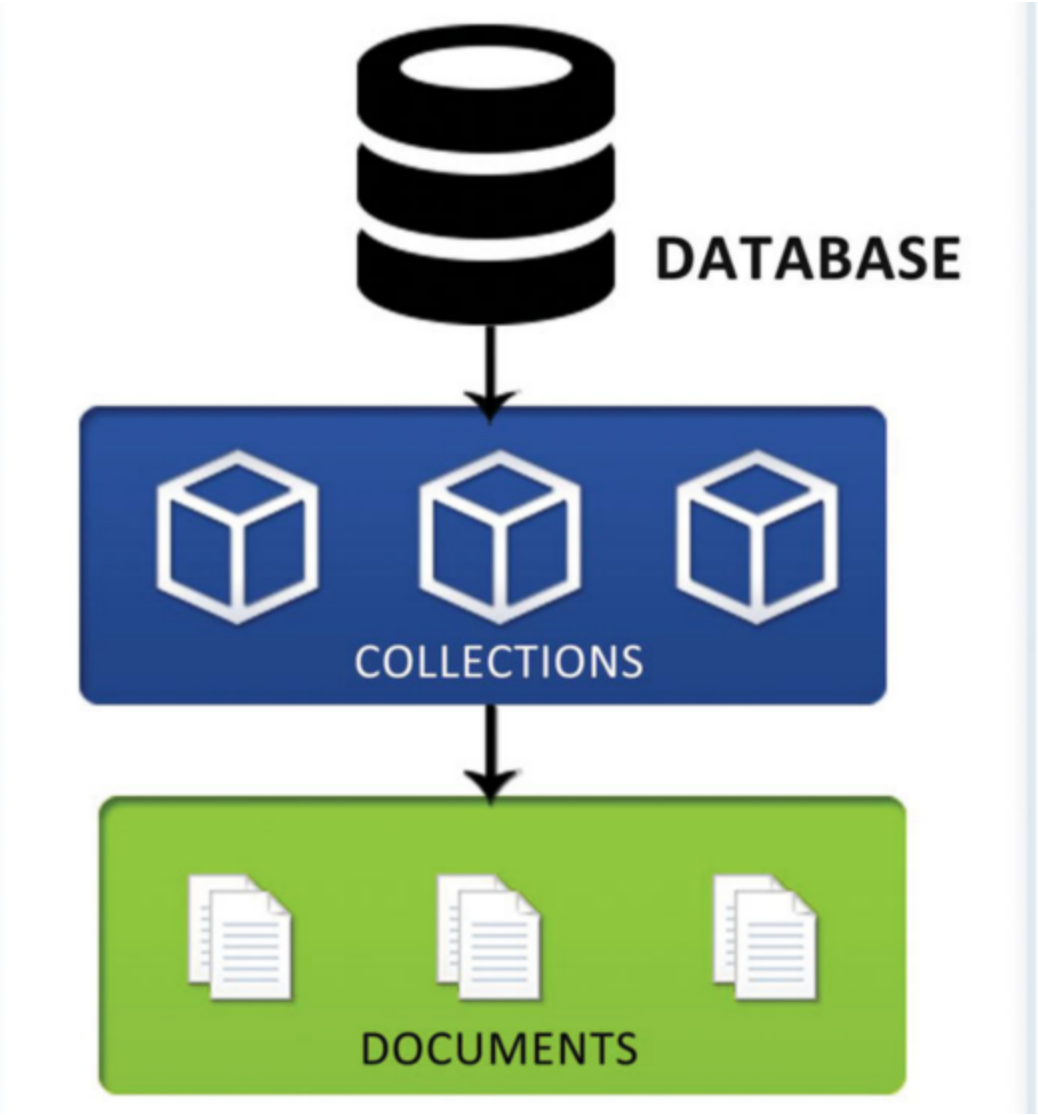
You have seen how to use selectors to filter out documents within the collection. In the above example, the `find()` command returns all fields of the documents matching the selector.

Let's add a projector to the query document where, in addition to the selector, you will also mention specific details or fields that need to be displayed.

Suppose you want to display the first name and age of all female employees. In this case, along with th selector, a projector is also used.

Execute the following command to return the desired result set:

```
> db.users.find({"Gender":"F"}, {"Name":1,"Age":1})
{ "_id" : ObjectId("52f4a826958073ea07e15071"), "Name" : "Test User", "Age" : 45 }
..........
```



Data model:

Documents within a collection can have different (or same) sets

of fields. This affords you more flexibility when dealing with data.

A MongoDB deployment can have many databases. Each database is a set of collections. Collections are similar to the concept of tables in SQL;

However, they are schemaless. Each collection can have multiple documents. In an RDBMS system, since the table structures and the data types for each column are fixed, you can only add data of a particular data type in a column. In MongoDB, a collection is a collection of documents where data is stored as key-value pairs.

{"Name": "ABC", "Phone": ["1111111", "222222"] }

Replication

Replication provides redundancy and increases data availability.
With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.
In some cases, replication can provide increased read capacity as clients can send read operations to different servers.

Maintaining copies of data in different data centers can increase data locality and availability for distributed applications.

You can also maintain additional copies for dedicated purposes, such as disaster recovery, reporting, or backup.

Master/Slave Replication

MongoDB replication is **the process of creating a copy of the same data set in more than one MongoDB server**.

- only for more than 50 node replications.
- one master and a number of slaves that replicate the data from the master.
- there's no restriction on the number of slaves within a cluster.
- thousands of slaves will overburden the master node, so in practical scenarios it's better to have less than a dozen slaves. In addition, this type of replication doesn't automate failover and provides less redundancy.

There are two main reasons behind a slave becoming out of sync:

• The slave shuts down or stops and restarts later. During this time, the oplog may have deleted the log of operations required to be applied on the slave.

• The slave is slow in executing the updates that are available from the master.

Replica Set

Replica sets are basically a type of master-slave replication but they provide automatic failover.

A replica set has one master, which is termed as primary, and multiple slaves, which are termed as secondary in the replica set context;

Sharding

One of the important factors when designing the application model is whether to partition the data or not. This is implemented using sharding in MongoDB. Sharding is also referred to as partitioning of data. In MongoDB, a collection is partitioned with its

documents distributed across clusters of machines, which are referred to as **shards.**

DATABASE RESTORE

```
C:\Users\DELL>mongo
MongoDB shell version v5.0.5
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("204fbb7a-6006-4849-8b0d-a6438f30a941") }
MongoDB server version: 5.0.5
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be r
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
---
The server generated these startup warnings when booting:
        2023-01-15T22:25:24.010+05:30: Access control is not enabled for the database. Read and write a
 configuration is unrestricted
```

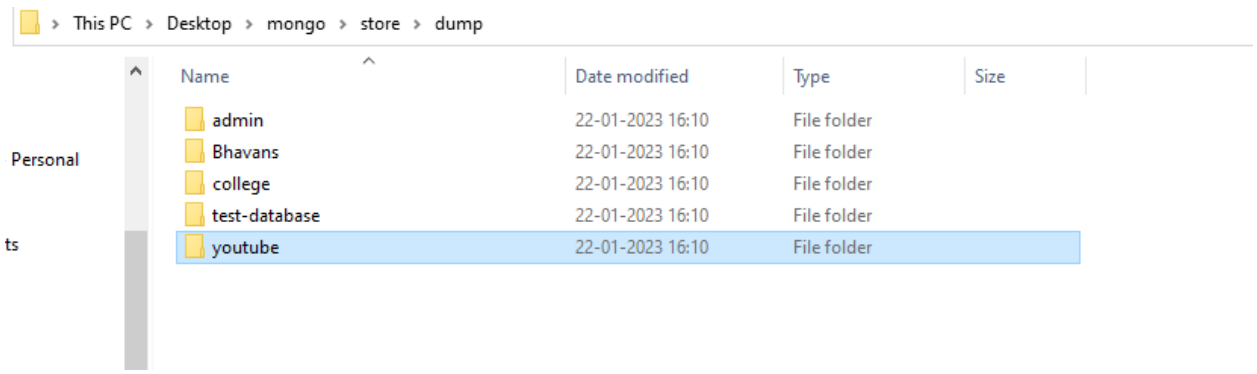
```
> show dbs
Bhavans        0.000GB
admin          0.000GB
college        0.000GB
config         0.000GB
local          0.000GB
test-database  0.000GB
> use youtube
switched to db youtube
> db.mychannel.insert({name:'rapper'})
WriteResult({ "nInserted" : 1 })
> db.tv.insert({name:'zee'})
WriteResult({ "nInserted" : 1 })
> show collection
uncaught exception: Error: don't know how to show [collection] :
shellHelper.show@src/mongo/shell/utils.js:1211:11
shellHelper@src/mongo/shell/utils.js:838:15
@(shellhelp2):1:1
> show collections
mychannel
tv
> show collections
mychannel
tv
```

```
> db.dropDatabase()
{ "ok" : 1 }
> show collections
> show dbs
Bhavans        0.000GB
admin          0.000GB
college        0.000GB
config         0.000GB
local          0.000GB
test-database  0.000GB
> show dbs
Bhavans        0.000GB
admin          0.000GB
college        0.000GB
config         0.000GB
local          0.000GB
test-database  0.000GB
youtube        0.000GB
> show collections
mychannel
tv
>
```

Command Prompt

```
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>cd C:\Users\DELL\Desktop\mongo\store

C:\Users\DELL\Desktop\mongo\store>mongodump
2023-01-22T16:10:02.350+0530    writing admin.system.version to dump\admin\system.version.bson
2023-01-22T16:10:02.356+0530    done dumping admin.system.version (1 document)
2023-01-22T16:10:02.359+0530    writing Bhavans.Students to dump\Bhavans\Students.bson
2023-01-22T16:10:02.361+0530    writing college.student to dump\college\student.bson
2023-01-22T16:10:02.362+0530    writing test-database.test-collection to dump\test-database\test-collection.bson
2023-01-22T16:10:02.364+0530    writing youtube.mychannel to dump\youtube\mychannel.bson
2023-01-22T16:10:02.369+0530    done dumping Bhavans.Students (3 documents)
2023-01-22T16:10:02.371+0530    done dumping college.student (3 documents)
2023-01-22T16:10:02.373+0530    writing youtube.tv to dump\youtube\tv.bson
2023-01-22T16:10:02.375+0530    done dumping test-database.test-collection (1 document)
2023-01-22T16:10:02.376+0530    done dumping youtube.mychannel (1 document)
2023-01-22T16:10:02.379+0530    done dumping youtube.tv (1 document)

C:\Users\DELL\Desktop\mongo\store>mongorestore
2023-01-22T16:12:12.332+0530    using default 'dump' directory
2023-01-22T16:12:12.340+0530    preparing collections to restore from
2023-01-22T16:12:12.360+0530    reading metadata for Bhavans.Students from dump\Bhavans\Students.metadata.json
2023-01-22T16:12:12.365+0530    reading metadata for college.student from dump\college\student.metadata.json
2023-01-22T16:12:12.371+0530    reading metadata for test-database.test-collection from dump\test-database\test-collection.metada
2023-01-22T16:12:12.374+0530    reading metadata for youtube.mychannel from dump\youtube\mychannel.metadata.json
2023-01-22T16:12:12.378+0530    reading metadata for youtube.tv from dump\youtube\tv.metadata.json
2023-01-22T16:12:12.387+0530    restoring to existing collection Bhavans.Students without dropping
2023-01-22T16:12:12.388+0530    restoring Bhavans.Students from dump\Bhavans\Students.bson
2023-01-22T16:12:12.391+0530    restoring to existing collection college.student without dropping
2023-01-22T16:12:12.404+0530    restoring college.student from dump\college\student.bson
2023-01-22T16:12:12.405+0530    restoring to existing collection test-database.test-collection without dropping
```

```
2023-01-22T16:12:12.545+0530    restoring youtube.tv from dump\youtube\tv.bson
2023-01-22T16:12:12.561+0530    finished restoring youtube.tv (1 document, 0 failures)
2023-01-22T16:12:12.561+0530    no indexes to restore for collection test-database.test-collection
2023-01-22T16:12:12.563+0530    no indexes to restore for collection college.student
2023-01-22T16:12:12.564+0530    no indexes to restore for collection youtube.mychannel
2023-01-22T16:12:12.566+0530    no indexes to restore for collection youtube.tv
2023-01-22T16:12:12.567+0530    no indexes to restore for collection Bhavans.Students
2023-01-22T16:12:12.568+0530    2 document(s) restored successfully. 7 document(s) failed to restore.

C:\Users\DELL\Desktop\mongo\store>
```



In order to make MongoDB high performance and fast, certain features commonly available in RDBMS systems are not available in MongoDB.

MongoDB is a document-oriented DBMS where data is stored as documents.

It does not support JOINs, and it does not have fully generalized transactions.

However, it does provide support for secondary indexes, it enables users to query using query documents, and it provides support for atomic updates at a per document level.

It provides a replica set, which is a form of master-slave replication with automated failover, and it has built-in horizontal scaling.

Regular Expressions

```
> db.students.insert({Name:"Student1", Age:30, Gender:"M", Class: "Biology", Score:90})
> db.students.insert({Name:"Student2", Age:30, Gender:"M", Class: "Chemistry", Score:90})
> db.students.insert({Name:"Test1", Age:30, Gender:"M", Class: "Chemistry", Score:90})
> db.students.insert({Name:"Test2", Age:30, Gender:"M", Class: "Chemistry", Score:90})
> db.students.insert({Name:"Test3", Age:30, Gender:"M", Class: "Chemistry", Score:90})
```

1. find all students with names starting with “St” or “Te”

```
> db.students.find({"Name":/(St|Te)*/i, "Class":/(Che)/i})
{ "_id" : ObjectId("52f89ecae451bb7a56e59086"), "Name" : "Student2", "Age" : 30,
"Gender" : "M", "Class" : "Chemistry", "Score" : 90 }
.......................
{ "_id" : ObjectId("52f89f06e451bb7a56e59089"), "Name" : "Test3", "Age" : 30,
"Gender" : "M", "Class" : "Chemistry", "Score" : 90 }
>
```

"Name":/(St|Te)*/i

//i indicates that the regex is case insensitive.

(St|Te)* means the Name string must start with either “St” or “Te”.

The * at the end means it will match anything after that.

When you put everything together, you are doing a case insensitive match of names that have either “St” or “Te” at the beginning of them. In the regex for the Class also the same Regex is issued.

Fetch Students with names as student1, student2 and who are male students with age >=25.

Ex: >db.students.find({"Name":/(student*)/i,"Age":{"$gte":25},"Gender":"M"})

{ "_id" : ObjectId("52f89eb1e451bb7a56e59085"), "Name" : "Student1", "Age" : 30,"Gender" : "M", "Class" : "Biology", "Score" : 90 }

{ "_id" : ObjectId("52f89ecae451bb7a56e59086"), "Name" : "Student2", "Age" : 30,"Gender" : "M", "Class" : "Chemistry", "Score" : 90 }

MAP REDUCE:

MapReduce is a framework that is used to process problems that are highly distributable across enormous datasets and are run using multiple nodes. If all

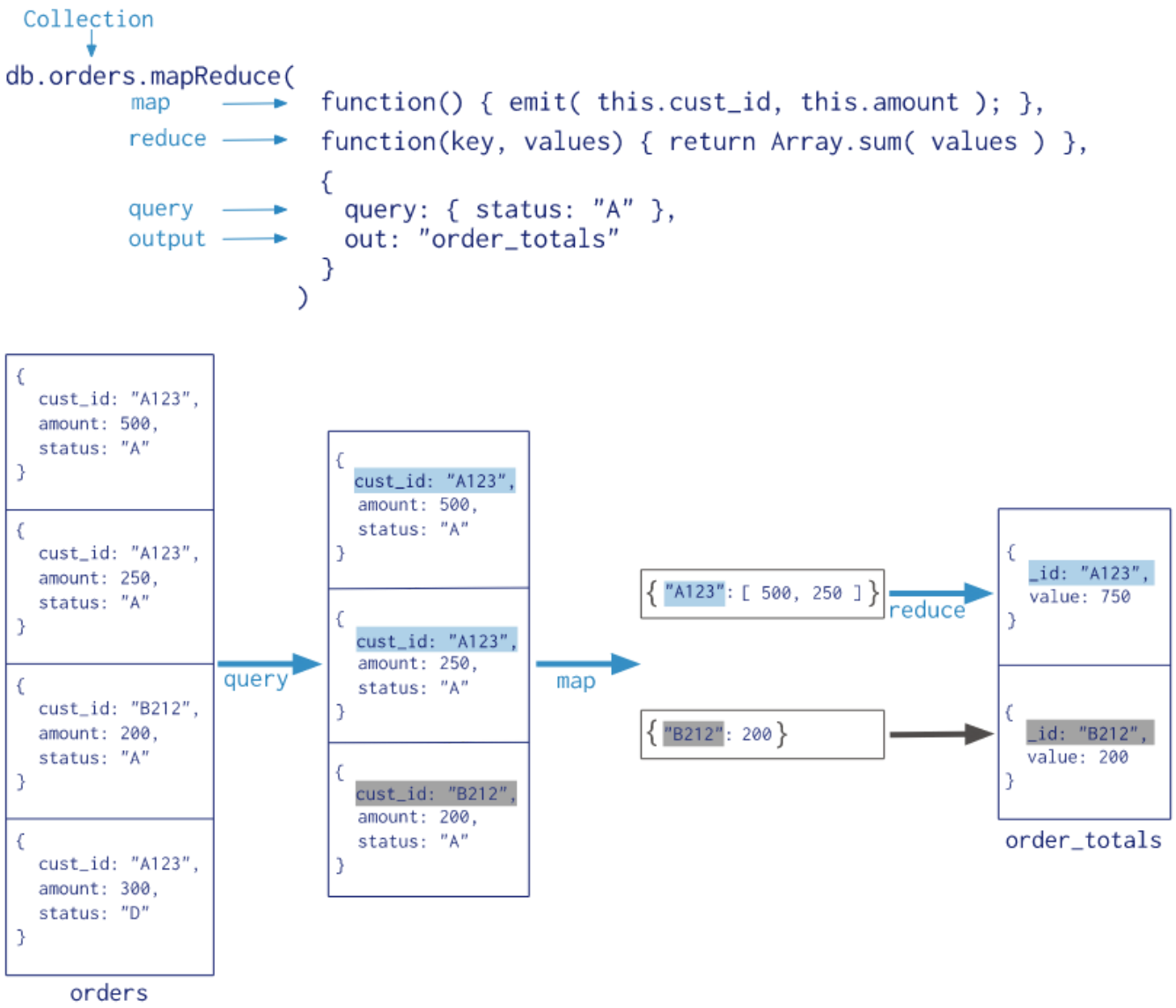
the nodes have the same hardware, these nodes are collectively referred to as a cluster; otherwise, it's referred to as a grid.

This processing can occur on structured data (data stored in a database) and unstructured data (data stored in a file system).

"Map": In this step, the node that is acting as the master takes the input parameter and divides the big problem into multiple small sub-problems. These sub-problems are then distributed across the worker nodes. The worker nodes might further divide the problem into sub-problems. This leads to a multi-level tree structure.

The worker nodes will then work on the sub-problems within them and return the answer back to the master node.

• "**Reduce**": In this step, all the sub-problems' answers are available with the master node, which then combines all the answers and produces the final output, which is the answer to the big problem you were trying to solve.



Using embedded schema

db.posts.find({'comments.author': 'author2'},{'comments': 1})

db.comments.find({"author": "author2"})

Schema Design

Table 10-1. *Log File*

Node UUID	IP Address	Node Name	MIB	Time Stamp (ms)	Metric Ve
3beb1a8b-040d-4b46-932a-2d31bd353186	10.161.1.73	corp_xyz_sardar	IFU	1369221223384	0.2

The following is the simplest way to store each line as text:

{

_id: ObjectId(...),

line: '10.161.1.73 - corp_xyz_sardar [15/July/2015:13:55:36 -0700] "Interface Util" ...

}

Inserting Data

The method used for inserting data depends on your application write concerns.

1. If you are looking for fast insertion speed and can compromise on the data safety, then the following command can be used:

> db.perfpoc.insert({Host:"Host1", GeneratedOn: new ISODate("2015-07-15T12:02Z"),

ParameterName:"CPU",Value:13.13},w=0)>

Although this command is the fastest option available, since it doesn't wait for any acknowledgment of whether the operation was successful or not, you risk losing data.

2. If you want just an acknowledgment that at least the data is getting saved, you can issue the following command:

>db.perfpoc.insert({Host:"Host1", GeneratedOn: new ISODate("2015-07-15T12:07Z"),

ParameterName:"CPU",Value:13.23},w=1)>

Although this command acknowledges that the data is saved, it will not provide safety against any data loss because it is not journaled.

3. If your primary focus is to trade off increased insertion speed for data safety guarantees, you can issue the following command:

> db.perfpoc.insert({Host:"Host1", GeneratedOn: new ISODate("2015-07-15T12:09Z"),

ParameterName:"CPU",Value:30.01},j=true,w=2)>

MongoDB Architecture

Core Processes

The core components in the MongoDB package are

• mongod , which is the core database process

• mongos , which is the controller and query router for sharded clusters

• mongo , which is the interactive MongoDB shell

MongoDB Tools

Apart from the core services, there are various tools that are available as part of the MongoDB installation:

• mongodump : This utility is used as part of an effective backup strategy. It creates a binary export of the database contents.

• mongorestore : The binary database dump created by the mongodump utility is

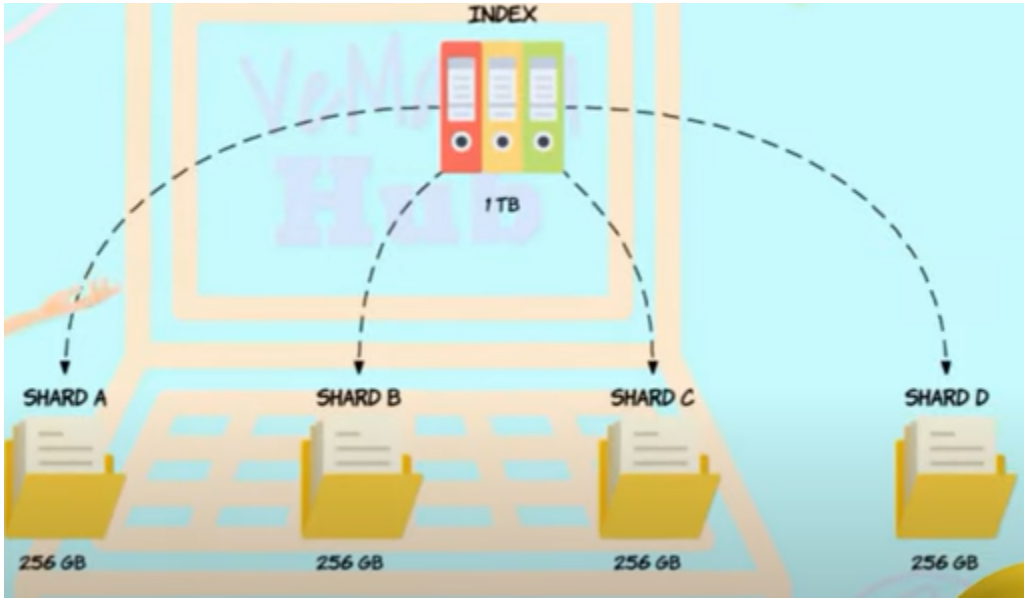
imported to a new or an existing database using the mongorestore utility.

• bsondump : This utility converts the BSON files into human-readable formats such as JSON and CSV. For example, this utility can be used to read the output file generated by mongodump.

• mongoimport , mongoexport : mongoimport provides a method for taking data in JSON, CSV, or TSV formats and importing it into a mongod instance. Mongoexport provides a method to export data from a mongod instance into JSON, CSV, or TSV formats.

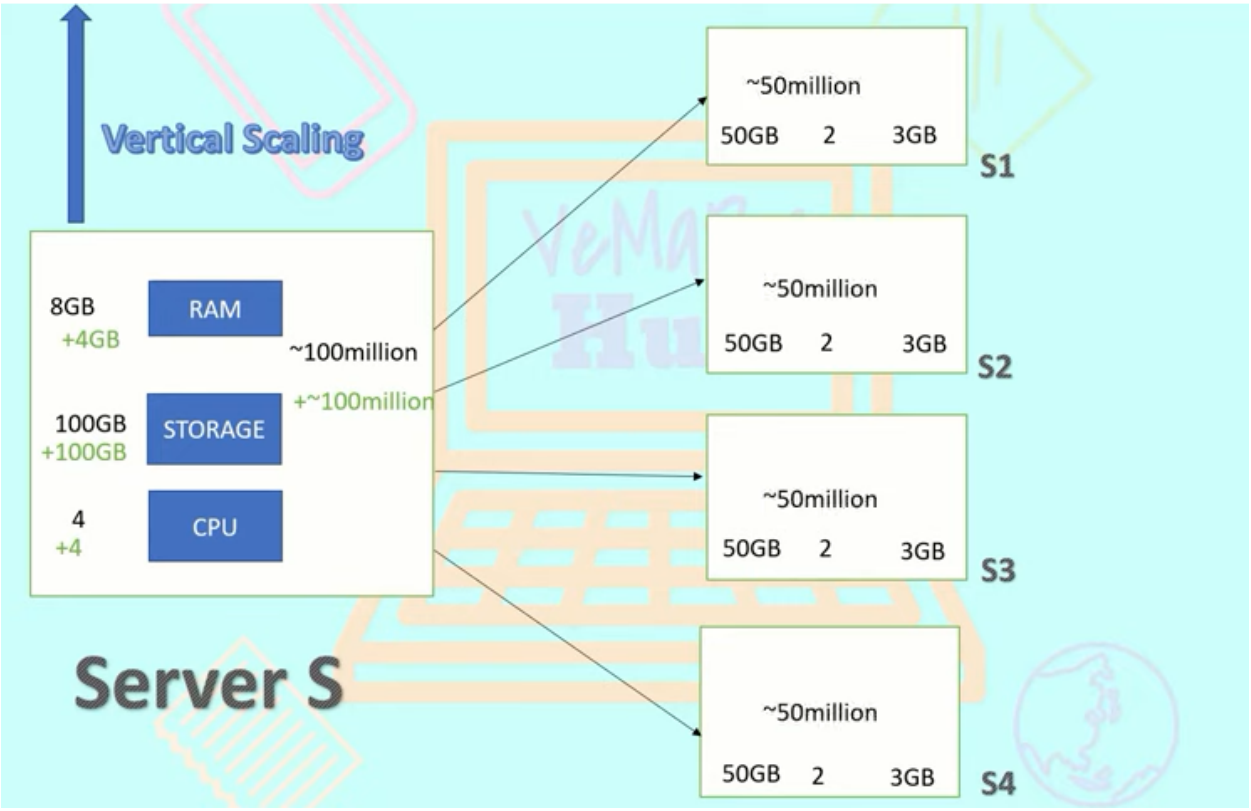
• mongostat , mongotop , mongosniff: These utilities provide diagnostic information related to the current operation of a mongod instance.

Sharding:

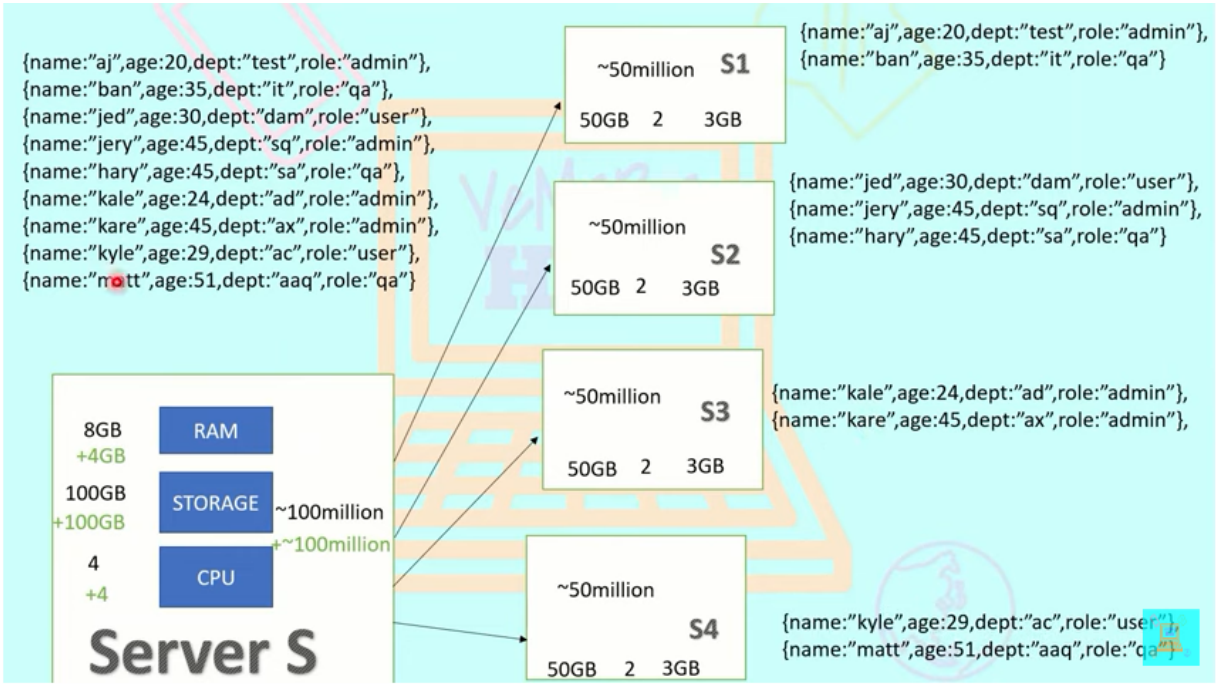


Vertical scaling

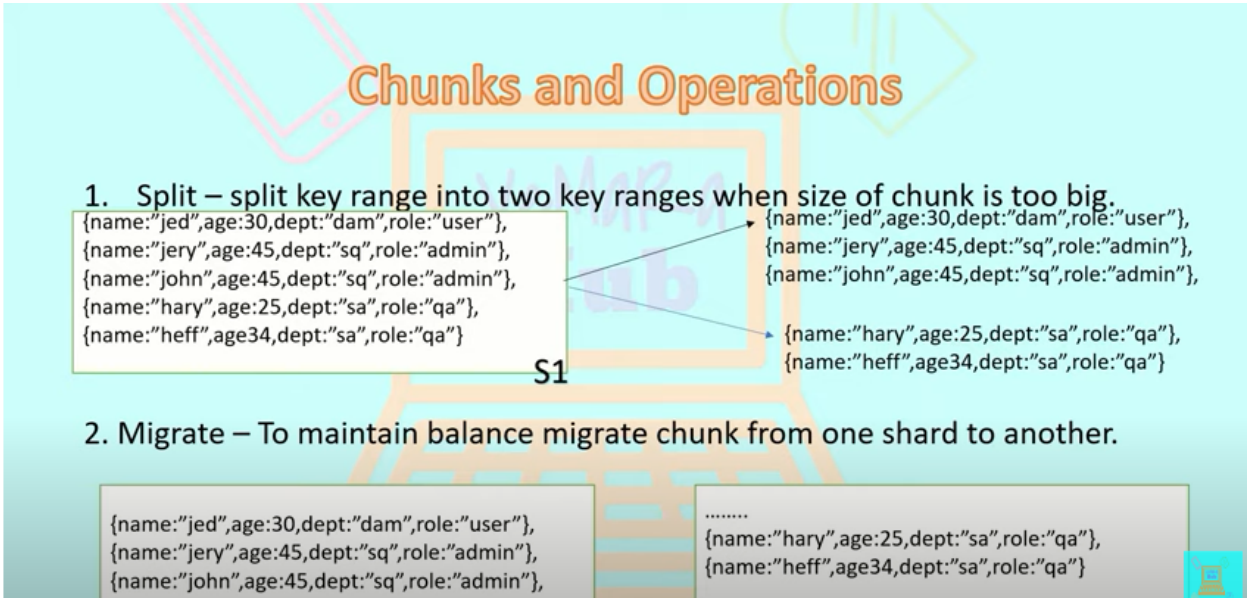
Horizontal scaling



Horizontal scaling → sharding (each shards)



Chunks of data can be moved

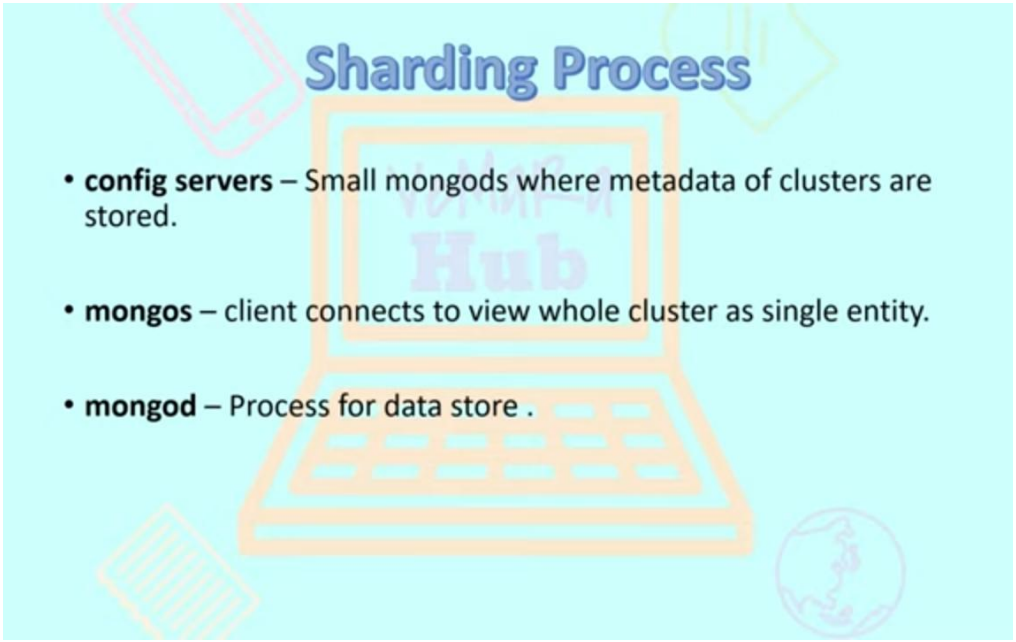


Chunk Splitting

Chunk splitting is one of the processes that ensures the chunks are of the specified size.
As you have seen, a shard key is chosen and it is used to identify how the documents will be distributed across the shards.

The documents are further grouped into chunks of 64MB (default and is configurable) and are stored in the shards based on the range it is hosting.

If the size of the chunk changes due to an insert or update operation, and exceeds the default chunk size, then the chunk is split into two smaller chunks by the mongos.



Primary and Secondary Members

Before you move ahead and look at how the replica set functions, let's look at the type of members that a replica set can have. There are two types of members: primary members and secondary members .

• **Primary member** : A replica set can have only one primary, which is elected by the voting nodes in the replica set. Any node with associated priority as 1 can be elected

as a primary. The client redirects all the write operations to the primary member, which is then later replicated to the secondary members.

• **Secondary member**: A normal secondary member holds the copy of the data. The secondary member can vote and also can be a candidate for being promoted to primary in case of failover of the current primary.

Data Replication Process

- The members of a replica set replicate data continuously.
- Every member, including the primary member, maintains an oplog.
- An oplog is a capped collection where the members maintain a record of all the operations that are performed on the data set.

OPLOG

ts : This stores the timestamp when the operations are performed. It's an internal type and is composed of a 4-byte timestamp and a 4-byte incrementing counter.

• op : This stores information about the type of operation performed. The value is stored as 1-byte code (e.g. it will store an "I" for an insert operation).

• ns : This key stores the collection namespace on which the operation was performed.

• o : This key specifies the operation that is performed. In case of an insert, this will store the document to insert.

In MongoDB, the scaling is handled by scaling out the data horizontally (i.e. partitioning the data across multiple commodity servers), which is also called sharding (horizontal scaling).

Sharding addresses the challenges of scaling to support large data sets and high throughput by horizontally dividing the datasets across servers where each server is responsible for handling its part of data and no one server is burdened. These servers are also called shards. Every shard is an independent database. All the shards collectively make up a single logical database .

Sharding reduces the operations count handled by each shard. For example, when data is inserted, only

the shards responsible for storing those records need to be accessed.

The processes that need to be handled by each shard reduce as the cluster grows because the subset of

data that the shard holds reduces. This leads to an increase in the throughput and capacity horizontally.

Sharding Components

The following are the components of a sharded cluster:

• Shards

- The shard is the component where the actual data is stored. For the sharded cluster, it holds a subset of data and can either be a mongod or a replica set.
- All shard's data combined together forms the complete dataset for the sharded cluster.

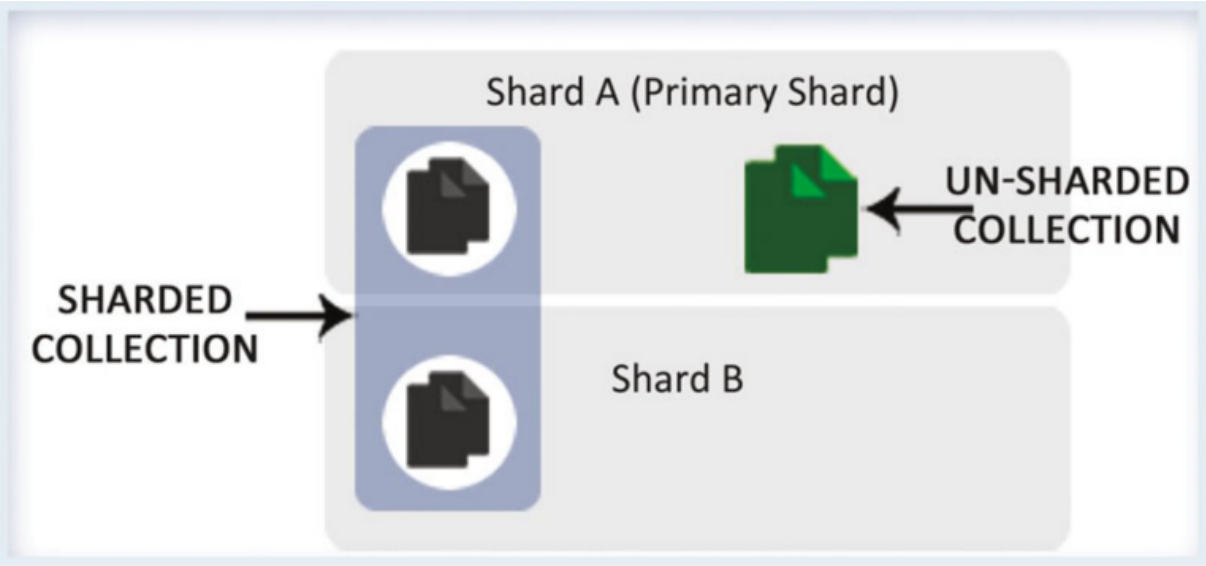


Figure 7-16. *Primary shard*

• mongos

- The mongos act as the routers. They are responsible for routing the read and write request from the
- application to the shards.
-

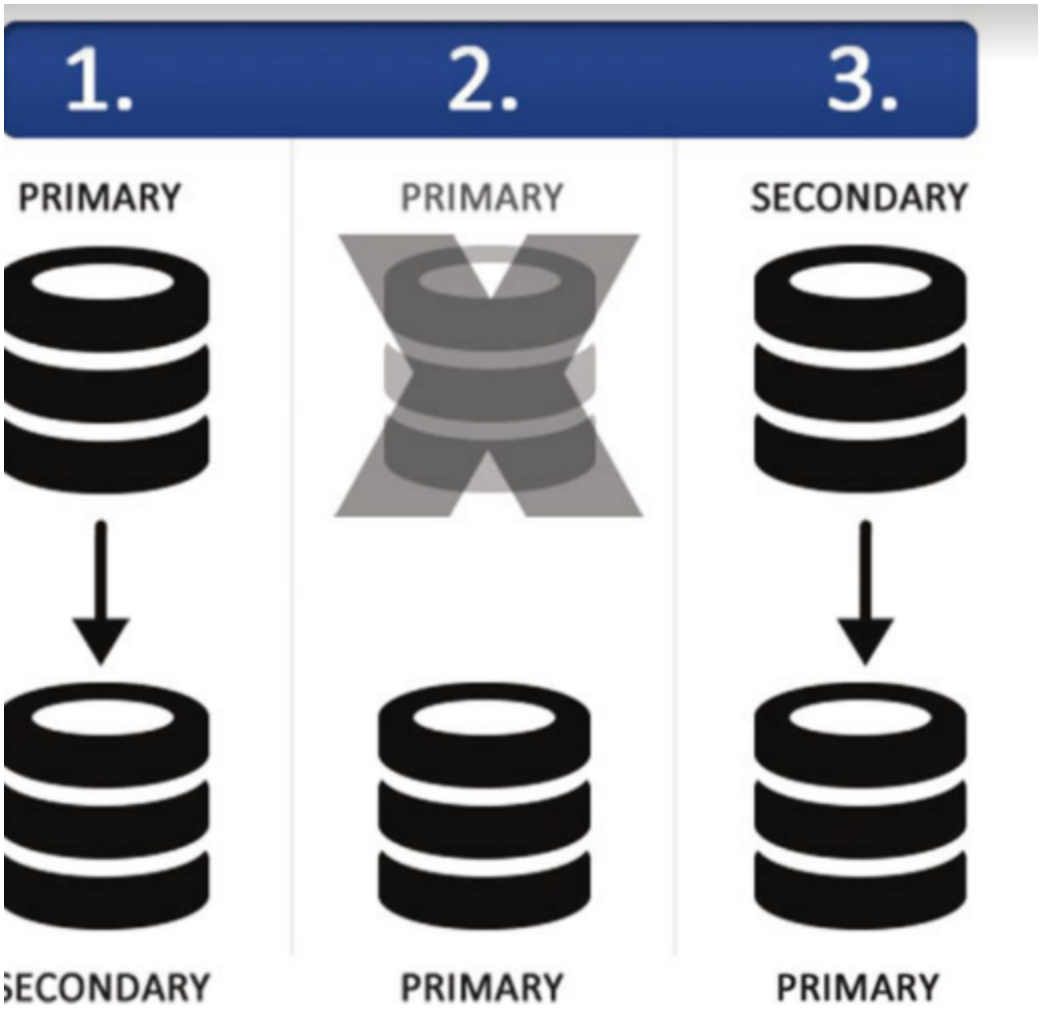
• Config servers

- Config servers are special mongods that hold the sharded cluster's metadata.
- This metadata depicts the sharded system state and organization.

- The config server stores data for a single sharded cluster. The config servers should be available for the proper functioning of the cluster.
- One config server can lead to a cluster's single point of failure. For production deployment it's recommended to have at least three config servers, so that the cluster keeps functioning even if one config server is not accessible.
- A config server stores the data in the config database, which enables routing of the client requests to the respective data. This database should not be updated.

Replica Set:

Two-member replica set failover



1. The primary goes down, and the secondary is promoted as primary.

2. The original primary comes up, it acts as slave, and becomes the secondary node.

Monitoring the Config Servers

- The config server, as you know by now, stores the metadata of the sharded cluster.
- The mongos caches the data and routes the request to the respective shards.
- If the config server goes down but there's a running mongos instance, there's no immediate impact on the shard cluster and it will remain available for a while.
- However, you won't be able to perform operations like chunk migration or restart a new mongos.
- In the long run, the unavailability of the config server can severely impact the availability of the cluster.
- To ensure that the cluster remains balanced and available, you should monitor the config servers.

Monitoring the Shard Status Balancing and Chunk Distribution

- db.printShardingStatus() or sh.status()

  command in the mongos mongo shell to ensure that the process is working effectively.

Lock status

use config

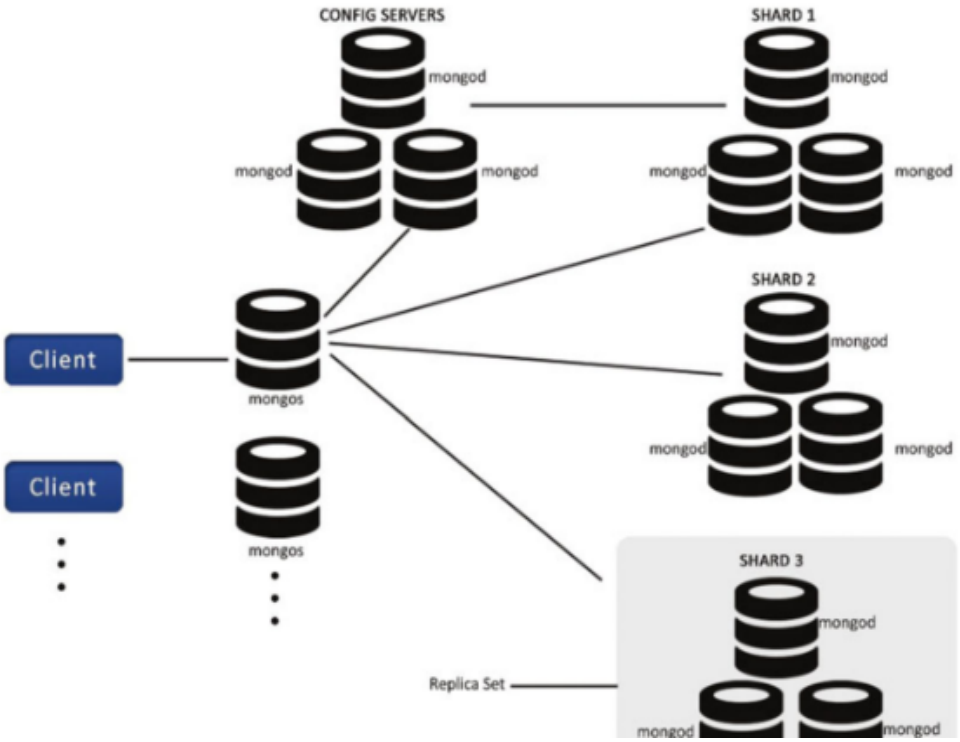
db.locks.find()

Application's production cluster should have the following components :

1. At least two mongos instances, but you can have more as per need.

2. Three config servers, each on a separate system.

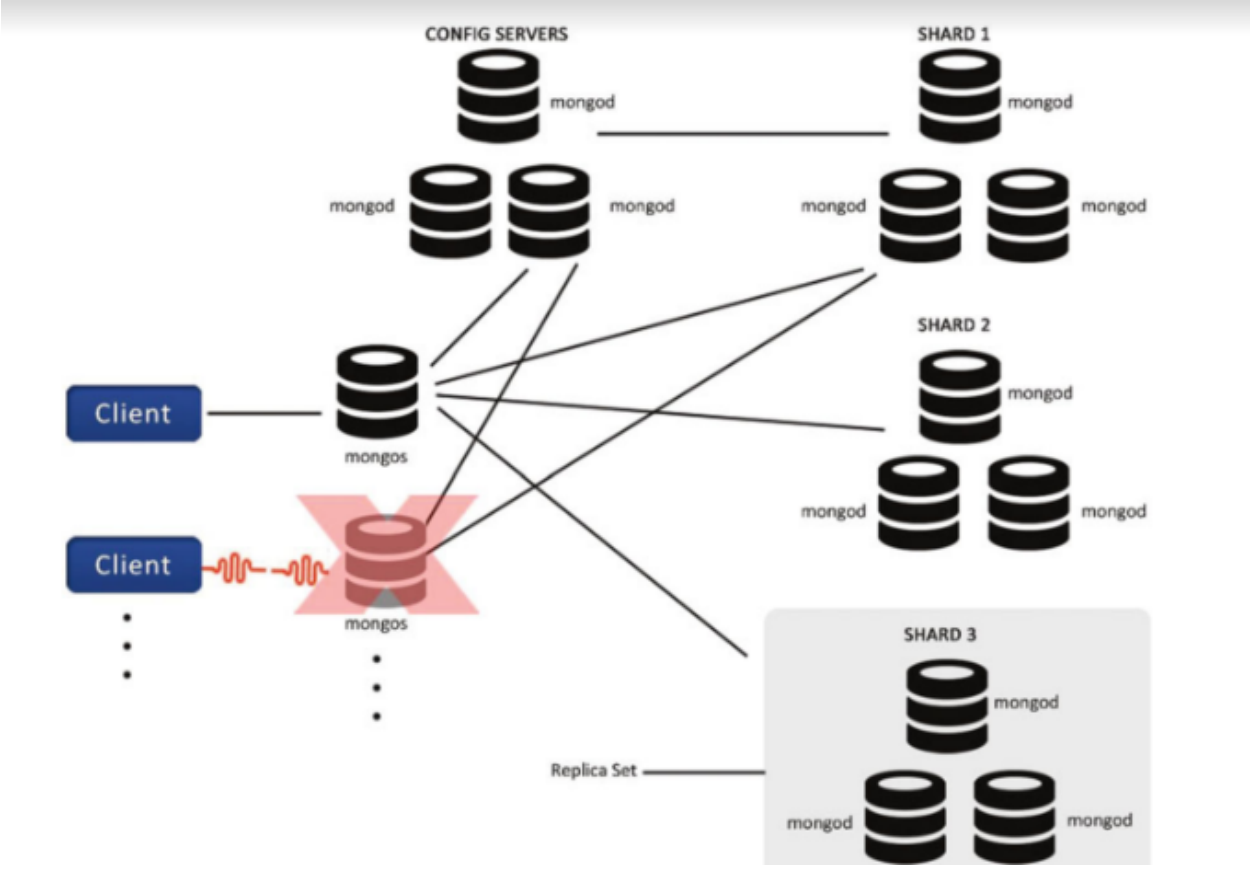
3. Two or more replica sets serving as shards. The replica sets are distributed across geographies with read concern set to nearest.


CONFIG SERVERS
mongod
mongod
mongod
SHARD 1
mongod
mongod
mongod
SHARD 2
mongod
mongod
mongod
Client
Client
mongos
mongos
SHARD 3
mongod
mongod
mongod
Replica Set

Scenario 1

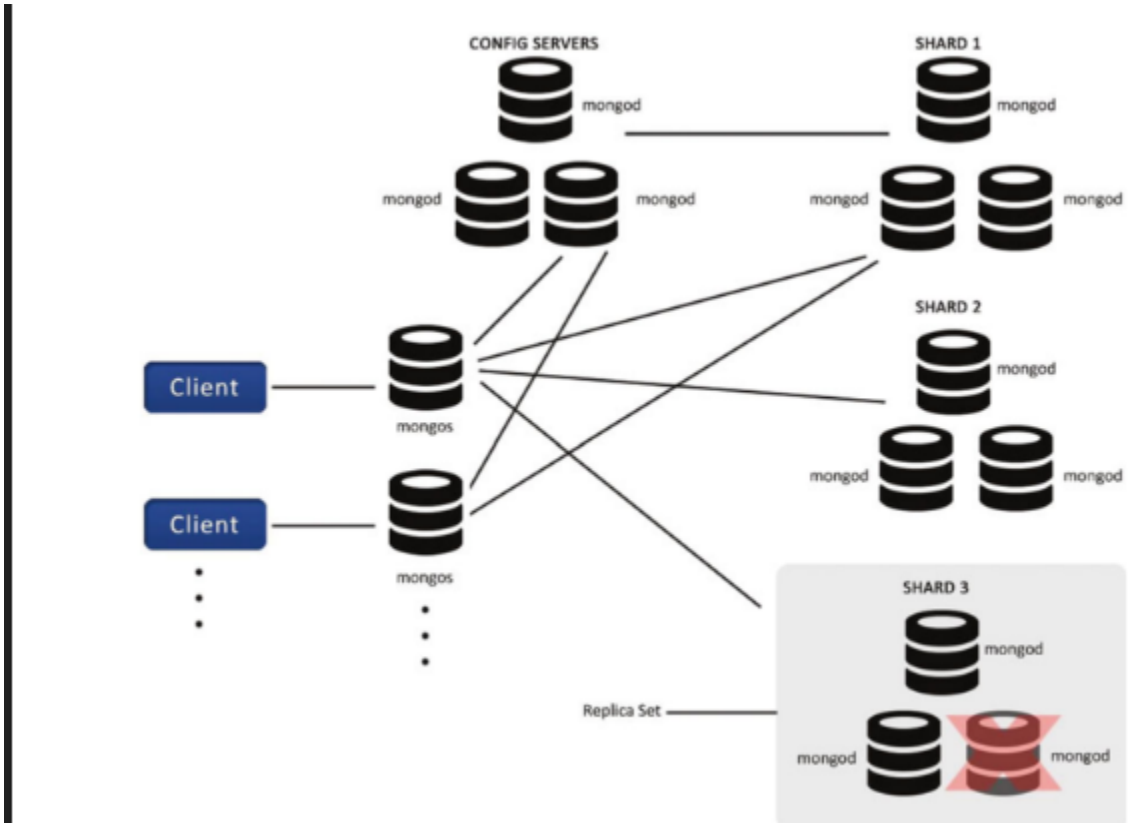
Mongos become unavailable:

- The application server where mongos has gone down will not be able to communicate with the cluster but it will not lead to any data loss since the mongos doesn't maintain any data of its own.
- The mongos can restart, and while restarting, it can sync up with the config servers to cache the cluster metadata, and the application can normally start its operations



Scenario 2

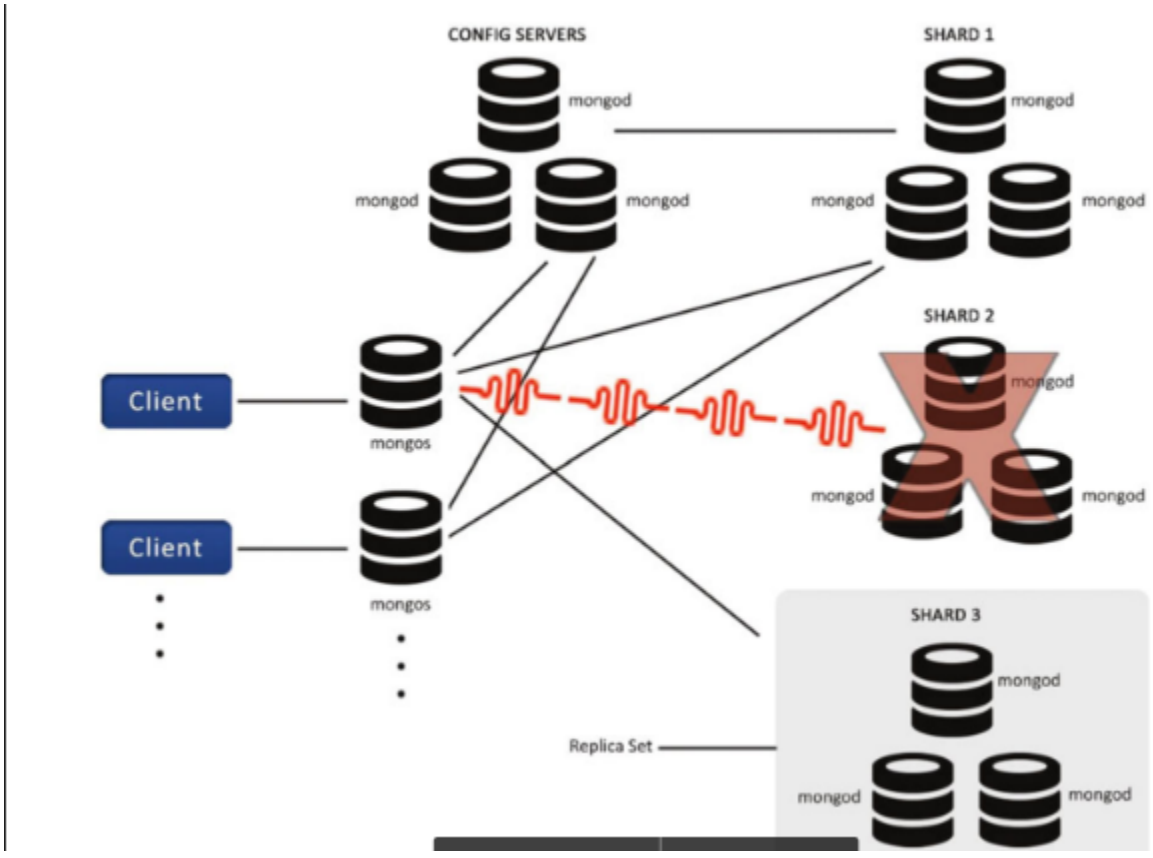
One of the mongod of the replica set becomes unavailable in a shard: Since you used replica sets to provide high availability, there is no data loss. If a primary node is down, a new primary is chosen, whereas if it's a secondary node, then it is disconnected and the functioning continues normally.



Scenario 3

If one of the shard becomes unavailable: In this scenario, the data on the shard will be unavailable, but the other shards will be available, so it won't stop the application.

The application can continue with its read/ write operations; however, the partial results must be dealt with within the application. In parallel, the shard should attempt to recover as soon as possible

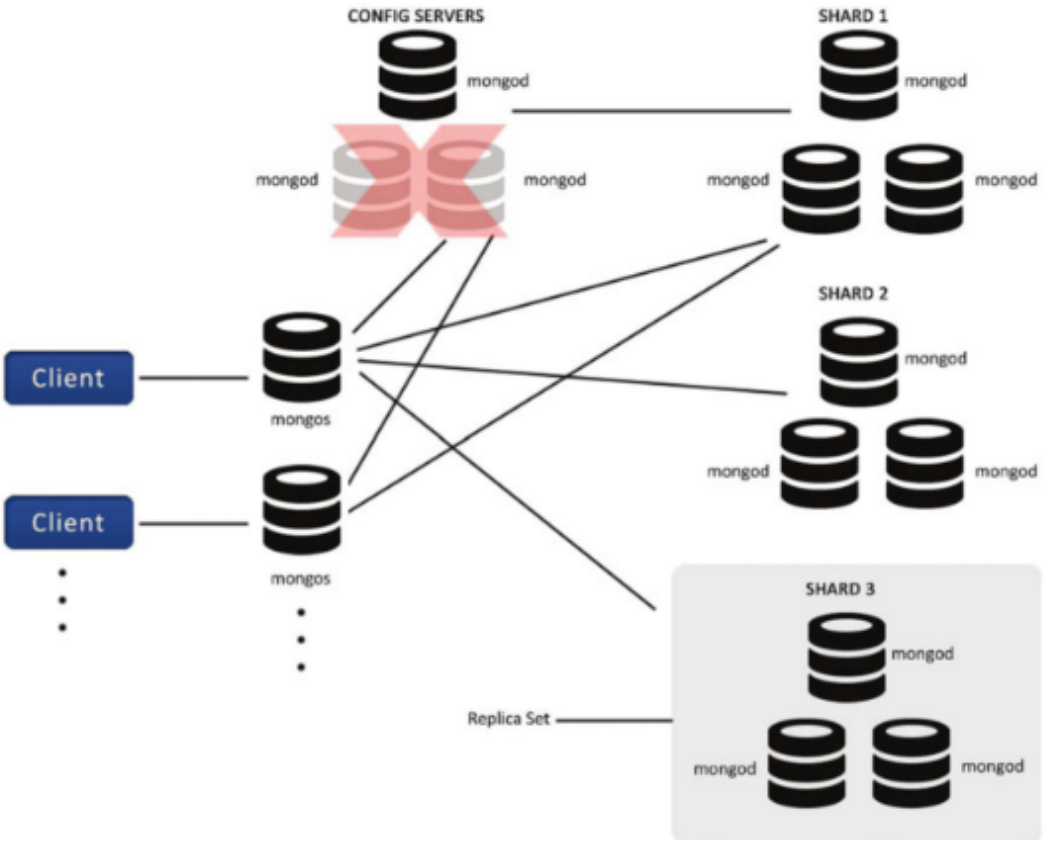


Scenario 4

Only one config server is available out of three:

In this scenario, although the cluster will become read- only, it will not serve any operations that might lead to changes in the cluster structure, thereby leading to a change of metadata such as chunk migration or chunk splitting.

The config servers should be replaced ASAP because if all config servers become unavailable, this will lead to an inoperable cluster



Explicitly Creating Collections :

db.createCollection("users")

Inserting Documents Using Loop

for(var i=1; i<=20; i++) db.users.insert({"Name" : "Test1" + i, "Age": 20+i,

"Department" : "CS", "Country" : "India"})

db.users.find()

Inserting by Explicitly Specifying _id :

db.users.insert({"_id":1, "Name": "Bhavans"})

db.users.find()

read the data:

user1 = {FName: "Test", LName: "User", Age:30, Gender: "M", Country: "US"}

 # Two variables are created which has some record , the insertion was done using a variable (in this case its user 1 and user 2 variable to print) using find , display the collection and check

user1 = {FName: "Test", LName: "User", Age:30, Gender: "M", Country: "US"}

user2 = {Name: "Test User", Age:45, Gender: "F", Country: "US"}

db.users.insert(user1)

db.users.insert(user2)

for(var i=1; i<=20; i++) db.users.insert({"Name" : "Test User" + i, "Age": 10+i,

"Gender" : "F", "Country" : "India"})

db.users.find()

Help command in mongo shell:

C:\practicalmongodb\bin\mongod.exe --help

for help and startup options

```
C:\Users\LAB01-PC30>mongod.exe --help
Options:
  --networkMessageCompressors arg (=snappy,zstd,zlib)
                                        Comma-separated list of compressors to
                                        use for network messages

General options:
  -h [ --help ]                         Show this usage information
  --version                             Show version information
  -f [ --config ] arg                   Configuration file specifying
                                        additional options
  --configExpand arg                    Process expansion directives in config
                                        file (none, exec, rest)
  --port arg                            Specify port number - 27017 by default
  --ipv6                                Enable IPv6 support (disabled by
                                        default)
  --listenBacklog arg (=2147483647)     Set socket listen backlog size
  --maxConns arg (=1000000)             Max number of simultaneous connections
  --pidfilepath arg                     Full path to pidfile (if not set, no
                                        pidfile is created)
  --timeZoneInfo arg                    Full path to time zone info directory,
                                        e.g. /usr/share/zoneinfo
  -v [ --verbose ] [=arg(=v)]           Be more verbose (include multiple times
                                        for more verbosity e.g. -vvvvv)
  --quiet                               Quieter output
  --logpath arg                         Log file to send write to instead of
                                        stdout - has to be a file, not
                                        directory
  --logappend                           Append to logpath instead of
                                        over-writing
  --logRotate arg                       Set the log rotation behavior
                                        (rename|reopen)
  --timeStampFormat arg                 Desired format for timestamps in log
                                        messages. One of iso8601-utc or
                                        iso8601-local
  --setParameter arg                    Set a configurable parameter
  --bind_ip arg                         Comma separated list of ip addresses to
                                        listen on - localhost by default
  --bind_ip_all                         Bind to all ip addresses
  --noauth                              Run without security
```

Switching to admin user:

>db = db.getStudentDB('admin')

Admin

The user needs to be created with either of the roles: userAdminAnyDatabase or userAdmin:

>db.createUser({user: "AdminUser", pwd: "password", roles:["userAdminAnyDatabase"]})

Insertion will also work with variable:

 # Create a variable and insert the variable in the collection as a record.

> user1 = {FName: "Test", LName: "User", Age:30, Gender: "M", Country: "US"}

> user2 = {Name: "Test User", Age:45, Gender: "F", Country: "US"}

> db.users.insert(user1)

> db.users.insert(user2)

Update queries:

1. **To update the country to UK for all female employees:**

   db.users.update({"Gender":"F"}, {$set:{"Country":"UK"}})

Setting multi option true

By default only the first document will be updated with the use of update function since multi was not set to true. If set true , it updates all the records based on the condition mentioned.

>db.users.update({"Gender":"F"},{$set:{"Country":"UK"}},{multi:true})

Delete:

db.users.remove({"Gender":"M"})

skip( )

If the requirement is to skip the first two records and return the third and fourth user, the skip command is used. The following command needs to be executed:

>db.users.find({"Gender":"F",$or:[{"Country":"India"},
{"Country":"US"}]}).limit(2).skip(2)

Examples of sort:

> db.testindx.find().sort({"Age":1})

```
> db.testindx.find().sort({"Age":1,"Name":1})

> db.testindx.find().sort({"Age":1,"Name":1, "Class":1})

> db.testindx.find().sort({"Gender":1, "Age":1, "Name": 1})
```