Contents to be covered

- Need of a database
- Different types of database,
- Relational vs Non-relational database

Need of a database

- In our day to day life as an end user we discover online applications which try to make our lives more convenient. As we know about these just get ourselves register ourselves for that application.
- After the one-time registration, whenever we want to use that app again, we just need to log in with user name and password sometime just a mobile based OTP, on the other end the application system automatically stores all our data that was provided during the registration process and submitted subsequently.

Need of a database

- All this is possible because of the database in which all our information or data gets stored when we register for any application. It may be our Facebook, Whatsapp, Instagram, email services or airline or hospitality application etc.
- when we browse various ecommerce websites and view millions of product items available on various online shopping applications like Amazon, Flipkart etc or post our status, photos videos on Facebook, twitter, youtube etc to let others view them, it is only possible because of the databases.

Need of a database

- *At present all the applications weather a desktop or mobile one, databases acts as a key component for collecting, storing and retrieving of these data in a safe and efficient manner with outmost security and privacy. The task of choosing the right option is tedious process.*

Need of a database

- In physical world and go back and analyses the past, before the implementation of computers and databases, departments like electricity boards, insurance offices, Banks, Public health departments etc stores a large heaps of files containing the data of all of their users/ customers.
- It was a quite troublesome to get details pertaining to a customer, e.g. consumption of electricity, payments made or pending towards bills generated, etc, if the names were not listed alphabetically or data in any other order like in ascending order of account number.
- Even the management of the physical data was done is just like it's done in computer databases but not in all cases. And if the data is not present or not in an organized form, then the getting any information from this physical data is taking so long time.

Need of a database

- The data stored in a computer database is organized in such a way that the stored data helps in easy retrieval of information along with storage and editing. The data stored in physical files can get lost when the papers of these files get older and, hence, get destroyed.
- But in a database, we can store as long as we want in a very small space with security and this computer based Data will get lost only if the system crashes. To overcome this, we keep a backup that may be in form of backup data on external storage or in an online mode depending upon the application requirement as well as the budget.

Types of databases

- Depending upon the usage requirements, there are following types of databases available in the market −
  - Centralized database.
  - Distributed database.
  - Personal database.
  - NoSQL database.
  - Operational database.
  - Relational database.
  - Cloud database.
  - Object-oriented database.
  - Graph database.

Types of databases

1. Centralized Database

The information(data) is stored at a centralized location and the users from different locations can access this data. This type of database contains application procedures that help the users to access the data even from a remote location.

Various kinds of authentication procedures are applied for the verification and validation of end users, likewise, a registration number is provided by the application procedures which keep a track and record of data usage. The local area office handles this thing.

Types of databases

2. Distributed Database

Just opposite of the centralized database concept, the distributed database has contributions from the common database as well as the information captured by local computers also. The data is not at one place and is distributed at various sites of an organization. These sites are connected to each other with the help of communication links which helps them to access the distributed data easily.

Types of databases

- You can imagine a distributed database as a one in which various portions of a database are stored in multiple different locations (physical) along with the application procedures which are replicated and distributed among various points in a network.
- There are two kinds of distributed database, viz. homogenous and heterogeneous. The databases which have same underlying hardware and run over same operating systems and application procedures are known as homogeneous DDB, for eg. All physical locations in a DDB. Whereas, the operating systems, underlying hardware as well as application procedures can be different at various sites of a DDB which is known as heterogeneous DDB.

Types of databases

3.Personal Database

Data is collected and stored on personal computers which is small and easily manageable. The data is generally used by the same department of an organization and is accessed by a small group of people.

Types of databases

4.NoSQL Database

These are used for large sets of distributed data. There are some big data performance issues which are not effectively handled by relational databases, such kind of issues are easily managed by NoSQL databases. There are very efficient in analyzing large size unstructured data that may be stored at multiple virtual servers of the cloud.

Types of databases

5.Operational Database

Information related to operations of an enterprise is stored inside this database. Functional lines like marketing, employee relations, customer service etc. require such kind of databases.

Types of databases

6.Relational Databases

These databases are categorized by a set of tables where data gets fit into a pre-defined category. The table consists of rows and columns where the column has an entry for data for a specific category and rows contains instance for that data defined according to the category.

The Structured Query Language (SQL) is the standard user and application program interface for a relational database.

Types of databases

There are various simple operations that can be applied over the table which makes these databases easier to extend, join two databases with a common relation and modify all existing applications.

7.Cloud Databases

Now a day, data has been specifically getting stored over clouds also known as a virtual environment, either in a hybrid cloud, public or private cloud. A cloud database is a database that has been optimized or built for such a virtualized environment.

Types of databases

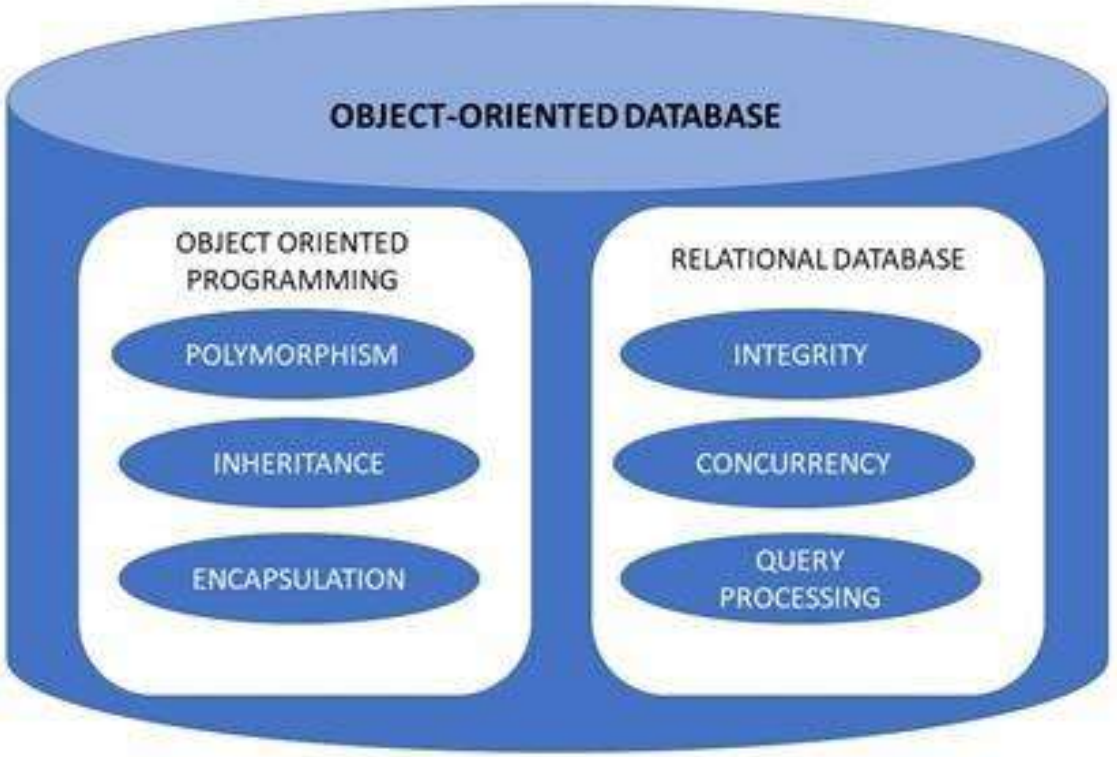
- There are various benefits of a cloud database, some of which are the ability to pay for storage capacity and bandwidth on a per-user basis, and they provide scalability on demand, along with high availability.
- A cloud database also gives enterprises the opportunity to support business applications in a software-as-a-service deployment.

Types of databases

8.Object-Oriented Databases

An object-oriented database is a collection of object-oriented programming and relational database. There are various items which are created using object-oriented programming languages like C++, Java which can be stored in relational databases, but object-oriented databases are well-suited for those items.

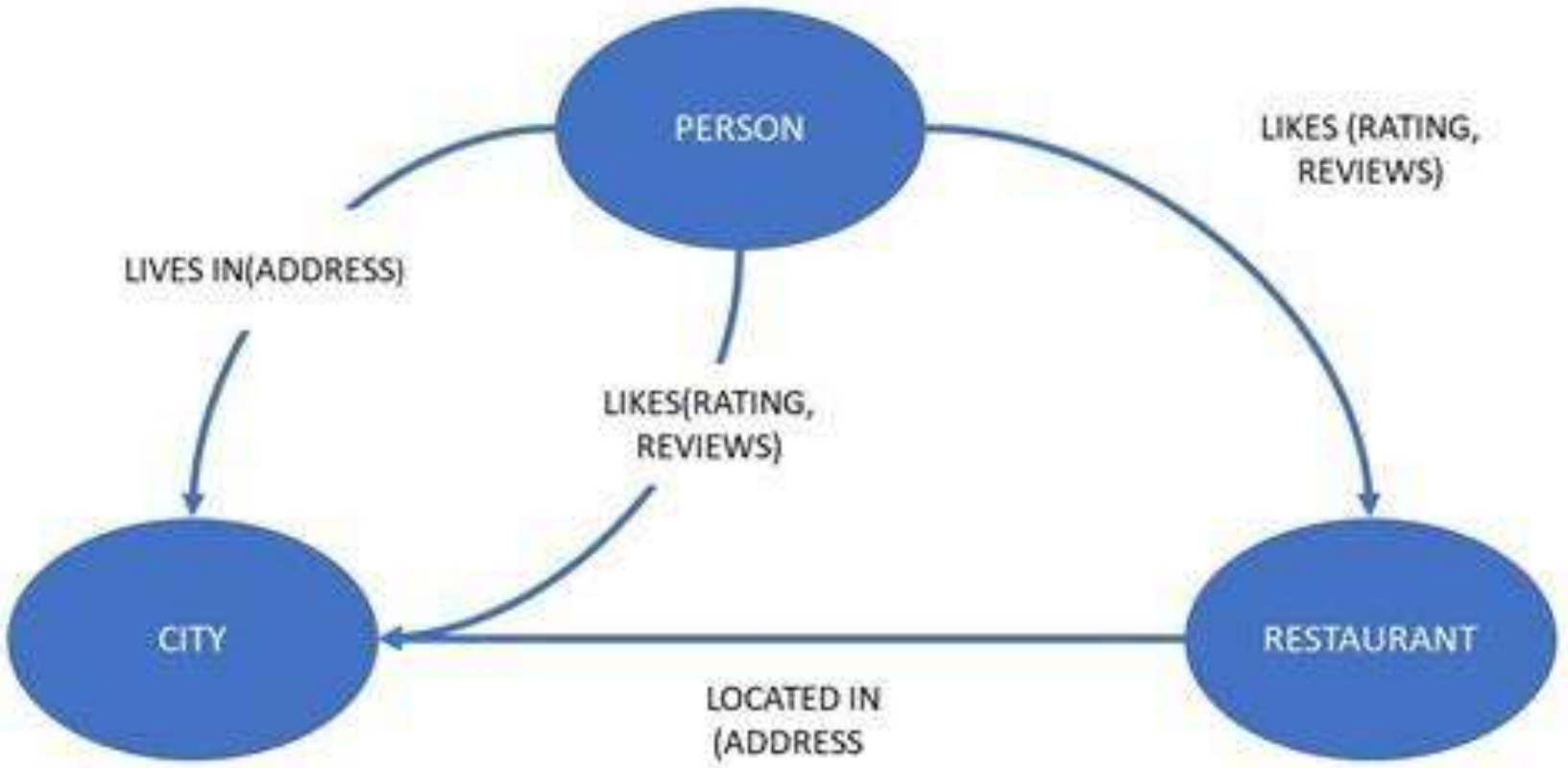
An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.

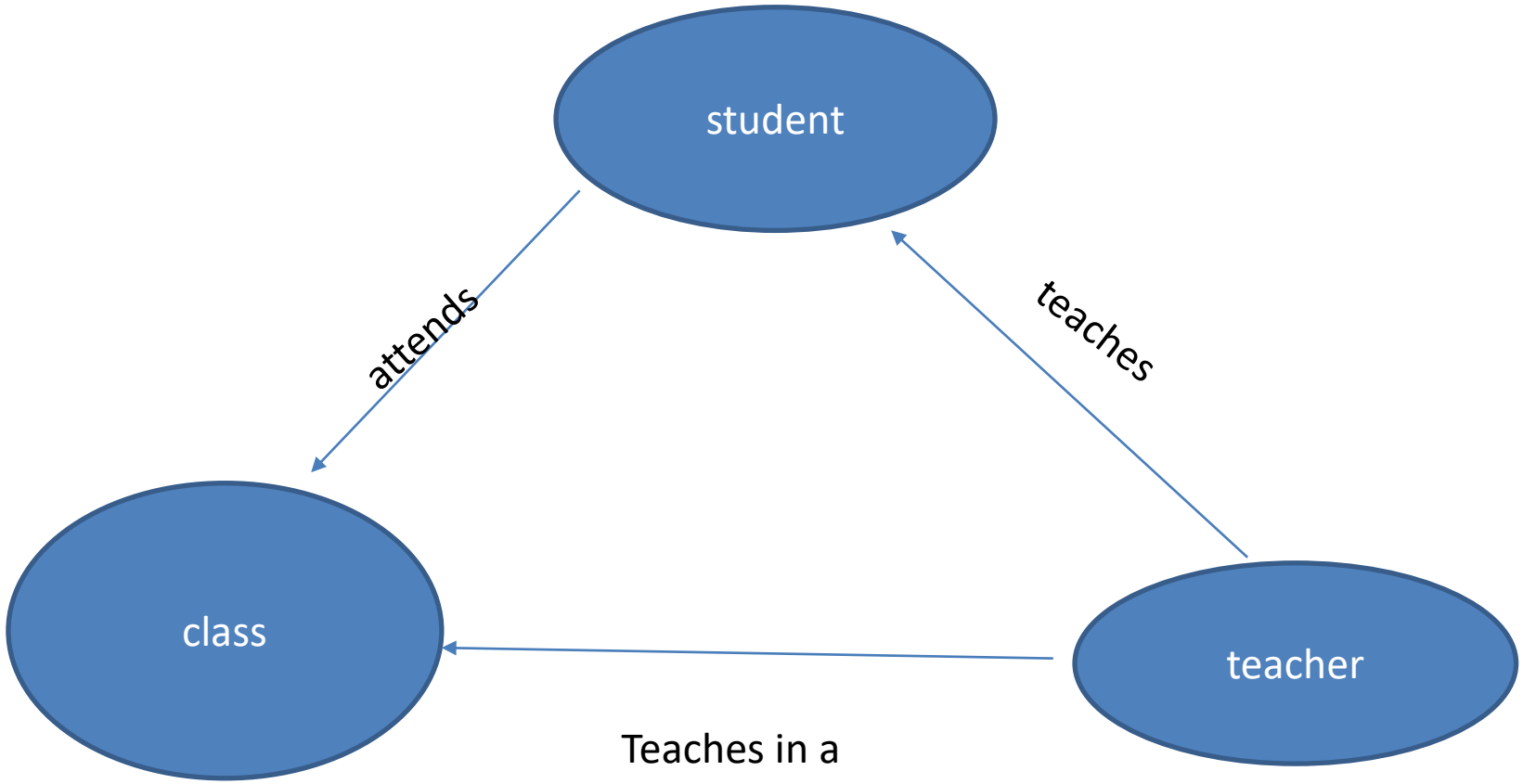

OBJECT-ORIENTED DATABASE
OBJECT ORIENTED PROGRAMMING
POLYMORPHISM
INHERITANCE
ENCAPSULATION
RELATIONAL DATABASE
INTEGRITY
CONCURRENCY
QUERY PROCESSING

Types of databases

9.Graph Databases

- The graph is a collection of nodes and edges where each node is used to represent an entity and each edge describes the relationship between entities. A graph-oriented database, or graph database, is a type of NoSQL database that uses graph theory to store, map and query relationships.
- Graph databases are basically used for analyzing interconnections. For example, companies might use a graph database to mine data about customers from social media.


PERSON
LIKES (RATING, REVIEWS)
LIVES IN(ADDRESS)
LIKES(RATING, REVIEWS)
CITY
RESTAURANT
LOCATED IN (ADDRESS


student
attends
teaches
class
teacher
Teaches in a

Relational vs. non-relational database

- **Relational databases** are the most prominent modern types of databases. Oracle, MySQL, PostgreSQL, or SQL Server are some examples of RDBMS. To access and manipulate the data in RDBMS, SQL (Structured Query Language) is required.
- SQL has well-established standards and allows your data to be easily portable. In relational database data is stored in **tables** consisting of columns and rows.
- Every row represents an individual record, and a column stands for a field with a data type assigned to it. Primary and foreign keys are used to link Tables having related information.

- Student

Rollno(PK)	Name	Phone number
1	Aaa	56654545656
2	Bbb	546545454

Rollno(FK)	marks	result
1	568	Pass
2	345	pass

Relational vs. non-relational database

- Now-a-days **non-relational** databases are also gaining popularity and being used so commonly. The main reason for this is the growing need for unstructured data storage like data from Social Media etc. In the era of big data, diversified information is used. diversity.
- Data now also mean images, videos, and even posts on social media networks and to work with non-tabular data, non-relational database is needed. These are also referred as **NoSQL databases as these** do not support SQL queries.

- RDBMS

Student rollno	Name	Marks
1	Kkk	56
2	Lll	85

- Non-relational
  - {“rollno”:1,”name”:rrr”,”marks”:67}
  - {“rollno”:2,”address”:”mumbai”,”phoneno”:2343423424}
  - {“rollno”:3,”hobbies”:”cricket”,”awards”:”volleyball”}

Advantages and the Disadvantages of Relational and Non-Relational Databases

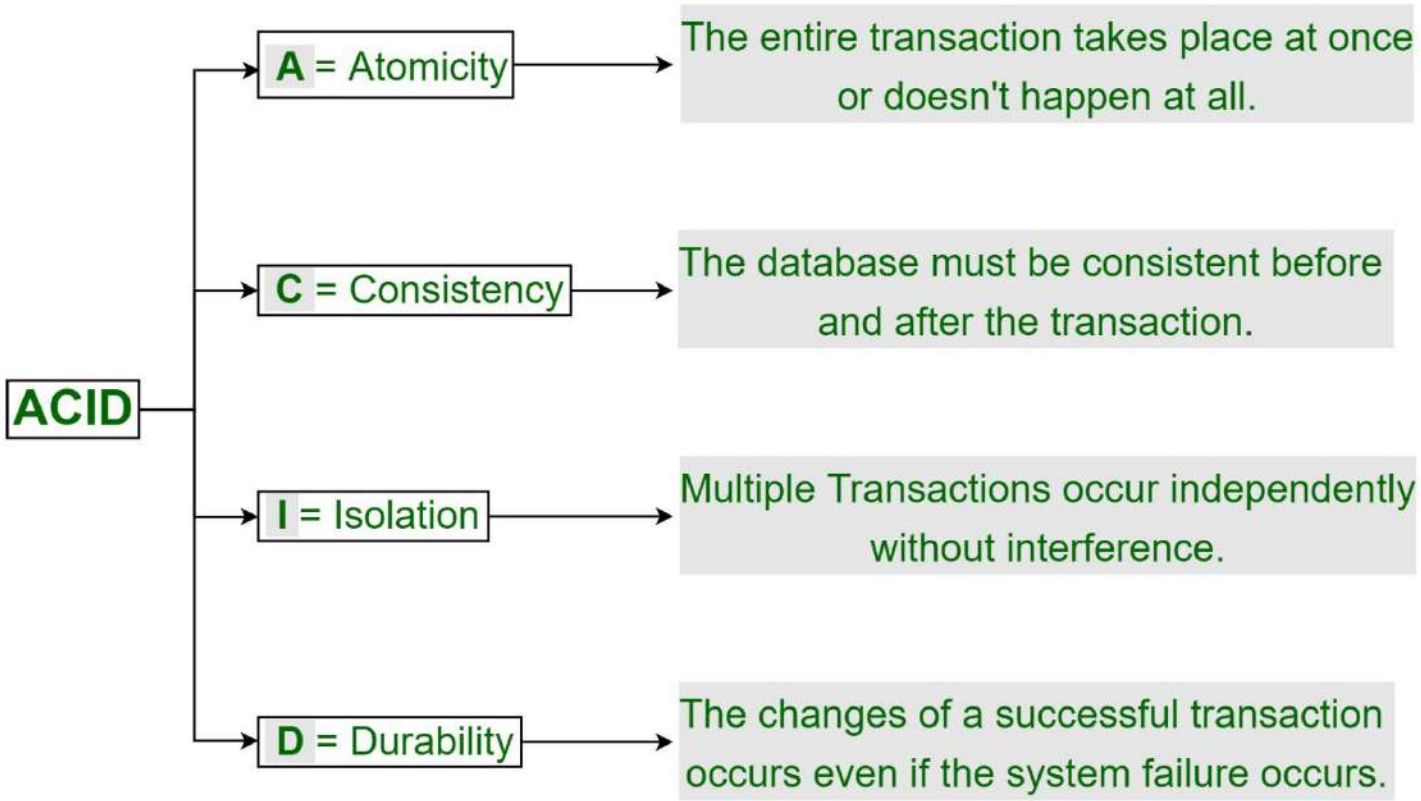
- **Relational databases** built around relational algebra and tuple relational calculus and are optimized for writes, consistency and availability.
- **Advantages**:
  - Simplicity, ease of data retrieval, data integrity,
  - Single uniform language (DDL) for different roles (developer, user, DBA).
  - Single standardized language for different RDBMS.
  - sticks to ACID principles (atomicity, consistency, isolation, durability), and thus ensures stability, security, and predictability both for the entire database and every individual transaction

ACID Properties in DBMS

- A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

ACID Properties in DBMS



Advantages and the Disadvantages of Relational and Non-Relational Databases

- **Relational databases**
- **Disadvantages:**
  - expensive to set up and maintain
  - have limits to field lengths. This can be cumbersome for storing a large amount of information in one field.
  - very difficult to scale as much as a database grows larger
  - Multiple databases can easily become islands of information and difficult to connect the databases where they can talk to each other.

Advantages and the Disadvantages of Relational and Non-Relational Databases

- **Non-relational databases** are schema-free and built on distributed systems, which makes it easy to scale and shard. These are optimized for reads.
- **Advantages:**
  - Flexible to store large volumes of structured, semi-structured, and unstructured data.
  - Can scale out architecture efficiently without expensive overhead.
  - Available as open source and free of cost in many cases

Advantages and the Disadvantages of Relational and Non-Relational Databases

- **Disadvantages**:
  - ACID transactions are not supported by non-relational databases. Instead, they rely on “eventual consistency”. The performance benefits of these databases mean there’s a cost of consistency.
  - There are many types of NoSQL databases but there is almost no uniformity among them.
  - No specific programming interface to the different databases. Each one varies in query language with another.
  - Not all non-relational databases are good at automating the process of sharding, or spreading the database across multiple nodes.

Contents to be covered

- Introduction to NoSQL Databases
- NoSQL features,
- Different types of NoSQL databases
- Introduction to MongoDB,
- MongoDB architecture,

NoSQL

- **NoSQL** is a non-relational DBMS, that does not require a fixed schema, avoids joins, and is easy to scale. The purpose of using a NoSQL database is for distributed data stores with large amount of data storage needs.
- NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook, 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.

RDBMS

Student name	Rollno	Phone number
Aaa	1	9856475896
Bbb	2	5646586465

Rollno	Name	marks
1	Aaa	45
2	Bbb	56

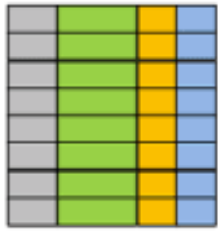
NoSQL

```
{
    name:”aaa”,
    Rollno:1,
    Phoneno.:34534534554,
    Academic:  {
                marks:45 },
    }
```

NoSQL

- Traditional RDBMS uses SQL syntax to store and retrieve data for further insights.
- Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

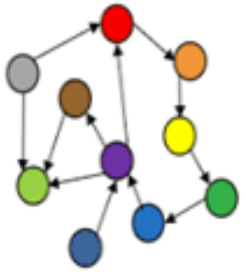
Relational



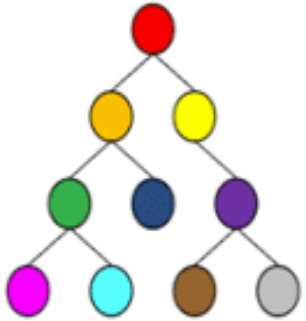
NoSQL Databases

Column-Family

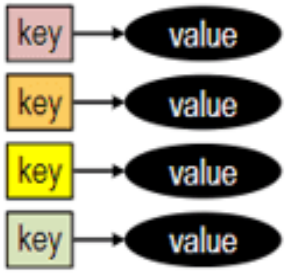
Graph



Document



Key-Value



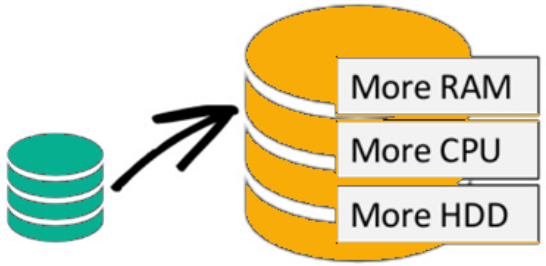
Why NoSQL?

- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

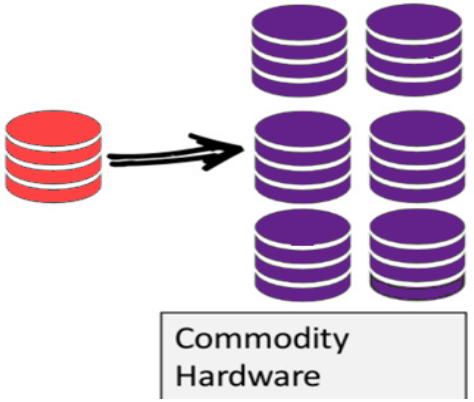
Why NoSQL?

- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out.“

Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):



Why NoSQL?

- NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

Features of NoSQL

- **Non-relational**
  - NoSQL databases never follow the relational model
  - Never provide tables with flat fixed-column records
  - Doesn't require data normalization
  - No complex features like query languages, query planners,referential integrity joins, ACID

Rollno	Name	Marks	Phone number
1	Rahul	45	9888254586, 4568954254
2	Smita	56	9856745896, 2568935984

Rollno	Name	Marks	Phone number
1	Rahul	45	9888254586
1	Rahul	45	4568954254
2	Smita	56	9856745896
2	Smita	56	2568935984

Features of NoSQL

- **Schema-free**
  - NoSQL databases are either schema-free or have relaxed schemas
  - Do not require any sort of definition of the schema of the data
  - Offers heterogeneous structures of data in the same domain

Features of NoSQL

- **Simple API**
  - Offers easy to use interfaces for storage and querying data provided
  - APIs allow low-level data manipulation & selection methods
  - Mostly used no standard based query language

Features of NoSQL

- **Distributed**
  - Multiple NoSQL databases can be executed in a distributed fashion
  - Offers auto-scaling
  - Often ACID concept can be sacrificed for scalability and throughput

Types of NoSQL databases

- There are mainly four categories of NoSQL databases. Each of these categories has its unique attributes and limitations. No specific database is better to solve all problems. select a database based on your product needs.
  - Key-value Pair Based
  - Column-oriented Graph
  - Graphs based
  - Document-oriented

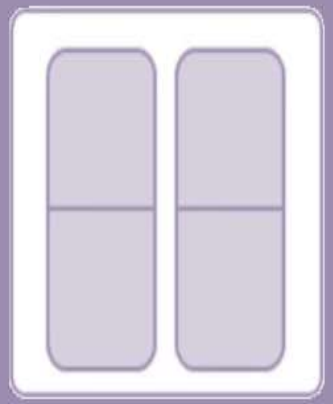
Key Value

Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based

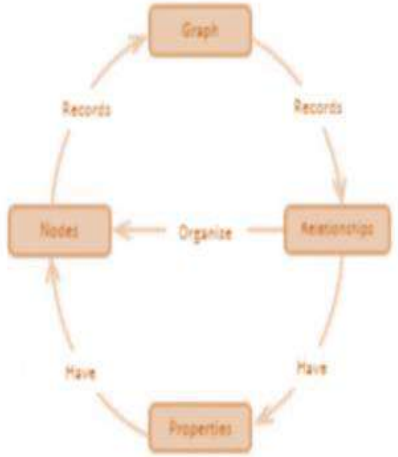
Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based



Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based



Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Types of NoSQL databases

- **Key Value Pair Based**
  - Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
  - Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
  - For example, a key-value pair may contain a key like “student name" associated with a value like “rahul".

Key Value Pair Based

- It is one of the most basic types of NoSQL databases. This kind of NoSQL database is used as a collection, dictionaries etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
- Shopping cart

  {
  product name : soap
  Product qty : 5
  Product price : 15 }
- Redis, Dynamo, Riak are some examples of key-value store Databases.

Types of NoSQL databases

- **Column-based**
  - Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.

Column-based

Column family			
Row key	Column name		
	Key	Key	Key
	Value	Value	Value
	Column name		
	Key	Key	Key
	Value	Value	value

student					
ID:1	Personal data				
	Name	Gender	age	Address	Phone no.
	Rahul	Male	23	dsfsd	895768
	Academic record				
	First semester	Second semester	Third semester	total	
	75	85	56	Sum(	
ID:2	Personal data				
	Name	Gender	age		
	Raj	Male	34		
	Academic record				
	First semester	Second semester	Third semester		
	75	45	59		

Column-based

- They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.
- Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, Library card catalogs,
- HBase, Cassandra, Hypertable are examples of column based database.

C language				
Name	Author	Price	Qty	Id
let us c	Yashwant	88	5	A01
C++				
Name	AUTHOR	QTY	ID	
C++	ROBERT	4	dasfaf	

Types of NoSQL databases

- **Document-Oriented:**
  - Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

Document-Oriented:

RDBMS

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Name	Age	Designation	salary	incentive
Rahul	23	Manager	2000	
Vijay	25	Officer	3000	
Raj	25	Account officer	4000	4000

Document1

{
Name: “rahul”,
Age:23,
Designation:”manager”,
Salary:2000
}

Document2

{
Name: “vijay”,
Age:25
Designation:
“officer”,
Salary:3000
}

Document3

{
Name:”raj”
Age:25
Designation:”account officer”
Salary:4000
Incentive:4000
}

Relational Vs. Document

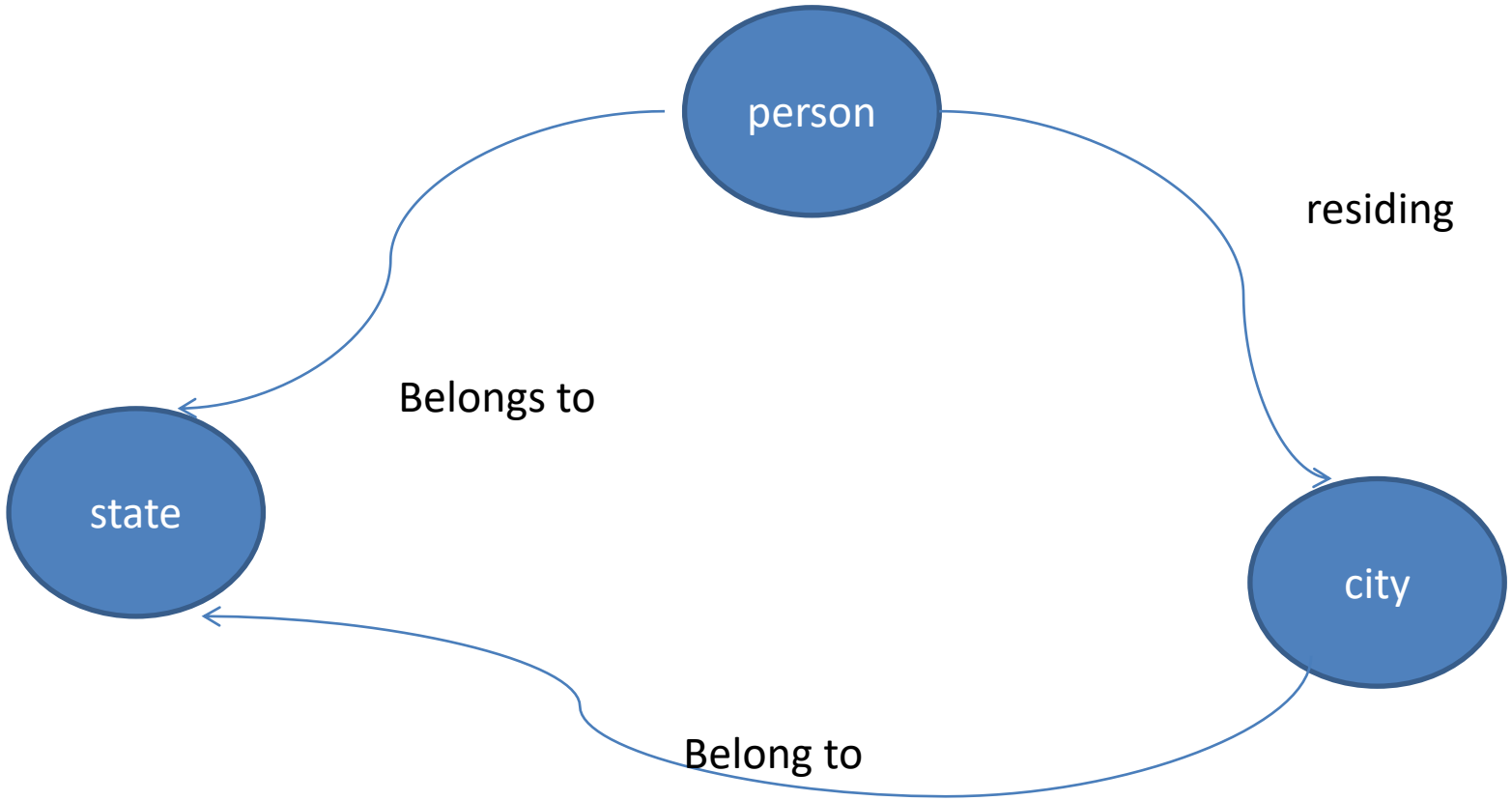
- In the left diagram there are rows and columns, and in the right, a document database which has a similar structure to JSON.
- Now for the relational database, there is the need to know what columns are there and so on. However, for a document database, there is data store like JSON object. There is no requirement to define which make it flexible.

Relational Vs. Document

- The document type is mostly used for blogging platforms & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.
- Amazon SimpleDB, CouchDB, MongoDB, Lotus Notes are popular Document originated DBMS systems.

Graph-Based

- A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.
- Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.


person
residing
Belongs to
state
city
Belong to

Introduction to MongoDB,

- MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents.
- Documents consist of key-value pairs which are the basic unit of data in MongoDB.
- Collections contain sets of documents and function which is the equivalent of relational database tables.
- MongoDB is a database which came into light around the mid-2000s

MongoDB Features

- Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.

MongoDB Features

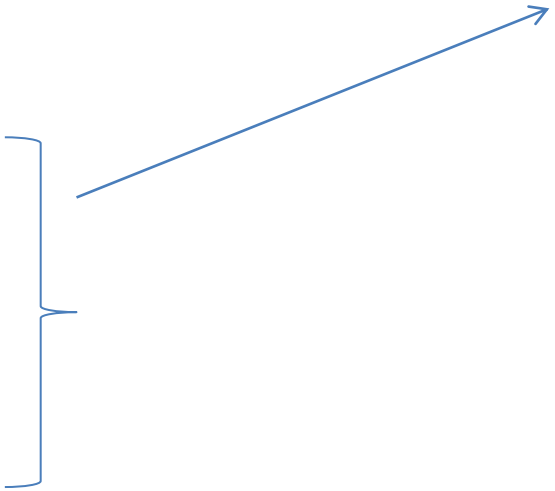
- The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created as needed
- The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
- Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database

MongoDB Example

- The below example shows how a document can be modeled in MongoDB.
  - The _id field is added by MongoDB to uniquely identify the document in the collection.
- the Order Data (OrderID, Product, and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in MongoDB.

document customer

```
{
_id:1,
CustomerName:indu
Order:
        {
        OrderID:111
        Product:book
        Quantity:5
        }
}
```



Embedded document

student

```
{
    Name:rahul
    Age:23
    Rollno:12
    Marks:
        {  eng:34
           hindi:45
           math:50
        }
    }
```

Key Components of MongoDB Architecture

- **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDBMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.
- **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
- **Database** – This is a container for collections like in RDBMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.

Key Components of MongoDB Architecture

- **Document** - A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
- **Field** - A name-value pair in a document. A document has zero or more fields. Fields are similar to columns in relational databases.
- The diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair's defined in the document.
- Data stored as key-value pair

Key Components of MongoDB Architecture

```
{       CustomerID:111,
        CustomerName: “indu”
        OrderID: 111
}
```

- **JSON** – This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.
- The _id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

Contents to be covered

- Data modelling in MongoDB,
- Advantages of MongoDB over RDBMS,
- Mongo Shell,
- Configuration file in MongoDB,

Data Modelling in MongoDB

- As we have seen from the Introduction section, the data in MongoDB has a flexible schema.
- in SQL databases, where you must have a table's schema declared before inserting data, MongoDB's collections do not enforce document structure.
- This sort of flexibility is what makes MongoDB so powerful.

Data Modelling in MongoDB

- When modeling data in Mongo, keep the following things in mind
  - **What are the needs of the application** – Look at the business needs of the application and see what data and the type of data needed for the application. Based on this, ensure that the structure of the document is decided accordingly.
  - **What are data retrieval patterns** – If you foresee a heavy query usage then consider the use of indexes in your data model to improve the efficiency of queries.
  - Are frequent inserts, updates and removals happening in the database? Reconsider the use of indexes or incorporate sharding if required in your data modeling design to improve the efficiency of your overall MongoDB environment.

Difference between MongoDB & RDBMS

Below are some of the key term differences between MongoDB and RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	Embedded documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in MongoDB.

Difference between MongoDB & RDBMS

- Apart from the terms differences, a few other differences are shown below
  - Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB.
  - RDBMS requires that data be normalized first so that it can prevent orphan records and duplicates .Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes.
  - As databases start to grow, performance can start becoming an issue. Again this is not an explicit requirement in MongoDB. MongoDB is flexible and does not need the data to be normalized first.

Name	Age	Address	Gender	Mobile
Aaa	45	Hghjghj	F	985632147
Aaa	45	Hghjghj	F	845759635

Advantages of MongoDB over RDBMS

- **Ease of scale-out** – MongoDB is easy to scale.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

MongoDB

- MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.
- **Database**
  - Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

MongoDB

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

MongoDB

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

- Student collections
- {“name”:”aaa”,”rollno”:1,”marks”:45}
- {“name”:”bbb”,”rollno”:2,”hobbies”:”playing cricket”}

The Following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Sample Document

Following example shows the document structure , which is simply a comma separated key value pair.

{

“_id”:”34893483478965767ashdhsda”,

“name”:”abc”,

“rollno”:1,

“phone number”:6535675324,

“Marks”:

{ “eng”:45,”maths”:67,”science”:67}

}

ID field

- **_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document.
- These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.
- 12 bytes=8*12 bits=96 bits / 4=24 digit hexadecimal
- When you convert 12 bytes to hexadecimal it becomes 24 digit number
- 48/4=12

Installtion of mongodb

- Install MongoDB Community Edition
- **Procedure**
  - Follow these steps to install MongoDB Community Edition using the MongoDB Installer wizard. The installation process installs both the MongoDB binaries as well as the default configuration file <install directory>\bin\mongod.cfg.
- **Download the installer.**
  - Download the MongoDB Community .msi installer from the following link:

Installtion of mongodb

➤ go to website mongodb.com

- In the **Version** dropdown, select the version of MongoDB to download.
- In the **Platform** dropdown, select **Windows**.
- In the **Package** dropdown, select **msi**.
- Click **Download**.

Installtion of mongodb

Run the MongoDB installer.

For example, from the Windows Explorer/File Explorer:

Go to the directory where you downloaded the MongoDB installer (.msi file). By default, this is your Downloads directory.

Double-click the .msi file.

Follow the MongoDB Community Edition installation wizard.

Installtion of mongodb

- The wizard steps you through the installation of MongoDB and MongoDB Compass.
- **Choose Setup Type**
  - You can choose either the **Complete** (recommended for most users) or **Custom** setup type. The **Complete** setup option installs MongoDB and the MongoDB tools to the default location. The **Custom** setup option allows you to specify which executables are installed and where.
- **Service Configuration**
  - Starting in MongoDB 4.0, you can set up MongoDB as a Windows service during the install or just install the binaries.

Installtion of mongodb

- **MongoDB Service**
  - Starting in MongoDB 4.0, you can configure and start MongoDB as a Windows service during the install, and the MongoDB service is started upon successful installation.

Installtion of mongodb

- Select **Install MongoD as a Service**

MongoDB as a service.

  - Select either:
    - **Run the service as Network Service user** (Default)
      - This is a Windows user account that is built-in to Windows

  or

    - **Run the service as a local or domain user**
      - For an existing local user account, specify a period (i.e. .) for the **Account Domain** and specify the **Account Name** and the **Account Password** for the user.

Installtion of mongodb

- For an existing domain user, specify the **Account Domain**, the **Account Name** and the **Account Password** for that user.

- **Service Name**. Specify the service name. Default name is MongoDB. If you already have a service with the specified name, you must choose another name.
- **Data Directory**. Specify the data directory, which corresponds to the --dbpath. If the directory does not exist, the installer will create the directory and sets the directory access to the service user.
- **Log Directory**. Specify the Log directory, which corresponds to the --logpath. If the directory does not exist, the installer will create the directory and sets the directory access to the service user.

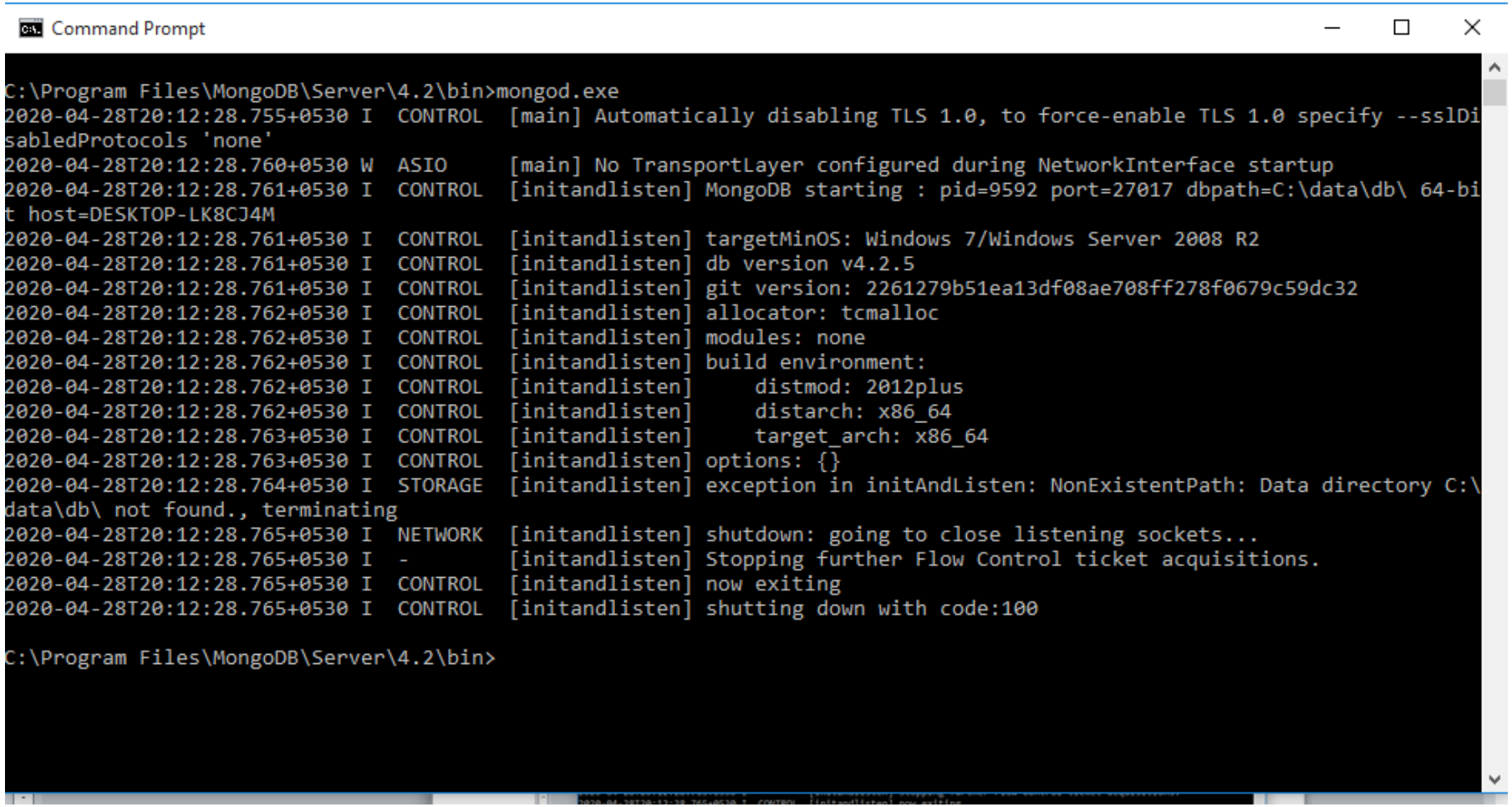
Installtion of mongodb

- **Install MongoDB Compass**
  - *Optional*. To have the wizard install MongoDB Compass, select **Install MongoDB Compass** (Default).
  - When ready, click **Install**.

Mongo Shell

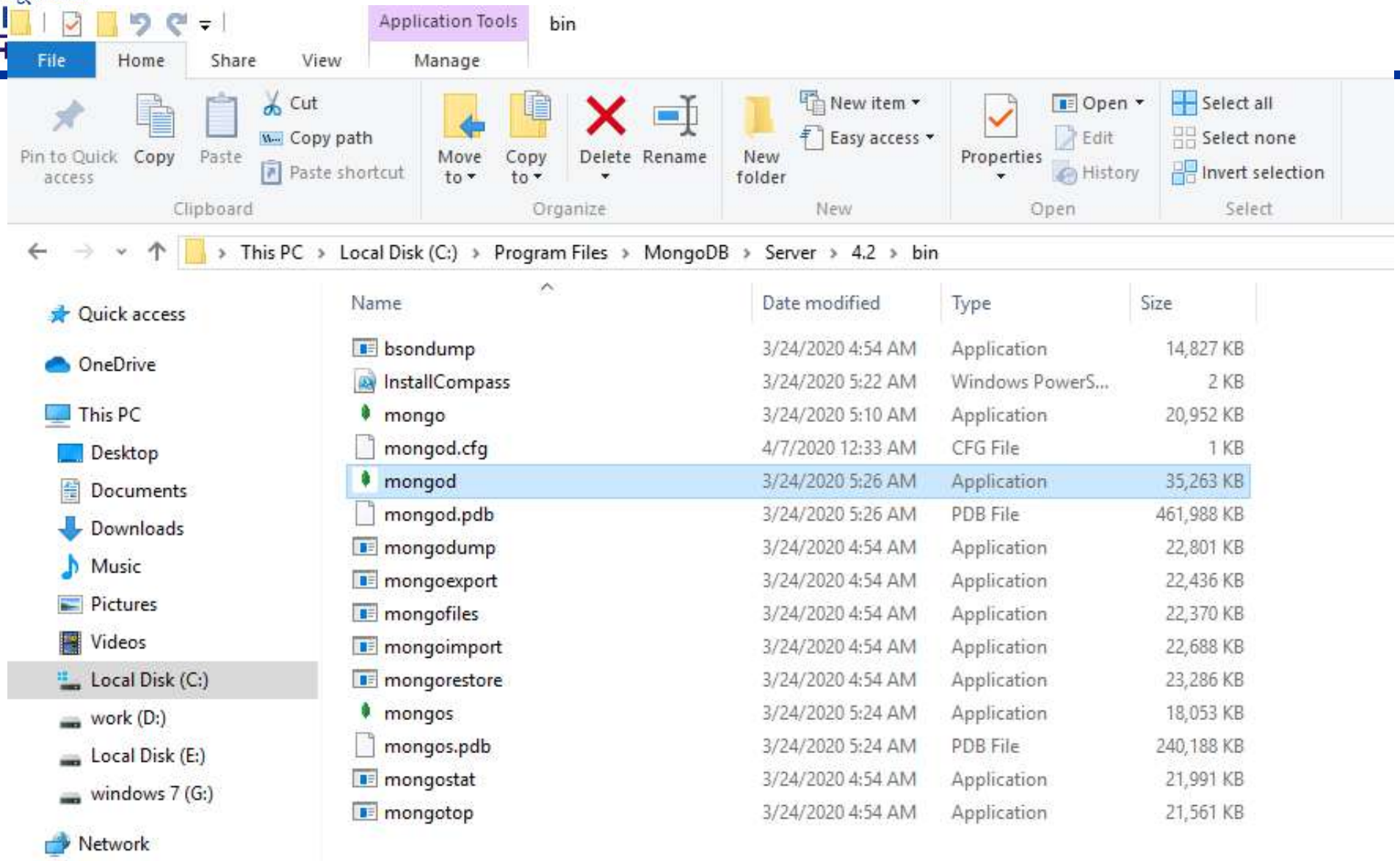
- **Mongo shell or MongoDB Shell** is an interactive JavaScript interface to MongoDB. It is used to perform administrative operations and also query and update data.
- The mongo shell is as part of the MongoDB Server installation.
- Mongo Shell is also provided as a standalone package by MongoDB.
- Prior to connecting to MongoDB one has to ensure that MongoDB is running.

To start MongoDB, Run the following command



OR

- Run the **mongod.exe** file on command prompt (path of the exe file is C:\Program Files\MongoDB\Server\4.2\bin)

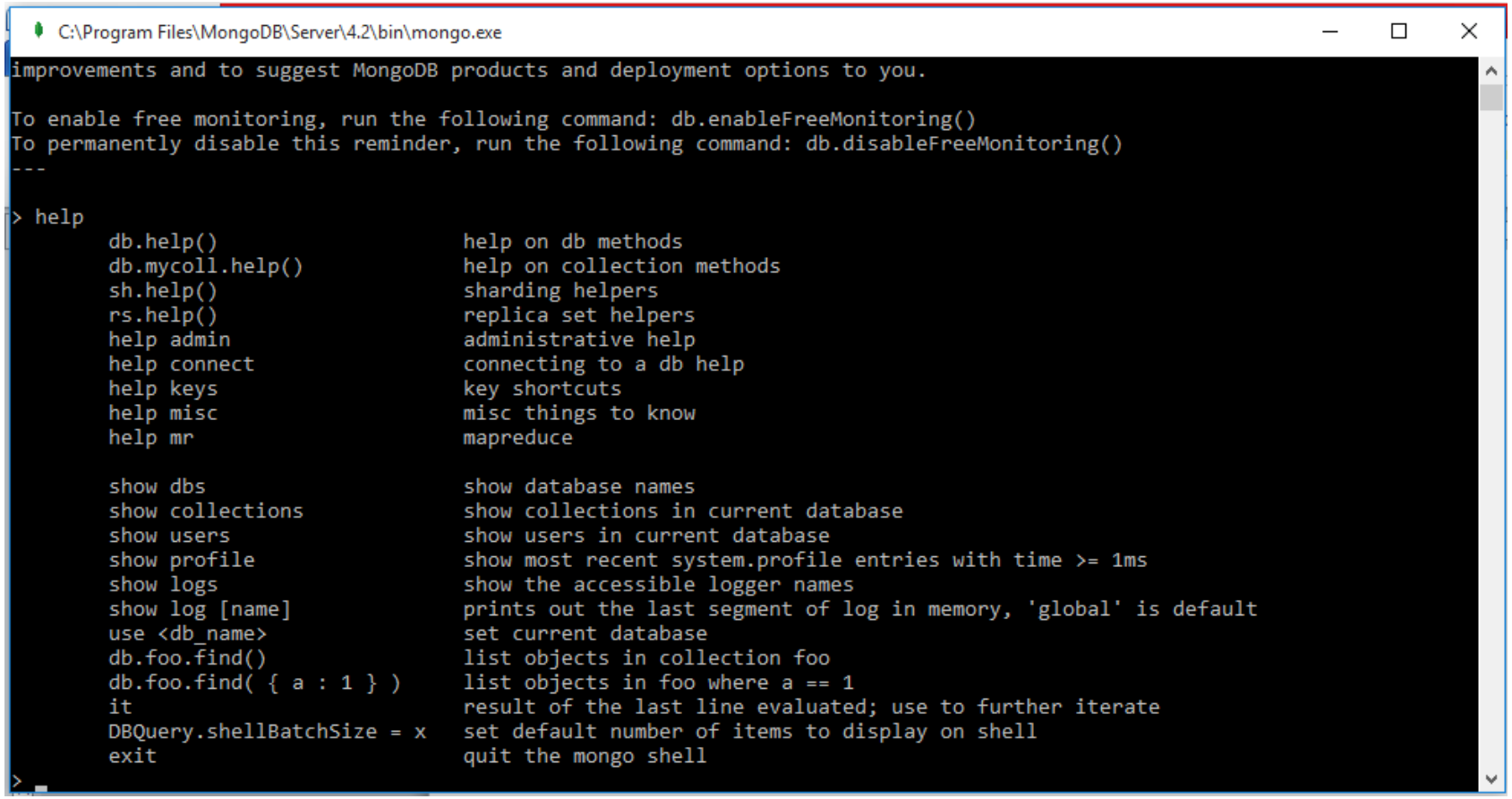

Application Tools
bin
File
Home
Share
View
Manage
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC › Local Disk (C:) › Program Files › MongoDB › Server › 4.2 › bin
Quick access
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
work (D:)
Local Disk (E:)
windows 7 (G:)
Network
Name
Date modified
Type
Size
bsondump 3/24/2020 4:54 AM Application 14,827 KB
InstallCompass 3/24/2020 5:22 AM Windows PowerS... 2 KB
mongo 3/24/2020 5:10 AM Application 20,952 KB
mongod.cfg 4/7/2020 12:33 AM CFG File 1 KB
mongod 3/24/2020 5:26 AM Application 35,263 KB
mongod.pdb 3/24/2020 5:26 AM PDB File 461,988 KB
mongodump 3/24/2020 4:54 AM Application 22,801 KB
mongoexport 3/24/2020 4:54 AM Application 22,436 KB
mongofiles 3/24/2020 4:54 AM Application 22,370 KB
mongoimport 3/24/2020 4:54 AM Application 22,688 KB
mongorestore 3/24/2020 4:54 AM Application 23,286 KB
mongos 3/24/2020 5:24 AM Application 18,053 KB
mongos.pdb 3/24/2020 5:24 AM PDB File 240,188 KB
mongostat 3/24/2020 4:54 AM Application 21,991 KB
mongotop 3/24/2020 4:54 AM Application 21,561 KB

Running Mongo Shell

- Once you are sure that MongoDB is running,
- Run the **mongo.exe** file from the same path (C:\Program Files\MongoDB\Server\4.2\bin) from command prompt or using Run or using Windows explorer.
- **Once executed, the Mongo Shell will open. Now you can execute desired commands.**

Example: Running Help Command on the Mongo Shell

Various help options will be displayed as shown in the figure below:



Configuration file in MongoDB

- When we install the full version of MongoDB, configuration file **mongod.cfg** is automatically created with default setting and is stored in bin folder of the MongoDB server.
- We have installed MongoDb on C:\ drive at the default path and the location of the configuration file is **C:\Program Files\MongoDB\Server\4.2\bin .**

Configuration file in MongoDB

- If we open this file, the content of the file is as under:

```
# mongod.conf
 # for documentation of all options, see:
#http://docs.mongodb.org/manual/reference/configuration-options/
# Where and how to store data.
storage:
dbPath: C:\Program Files\MongoDB\Server\4.2\data
journal:
enabled: true
# engine:
# mmapv1:
# wiredTiger:
```

Configuration file in MongoDB

```
# where to write logging data.
systemLog:
destination: file
 logAppend: true
path: C:\Program Files\MongoDB\Server\4.2\log\mongod.log
 #processManagement:
 #security:
 #operationProfiling:
 #replication:
 #sharding:
```

Configuration file in MongoDB

```
# network interfaces
net:
port: 27017
bindIp: 127.0.0.1
 ## Enterprise-Only Options:
 #auditLog:
 #snmp:
```

We may edit the path and settings or create our own configuration file with these settings. Storage path and system log path is mandatory.

Contents to be covered

- JSON File format for storing documents
- Introduction to Documents, Collections,
- Database Commands in Mongodb,
- Update and save method
- Quering documnets

JSON File format for storing documents

- JSON stands for ***JavaScript Object Notation. Json*** is data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. We call JSON as syntax for storing and exchanging data.
- JSON is a text format which is completely language independent and written with JavaScript object notation. JSON uses conventions that are familiar to programmers of the C- family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.
- These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- As a collection of name/value or Attribute/Value pairs. This may be an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. This may be an array, vector, list, or sequence.
- JSON is universal data structures which are virtually supported by all modern programming in one form or another. These JSON structure supports the data format which is interchangeable with programming languages.

JSON

- **Example 1:** An *object* is an unordered set of name/value pairs. In JSON, an object starts with **{ *left brace*** and ends with **} *right brace*.** Each name is followed by **:*colon*** and the name/value pairs are separated by **,*comma*.**
- {"name": "Ajay", "age": 21, "city": "Gorakhpur" }
- { "name": "Mohit", "age": 22, "city": "Deoria" }
- { "name": "Jeetut", "age": 20, "city": "Sonauli" }

JSON

- **Example 2 :** An array is an ordered collection of values. An array begins with ***[left bracket*** and ends with ***] right bracket***. Here, values are separated by ***,comma***. Here comes some arrays:

  [“Graduation”, “Post Graduation”, “NIELIT Course”]

  [“BSc”, “BCom”, “BA”, “BBA”, “BCA”]

  [“MSc”, “MCom”, “MA”, “MBA”, “MCA”]

  [“CCC”, “O Level IT”, “A Level IT”, “CHM O Level”, “MAT O Level”]

JSON File format for storing documents

Example 3 :

- Array of Objects

```
[ {"name": "Ajay", "age": 21, "city": "Gorakhpur" },
{ "name": "Mohit", "age": 22, "city": "Deoria" },
{ "name": "Jeetut", "age": 20, "city": "Sonauli" } ]
```

Exchanging Data with JSON File

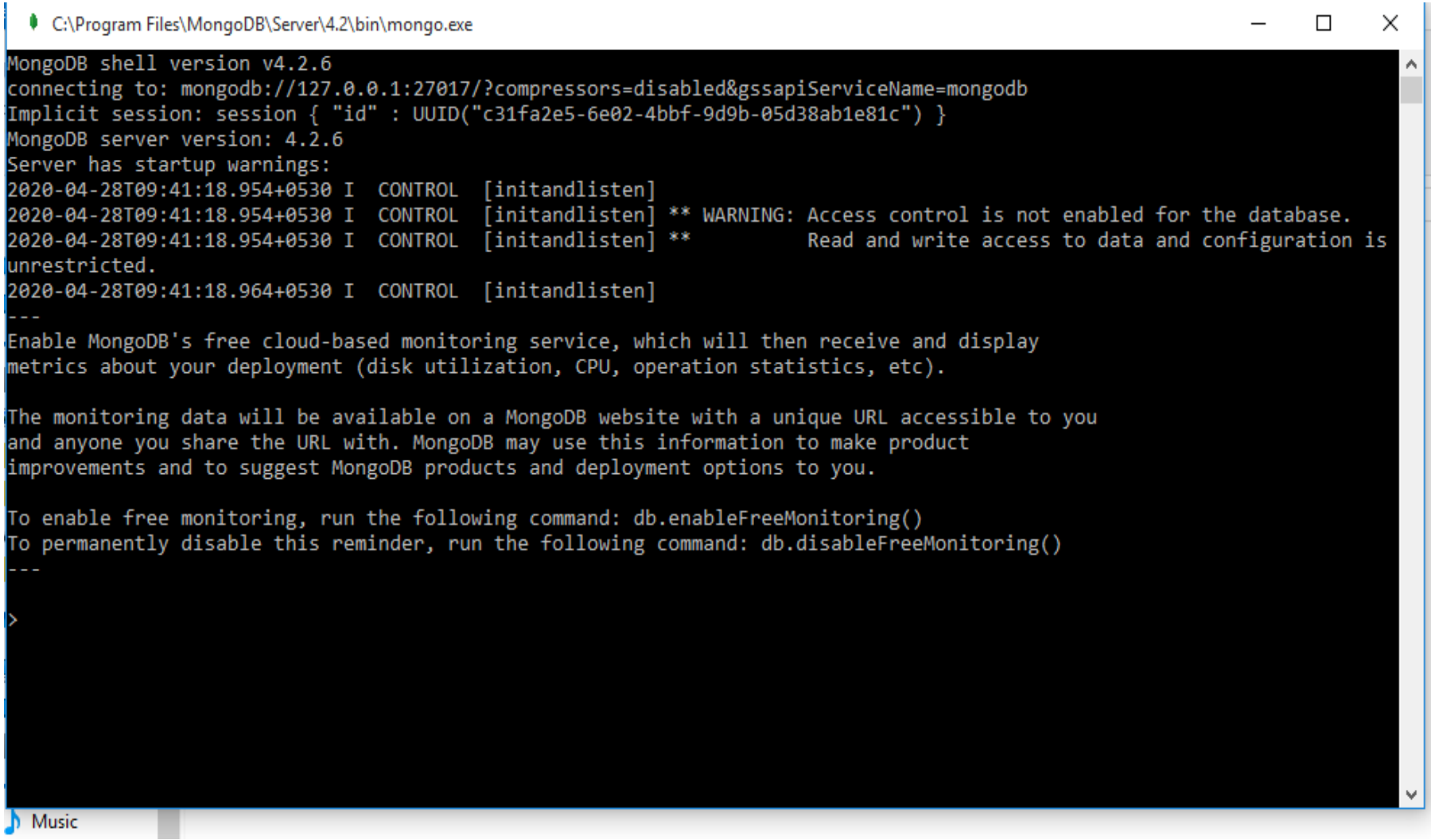
- It is worth to mention here that exchange of data between a browser and a server can only be text.
- JSON is text format and hence we can convert any JavaScript object into JSON, and send JSON to the server.
- Similarly, we may also convert any JSON received from the server into JavaScript objects.
- With this technique, we may work with the data with JSON file as JavaScript objects, with no complicated parsing and translations.

Introduction to Documents, Collections,

- MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents.
- **Databases**
  - In MongoDB, databases hold one or more collections of documents.
- **Collections**
  - MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

Commands in MongoDB

- All the commands will be written in MongoDB shell. Alternatively the operations may be executed in Compass GUI. We are using commands in shell.
- OPEN mongo shell by using the shell application file **mongo.exe** from at C:\Program Files\MongoDB\Server\4.2\bin (or the directory you have specified)
- Mongo screen will open


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
MongoDB shell version v4.2.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("c31fa2e5-6e02-4bbf-9d9b-05d38ab1e81c") }
MongoDB server version: 4.2.6
Server has startup warnings:
2020-04-28T09:41:18.954+0530 I CONTROL [initandlisten]
2020-04-28T09:41:18.954+0530 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-04-28T09:41:18.954+0530 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2020-04-28T09:41:18.964+0530 I CONTROL [initandlisten]

Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

>
Music

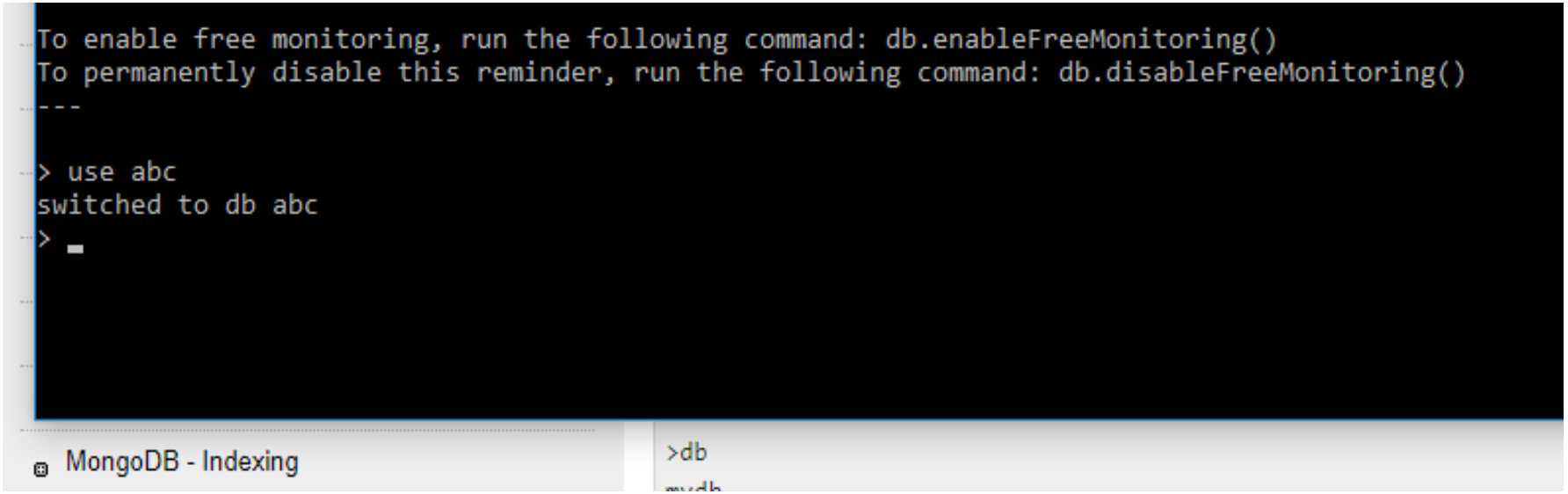
Create database

- All commands can be written and executed on this MongoDB Shell prompt.
  - **use DATABASE_NAME** command is used to create database.
  - This will open/ or switch the specified database if exists otherwise will create a database with the given name and switch to that database.
- **Syntax**

  use DATABASE_NAME
- **Example:**

  Use abc

Create database



Show database

- **Show database (show dbs) command**
  - To display the list of databases, show command is used,
- **Syntax**

  Show dbs

  - This will list all the non-empty databases.
  - To display database, you need to insert at least one document into it.
  - Just created database (abc) is not present in list shown below as it is not having any document in it.

Show database

```
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> use abc
switched to db abc
> show dbs
abd     0.000GB
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

The dropDatabase() Method

- MongoDB **db.dropDatabase()** command is used to drop a existing database.
- **Syntax**
- Basic syntax of **dropDatabase()** command is as follows –

  db.dropDatabase()
- This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Create Collection command

- createCollection() method is used to create collection in the selected database.

Syntax

db.createCollection(name, options)

Example

To run this method / command database must be in use. We are using our newly created **abc** database.

db.createCollection("college")

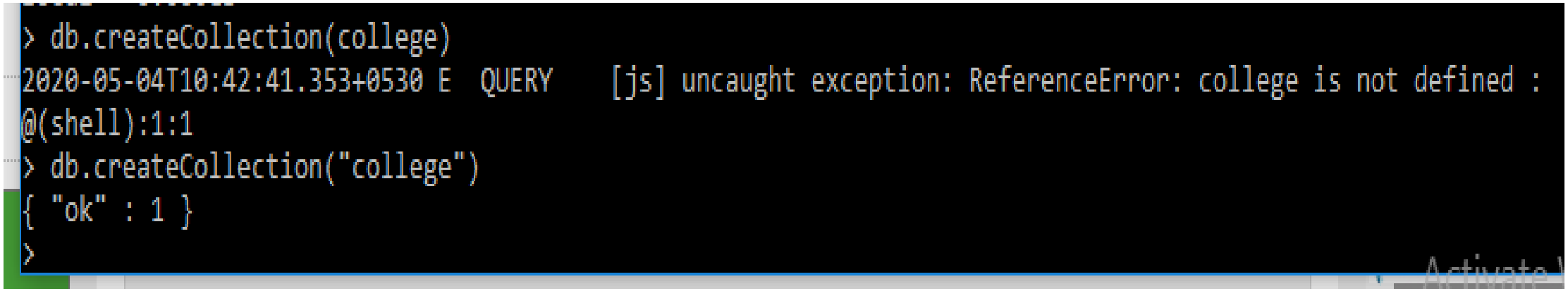
this will result status as

```
{ "ok" : 1 }
```

Create Collection command

Parameter	Type	Description
Name	String	Name parameter specifies the name of the collection to be created. It is of String type and should be specified in quotes.
Options (it is optional parameter)	Document	Options parameter is used to specify options about memory size and indexing. This parameter is optional.

Create Collection command



Create Collection command

- **Note:** In the above screen, 1st we write collection name without quotes **db.createcollection(college)** which is wrong and in that case some exceptions has been raised.
- ***In MongoDB, collection creation is not necessary before inserting document. MongoDB creates collection automatically, when you insert some document.***

The drop() Method

- MongoDB's **db.collection.drop()** is used to drop a collection from the database.
- Syntax
- Basic syntax of **drop()** command is as follows –
- db.COLLECTION_NAME.drop()

Show collections command

- **Show collections command**
  - **Show collections** is used to display / list the created collection in the selected database.
- **Syntax**

  Show collections
- If we run this command, it will display our recently created collection **“college”.**

Show collections command

```
> show collections
college
```

MongoDB supports many datatypes. Some of them are –

- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** – This type is used to store a boolean (true/ false) value.
- **Double** – This type is used to store floating point values.

MongoDB supports many datatypes. Some of them are –

- **Min/ Max keys** – This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** – This type is used to store arrays or list or multiple values into one key.
- **Object** – This datatype is used for embedded documents.

Inserting and Saving Documents

- **Insert() command**
  - Insert command is used to insert document in form of name/value combination to the specified collection. If collection is not already created, it will create the specified collection.
- **Syntax**

db.COLLECTION_NAME.insert (document)

Inserting and Saving Documents

where

- **COLLECTION_NAME** is the name of the collection in which document has to be inserted.
- **document** is document(key/value combination) or array of documents to insert into the collection.

Inserting and Saving Documents

_id Field

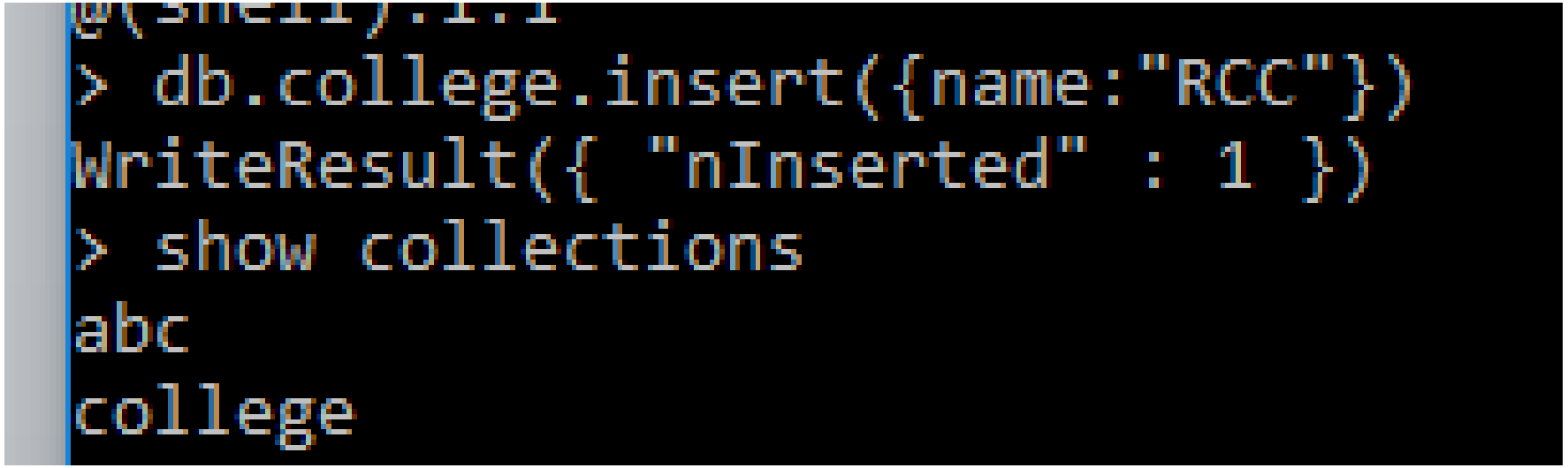
- We may give an **_id** filed while inserting the document and this shall be unique within the collection to avoid duplicate key error.
- **If no _id field is specified, then MongoDB will add the** _id **field and assign a unique** ObjectId **to the document**

Example-1:

```
db.college.insert({"name" : "RCC"})
```

This will show the following result after successful execution of the command:

```
WriteResult({ "nInserted" : 1 })
```


> db.college.insert({name:"RCC"})
WriteResult({ "nInserted" : 1 })
> show collections
abc
college

Example-2:

db.college.insert({course:"A level",duration:"2 years", fees:"12000"})

This will show the following result after successful execution of the command:

WriteResult({ "nInserted" : 1 })

Example-2:

```
> db.college.insert({course:"A level",duration:"2 years",fees:"12000"})
WriteResult({ "nInserted" : 1 })
```

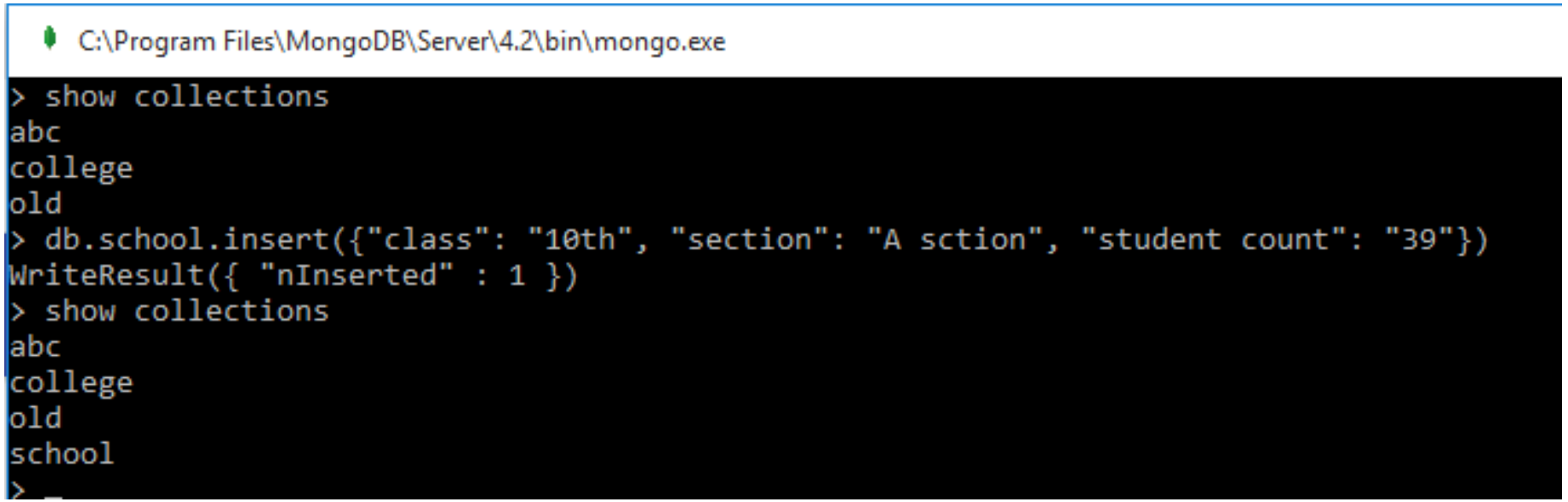
Example-3:

- Lets insert a document into a collection which has not been created. For example let collection name as **school**.

```
db.school.insert({"class": "10th", "section": "A section", "student count": "39"})
```

- Once this command is executed, a new collection named as school will be automatically created.

Example-3:


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> show collections
abc
college
old
> db.school.insert({"class": "10th", "section": "A sction", "student count": "39"})
WriteResult({ "nInserted" : 1 })
> show collections
abc
college
old
school
>

The insertOne() method

- If you need to insert only one document into a collection you can use this method.

- Syntax
- The basic syntax of insert() command is as follows –

- >db.COLLECTION_NAME.insertOne(document)

Batch insert or Inserting Multiple documents into a collection

To Insert multiple documents or Batch insert into a collection **insertMany()** method is available to do this task.

Syntax:

```
db.collection.insertMany (
   [ <document 1> , <document 2>, ... ]
   )
```

Where

Each of these document should be separated by a **, comma**

Batch insert or Inserting Multiple documents into a collection

- [<document 1> , <document 2>, ...] is an array of the various s documents which needs to be inserted into the collection.
- **On Successful execution, it returns a**
  - A boolean acknowledged as true if the operation is successful or false if uncessful.
  - An array of _id for each successfully inserted documents

The insertMany() method

- You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.
- Example
- Following example inserts three different documents into the empDetails collection using the insertMany() method.

Read Operations

- Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:
- db.collection.find()

Find() command

- Find () command is used to see the records/documents in the specified collection.

- **Syntax**

db.COLLECTION_NAME.find()

Example:

db.college.find()

Find example

- Lets insert a record with specified **_id**

  db.school.insert({"_id: "01", "class": "10th", "section": "B section", "student count": "27"})

find example:

```
>
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
>
>
> db.school.insert({"_id":"01","class":"10th","section":"B Section","student count":"27"})
WriteResult({ "nInserted" : 1 })
>
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
{ "_id" : "01", "class" : "10th", "section" : "B Section", "student count" : "27" }
>
```

- Now a record with **_id** as **01** (specified in insert command) has been inserted in the collection. Find command used here has been described below.

The pretty() Method

- To display the results in a formatted way, you can use pretty() method.
- Syntax
- >db.COLLECTION_NAME.find().pretty()

The findOne() method

- Apart from the find() method, there is findOne() method, that returns only one document.

- Syntax
- >db.COLLECTIONNAME.findOne()

Update and save

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

- MongoDB Update() Method
- The update() method updates the values in the existing document.

- Syntax
- The basic syntax of update() method is as follows −

- >db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

Save() command

- Save() command may also be used to insert the document. It has the same syntax as of insert() command.
- **Syntax**

  db.college.save(document)

  – If **_id** is not specified in the document then **save()** method will work same as **insert()** method.

  – If **_id** feild is specified then it will replace whole data of document having the _id as specified in save() method if matched else Insert the document into the collection.

Example-1

- Lets save a record in the school collection with **_id** as **02,** and then display the documents in the collection.

db.school.save({"_id":"02","class":"10th","section":"C Section","student count":"37"})

this will display

WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : "02" })

It will display details like "nMatched" : 0, i.e. no record is matched with the specified _id, so a new record is "nUpserted" : 1 with , "_id" : "02".

Example-2

Lets try to save a record in the school collection with **_id** as **01,** and then display the documents in the collection.

db.school.save({"_id":"01","class":"10th","section":"D Section","student count":"29"})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

It will display details like "nMatched" : 1 which matched with the specified _id, and the same is modified "nModified" : 1.

Quering documents

- Find is used to search for documents
- It can be based on some criteria
- syntax

db.collection.find(criteria)

Eg

db.student.find({name:”tina”})

Select All Documents in a Collection

- To select all documents in the collection, pass an empty document as the query filter parameter to the find method. The query filter parameter determines the select criteria:
- db.employee.find({})

Specify Equality Condition

- To specify equality conditions, use <field>:<value> expressions in the query filter document:

- db.employee.find({ designation: “manager" })

comparison query operators for find

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

Specifies equality condition.

- The $eq operator matches documents where the value of a field equals the specified value.

  { <field>: { $eq: <value> } }

- To find all the doucments where marks is equal to 50
- db.student.find({ marks: { $eq: 50 } })

$gt

- $gt selects those documents where the value of the field is greater than (i.e. >) the specified value.
- *syntax*:

  {field: {$gt: value} }
- Eg:
- db.student.find({ marks: { $gt: 20 } })

$gte

- $gte selects the documents where the value of the field is greater than or equal to (i.e. >=) a specified value (e.g. value.)

- *Syntax*:

{field: {$gte: value} }

- db.student.find({ marks: { $gte: 20 } })

$in

- The $in operator selects the documents where the value of a field equals any value in the specified array. To specify an $in expression, use the following prototype

  { field: { $in: [<value1>, <value2>, ... <valueN>] } }

- db.student.find({ marks: { $in: [50, 75] } })

$lt

- *Syntax*: **{field: {$lt: value} }**
- $lt selects the documents where the value of the field is less than (i.e. <) the specified value.
- db.student.find({ marks: { $lt: 20 } })

$lte

- $lte selects the documents where the value of the field is less than or equal to (i.e. <=) the specified value
- ***Syntax*:**

  { field: { $lte: value} }

- db.student.find({ marks: { $lte: 20 } })

$ne

- $ne selects the documents where the value of the field is not equal to the specified value. This includes documents that do not contain the field.
- ***Syntax*:**

  {field: {$ne: value} }

- db.student.find({ marks: { $ne: 20 } })

logical operators with find

Logical

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

logical operators

- *Syntax*:

{ $and: [{ <expression1> }, { <expression2> } , ... , { <expression N> }] }

- $and performs a logical AND operation on an array of *one or more* expressions (e.g. <expression1>, <expression2>, etc.) and selects the documents that satisfy *all* the expressions in the array

- db.student.find({ $and: [{ marks: { $ne: 50 } }, { rollno: { $lt:4 } }] })

logical operators

- The $or operator performs a logical OR operation on an array of *two 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> }] }

The Limit() Method

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

- Syntax
- The basic syntax of limit() method is as follows −

- >db.COLLECTION_NAME.find().limit(NUMBER)

MongoDB Skip() Method

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

- Syntax
- The basic syntax of skip() method is as follows −

- >db.COLLECTION_NAME.find().skip(NUMBER)

The sort() Method

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

- Syntax
- The basic syntax of sort() method is as follows –

- >db.COLLECTION_NAME.find().sort({KEY:1})

Contents to be covered

- projection
- Updating Documents
- Operator and Modifiers
- Removing Documents
- Document Replacement

projection

- By default, queries in MongoDB return all fields in matching documents. To limit the amount of data that MongoDB sends to applications, you can include a projection document to specify or restrict field

projection

- If you do not specify a projection document, the db.collection.find() method returns all fields in the matching documents.

Syntax:

db.collectionname.find({criteria},{field:1})

– db.marks.find({ rollno:11 }, { name: 1,marks: 1 })

Updating Documents

- **update() Method : Updating the values in document**
  - The update() method is used to update the value in the existing document of specified collection.

Syntax

db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

where

SELECTION_CRITERIA -- The selection criteria for the updating the document

UPDATED_DATA -- The modifications to apply to the document.

Operator and Modifiers

Update Operators

Update methods supports various field and array operators. It has the following syntax :

Syntax

```
{
 <operator1>: { <field1>: <value1>, ... },
   <operator2>: { <field2>: <value2>, ... },
   ...
}
```

Operator and Modifiers

- Where operator can be field or array operator.

•Field operators

Name	Description
$set	Sets the value of a field in a document.
$inc	Increments the value of the specified field by the specified amount.
$min	if the given value is less than the existing field value, then only updates the field
$max	if the given value is greater than the existing field value, then only updates the field
$mul	Used to multiply the value of the specified field by the given amount.
$rename	Renames the specified field.

$set

It may be simply given with $set: parameter.

Example 1

db.school.update({"class":"10th"}, {$set:{class:"11th"}})

after this command, record with class 10th will be updated to class 11th

```
>
>
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "10th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "9th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "9th", "section" : "B section", "student count" : "39" }
>
> db.school.update({"class":"10th"}, {$set:{class:"11th"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "9th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "9th", "section" : "B section", "student count" : "39" }
>
```

Example 2 :

In case of multiple matching criteria, only 1st document will be updated

db.school.update({"class":"12th"}, {$set:{class:"9th"}})

after this command, 1st record with class 12th will be updated to class 9th

```
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "10th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "9th", "section" : "B section", "student count" : "39" }
>
>
> db.school.update({"class":"12th"}, {$set:{class:"9th"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "9th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "10th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "9th", "section" : "B section", "student count" : "39" }
```

- WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }) means 1 record matched and updated one record.

updateMany() Method :

The updateMany() method is used to update all the matching records of collection. All other parameter and system is similar to update() method.

Syntax

db.COLLECTION_NAME.updateMany(SELECTION_CRITERIA, UPDATED_DATA)

Example:

Lets update all the records of the matching criteria,

db.school.updateMany({"class":"9th"}, {$set:{class:"12th"}})

updateMany() Method :

```
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "9th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "10th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "9th", "section" : "B section", "student count" : "39" }
> db.school.updateMany({"class":"9th"}, {$set:{class:"12th"}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
>
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "10th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "12th", "section" : "B section", "student count" : "39" }
>
```

updateMany() Method :

It results as:

{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }

Its shows that two records are matched and 2 of them has been modified

$inc

- **Increments the value of the specified field by the specified amount.**
- syntax

db.collectionname.update({criteria},{$inc:{field:value}})

Eg:

db.student.update({rollno:4},{$inc:{marks:10}})

$min

- **if the given value is less than the existing field value, then only updates the field**
- **Syntax:**

db.collectionname.update({criteria},{$min:{field:value}})

- Eg
- db.student.update({rollno:4},{$min:{marks:10}})

$max

- **if the given value is greater than the existing field value, then only updates the field**

- db.collectionname.update({criteria},{$max:{field:value}})
- Eg
- db.student.update({rollno:4},{$max:{marks:70}})

$mul

- **Used to multiply the value of the specified field by the given amount.**

- db.collectionname.update({criteria},{$mul:{field:value}})
- Eg
- db.student.update({rollno:4},{$mul:{marks:10}})

$rename

- **Renames the specified field.**

- db.collectionname.update({criteria},{$rename:{field:value}})
- Eg
- db.student.update({rollno:4},{$rename:{“marks”:”score”}})

Array Operators

Name	Description
$addToSet	Adds given elements to the array, but the existing elements will not be added.
$push	Adds an item to the array.

Modifiers with array operator

Update methods supports following 4 modifiers to update documents based on the criteria.

Modifier name	Description
$each	If we wish to add multiple items to array in update, $each is used with $push and $addToSet operators
$position	Used to add element at a particular position in array. It is used with $push operator. Without $position, element is added at the end of the array.
$slice	Used to limit the size of updated arrays when used with $push, i.e. limit the number of elements.
$sort	Modifies the order of the elements of an array when used with $push operation.

Insert values into Array

- Lets insert the multiple values into an array. The arrays are represented in [ ]. For example and array of marks can be [45, 44, 43].
- The values in array can be inserted in [ ] using simple **db.collectionname.insert()** method.

Example:

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.result.find()
>
> db.result.insert({"name": "amit", "marks":[45, 48, 41]})
WriteResult({ "nInserted" : 1 })
>
> db.result.insert({"name": "indu", "marks":[42, 44]})
WriteResult({ "nInserted" : 1 })
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 45, 48, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44 ] }
>
```

operators used with array

- **{ $addToSet: { <field>: [<value1>, <value2> ...] } } }**

  $addToSet array operator to add multiple values to an array <field>. If the values do not exist in the <field> of array it adds to it, if exist do nothing.

- **{ $push: { <field>: [<value1>, <value2> ...] } } }**

  $push operator to append multiple values to an array <field>.

  If you use these operator without each then they will add the values as sub array

Syntax of $each modifier

- **Each modifier adds the value in the array**
- **{ $addToSet: { <field>: { $each: [<value1>, <value2> ...] } } }**

  $each modifier can be Used alongwith the $addToSet array operator to add multiple values to an array <field>. If the values do not exist in the <field> of array it adds to it, if exist do nothing.

- **{ $push: { <field>: { $each: [<value1>, <value2> ...] } } }**

  $each modifier can be Used with the $push operator to append multiple values to an array <field>.

$addToSet operator with $each modifier:

- Let update the Marks array and add elements to it. This is being done using **$each** modifier and **$addToSet** operator.
- With this only elements specified in the update, will be appended to the array, if they do not exist in the array, if for existing elements, do nothing.
- In the below example, we are adding 3 elements to the array marks only if the specified elements are not existing in the array for amit.

$addToSet operator with $each modifier:

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 45, 48, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 43, 47, 51 ] }
>
> db.result.update({"name":"amit"}, {$addToSet: {"marks":{$each: [42,41,48]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 45, 48, 41, 42 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 43, 47, 51 ] }
>
```

Examples of Update Operator & Modifiers

- **$each modifier:** Let update the Marks array and add elements to it. This is being done using **$each** modifier and **$push** operator. With this all the elements specified in the update, will be appended to the array.
- In the following example, 3 elements has been appended to the marks where name is indu.

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 45, 48, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44 ] }
>
> db.result.update({"name":"indu"}, {$push: {"marks":{$each: [43,47,51]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 45, 48, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 43, 47, 51 ] }
>
```

$slice and $sort modifier:

- We may use sort and slice modifier with Update also. Sort may sort the specified array in ascending or descending order and slice will limit the given no of elements in the array.
- In the following example we are appending some elements to marks array, then sorting it in descending order and slicing top 5 records in the collection.

Syntax of $sort modifier

```
{
  $push: {
     <field>: {
       $each: [ <value1>, <value2>, ... ],
       $sort: <sort specification>
     }
  }
}
```

Syntax of $sort modifier

where <sort specification> is

1 for ascending

-1 for descending.

It may also be used with field i.e. {field: 1 } or { field: -1 }.

- db.student.update({rollno:10},{$push:{"marks":{$each:[34,56], $sort:1}}})

$slice modifier

- It will slice the array to the specified value

Syntax of $slice modifier

```
{
  $push: {
     <field>: {
       $each: [ <value1>, <value2>, ... ],
       $slice: <num>
  }
   }
}
```

Syntax of $slice modifier

- The <num> can be:
  - Zero - To update the array <field> to an empty array [].
  - Negative - To update the array <field> to contain only the last <num> elements.
  - Positive - To update the array <field> contain only the first <num> elements

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 45, 48, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 43, 47, 51 ] }
>
> db.result.update({name:"amit"},{$push:{marks:{$each:[40,78,89],$sort:-1,$slice:5}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 89, 78, 48, 45, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 43, 47, 51 ] }
>
```

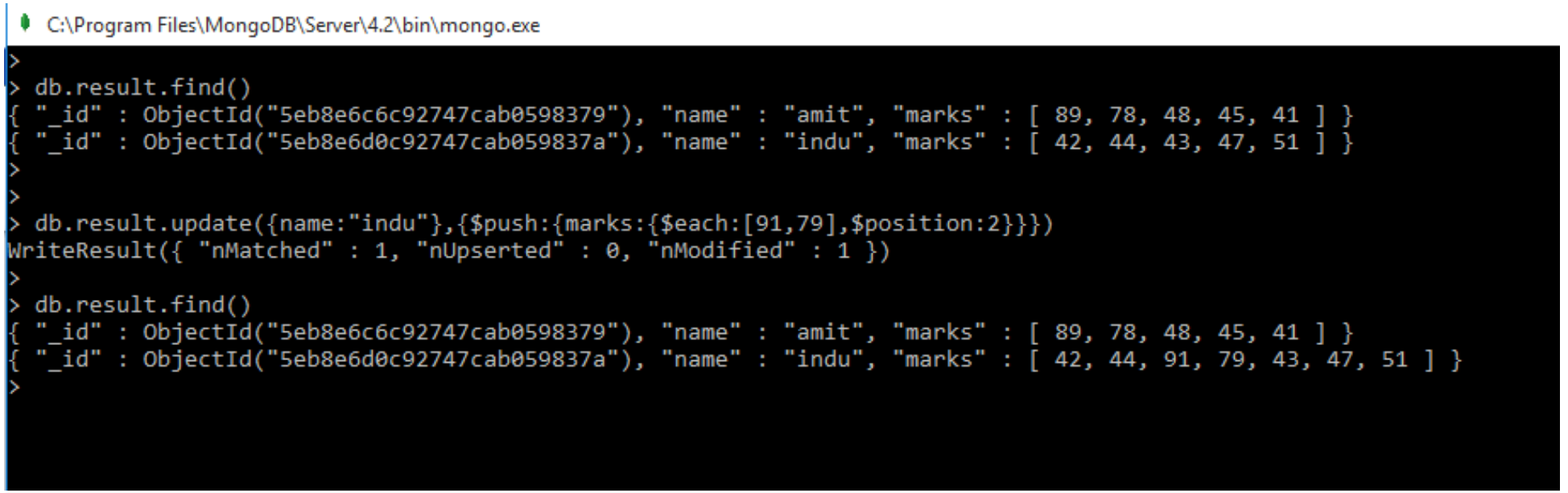
Syntax of $position modifier

```
{
  $push: {
    <field>: {
      $each: [ <value1>, <value2>, ... ],
      $position: <num>
    }
  }
}
```

- Where <num> is the position for the element in the array, and array index starts with 0.

$position modifier:

- Using the **$position** modifier, we may specify the element location in the array at which the **$push** operator inserts elements. **$position** modifier is always used with **$each** modifier.
- In the following example, we are adding elements at the position 2 in the array marks for indu.


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [89, 78, 48, 45, 41] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [42, 44, 43, 47, 51] }
>
>
> db.result.update({name:"indu"},{$push:{marks:{$each:[91,79],$position:2}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [89, 78, 48, 45, 41] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [42, 44, 91, 79, 43, 47, 51] }
>

- If we specify a negative number in $position, it will add the element in the array counting the **$position** from the last element of the array. Example:

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 89, 78, 48, 45, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 91, 79, 43, 47, 51 ] }
>
> db.result.update({name:"indu"},{$push:{marks:{$each:[77, 78],$position:-2}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.result.find()
{ "_id" : ObjectId("5eb8e6c6c92747cab0598379"), "name" : "amit", "marks" : [ 89, 78, 48, 45, 41 ] }
{ "_id" : ObjectId("5eb8e6d0c92747cab059837a"), "name" : "indu", "marks" : [ 42, 44, 91, 79, 43, 77, 78, 47, 51 ] }
>
```

Removing Documents,

remove() method to remove documents from collection

remove() method is used to remove a document from collection.

Syntax

db.COLLECTION_NAME.remove(DELLETION_CRITTERIA, justOne)

this method has two parameters .

One is deletion criteria and second is justOne flag.

Removing Documents,

- **DELLETION_CRITTERIA** – (Optional) deletion criteria of the documents to be removed. To delete all documents in a collection, pass an empty document ({ }).
- **justOne** – (Optional) if set to true or 1, then remove only one document i.e. the 1st document. Optional. If not specified i,e, the default value of false and it will delete all documents matching the deletion criteria.
- **Example1:Remove only one document**

  db.school.remove({"class":"12th"},1)
- will remove 1st record of class 12th.

Removing Documents,

```
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
{ "_id" : "01", "class" : "10th", "section" : "D Section", "student count" : "29" }
{ "_id" : "02", "class" : "10th", "section" : "C Section", "student count" : "37" }
{ "_id" : ObjectId("5eb2446ddb4e42dfda38301b"), "class" : "12th", "section" : "A section", "student count" : "33" }
{ "_id" : ObjectId("5eb24479db4e42dfda38301c"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb2448adb4e42dfda38301d"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb244a2db4e42dfda38301e"), "class" : "9th", "section" : "A section", "student count" : "40" }
>
>
> db.school.remove({"class":"12th"},1)
WriteResult({ "nRemoved" : 1 })
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
{ "_id" : "01", "class" : "10th", "section" : "D Section", "student count" : "29" }
{ "_id" : "02", "class" : "10th", "section" : "C Section", "student count" : "37" }
{ "_id" : ObjectId("5eb24479db4e42dfda38301c"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb2448adb4e42dfda38301d"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb244a2db4e42dfda38301e"), "class" : "9th", "section" : "A section", "student count" : "40" }
>
```

Example2:Remove all records of the matching criteria

db.school.remove({"class":"12th"})

will remove all the record of class 12th.

Example2:Remove all records of the matching criteria

```
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
{ "_id" : "01", "class" : "10th", "section" : "D Section", "student count" : "29" }
{ "_id" : "02", "class" : "10th", "section" : "C Section", "student count" : "37" }
{ "_id" : ObjectId("5eb24479db4e42dfda38301c"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb2448adb4e42dfda38301d"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb244a2db4e42dfda38301e"), "class" : "9th", "section" : "A section", "student count" : "40" }
>
>
> db.school.remove({"class":"12th"})
WriteResult({ "nRemoved" : 2 })
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
{ "_id" : "01", "class" : "10th", "section" : "D Section", "student count" : "29" }
{ "_id" : "02", "class" : "10th", "section" : "C Section", "student count" : "37" }
{ "_id" : ObjectId("5eb244a2db4e42dfda38301e"), "class" : "9th", "section" : "A section", "student count" : "40" }
>
```

- **WriteResult({ "nRemoved" : 2 })**
- Here it has removed all the 2 records of matching criteria.

Example 3: Remove all document if no criteria is specified

db.school.remove({})

will remove all the record from the collection.

WriteResult({ "nRemoved" : 2 })

Here it has removed all the 2 records of matching criteria

Example 3: Remove all document if no criteria is specified

```
>
> db.school.find()
{ "_id" : ObjectId("5eb0f690ffe73fbdd0e2b1e5"), "class" : "10th", "section" : "A sction", "student count" : "39" }
{ "_id" : "01", "class" : "10th", "section" : "D Section", "student count" : "29" }
{ "_id" : "02", "class" : "10th", "section" : "C Section", "student count" : "37" }
{ "_id" : ObjectId("5eb244a2db4e42dfda38301e"), "class" : "9th", "section" : "A section", "student count" : "40" }
>
>
> db.school.remove({ })
WriteResult({ "nRemoved" : 4 })
> db.school.find()
>
```

WriteResult({ "nRemoved" : 4 })

Here it has removed all the 4 records from collection and now the collection is empty.

Document Replacement

- **Replacing Document in collection**
  - db.collection.replaceOne() method is used to replace the entire content of a document except the _id field.
  - Based on the condition, whole document is replaced for the 1st matching record if there are multiple records matching the given criteria.
  - The _id field remains unchanged after the replace. This method requires only field/value pairs.

Document Replacement

- **Syntax**

  db.COLLECTION_NAME.replaceOne(SELECTION_CRITERIA, Replaced_DATA)

  where

  SELECTION_CRITERIA -- The selection criteria for the replacing the document

  Replaced_DATA -- The replacement document on the matching criteria.

- The replacement document may also have different fields from the original document. _id filed may be given in new document to be replaced but it has no significance.
- **In the following example**, we are replacing document matching with criteria “class”:”10th” with a new document having only 2 fields class and student count in school collection:

Document Replacement

```
> db.school.find();
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "10th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "12th", "section" : "B section", "student count" : "39" }
>
>
> db.school.replaceOne({class:"10th"},{"class":"10th","student count":"NIL"})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
>
>
> db.school.find()
{ "_id" : ObjectId("5eb24d83db4e42dfda38301f"), "class" : "11th", "section" : "A section", "student count" : "40" }
{ "_id" : ObjectId("5eb24d89db4e42dfda383020"), "class" : "12th", "section" : "C section", "student count" : "30" }
{ "_id" : ObjectId("5eb24d8fdb4e42dfda383021"), "class" : "12th", "section" : "B section", "student count" : "35" }
{ "_id" : ObjectId("5eb24ef5db4e42dfda383022"), "class" : "10th", "student count" : "NIL" }
{ "_id" : ObjectId("5eb24f1ddb4e42dfda383023"), "class" : "12th", "section" : "B section", "student count" : "39" }
>
```

Indexing in MongoDb

- In MongoDB, index is a special type of data structure which is used to easily and quickly locate the record in a given table of the database without traversing every record in the give table. Indexed may be generated for one or multiple fields of the table. Binary Tree data structures are used by indexes.

35

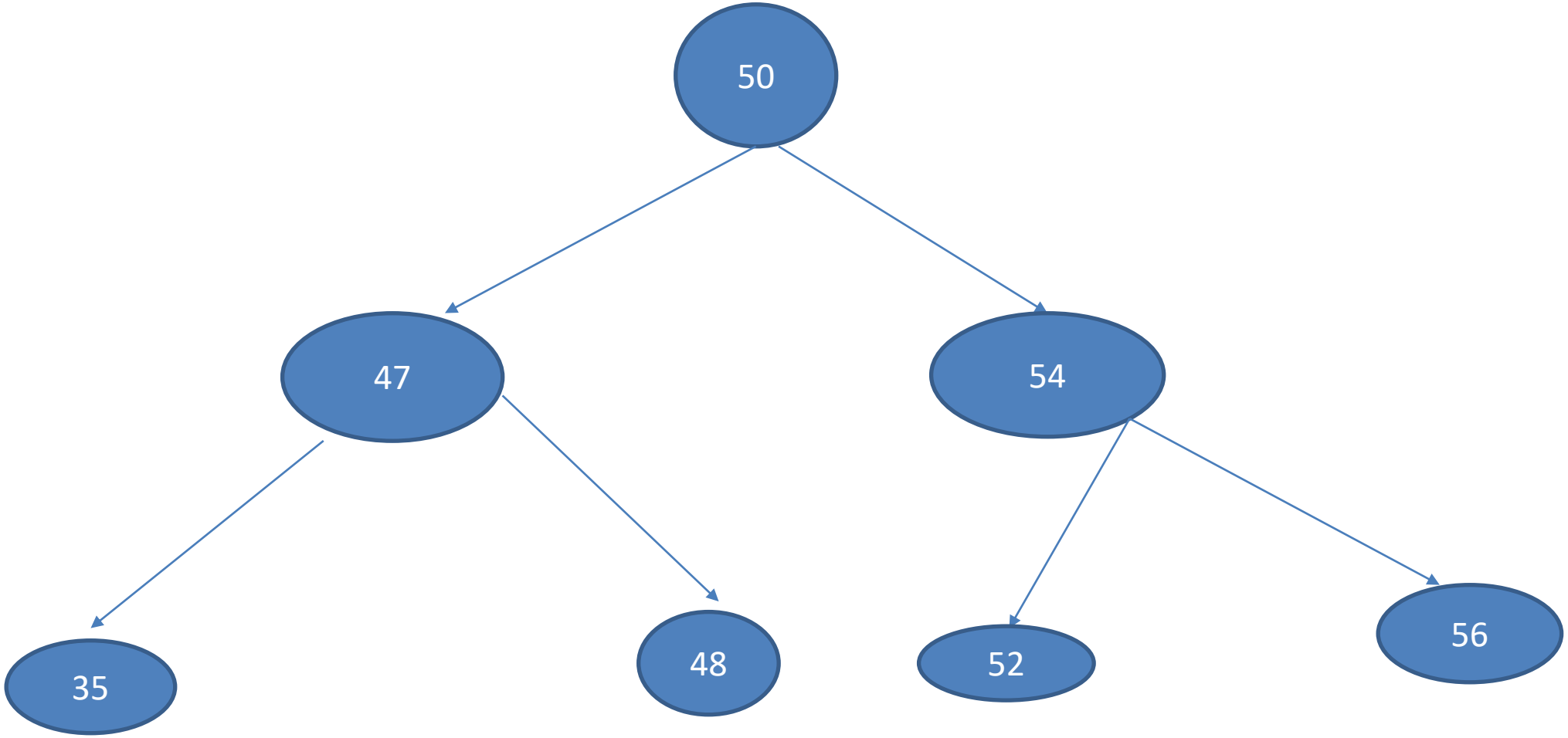
46

48

50

52

54


50
47
54
35
48
52
56

Indexes

- plays a vital role in execution of queries in MongoDB. If no index is defined, MongoDB has to scan every record/ document of the specified collection and if index is there, it has to scan less number of documents resulting in a faster result to select those documents that match the query criteria. The index in MongoDB is similar to the indexes used in other relational databases.

Indexing in MongoDb

- In MongoDB, indexes are special data structure, which stores the value of a specific field or set of fields of a collection, ordered by the value of the field.
- The ordering of the index, ascending or descending, supports efficient equality matches and range-based query operations apart from returning sorted results by using the ordering in the index.
- The indexes are defined at collection level in MongoDB on any field or sub-field of the documents. MongoDB indexes also uses a B-tree data structure.
- It is very important and suggested to create the index on the field that will be frequently searched in a collection.

Indexing in MongoDb

Default Index in MongoDB i.e. _id

- By default, MongoDB creates a _id field if not specified in each collection and create a unique index on the **_id** field during the creation of a collection.
- The _id field acts like Primary Key of other RDBMS and prevents clients from inserting two documents with the same value for the _id field. The index on the _id field automatically created by MongoDB cannot be removed or dropped.

Single Field Index

- MongoDb provides the facility to create index on any user defied key in the collection apart from the default **_id** index. This user created index can be ascending/descending indexes on a single field of a document as defined by the user.
- **Syntax**

  db.collection.createIndex({ <field>: <Value>})

Where

Field : is the name of the key in the collection

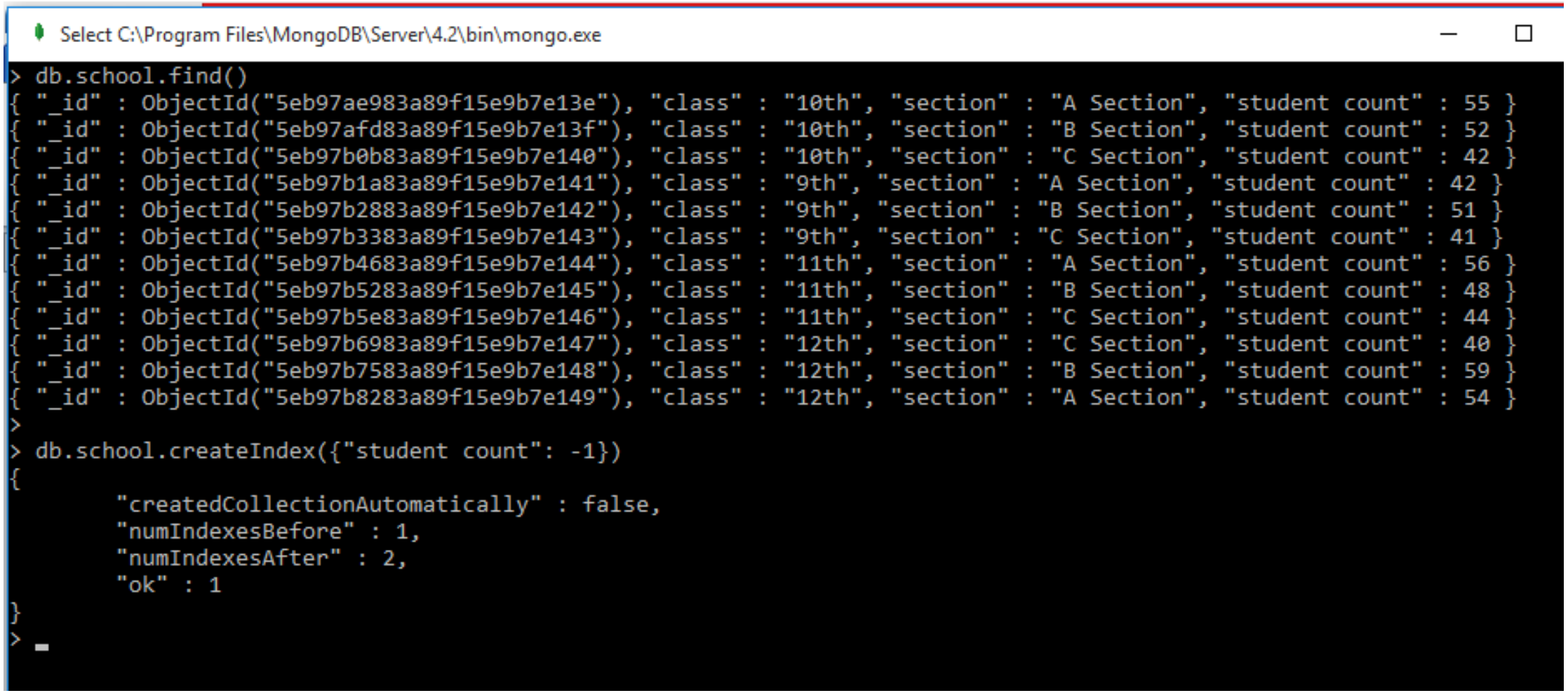
Value : 1 for ascending , -1 for descending

Single Field Index

- **Example**
  - Command to create a descending order index on field “student count”

db.school.createIndex({"student count": -1})

Single Field Index



Single Field Index

- After executing the command, it displays

  "createdCollectionAutomatically" : false -- It is created by user

  "numIndexesBefore" : 1 -- No of indexes before the command (only _id)

  "numIndexesAfter" : 2 -- No of indexes after the command (_id and the other one)

  "ok" : 1 – Command executed Successfully

- Here, an index has been created on “student count” which means when someone searches the document based on the “student count”, the search will be faster because the created index will be used for this search.

Finding the indexes in a collection

- **getIndexes()** method in MongoDB is used to find all the indexes created on a collection.
- **Syntax**

  db.collection_name.getIndexes()
- **Example:**

  – To get all the indexes of **student** collection, the command will be:

  > db.school.getIndexes()

Finding the indexes in a collection

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.school.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "abc.school"
        },
        {
                "v" : 2,
                "key" : {
                        "student count" : -1
                },
                "name" : "student count_-1",
                "ns" : "abc.school"
        },
        {
                "v" : 2,
                "key" : {
                        "class" : -1,
                        "section" : 1
                },
                "name" : "class_-1_section_1",
                "ns" : "abc.school"
        }
]
>
```

Finding the indexes in a collection

- The above example shows that **three** indexes are there in "**student**" collection. These are
  - the default index created on **_id** with name **"_id_",**
  - **The single index we have created on "student count" field with name**

    "student count_-1",
  - The compound index we have created on "class" and "student count" with name **"class_-1_section_1",**

Compound Indexes

- A compound index is an index on two or more fields of a collection, and it can support queries based on those fields. In compound indexes, Fields can be sorted inside other fields.

Syntax

db.collection.createIndex({ <field1>: <value>, <field2>: <value>, ... })

Where

Field : is the name of the key in the collection

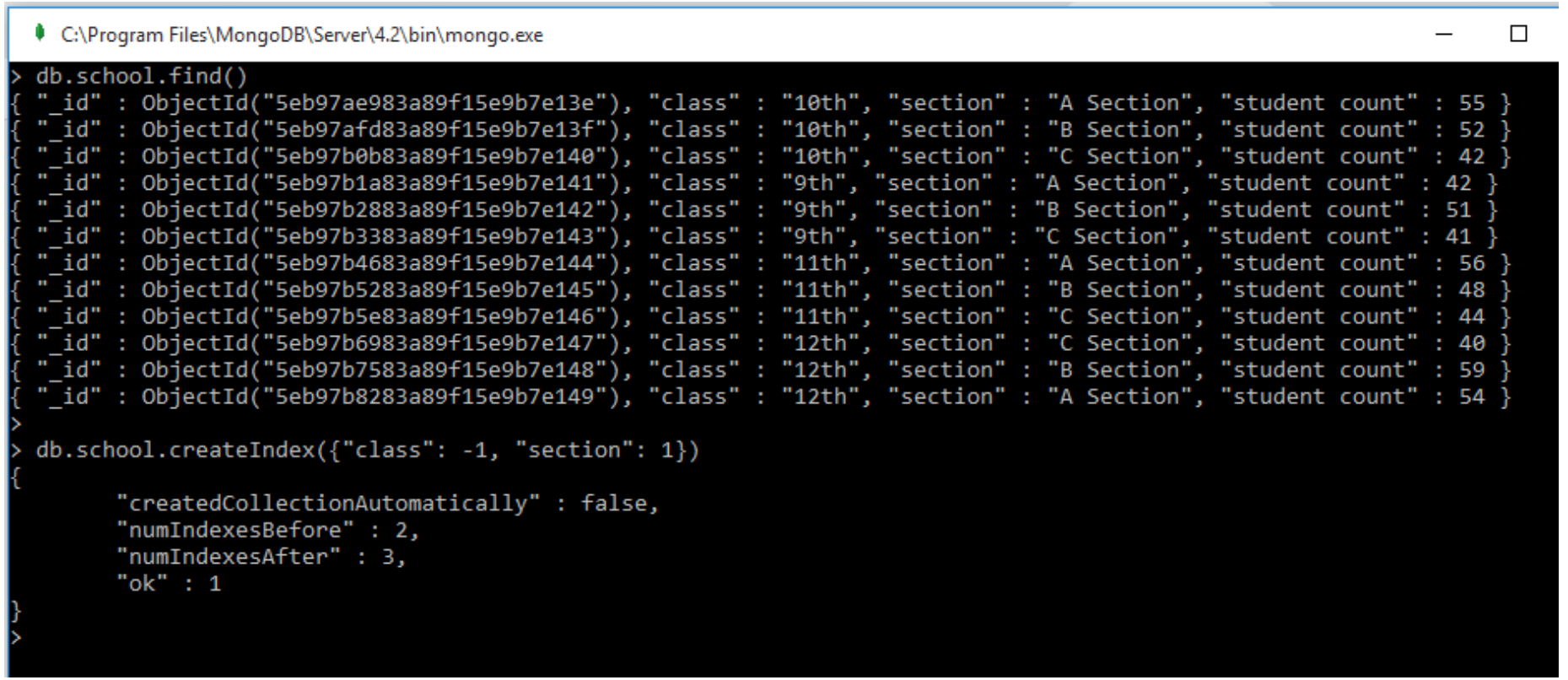
Value : 1 for ascending , -1 for descending

Compound Indexes

- **Example**
  - We have to create a compound index on two fields, i.e. class (descending order) and section(ascending order), to do so, the command will be

db.school.createIndex({"class": -1, "section": 1})

Compound Indexes



Compound Indexes

- After executing the command, it displays

  "createdCollectionAutomatically" : false -- It is created by user

  "numIndexesBefore" : 2 -- No of indexes before the command (_id and "student count" created just before)

  "numIndexesAfter" : 3 -- No of indexes after the command (_id, "student count" and compound index of "**class and section**")

  "ok" : 1 – Command executed Successfully

- This compound index of "**class + section**" sorts the collection first by "**class**" and then, within each "**class**" value, sort by "**section**".

Drop index(es) in a collection

- **Dropping a index:**
  - **dropIndex()** method is used to drop i.e. delete or remove a particular index on a collection.
- **Syntax**

```
db.collection_name.dropIndex(index)
```

Drop index(es) in a collection

- where index can be either “index_name”

  for example, "**student count_-1**" or **"class_-1_section_1",**

  OR

- “index specification document”

  for example, **{ "student count" : -1} or { class" : -1, "section" : 1}**

Example:

- **db.school.dropIndex("student count_-1")** will delete the index with index name "**student count_-1**"

```
> db.school.dropIndex("student count_-1")
{ "nIndexesWas" : 3, "ok" : 1 }
>
> db.school.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "abc.school"
        },
        {
                "v" : 2,
                "key" : {
                        "class" : -1,
                        "section" : 1
                },
                "name" : "class_-1_section_1",
                "ns" : "abc.school"
        }
]
```

Example:

- **nIndexesWas: 3 --**shows how many indexes were there before this command got executed i.e. 3
- **ok: 1:** -- means the command is executed successfully.
- For dropping a compound index:
  - **dropIndexes()** method is are used to drop i.e. delete or remove, all the indexes on a collection except the index on the **_id** field.

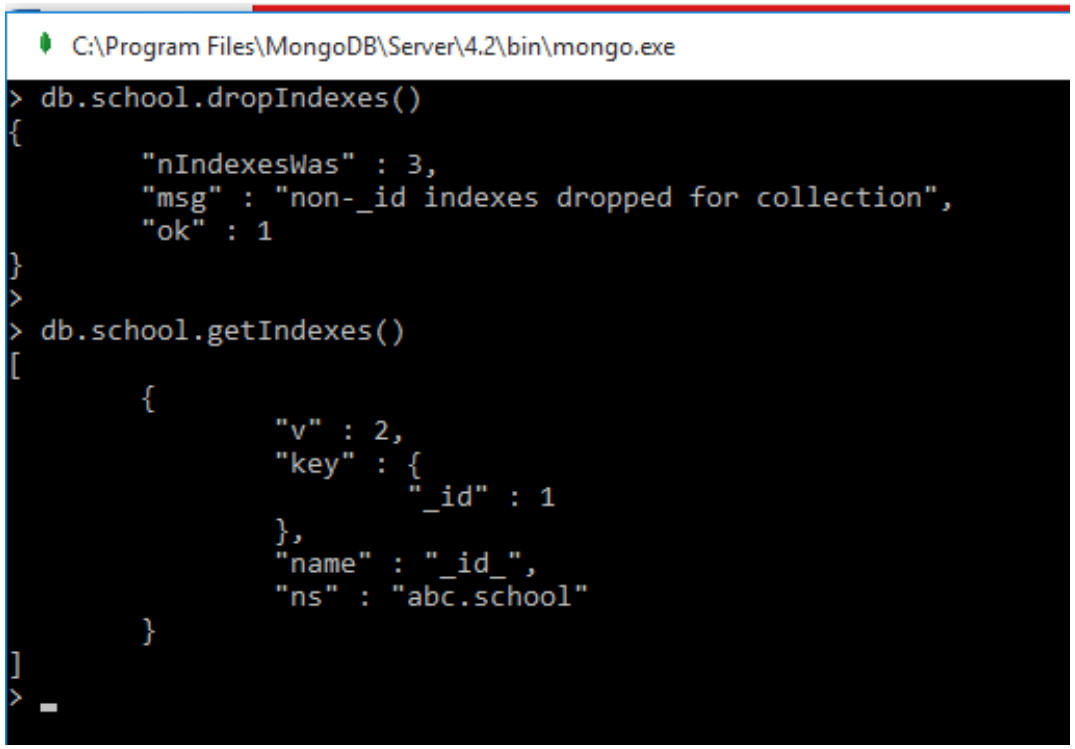
Syntax

db.collection_name.dropIndexes()

– without any parameter it will delete all indexes except on _id field, if we specify the “index_name” or “index specification document” it will delete that particular index and act like **db.collection_name.dropIndex()** method

Example

- **db.student.dropIndexes()** will delete all the indexes.



- **Msg: "non-_id indexes dropped for collection"** – means that the default index _id will still remain and cannot be dropped.
- **nIndexesWas: 3 --**shows how many indexes were there before this command got executed i.e. 3
- **ok: 1:** -- means the command is executed successfully.

Multikey Index

- MongoDB uses multikey indexes to index the content stored in arrays. To index a field that holds an array value, MongoDB creates an index key for each element in the array, i.e. creates separate index entries for every element of the array
- These *multikey* indexes support efficient queries against array fields. Multikey indexes can be constructed over arrays that hold both scalar values (e.g. strings, numbers) *and* nested documents.
- The multikey indexes, created by MongoDB, allow queries to select documents that contain arrays by matching on element or elements of the arrays

Syntax

db.collection_name.createIndex({ <field>: value })

- where
  - **Field :** is the name of the key in the collection
  - **Value :** 1 for ascending , -1 for descending
- While Creating an Index, it is not required to explicitly specify the multikey type, MongoDB automatically determines whether to create a multikey index, if the specified indexed field contains an array value.

Example:

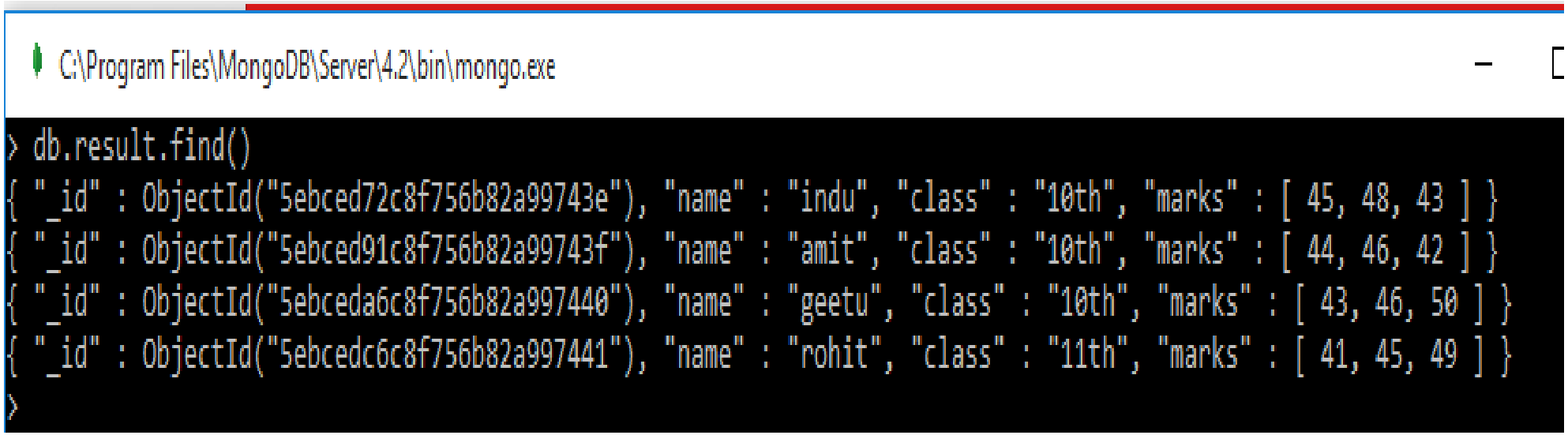
- Lets consider a collection **result** with following documents:

{ "_id" : ObjectId("5ebced72c8f756b82a99743e"), "name" : "indu", "class" : "10th", "marks" : [45, 48, 43] },

{ "_id" : ObjectId("5ebced91c8f756b82a99743f"), "name" : "amit", "class" : "10th", "marks" : [44, 46, 42] },

{ "_id" : ObjectId("5ebceda6c8f756b82a997440"), "name" : "geetu", "class" : "10th", "marks" : [43, 46, 50] },

{ "_id" : ObjectId("5ebcedc6c8f756b82a997441"), "name" : "rohit", "class" : "11th", "marks" : [41, 45, 49] }


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.result.find()
{ "_id" : ObjectId("5ebced72c8f756b82a99743e"), "name" : "indu", "class" : "10th", "marks" : [45, 48, 43] }
{ "_id" : ObjectId("5ebced91c8f756b82a99743f"), "name" : "amit", "class" : "10th", "marks" : [44, 46, 42] }
{ "_id" : ObjectId("5ebceda6c8f756b82a997440"), "name" : "geetu", "class" : "10th", "marks" : [43, 46, 50] }
{ "_id" : ObjectId("5ebcedc6c8f756b82a997441"), "name" : "rohit", "class" : "11th", "marks" : [41, 45, 49] }
>

- Lets create an index on “**marks**” key, which is an array on descending order
- **> db.result.createIndex({"marks" : -1})**

```
> db.result.createIndex({"marks" : -1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

- And if we run the command **getindexes**(), it will display all the indexes in the collection **result**

> db.result.getIndexes()

```
> db.result.createIndex({"marks" : -1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.result.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "abc.result"
        },
        {
                "v" : 2,
                "key" : {
                        "marks" : -1
                },
                "name" : "marks_-1",
                "ns" : "abc.result"
        }
]
```

- It shows that an index on **marks key** has been created with name "**marks_-1**" and stored internally, but it will be a multikey index as it has been created on an array key.

Compound Multikey Index

- Just like, creating a compound index, we may create a compound multikey index by specifying multiple keys in createIndex method

Syntax

db.collection.createIndex({ <field1>: <value>, <field2>: <value>, ... })

where

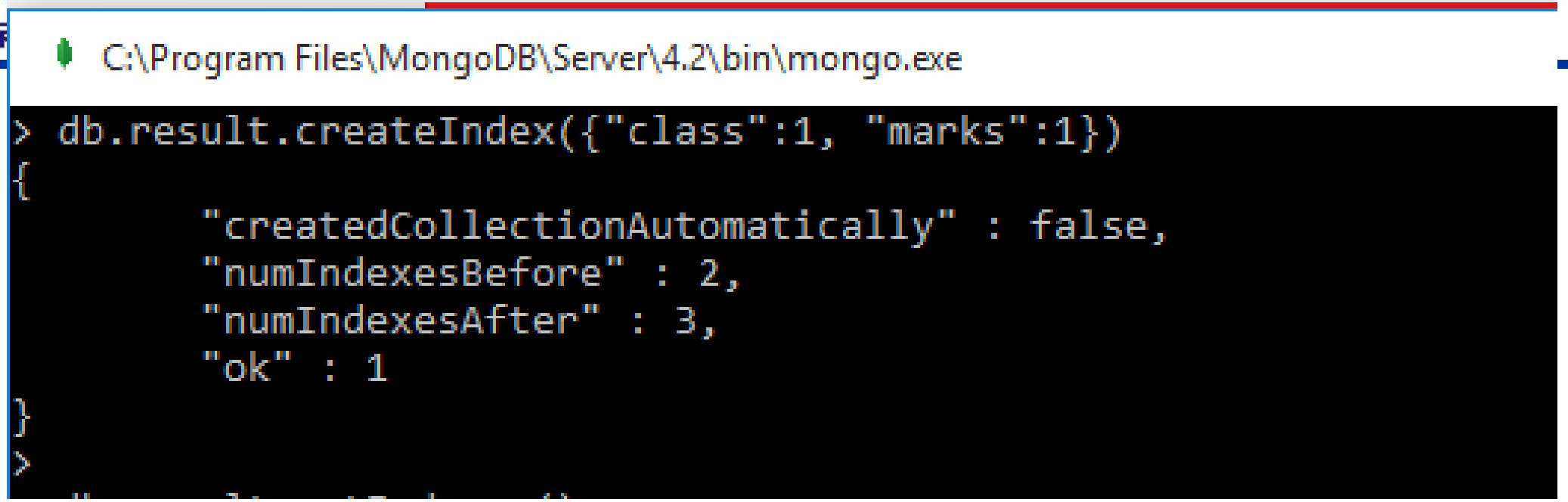
Field : is the name of the key in the collection

Value : 1 for ascending , -1 for descending

Example

- Lets create a compound multikey index on class and marks key, both in ascending order.

> db.result.createIndex({"class":1, "marks":1})



- This will create a compound index combining both the specified key , both in ascending order as we have specified the value as 1 in both the cases.

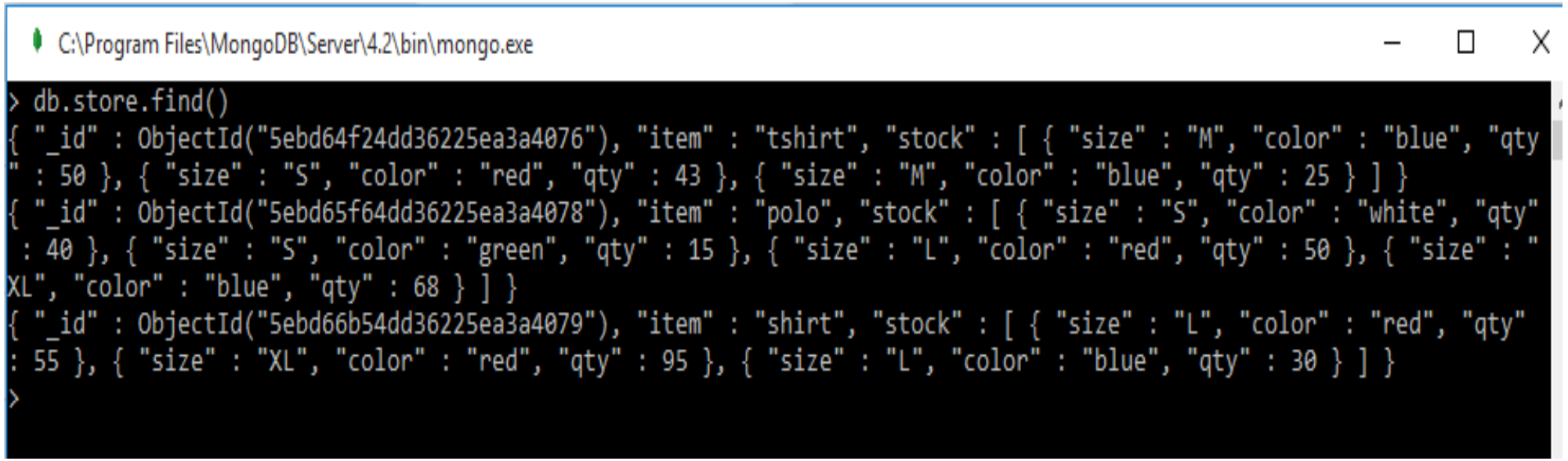
- And if we display the created indexes on the result collection, with getIndexes() method, it will display both the created indexes.

```
> db.result.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "abc.result"
        },
        {
                "v" : 2,
                "key" : {
                        "marks" : -1
                },
                "name" : "marks_-1",
                "ns" : "abc.result"
        },
        {
                "v" : 2,
                "key" : {
                        "class" : 1,
                        "marks" : 1
                },
                "name" : "class_1_marks_1",
                "ns" : "abc.result"
        }
]
>
```

Multikey Indexes on array fields with Objects

- We may also create multikey indexes on array fields that contain nested objects i.e. creating Index inside an array object in a document.
- Lets consider the collection **stock** having the documents as under:

{ "_id" : ObjectId("5ebd64f24dd36225ea3a4076"), "item" : "tshirt", "stock" : [{ "size" : "M", "color" : "blue", "qty" : 50 }, { "size" : "S", "color" : "red", "qty" : 43 }, { "size" : "M", "color" : "blue", "qty" : 25 }] },
{ "_id" : ObjectId("5ebd65f64dd36225ea3a4078"), "item" : "polo", "stock" : [{ "size" : "S", "color" : "white", "qty" : 40 }, { "size" : "S", "color" : "green", "qty" : 15 }, { "size" : "L", "color" : "red", "qty" : 50 }, { "size" : "XL", "color" : "blue", "qty" : 68 }] },
{ "_id" : ObjectId("5ebd66b54dd36225ea3a4079"), "item" : "shirt", "stock" : [{ "size" : "L", "color" : "red", "qty" : 55 }, { "size" : "XL", "color" : "red", "qty" : 95 }, { "size" : "L", "color" : "blue", "qty" : 30 }] }


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.store.find()
{ "_id" : ObjectId("5ebd64f24dd36225ea3a4076"), "item" : "tshirt", "stock" : [{ "size" : "M", "color" : "blue", "qty" : 50 }, { "size" : "S", "color" : "red", "qty" : 43 }, { "size" : "M", "color" : "blue", "qty" : 25 }] }
{ "_id" : ObjectId("5ebd65f64dd36225ea3a4078"), "item" : "polo", "stock" : [{ "size" : "S", "color" : "white", "qty" : 40 }, { "size" : "S", "color" : "green", "qty" : 15 }, { "size" : "L", "color" : "red", "qty" : 50 }, { "size" : "XL", "color" : "blue", "qty" : 68 }] }
{ "_id" : ObjectId("5ebd66b54dd36225ea3a4079"), "item" : "shirt", "stock" : [{ "size" : "L", "color" : "red", "qty" : 55 }, { "size" : "XL", "color" : "red", "qty" : 95 }, { "size" : "L", "color" : "blue", "qty" : 30 }] }
>

- This **stock** collection has an array **stock** which is an object having 3 fields **"size" , "color"** and **"qty" .**
- Lets display only one document using findOne() from the stock collection

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.store.findOne()
{
        "_id" : ObjectId("5ebd64f24dd36225ea3a4076"),
        "item" : "tshirt",
        "stock" : [
                {
                        "size" : "M",
                        "color" : "blue",
                        "qty" : 50
                },
                {
                        "size" : "S",
                        "color" : "red",
                        "qty" : 43
                },
                {
                        "size" : "M",
                        "color" : "blue",
                        "qty" : 25
                }
        ]
}
>
```

- This clearly shows that the **stock** is an array object which has **size, color** and **qty** (quantity) fields. Store owner always want to query quantity available for each color and size. Therefore, they may use query command on finding **stock.qty** or **stock.size** data frequently. In order to meet the demand & supply ratio, it is good to create an index on these fields.
- To create a simple multikey index on the array object field **stock.qty,** command will be

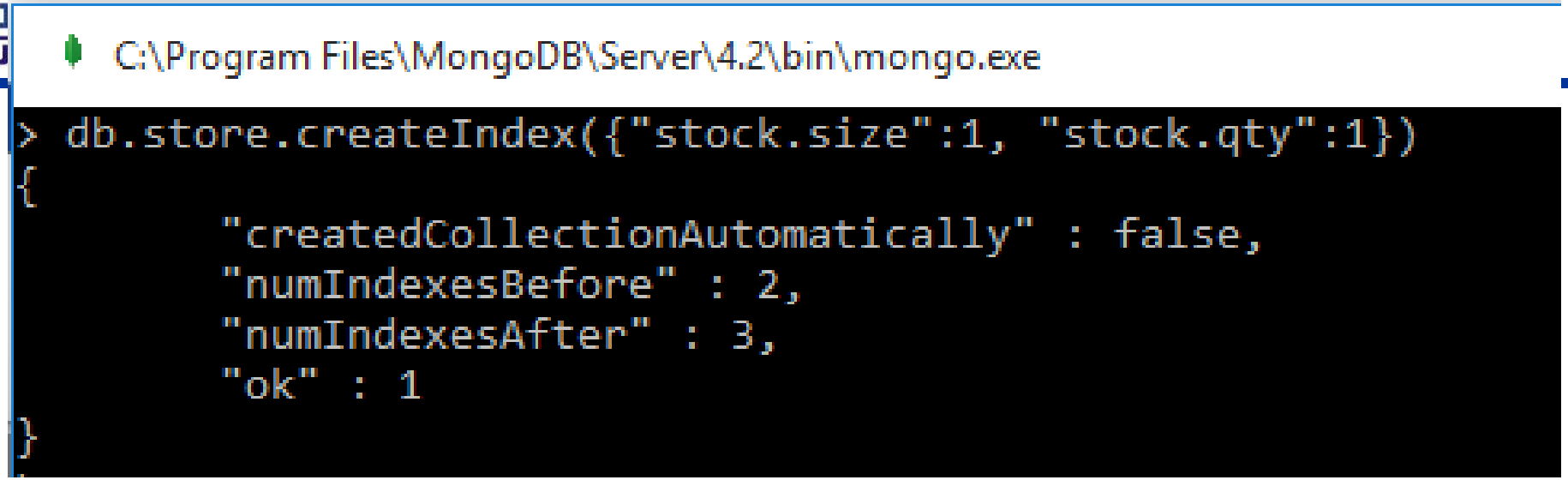
db.store.createIndex({"stock.qty":1})

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.store.createIndex({"stock.qty":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

- We can also create a compound multikey index on the stock.size and stock.qty fields on store collection using the following command:

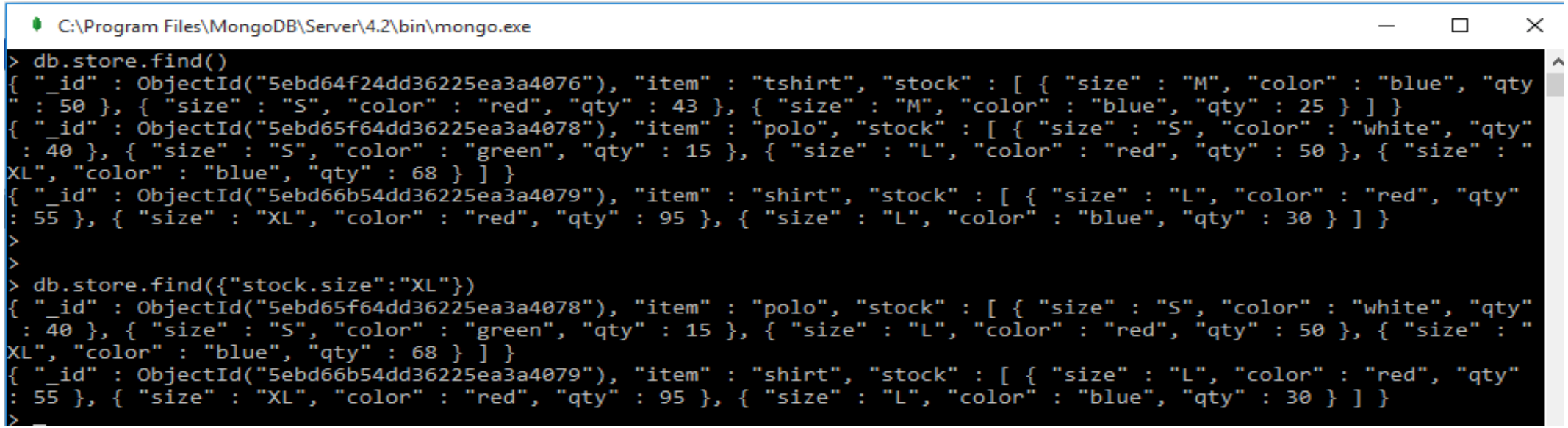
db.store.createIndex({ "stock.size": 1, "stock.qty": 1 })



- These kind of compound multikey index may support queries with predicates that include both indexed fields as well as predicates that include only the index prefix "**stock.size**".

Example-1

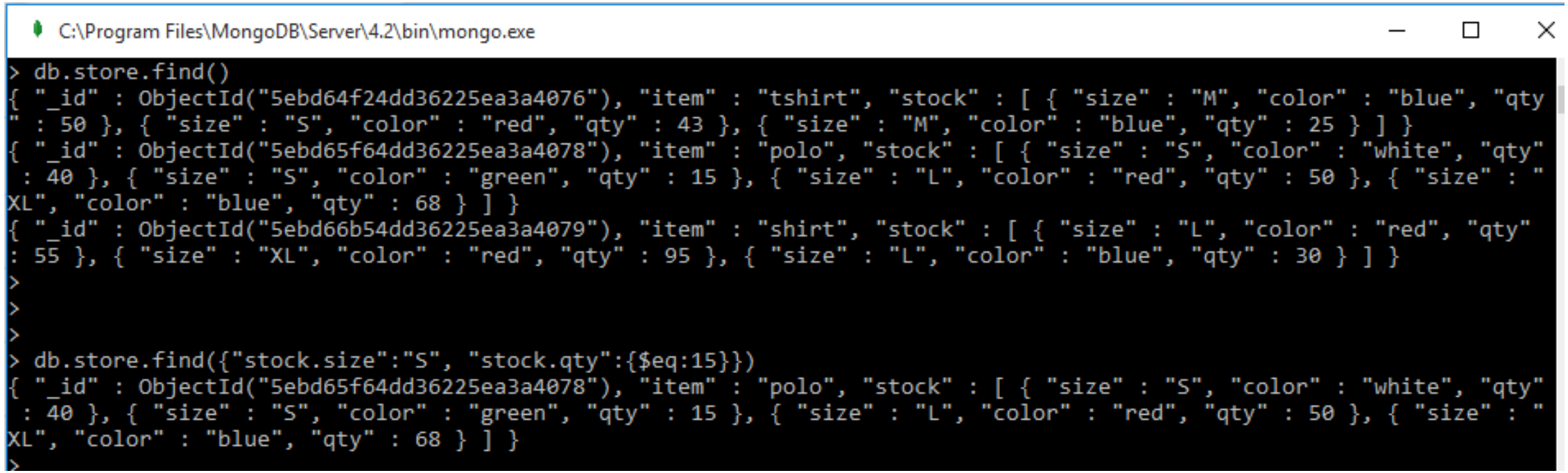
- **db.store.find({"stock.size":"XL"})**



- This is to find all stock of size XL in the collection.

Example-2

- db.store.find({"stock.size":"S", "stock.qty":{$eq:15}})



Contents to be covered

- Aggregation Framework
- Pipeline Operations- $match, $sort , $group

Aggregation Framework

- Aggregation is an operation used to process the data that returns the computed results.
- In Simple words, Aggregation groups the data from multiple documents in a collection and operates in several ways on those grouped data in order to return one combined result i.e. total number(sum), average, minimum, maximum etc out of the group selected.
- In SQL **count**(*) and with "**group by"** is an equivalent of MongoDB aggregation. In MongoDB, **aggregate()** method is used for the aggregation

Aggregation Pipeline

- The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results.
- **Pipeline**
  - The MongoDB aggregation pipeline consists of stages (aggregation states). Each stage transforms the documents as they pass through the pipeline. Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents based on the various operators and functions etc.

Aggregation Pipeline

- The most basic pipeline stages provide ***filters*** that operate like queries and ***document transformations*** that modify the form of the output document.
- Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use operators for tasks such as calculating the average, Sum, Min, MAX or concatenating a string also.
- The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.

Aggregation pipeline

Syntax

```
db.COLLECTION_NAME.aggregate(pipeline)
```

Aggregation pipeline operators

Name	Description
$match	The $match operator filters the documents stream to pass only those documents that match the specified condition(s) to the next pipeline stage. $match uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).
$group	In MongoDB, the $group operator groups the input documents by the specified expression and groups the document for each distinct grouping. An identifier (_id) field in the output documents contains the distinct group by key. The output documents can also contain computed fields that hold the values of some accumulator expression grouped by the $group‘s _id(identifier) field.

Different expressions used by Aggregate function

Expression	Description
$sum	Summates the defined values from all the documents in a collection
$avg	Calculates the average values from all the documents in a collection
$min	Return the minimum of all values of documents in a collection
$max	Return the maximum of all values of documents in a collection

Aggregation Pipeline

- **Example:** lets take a collection marks, having marks of various subjects for each students in various class given as under:

{ "_id" : ObjectId("5ec103443b6e4f8f5b4f1148"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "computer", "marks" : 48 },

{ "_id" : ObjectId("5ec103583b6e4f8f5b4f1149"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "english", "marks" : 44 },

Aggregation Pipeline

{ "_id" : ObjectId("5ec1036c3b6e4f8f5b4f114a"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "hindi", "marks" : 41 },

{ "_id" : ObjectId("5ec103913b6e4f8f5b4f114b"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "computer", "marks" : 41 },

{ "_id" : ObjectId("5ec103a33b6e4f8f5b4f114c"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "english", "marks" : 43 },

{ "_id" : ObjectId("5ec103b53b6e4f8f5b4f114d"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "hindi", "marks" : 43 },

Aggregation Pipeline

{ "_id" : ObjectId("5ec103cf3b6e4f8f5b4f114e"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "hindi", "marks" : 45 },

{ "_id" : ObjectId("5ec103e73b6e4f8f5b4f114f"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "english", "marks" : 39 },

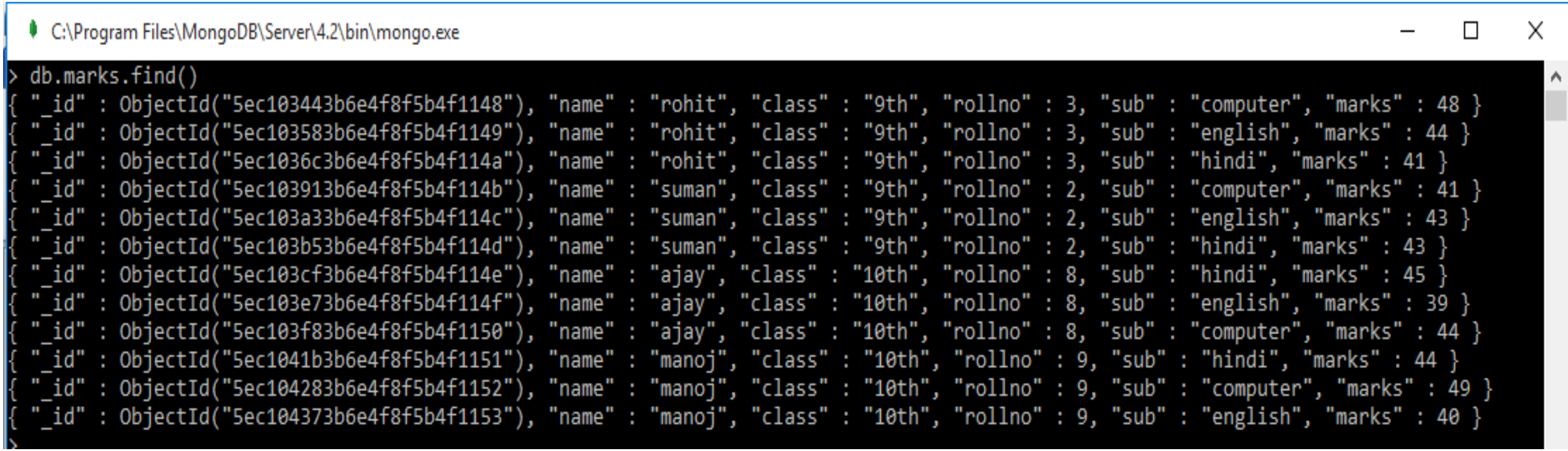
{ "_id" : ObjectId("5ec103f83b6e4f8f5b4f1150"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "computer", "marks" : 44 },

{ "_id" : ObjectId("5ec1041b3b6e4f8f5b4f1151"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "hindi", "marks" : 44 },

{ "_id" : ObjectId("5ec104283b6e4f8f5b4f1152"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "computer", "marks" : 49 },

{ "_id" : ObjectId("5ec104373b6e4f8f5b4f1153"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "english", "marks" : 40 }

Aggregation Pipeline



Aggregation Pipeline

- **Now, lets execute the following aggregate command:**

db.marks.aggregate({$match:{"class":"10th"}},{$group:{_id:"$name", "Total_Marks":{$sum:“$marks"}}})

```
>
> db.marks.aggregate([{$match:{"class":"10th"}},{$group:{_id:"$name","Total_Marks":{$sum:"$marks"}}}])
{ "_id" : "manoj", "Total_Marks" : 133 }
{ "_id" : "ajay", "Total_Marks" : 128 }
>
```

Aggregation Pipeline

- **This command executed in two states,**
  - **first Stage**: The **$match** stage filters the documents by the status field and passes to the next stage those documents that have class equal to "10th".
  - **Second Stage**: The **$group** stage groups the documents by the sub field to calculate the sum of the amount for marks.
- This resulted into:
  - **{ "_id" : "manoj", "Total_Marks" : 133 }**
  - **{ "_id" : "ajay", "Total_Marks" : 128 }**

Aggregation Pipeline

- **Now Run another, aggregate command and see the output**
- **db.marks.aggregate({$match:{"class":"9th"}},{$group:{_id:“$sub",Max_marks:{$max:“$marks"}}})**

```
> db.marks.aggregate([{$match:{"class":"9th"}},{$group:{_id:"$sub",Max_marks:{$max:"$marks"}}}])
{ "_id" : "english", "Max_marks" : 44 }
{ "_id" : "computer", "Max_marks" : 48 }
{ "_id" : "hindi", "Max_marks" : 43 }
>
```

Aggregation Pipeline

- **Here, in state 1, students of class 10th are filtered out and then Maximum marks of each subject has been computed (filtered). The output is:**

```
{ "_id" : "english", "Max_marks" : 44 }
{ "_id" : "computer", "Max_marks" : 48 }
{ "_id" : "hindi", "Max_marks" : 43 }
```

Aggregation pipeline operators

Name	Description
$sort	$ sort is used to Reorders the document stream by a specified sort key. It only changes the order not the documents. For each input document, outputs one document is there..

$sort stage Operator

- $sort stage operator in aggregation Sorts all the input documents and returns them to the pipeline in sorted order.
- **Syntax**

{ $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }

$sort stage operator takes a document that specifies the field(s) to sort by and the respective sort order.

sort order can be:

- 1 Sort ascending.
- -1 Sort descending.

$sort stage Operator

- We may use multiple fields to sort the documents in different orders as specified. If sorting on multiple fields, sort order is evaluated from left to right.
- For example, if follow the syntax, documents are first sorted by <field1>, then documents with the same <field1> values are further sorted by <field2> and so on.

Example-1

- Lets sort the documents of marks collection, 1st on subject in ascending order then on marks in descending order within the subject. The command will be

db.marks.aggregate({$sort:{sub:1, marks:-1}})

Example-1

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.marks.aggregate( [ {$sort:{sub:1, marks:-1}}])
{ "_id" : ObjectId("5ec104283b6e4f8f5b4f1152"), "name" : "manoj", "class" : "10th", "rollno" : 9, "
sub" : "computer", "marks" : 49 }
{ "_id" : ObjectId("5ec103443b6e4f8f5b4f1148"), "name" : "rohit", "class" : "9th", "rollno" : 3, "s
ub" : "computer", "marks" : 48 }
{ "_id" : ObjectId("5ec103f83b6e4f8f5b4f1150"), "name" : "ajay", "class" : "10th", "rollno" : 8, "s
ub" : "computer", "marks" : 44 }
{ "_id" : ObjectId("5ec103913b6e4f8f5b4f114b"), "name" : "suman", "class" : "9th", "rollno" : 2, "s
ub" : "computer", "marks" : 41 }
{ "_id" : ObjectId("5ec103583b6e4f8f5b4f1149"), "name" : "rohit", "class" : "9th", "rollno" : 3, "s
ub" : "english", "marks" : 44 }
{ "_id" : ObjectId("5ec103a33b6e4f8f5b4f114c"), "name" : "suman", "class" : "9th", "rollno" : 2, "s
ub" : "english", "marks" : 43 }
{ "_id" : ObjectId("5ec104373b6e4f8f5b4f1153"), "name" : "manoj", "class" : "10th", "rollno" : 9, "
sub" : "english", "marks" : 40 }
{ "_id" : ObjectId("5ec103e73b6e4f8f5b4f114f"), "name" : "ajay", "class" : "10th", "rollno" : 8, "s
ub" : "english", "marks" : 39 }
{ "_id" : ObjectId("5ec103cf3b6e4f8f5b4f114e"), "name" : "ajay", "class" : "10th", "rollno" : 8, "s
ub" : "hindi", "marks" : 45 }
{ "_id" : ObjectId("5ec1041b3b6e4f8f5b4f1151"), "name" : "manoj", "class" : "10th", "rollno" : 9, "
sub" : "hindi", "marks" : 44 }
{ "_id" : ObjectId("5ec103b53b6e4f8f5b4f114d"), "name" : "suman", "class" : "9th", "rollno" : 2, "s
ub" : "hindi", "marks" : 43 }
{ "_id" : ObjectId("5ec1036c3b6e4f8f5b4f114a"), "name" : "rohit", "class" : "9th", "rollno" : 3, "s
ub" : "hindi", "marks" : 41 }
>
```

Aggregation Pipeline

- The aggregation pipeline can use indexes to improve its performance during some of its stages. Some of the pipeline stages which may take advantage of indexes are as under:
  - **$match** : The $match stage can use an index to filter documents if it occurs at the beginning of a pipeline.
  - **$sort** : The $sort stage can use an index as long as it is not preceded by a **$group** stage.

EXAMPLE

- If a collection named marks contains an index { sub: 1}, the following pipeline can use that index to find the first document of each group:
  - db.marks.aggregate({ $sort:{ sub : 1} })
  - db.marks.aggregate({$match:{“class”:”10th”}})

Contents to be covered

- Pipeline Operations-
  - $project
  - $unwind
  - $skip
  - $limit

pipeline operators

Name	Description
$project	The $project function in MongoDB passes along the documents with only the specified fields to the next stage in the pipeline, i.e. it Reshapes each document in the stream, such as by adding new fields or removing existing fields where the field may be the existing fields from the input documents or newly computed fields.
$unwind	The $unwind operator is used to deconstructing an array field from the input documents to output a document for each element. Each output document replaces the array with an element value i.e. Every output document is the input document with the value of the array field replaced by the element.

Project Function

In MongoDB Aggregation, $project function is passed along the documents with only the specified fields to the next stage in the pipeline. This field can be either the existing fields from the input documents or newly computed fields.

:

Syntax

{ $project: { <specifications> } }

Where, the specification contain

- the inclusion of fields,
- the suppression of the _id field,
- the addition of new fields, and
- the resetting the values of existing fields.

Project Function

- **Parameters**

Specification	Description
<field>: <1 or true>	Specify the inclusion of a field.
_id: <0 or false>	Specify the suppression of the _id field. By default _id filed is included in the output document.
<field>: <expression>	Add a new field or reset the value of an existing field.

Project Function

- **Further,**
  - A field that does not exist in the document are going to include, the $project ignores that field inclusion;
  - A new field can be added and value of an existing field can be reset by specifying the field name and set its value to some expression.

Project Function

- **For Example 1 & 2 below, Lets have a school collection as follows**

```
> db.school.find()
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb180"), "stu_id" : "A01", "stu_name" : "vinod", "class" : "2nd", "marks" : 60 }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb181"), "stu_id" : "A02", "stu_name" : "amit", "class" : "2nd", "marks" : 55 }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb182"), "Stu_name" : "arun" }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb183"), "stu_id" : "A04", "stu_name" : "bipin", "class" : "2nd", "marks" : 60 }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb184"), "stu_id" : "A05", "stu_name" : "anil", "class" : "3rd", "marks" : 55 }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb185"), "stu_id" : "A06", "stu_name" : "ajay", "class" : "3rd", "marks" : 70 }
```

Project Function

- **Example-1:** To include some Specific Fields in the Output Documents

db.school.aggregate({ $project : { Stu_id: 1 , stu_name : 1 } })

```
> db.school.aggregate([{$project:{stu_id:1,stu_name:1}}])
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb180"), "stu_id" : "A01", "stu_name" : "vinod" }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb181"), "stu_id" : "A02", "stu_name" : "amit" }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb182") }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb183"), "stu_id" : "A04", "stu_name" : "bipin" }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb184"), "stu_id" : "A05", "stu_name" : "anil" }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb185"), "stu_id" : "A06", "stu_name" : "ajay" }
>
```

- The command includes only the _id, std_id, and stu_name fields in its output documents as shown above. _id field is included by default.

Project Function

- **Example-2 :** To Suppress _id Field in the Output Documents

db.school.aggregate({ $project : { _id: 0, stu_id : 1 ,stu_name : 1 } })

```
> db.school.aggregate([{$project:{_id:0,stu_id:1,stu_name:1}}])
{ "stu_id" : "A01", "stu_name" : "vinod" }
{ "stu_id" : "A02", "stu_name" : "amit" }
{ }
{ "stu_id" : "A04", "stu_name" : "bipin" }
{ "stu_id" : "A05", "stu_name" : "anil" }
{ "stu_id" : "A06", "stu_name" : "ajay" }
>
```

Project Function

- The above command suppresses i.e. excludes the **_id** field but includes the std_id, and stu_name fields in its output documents as we have given value as 0 for _id field.

Example:

- To Include Specific Fields from Embedded Documents
- Lets 1st consider the School collection, the same has been updated and added an embedded document fees having tution, admission and books keys. Now the collection looks like:

Example:

```
> db.school.find()
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb180"), "stu_name" : "vinod", "class" : "2nd", "marks" : 60, "fees" : { "tution"
 : 6000, "admission" : 600, "books" : 700 } }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb181"), "stu_name" : "amit", "class" : "2nd", "marks" : 55, "fees" : { "tution"
: 4000, "admission" : 300, "books" : 300 } }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb182"), "Stu_name" : "arun" }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb183"), "stu_name" : "bipin", "class" : "2nd", "marks" : 45, "fees" : { "tution"
 : 2000, "admission" : 500, "books" : 700 } }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb184"), "stu_name" : "anil", "class" : "3rd", "marks" : 50, "fees" : { "tution"
: 5000, "admission" : 300, "books" : 900 } }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb185"), "stu_name" : "ajay", "class" : "3rd", "marks" : 50, "fees" : { "tution"
: 4500, "admission" : 600, "books" : 200 } }
>
```

Example:

- We wish to include only tuition fee component from the embedded document. We may use dot (.) notation for taking only the tution from fee **{fee.tution:1**} or we may write it like **{fee:{tution:1}}.** The command for this will be:

db.school.aggregate({ $project: { "fee.tution": 1 } })

or we may also write it as

db.schools.aggregate({ $project: { fees: { tution: 1 } } });

Example:

```
> db.school.aggregate([{$project:{"fees.tution":1}}])
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb180"), "fees" : { "tution" : 6000 } }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb181"), "fees" : { "tution" : 4000 } }
{ "_id" : ObjectId("5ec233f75ddaf5609c6cb182") }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb183"), "fees" : { "tution" : 2000 } }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb184"), "fees" : { "tution" : 5000 } }
{ "_id" : ObjectId("5ec236255ddaf5609c6cb185"), "fees" : { "tution" : 4500 } }
>
```

For adding new fields

- syntax

db.collectionname.aggregate({$project:{field:1,”newfield”:”$oldfield”}})

Command Prompt - mongo

```
... ...        }
... ...    ]
... ... )
> db.employee.aggregate([{$project: {emp_code:1,employeename: "$empname"}}])
{ "_id" : ObjectId("553f35a0eff0e6345e7c95e7"), "emp_code" : "E001" }
{ "_id" : ObjectId("553f35d1eff0e6345e7c95e8"), "emp_code" : "E002" }
{ "_id" : ObjectId("553f39edeff0e6345e7c95e9"), "emp_code" : "E003" }
{ "_id" : ObjectId("553f3c20eff0e6345e7c95ea"), "emp_code" : "E004" }
> db.employee.aggregate([{$project: {emp_code:1,employeename: "$emp_name"}}])
{ "_id" : ObjectId("553f35a0eff0e6345e7c95e7"), "emp_code" : "E001", "employeename" : "smita" }
{ "_id" : ObjectId("553f35d1eff0e6345e7c95e8"), "emp_code" : "E002", "employeename" : "ankit" }
{ "_id" : ObjectId("553f39edeff0e6345e7c95e9"), "emp_code" : "E003", "employeename" : "rohit" }
{ "_id" : ObjectId("553f3c20eff0e6345e7c95ea"), "emp_code" : "E004", "employeename" : "sarita" }
> db.employee.find()
{ "_id" : ObjectId("553f35a0eff0e6345e7c95e7"), "emp_code" : "E001", "emp_name" : "smita", "date_of_join" : "16/10/2010"
, "salary" : 8000 }
{ "_id" : ObjectId("553f35d1eff0e6345e7c95e8"), "emp_code" : "E002", "emp_name" : "ankit", "date_of_join" : "16/06/2011"
, "salary" : 6000 }
{ "_id" : ObjectId("553f39edeff0e6345e7c95e9"), "emp_code" : "E003", "emp_name" : "rohit", "date_of_join" : "25/12/2010"
, "salary" : 8000 }
{ "_id" : ObjectId("553f3c20eff0e6345e7c95ea"), "emp_code" : "E004", "emp_name" : "sarita", "date_of_join" : "16/10/2010
", "salary" : "8000" }
>
```

Command Prompt - mongo

```
{ "_id" : ObjectId("553f35a0eff0e6345e7c95e7"), "emp_code" : "E001", "employeename" : "smita" }
{ "_id" : ObjectId("553f35d1eff0e6345e7c95e8"), "emp_code" : "E002", "employeename" : "ankit" }
{ "_id" : ObjectId("553f39edeff0e6345e7c95e9"), "emp_code" : "E003", "employeename" : "rohit" }
{ "_id" : ObjectId("553f3c20eff0e6345e7c95ea"), "emp_code" : "E004", "employeename" : "sarita" }
> db.employee.find()
{ "_id" : ObjectId("553f35a0eff0e6345e7c95e7"), "emp_code" : "E001", "emp_name" : "smita", "date_of_join" : "16/10/2010"
, "salary" : 8000 }
{ "_id" : ObjectId("553f35d1eff0e6345e7c95e8"), "emp_code" : "E002", "emp_name" : "ankit", "date_of_join" : "16/06/2011"
, "salary" : 6000 }
{ "_id" : ObjectId("553f39edeff0e6345e7c95e9"), "emp_code" : "E003", "emp_name" : "rohit", "date_of_join" : "25/12/2010"
, "salary" : 8000 }
{ "_id" : ObjectId("553f3c20eff0e6345e7c95ea"), "emp_code" : "E004", "emp_name" : "sarita", "date_of_join" : "16/10/2010
", "salary" : "8000" }
> db.employee.aggregate([{$project: {emp_code:1,employeename:"$emp_name"}}])
{ "_id" : ObjectId("553f35a0eff0e6345e7c95e7"), "emp_code" : "E001", "employeename" : "smita" }
{ "_id" : ObjectId("553f35d1eff0e6345e7c95e8"), "emp_code" : "E002", "employeename" : "ankit" }
{ "_id" : ObjectId("553f39edeff0e6345e7c95e9"), "emp_code" : "E003", "employeename" : "rohit" }
{ "_id" : ObjectId("553f3c20eff0e6345e7c95ea"), "emp_code" : "E004", "employeename" : "sarita" }
>
```

$unwind stage operator

- $unwind is another very important operator in Aggregation. It is used to deconstruct an array field from the input documents to output a document for ***each*** element i.e. each output document is the input document with the value of the array field replaced by the element.
- **Syntax**
  - It has two different syntax based on what information is passed. We may pass a field path operand or a document operand to deconstruct an array field.

$unwind stage operator

- **Field Path Operand**

{ $unwind: <field path> }

– Where, field path is an array field prefix with a dollar sign and enclosed in quotes, passed to $unwind. And If the field value is null, missing, or an empty array, $unwind does not output any document.

$unwind stage operator

Document Operand with Options

```
{
  $unwind:
    {
      path: <field path>,
          preserveNullAndEmptyArrays: <boolean>
```

$unwind stage operator

```
    }
}
```

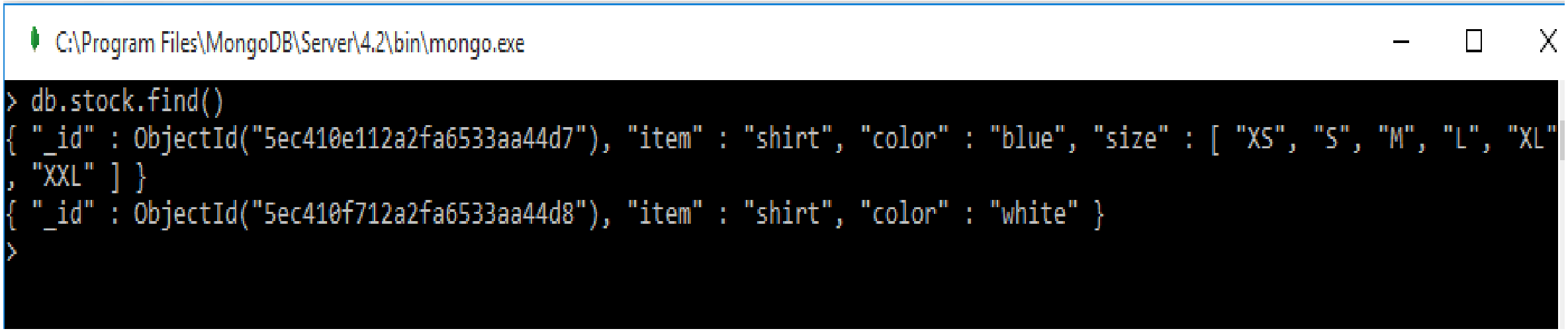
- Where,
  - **field path :** field path is an array field prefix with a dollar sign and enclosed in quotes
  - **preserveNullAndEmptyArrays(Optional) : True or False (**The default value is false).

$unwind stage operator

- **True:** If the path is null, missing, or an empty array, $unwind outputs the document.
- **False:** If path is null, missing, or an empty array, $unwind does not output a document.

Example

- Lets consider the stock collection having the following documents, where size is an array field.



Example

- Now, if we deconstruct the array size, with the following command
  - **db.stock.aggregate({$unwind:"$size"})**

is smiliar to

  - **db.stock.aggregate({ $unwind: { path: "$size" } })**

Example

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.stock.aggregate([{$unwind:"$size"}])
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "XS" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "S" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "M" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "L" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "XL" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "XXL" }
>
>
> db.stock.aggregate( [ { $unwind: { path: "$size" } } ] )
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "XS" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "S" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "M" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "L" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "XL" }
{ "_id" : ObjectId("5ec410e112a2fa6533aa44d7"), "item" : "shirt", "color" : "blue", "size" : "XXL" }
>
```

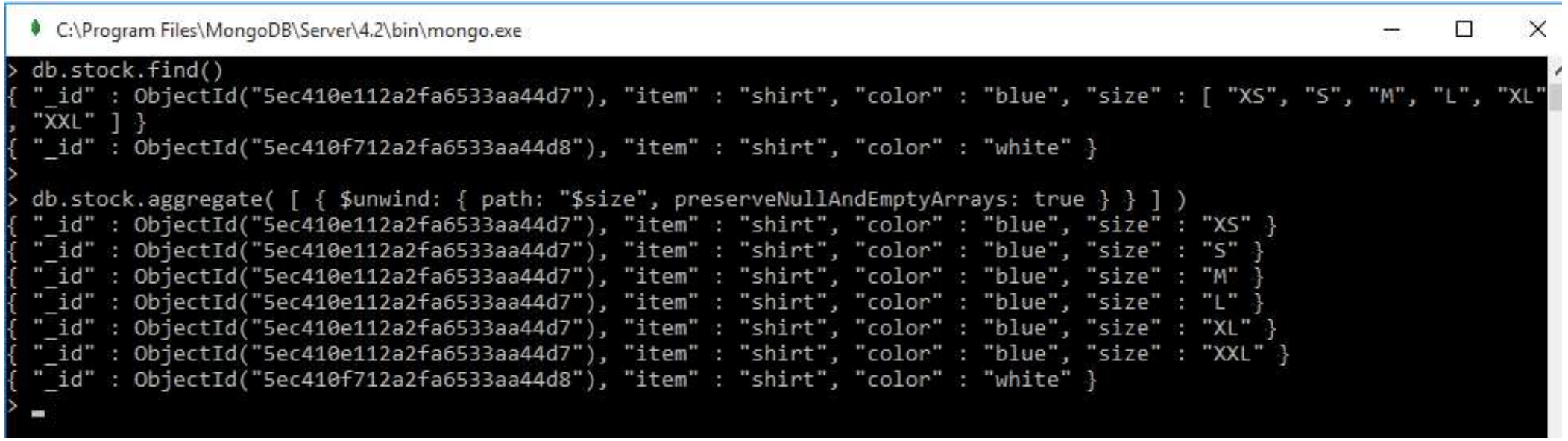
- All the not Null/missing Array are deconstructed and converted to element in output document. In this case the shirt with white color is not in output as it has missing array field.

Example

- Consider the output of the above example, and if we wish to display the documents with missing array field, we may use document operand with options syntax to display this.

db.stock.aggregate([{$unwind:{path:"$size",preserveNullAndEmptyArrays : true } }])

Example



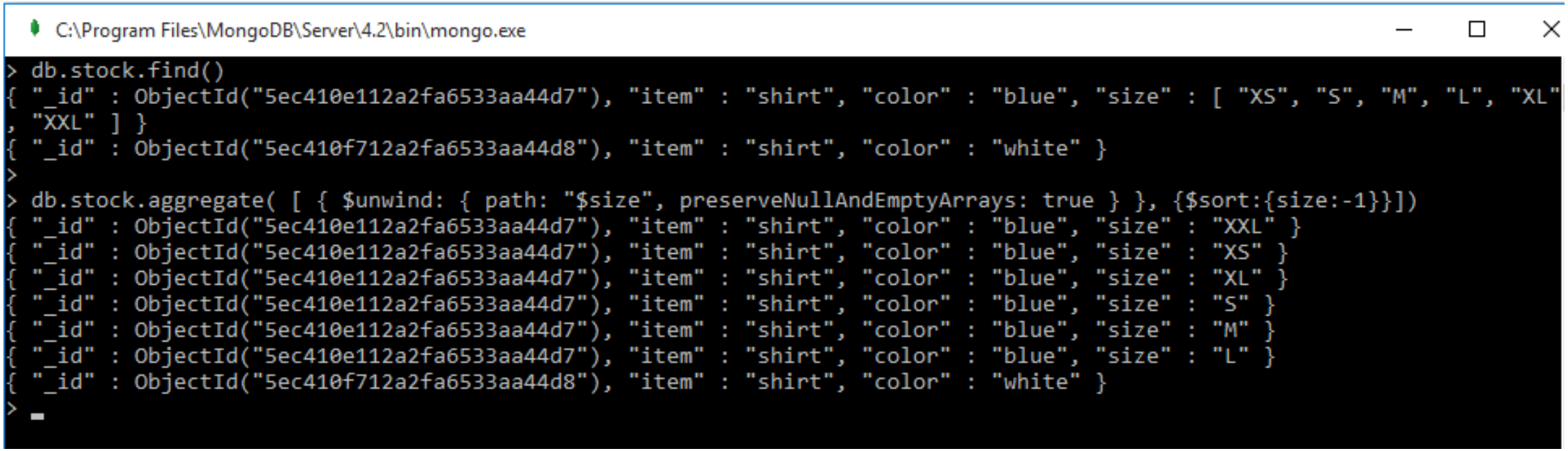
- Output documents also displays the input documents where the path field is missing

$sort stage Operator

- **Example**
  - We have stock collection, in the following example we first unwind the collection on Array and then sorted the document on the size element of output documents in descending order.

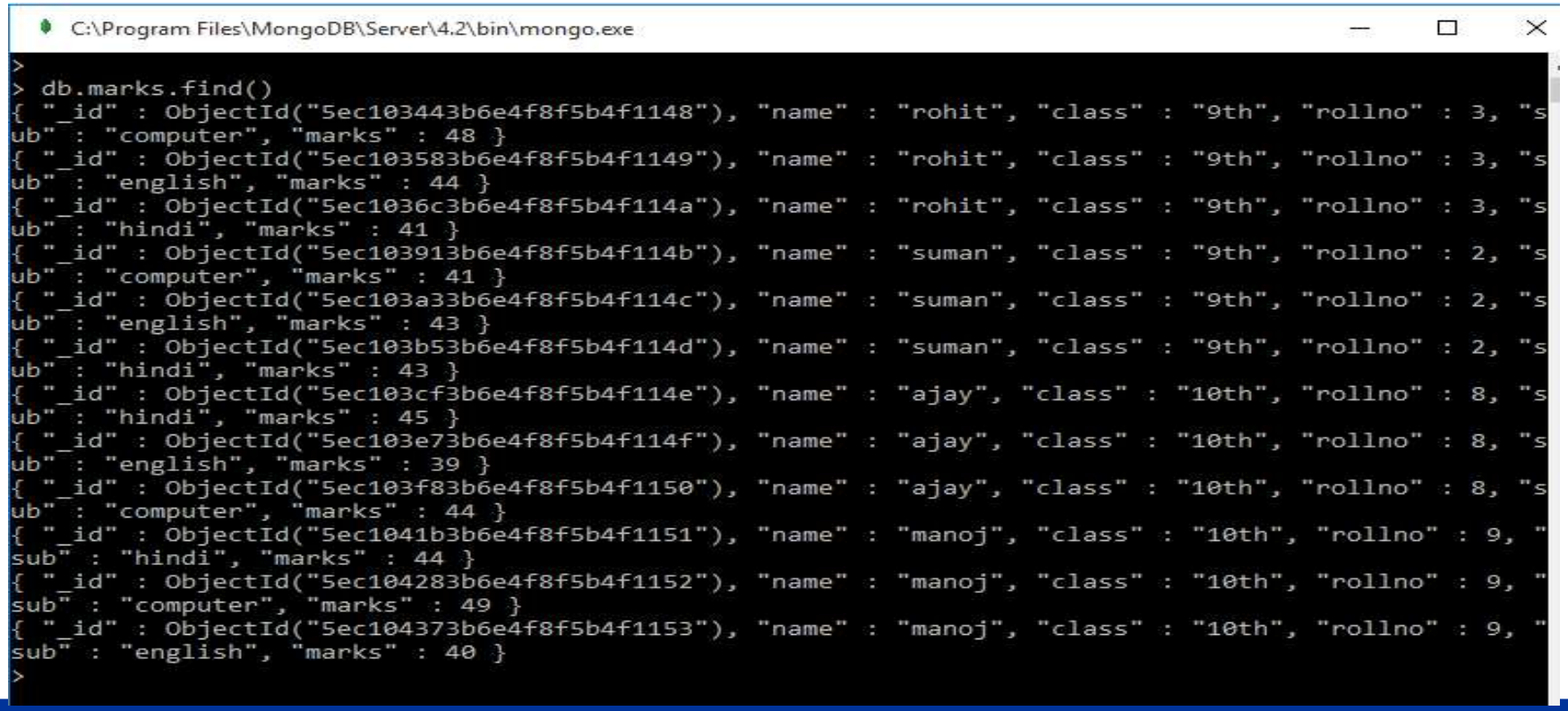
```
db.stock.aggregate( [ { $unwind: { path: "$size",
preserveNullAndEmptyArrays: true } }, {$sort:{size:-1}}])
```

$sort stage Operator



Example

- Lets consider marks collection, having the following documents:



pipeline operators

Name	Description
$skip	$skip operator Skips the first n documents where n is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first n documents) or one document (if after the first n documents).

$skip stage operator

- $skip stage operator skips the specified number of documents that pass into the stage and passes the remaining documents to the next stage in the pipeline. $skip has no effect on the content of the documents it passes along the pipeline.
- **Syntax**

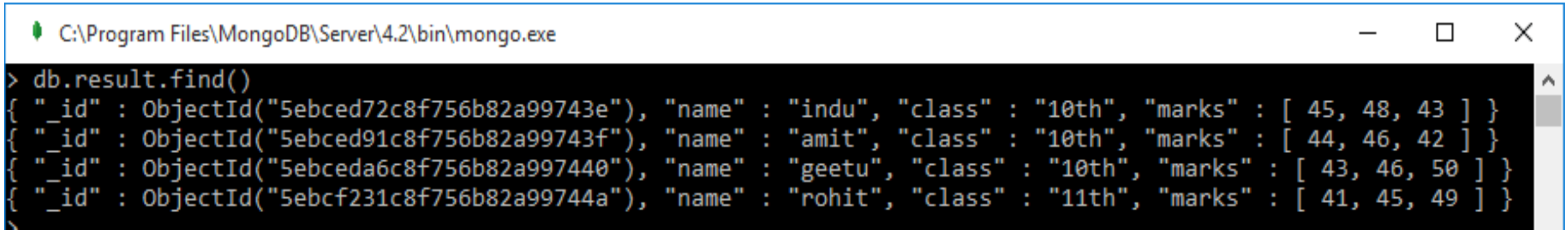
  { $skip: <positive integer> }

  Where,

  – Positive integer specifies the maximum number of documents to skip to be passed with $skip stage operator.

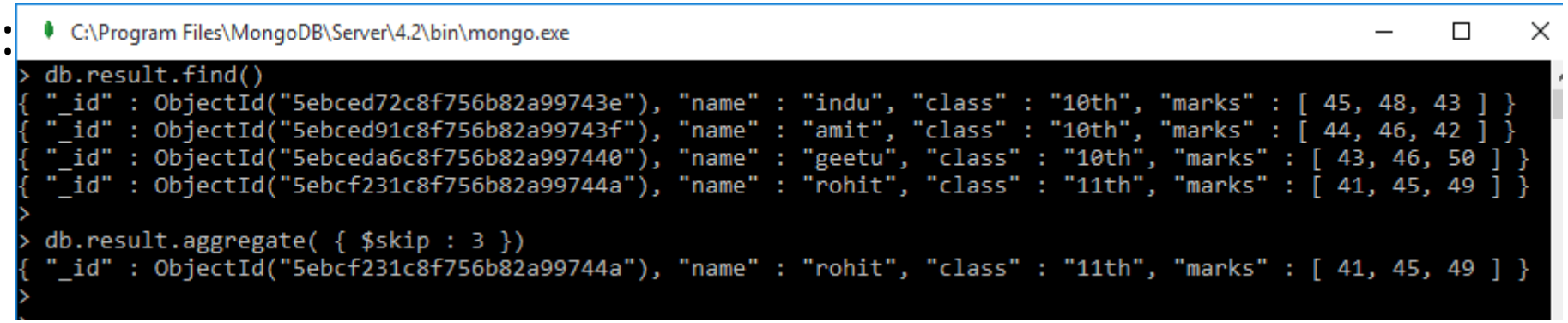
$skip stage operator

- **Example**
- Consider the result collection having 4 documents:



$skip stage operator

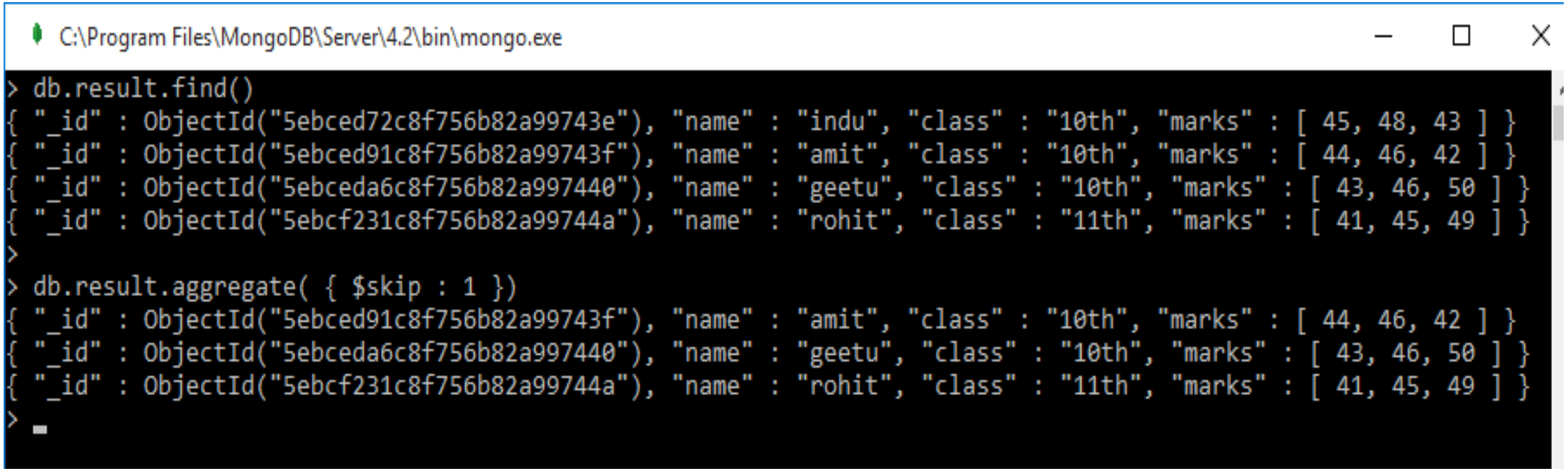
- If we want to skip first 3 records of result collection, then the command will be
- **db.result.aggregate({ $skip : 3 })**
  - This operation skips the first 3 documents passed to it by the pipeline



$skip stage operator

- If we want to skip only the first records of result collection, then the command will be
- **db.result.aggregate({ $skip : 1 })**
  - This operation skips the first documents passed to it by the pipeline :

$skip stage operator



pipeline operators

Name	Description
$limit	$limit operator Passes the first n documents unmodified to the pipeline where n is the specified limit. For each input document, outputs either one document (for the first n documents) or zero documents (after the first n documents).

$limit stage operator

- $limit stage operator returns the first N documents, where N is the specified limit and these output documents may be passed to the next stage in the pipeline.
- **Syntax**

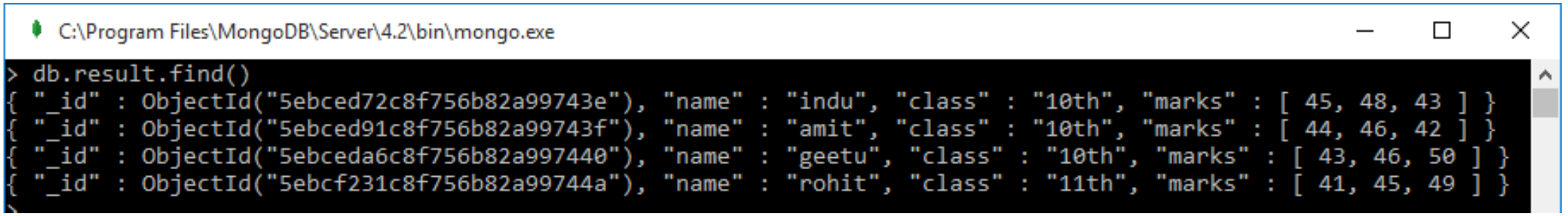
  { $limit: <positive integer> }

  Where,

  – positive integer specifies the maximum number of documents to be passed, i.e. no of documents for next stage in the pipeline.
- **Example**

  – Consider the result collection having 4 documents:

$limit stage operator



- If we want to pass only first 3 records of the result collection, then the command will be
- **db.result.aggregate({ $limit : 3 })**

$limit stage operator

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

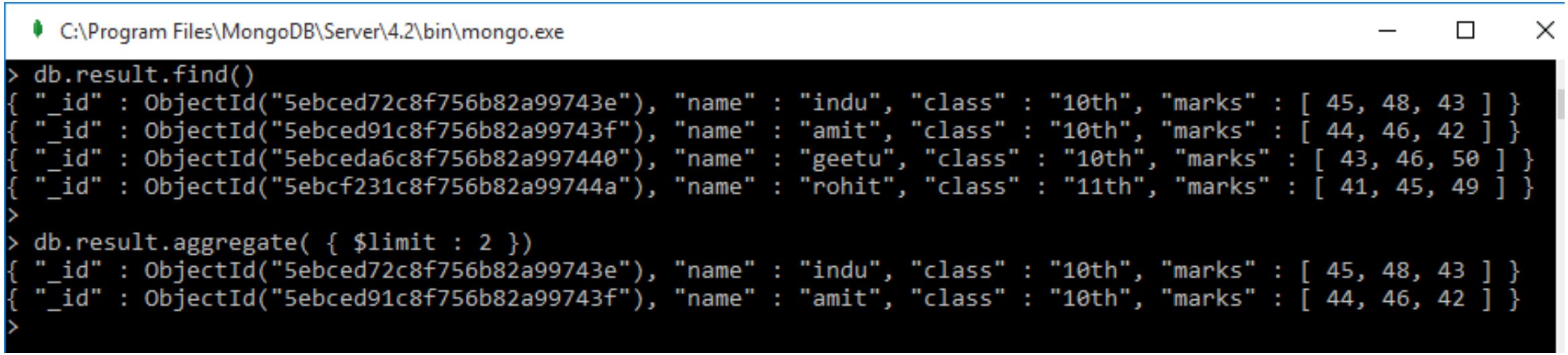
```
> db.result.find()
{ "_id" : ObjectId("5ebced72c8f756b82a99743e"), "name" : "indu", "class" : "10th", "marks" : [ 45, 48, 43 ] }
{ "_id" : ObjectId("5ebced91c8f756b82a99743f"), "name" : "amit", "class" : "10th", "marks" : [ 44, 46, 42 ] }
{ "_id" : ObjectId("5ebceda6c8f756b82a997440"), "name" : "geetu", "class" : "10th", "marks" : [ 43, 46, 50 ] }
{ "_id" : ObjectId("5ebcf231c8f756b82a99744a"), "name" : "rohit", "class" : "11th", "marks" : [ 41, 45, 49 ] }
>
> db.result.aggregate( { $limit : 3 })
{ "_id" : ObjectId("5ebced72c8f756b82a99743e"), "name" : "indu", "class" : "10th", "marks" : [ 45, 48, 43 ] }
{ "_id" : ObjectId("5ebced91c8f756b82a99743f"), "name" : "amit", "class" : "10th", "marks" : [ 44, 46, 42 ] }
{ "_id" : ObjectId("5ebceda6c8f756b82a997440"), "name" : "geetu", "class" : "10th", "marks" : [ 43, 46, 50 ] }
>
```

$limit stage operator

- The above command will pass only top 3 documents.
- If we want to pass only first 2 records of the result collection, then the command will be

db.result.aggregate({ $limit : 2 })

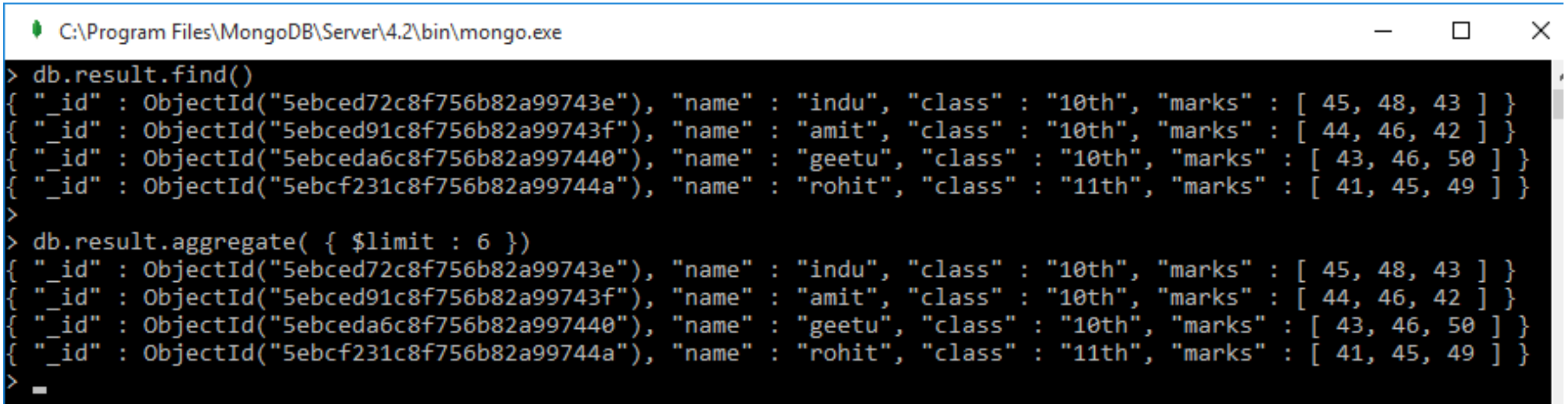
$limit stage operator



$limit stage operator

- If we want to pass only first 6 documents of the result collection, then the command will be
- **db.result.aggregate({ $limit : 6 })**
  - but if the collection is not having the specified documents, then all the documents available in the collection will be passed, no Error will be displayed. Here there are 4 documents which are passed:

$limit stage operator



Contents to be covered

- MapReduce,
- Aggregation commands

Map-Reduce

- **MapReduce** is the data processing mechanism for condensing large amount of data into useful aggregated results. This task is executed by **MapReduce** command which subsequently perform **map** and **reduce** operations.
- **MapReduce** is used for processing large data sets. In straightforward terms, the **MapReduce** command takes two primary inputs, the mapper function, and the reducer function.

Map-Reduce

- The **Mapper** function reads the collection of data and build the **Map** with the required fields that which are required to process and group them into one array. Subsequently, this **key-value** pair is fed into the **Reducer**, which further transform the values. The two phases of Map/Reduce is:
  - **map** phase: filter / transform / convert data
  - **reduce** phase: perform aggregations over the data

Syntax

```
db.collectionName.mapReduce(
   function() {emit(key,value);},                    //here is the map operation
   function(key,values) {return reduceFunction}, {
 //here is the reduce operation
      out: collection,
      query: document,
}
)
```

Syntax

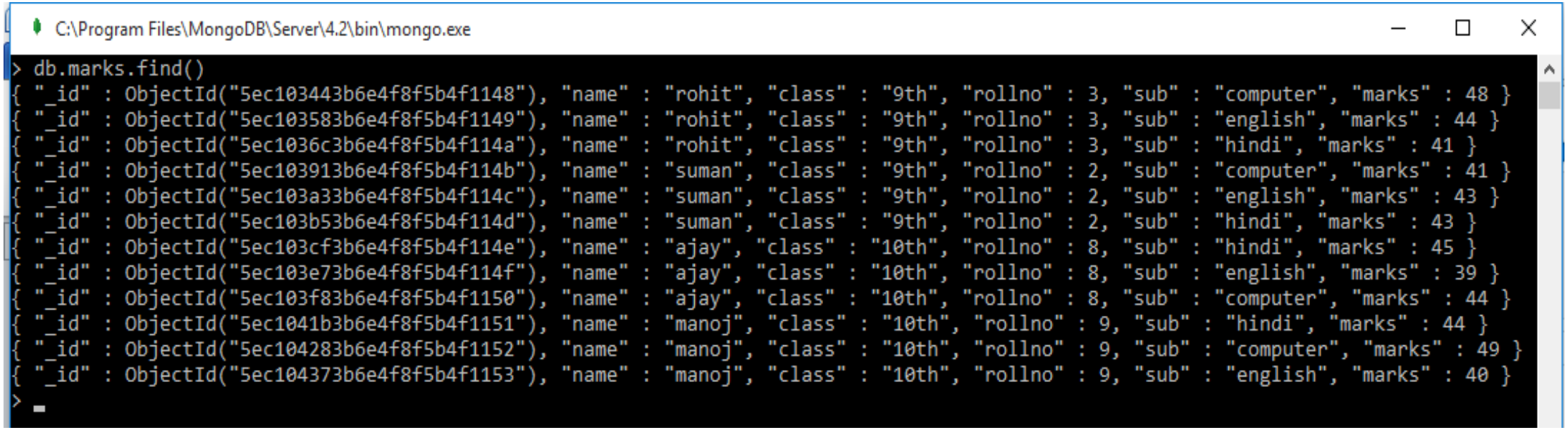
- Where,
  - **map** function maps a value with a key and emits a key-value pair. It's a JavaScript function
  - **reduce** function reduces or groups all the documents having the same key. It's a JavaScript function
  - **out** specifies the location of the map-reduce query result

Syntax

- **query** specifies the optional selection criteria for selecting documents
- The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

Map-Reduce

- Example:
- Lets have the marks collection having the student name, roll, no, class and subject marks in each document.



Map-Reduce

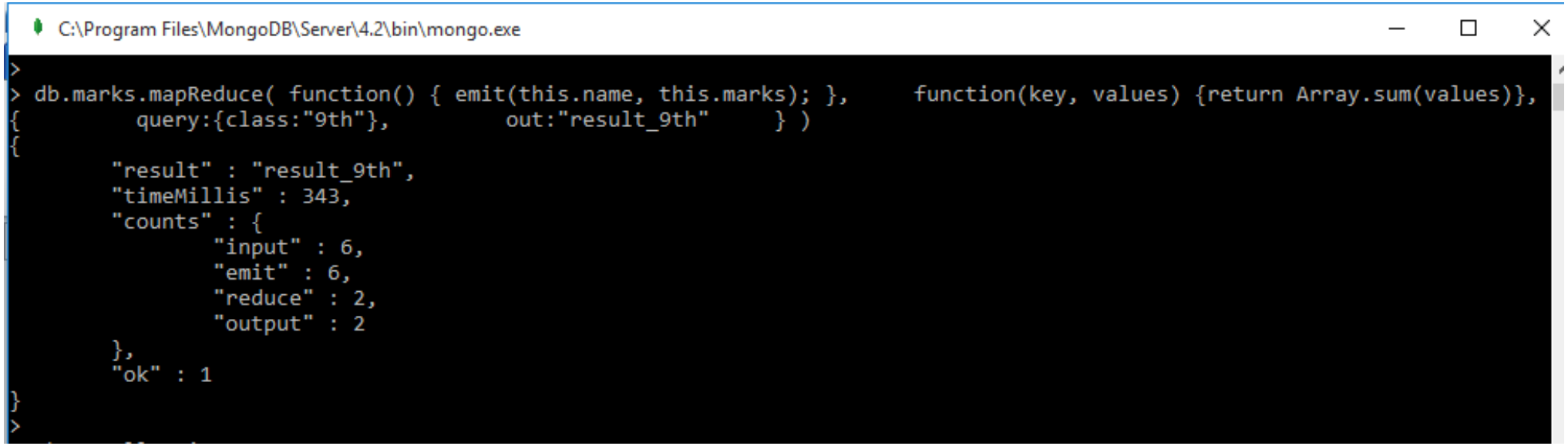
- Now, we will use a mapReduce function on **marks** collection to select all sum of marks of all the students in class **9th**. The candidates of the class will be grouped based on the "**name"** and then the sum the **marks** of each student. The output of the operation will be saved as collection "**result_9th".**
- The code will be

Map-Reduce

```
>db.marks.mapReduce(
 function() { emit(this.name,this.marks); },
function(key, values) {return Array.sum(values)},
      {
   query:{class:"9th"},
   out:"result_9th"
      }
      )
```

Map-Reduce

- As a result a new collection "result_9th" will be created and the same can be seen using show collections command. The documents in the newly created collection may be seen using find() command.



Map-Reduce

- The result of the operation displays that
  - Total 6 documents matched the query (class:“9th”),
  - The map function emitted 6 documents with key-value pairs and
  - Finally the reduce function grouped mapped documents having the same keys into 2 documents.

Map-Reduce

- We may list the collections, and also see the documents in the collection :

```
>
> show collections
abc
college
marks
old
result
result_9th
result_totals
school
stock
store
>
> db.result_9th.find()
{ "_id" : "rohit", "value" : 133 }
{ "_id" : "suman", "value" : 127 }
```

Map-Reduce

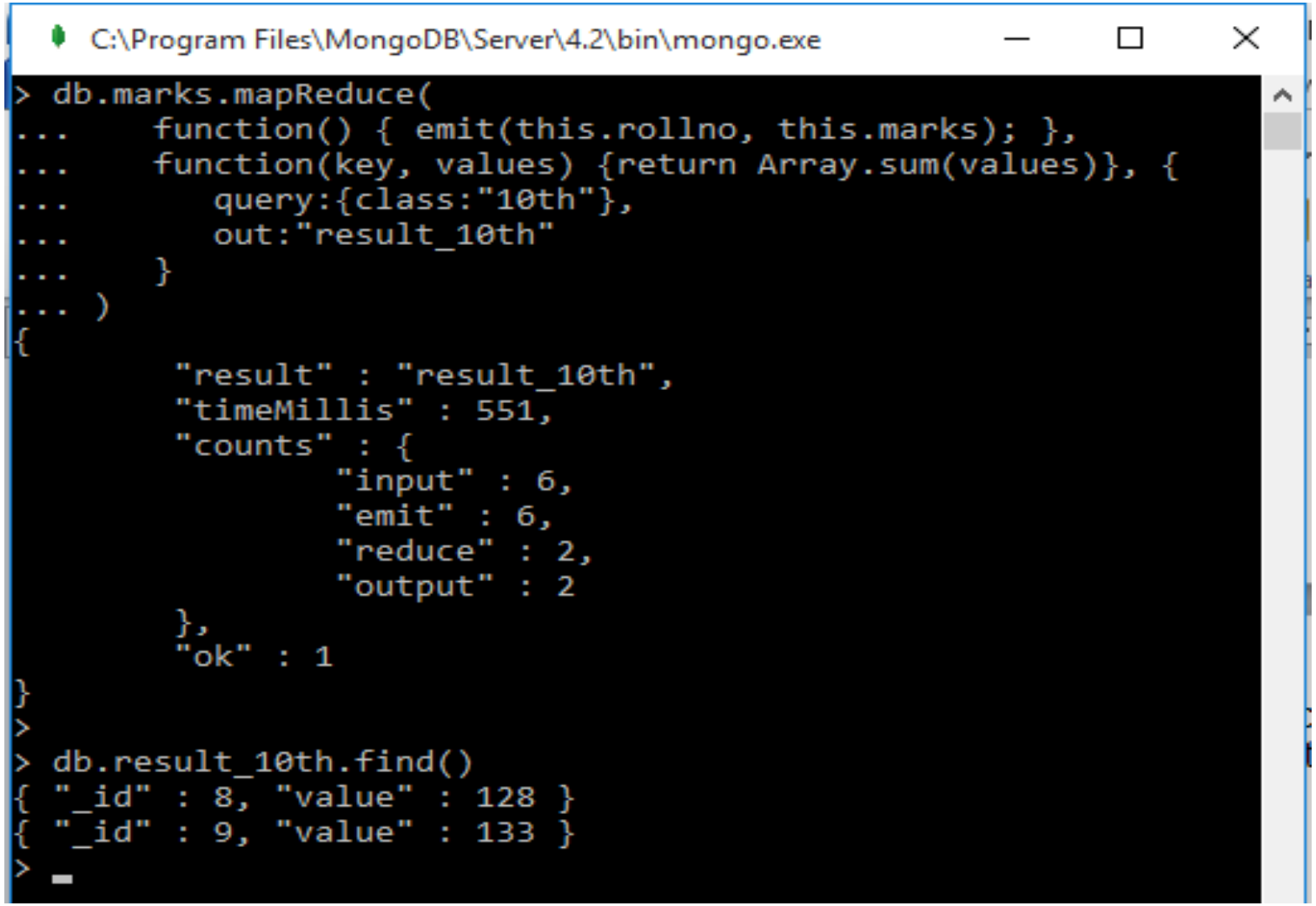
- Lets use mapReduce function on **marks** collection again to find sum of marks of all the students in class **10th**. The candidates of the class will be grouped based on the **rollno** and then the count their marks of each student. The output of the operation will be saved as collection “**result_10th”.**

>

Map-Reduce

```
db.marks.mapReduce(
   function() { emit(this.rollno, this.marks); },
function(key, values) {return Array.sum(values)}, {
     query:{class:"10th"},
     out:"result_10th"
        }
        )
```

Map-Reduce


C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.marks.mapReduce(
... function() { emit(this.rollno, this.marks); },
... function(key, values) {return Array.sum(values)}, {
... query:{class:"10th"},
... out:"result_10th"
... }
...)
{
"result" : "result_10th",
"timeMillis" : 551,
"counts" : {
"input" : 6,
"emit" : 6,
"reduce" : 2,
"output" : 2
},
"ok" : 1
}
>
> db.result_10th.find()
{ "_id" : 8, "value" : 128 }
{ "_id" : 9, "value" : 133 }
>

Map-Reduce

- **Note:**
  - In above examples, the resulted documents are saved in the collection and we have to run find() command to show them.
  - We can use find () command with the mapReducer command to display the resulted documents after the operation. The command for the this to find the result of 10^{th} class is

Map-Reduce

```
>db.marks.mapReduce(
  function() { emit(this.rollno, this.marks); },
 function(key, values) {return Array.sum(values)}, {
    query:{class:"10th"},
    out:"result_10th"
        })
```

Aggregation Commands

- Aggregation functions perform operations on groups of documents and return the computed result. Aggregation operation mainly groups the data/ values from the multiple documents and perform various operations on the grouped data and returns a single or multiple results.

Aggregation Commands

- MongoDB performs aggregate operations in one of the following three ways:
  - Pipeline.
  - Map Reduce
  - Single-purpose aggregate methods and commands.
- We have already studied about these. We will emphasis once again on the aggregation commands and methods.

Single-purpose aggregate commands

- These are used for specific aggregation operations on sets of data. Single-purpose aggregate commands and methods are less complex but with limited scope compared to pipeline and map reduce operations. Single-purpose aggregate commands provide straightforward semantics for common data processing options. These are
  - db.collection.count()
  - db.collection.distinct()
  - db.collection.estimatedDocumentCount()

Single-purpose aggregate commands

- All of these operations aggregate documents from a single collection. These operations provide simple access to common aggregation processes, but not as flexible and capable like aggregation pipeline and map-reduce.

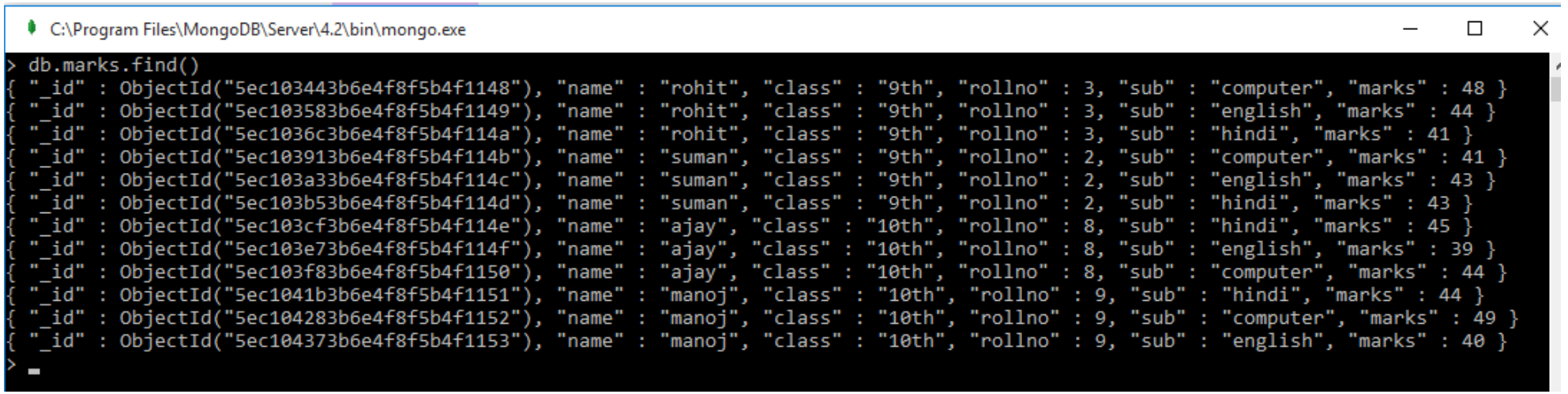
Single-purpose aggregate commands

- **db.collection.count() i.e. count**
  - The Count operation returns the count of documents from the collection. It takes a number of documents and depending on the match query returns the count of the documents. It is different from find() as find returns the documents while it returns the number of documents matching the query.
- **Syntax**
  - **db.collectionName.count(*query*)**

  where, **Query –** The selection Criteria and it is of document type

Single-purpose aggregate commands

- **Example:**
  - Lets Consider the marks collection having following documents:



Single-purpose aggregate commands

- To find no of documents where the **marks** is greater than 45. The command will be

  db.marks.count({"marks":{$gt:45}})

- **Output**

  2

- To find no of documents where the **marks** is greater than **40** and **class** is **10th**. The command will be

  db.marks.count({$and:[{"marks":{"$gt":40}},{"class":"10th"}]})

  Output

  4

Single-purpose aggregate commands

- **db.collection.estimatedDocumentCount()**
  - db.collection.estimatedDocumentCount() method returns the count of all documents in a collection or view.

Syntax

db.collectionName.estimatedDocumentCount()

- It does not take a query filter and instead uses metadata to return the count for a collection.

Single-purpose aggregate commands

Example:

- Lets Consider the marks collection having following documents:

Single-purpose aggregate commands

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

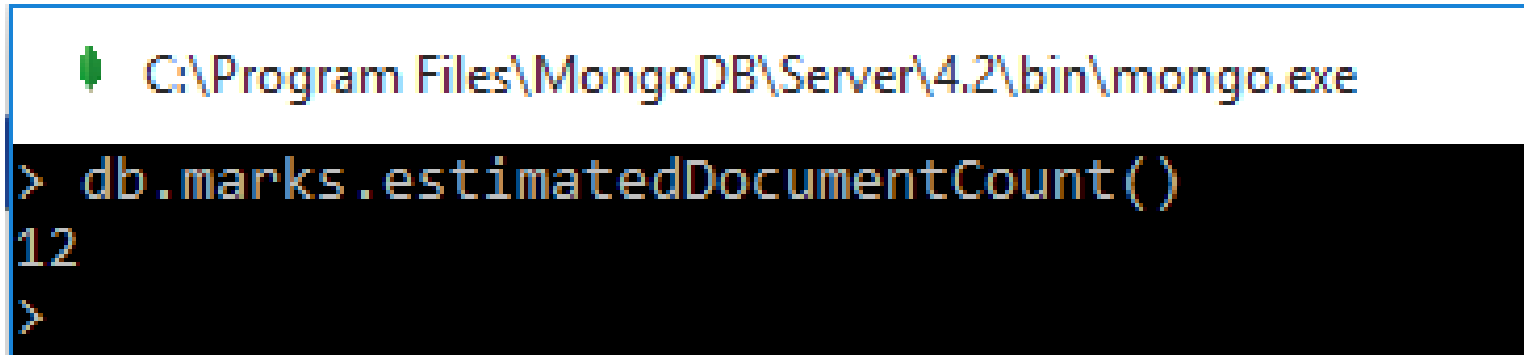
```
> db.marks.find()
{ "_id" : ObjectId("5ec103443b6e4f8f5b4f1148"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "computer", "marks" : 48 }
{ "_id" : ObjectId("5ec103583b6e4f8f5b4f1149"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "english", "marks" : 44 }
{ "_id" : ObjectId("5ec1036c3b6e4f8f5b4f114a"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "hindi", "marks" : 41 }
{ "_id" : ObjectId("5ec103913b6e4f8f5b4f114b"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "computer", "marks" : 41 }
{ "_id" : ObjectId("5ec103a33b6e4f8f5b4f114c"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "english", "marks" : 43 }
{ "_id" : ObjectId("5ec103b53b6e4f8f5b4f114d"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "hindi", "marks" : 43 }
{ "_id" : ObjectId("5ec103cf3b6e4f8f5b4f114e"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "hindi", "marks" : 45 }
{ "_id" : ObjectId("5ec103e73b6e4f8f5b4f114f"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "english", "marks" : 39 }
{ "_id" : ObjectId("5ec103f83b6e4f8f5b4f1150"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "computer", "marks" : 44 }
{ "_id" : ObjectId("5ec1041b3b6e4f8f5b4f1151"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "hindi", "marks" : 44 }
{ "_id" : ObjectId("5ec104283b6e4f8f5b4f1152"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "computer", "marks" : 49 }
{ "_id" : ObjectId("5ec104373b6e4f8f5b4f1153"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "english", "marks" : 40 }
>
```

Single-purpose aggregate commands

- To find no of documents in the marks collection, the command will be

  db.marks.estimatedDocumentCount()

Output



Single-purpose aggregate commands

- **db.collection.distinct()**

  The **db.collection.distinct() command** finds the distinct values for a specified field across a single collection or view and returns the results in an array. It takes a document and depending on the match query returns the unique values for a field.

- **Syntax**

  db.collectionName.distinct(field, *query***)**

  where,

  Field - The field for which to return distinct values.

  Query – specifies the documents from which to retrieve the distinct values. It is optional.

Single-purpose aggregate commands

- **For Array Fields :** If the value of the specified field is an array then the db.collectionName.distinct() considers each element of the array as a separate value.
- **For Embedded Documents:** We may find distinct values from embedded documents also by specifying the specific field e.g. stock.color
- **Example:**

  Lets consider the **marks** collection, having following documents

Single-purpose aggregate commands

C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe

```
> db.marks.find()
{ "_id" : ObjectId("5ec103443b6e4f8f5b4f1148"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "computer", "marks" : 48 }
{ "_id" : ObjectId("5ec103583b6e4f8f5b4f1149"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "english", "marks" : 44 }
{ "_id" : ObjectId("5ec1036c3b6e4f8f5b4f114a"), "name" : "rohit", "class" : "9th", "rollno" : 3, "sub" : "hindi", "marks" : 41 }
{ "_id" : ObjectId("5ec103913b6e4f8f5b4f114b"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "computer", "marks" : 41 }
{ "_id" : ObjectId("5ec103a33b6e4f8f5b4f114c"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "english", "marks" : 43 }
{ "_id" : ObjectId("5ec103b53b6e4f8f5b4f114d"), "name" : "suman", "class" : "9th", "rollno" : 2, "sub" : "hindi", "marks" : 43 }
{ "_id" : ObjectId("5ec103cf3b6e4f8f5b4f114e"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "hindi", "marks" : 45 }
{ "_id" : ObjectId("5ec103e73b6e4f8f5b4f114f"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "english", "marks" : 39 }
{ "_id" : ObjectId("5ec103f83b6e4f8f5b4f1150"), "name" : "ajay", "class" : "10th", "rollno" : 8, "sub" : "computer", "marks" : 44 }
{ "_id" : ObjectId("5ec1041b3b6e4f8f5b4f1151"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "hindi", "marks" : 44 }
{ "_id" : ObjectId("5ec104283b6e4f8f5b4f1152"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "computer", "marks" : 49 }
{ "_id" : ObjectId("5ec104373b6e4f8f5b4f1153"), "name" : "manoj", "class" : "10th", "rollno" : 9, "sub" : "english", "marks" : 40 }
>
```

Single-purpose aggregate commands

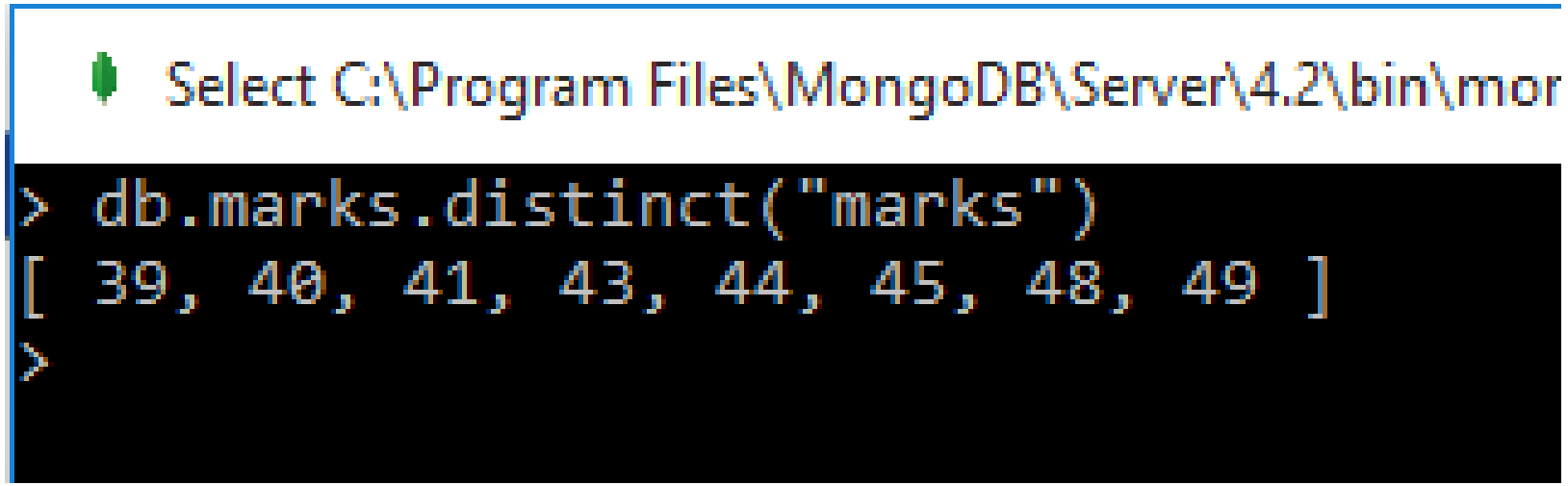
- To find the distinct marks among the various documents from the collection, the command is

  db.marks.distinct("marks")

-

Single-purpose aggregate commands

- **Output**



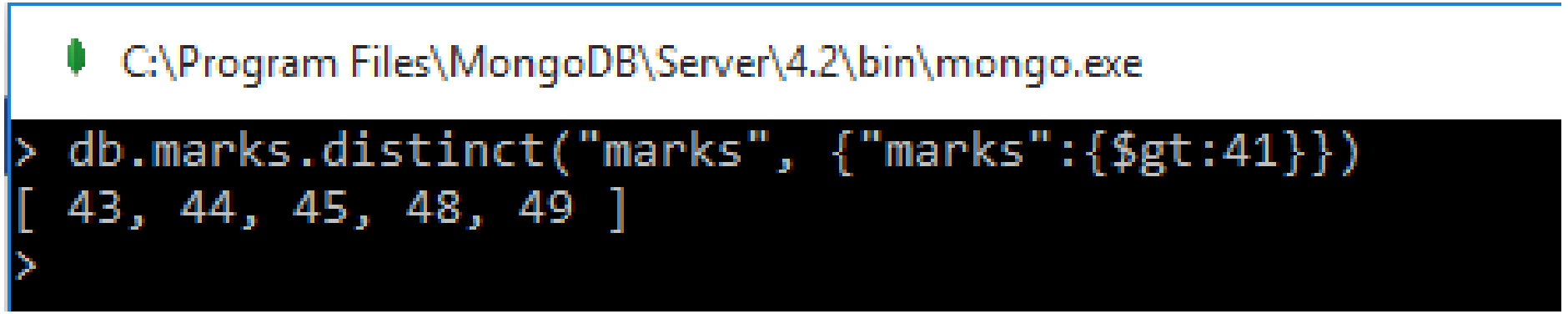
Single-purpose aggregate commands

- To find the distinct marks greater than 41 among the various documents from the collection, the command is

db.marks.distinct("marks", {"marks":{$gt:41}})

Single-purpose aggregate commands

- **Output**



Single-purpose aggregate commands

- To find the distinct marks for the students of the **class 9th** among the various documents matching the query from the collection, the command is

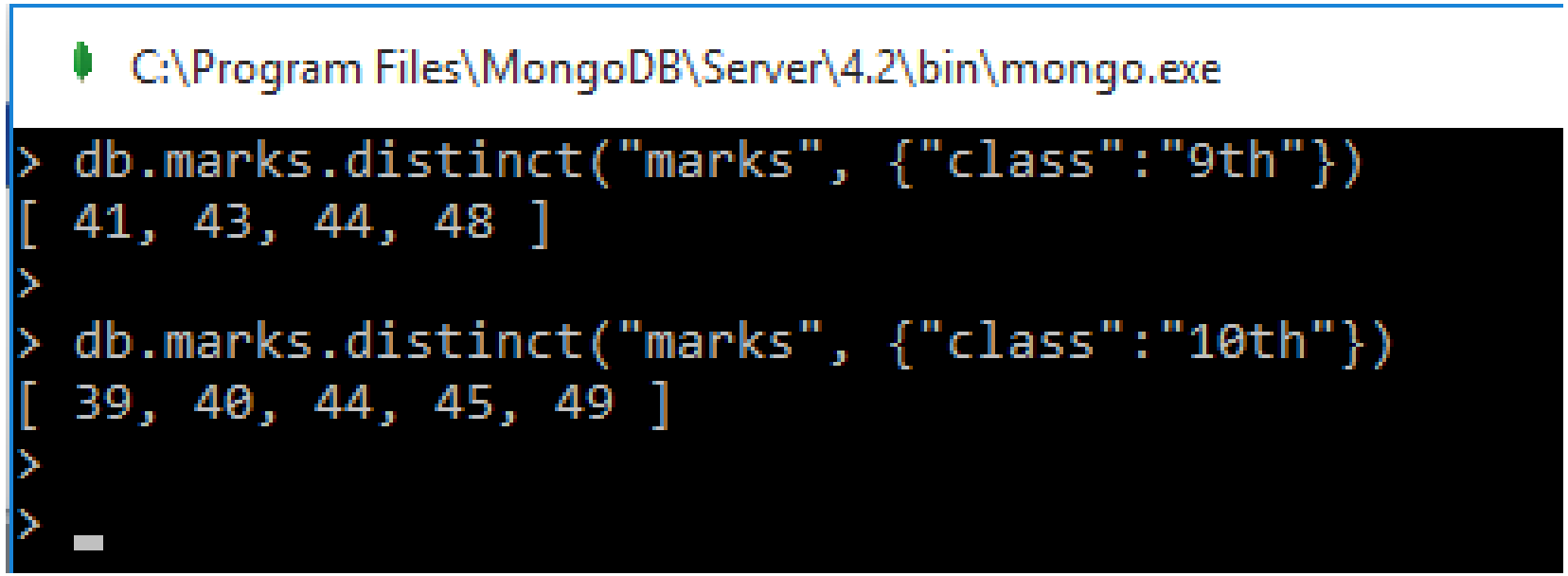
  db.marks.distinct("marks", {"class":"9th"})

- and for class 10th, it will be

  db.marks.distinct("marks", {"class":"9th"})

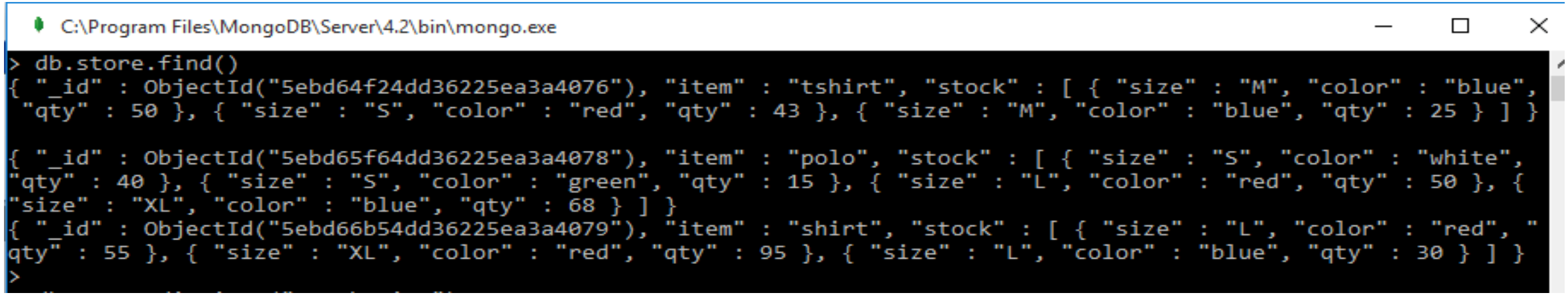
Single-purpose aggregate commands

- **Output**



Single-purpose aggregate commands

- **Example: To find distinct values in case of Embedded documents**
  - Lets consider the store collection having stock as embedded documents.



Single-purpose aggregate commands

- To find the distinct values of sizes from the stock, the command will be

  db.store.distinct("stock.size")

or

- to find the distinct color from the stock, the command will be

  db.store.distinct("stock.color")

-

Single-purpose aggregate commands

- **Output**

```
>
> db.store.distinct("stock.size")
[ "L", "M", "S", "XL" ]
>
> db.store.distinct("stock.color")
[ "blue", "green", "red", "white" ]
>
```

Single-purpose aggregate commands

- **Example: To find values in array fields**
  - Lets consider, the **result** collection having marks as an array field.

```
C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe
> db.result.find()
{ "_id" : ObjectId("5ebced72c8f756b82a99743e"), "name" : "indu", "class" : "10th", "marks" : [ 45, 48, 43 ] }
{ "_id" : ObjectId("5ebced91c8f756b82a99743f"), "name" : "amit", "class" : "10th", "marks" : [ 44, 46, 42 ] }
{ "_id" : ObjectId("5ebceda6c8f756b82a997440"), "name" : "geetu", "class" : "10th", "marks" : [ 43, 46, 50 ] }
{ "_id" : ObjectId("5ebcf231c8f756b82a99744a"), "name" : "rohit", "class" : "11th", "marks" : [ 41, 45, 49 ] }
>
```

Single-purpose aggregate commands

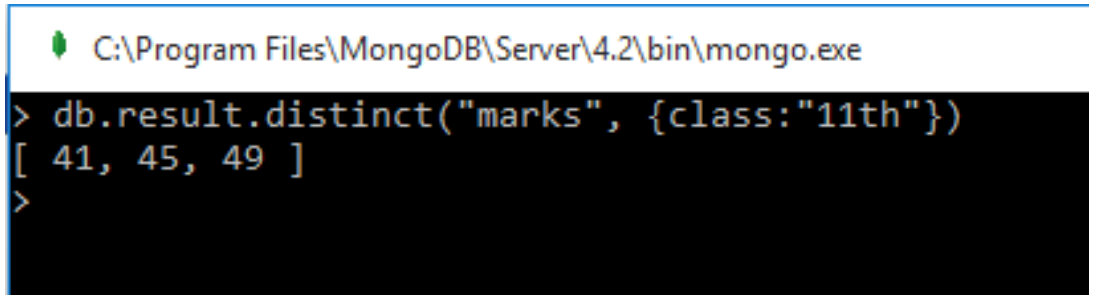
- To find all distinct marks, the command will be
  - **db.result.distinct("marks")**

```
>
> db.result.distinct("marks")
[ 41, 42, 43, 44, 45, 46, 48, 49, 50 ]
>
```

Single-purpose aggregate commands

- To find all distinct marks of Class 11th, the command will be

  db.result.distinct("marks", {class:"11th"})



Contents to be covered

- Backup and restore
- Export and import of data

Backup and Restore

- MongoDB is one of the most popular NoSQL database engines. It is famous for being scalable, powerful, reliable and easy to use.
- The backup and restore operations create or use MongoDB-specific binary data, which preserves not only the consistency and integrity of your data but also its specific MongoDB attributes.
- Backup is also required to meet any unforeseen circumstances apart from migration of database. For these tasks, backup and restore methods are useful as long as the source and target systems are compatible.

Creating Backup of Data

- **mongodump** command can be used to create database in MongoDB. This command will dump the entire data of your server into the dump directory. The dump directory is automatically created under the current directory i.e. bin. You may carry the .bson files to new system, copy the same and use them further.
- **Syntax**

  >mongodump

Creating Backup of Data

- **Steps to run the command**
  - Start mongod server.
  - Open Windows command Prompt
  - Go the bin directory of MongoDB Server. Where it has been installed. (for example c:\program files\mongodb\server\4.2\bin)

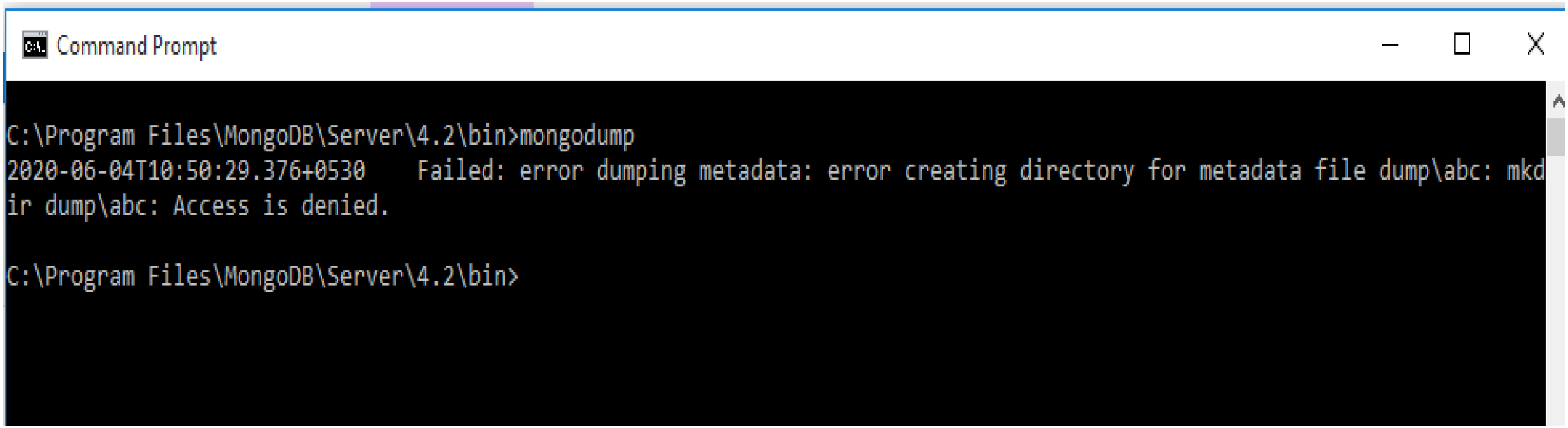
Creating Backup of Data

- Now, type the command **mongodump**

  >mongodump

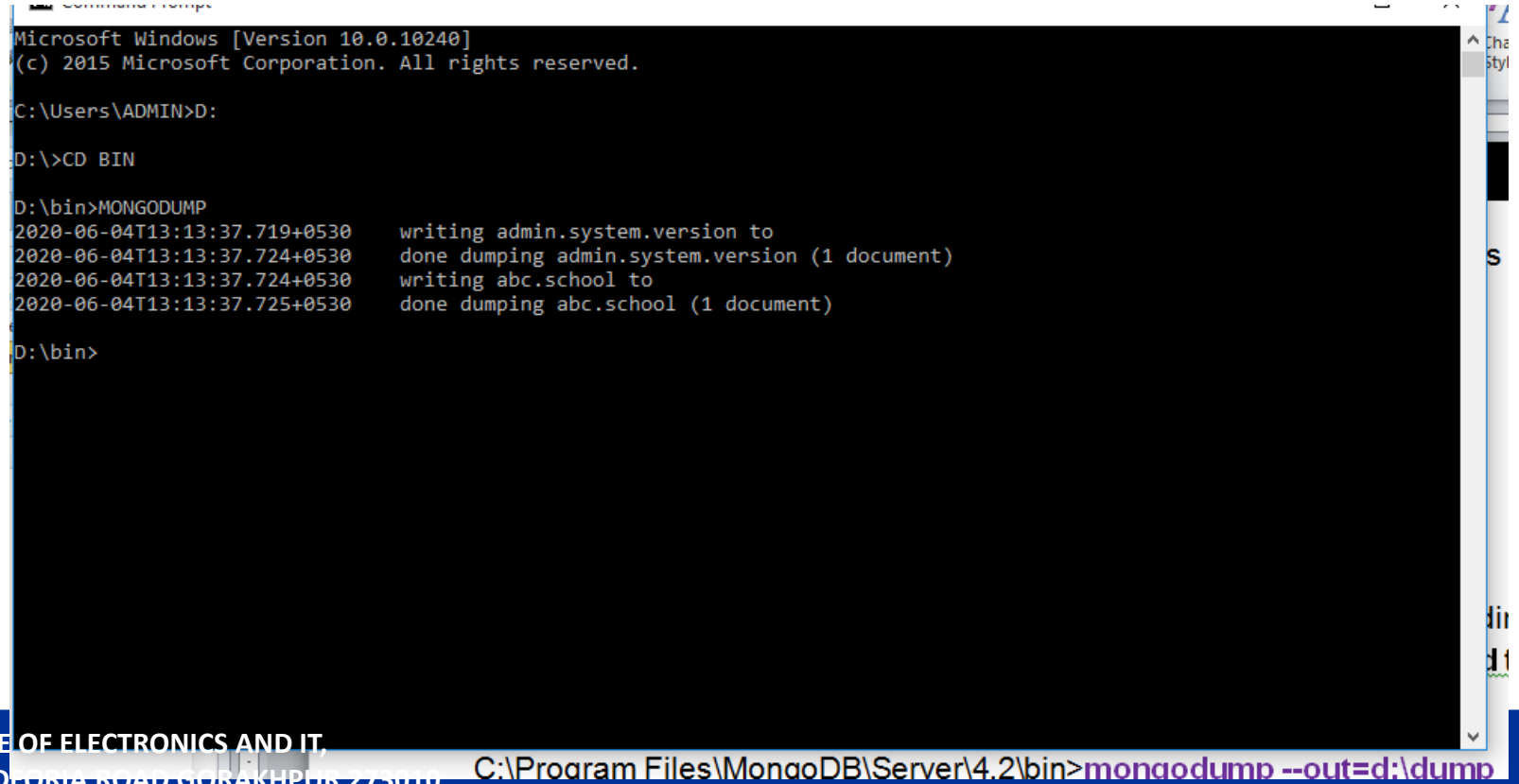
- By default, this command will connect to the server running at port **27017** (the default port) on local host and back all data of the server to directory **/bin/dump/. This command will work only if the directory has full access permission otherwise it will give you an error**. If successful, it will create separate folders for each database and these folders will further have dump files for each collections.

Creating Backup of Data



Creating Backup of Data

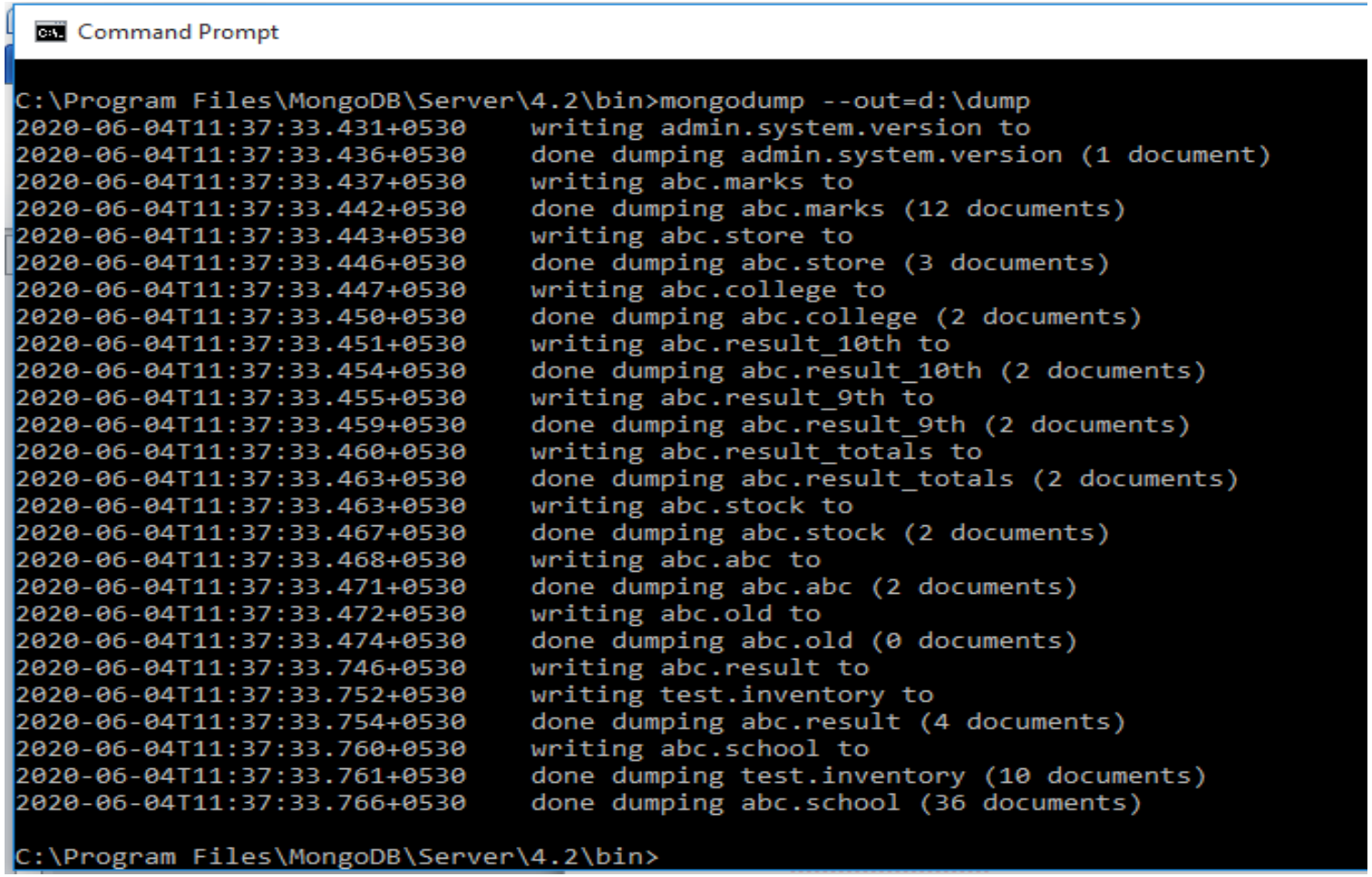
- On my system, it is giving error due to lack of permissions. Solution to this is to grant full permission to the files and folders of your system



Creating Backup of Data

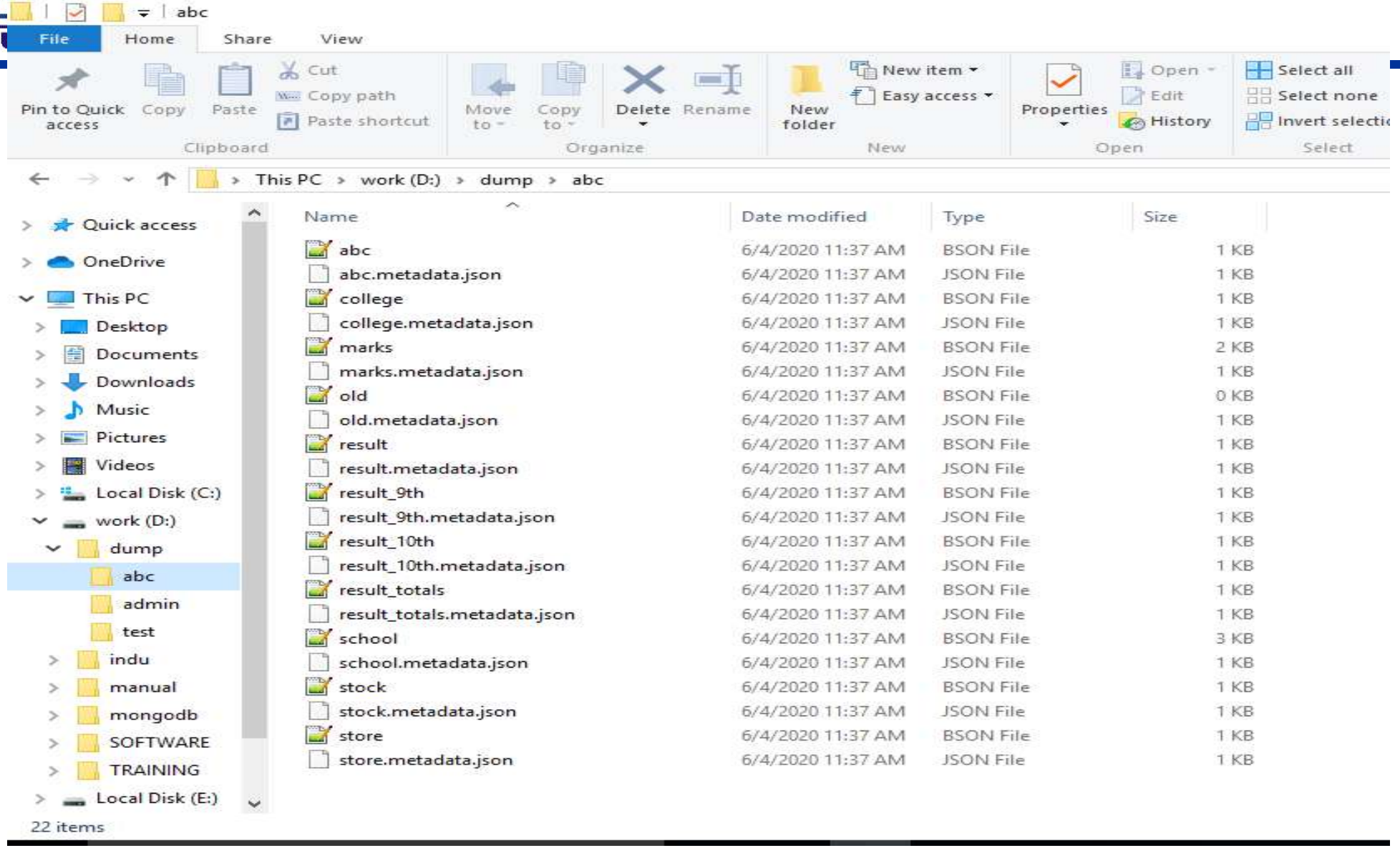
- **Alternatively just give an output path with mongodump command**:
- **Syntax**
  - **Mongodump --out=BACKUP_DIRECTORY**
  - Now run the command with –out parameter and give output path (dump directory as c:\dump). This will take full backup of all the databases. All the **command** to be run from bin folder only.
  - C:\Program Files\MongoDB\Server\4.2\bin>**mongodump --out=d:\dump**

Creating Backup of Data



Creating Backup of Data

- The command is successful, it will create separate create folders for each database and these folders will further have dump files for each collections.

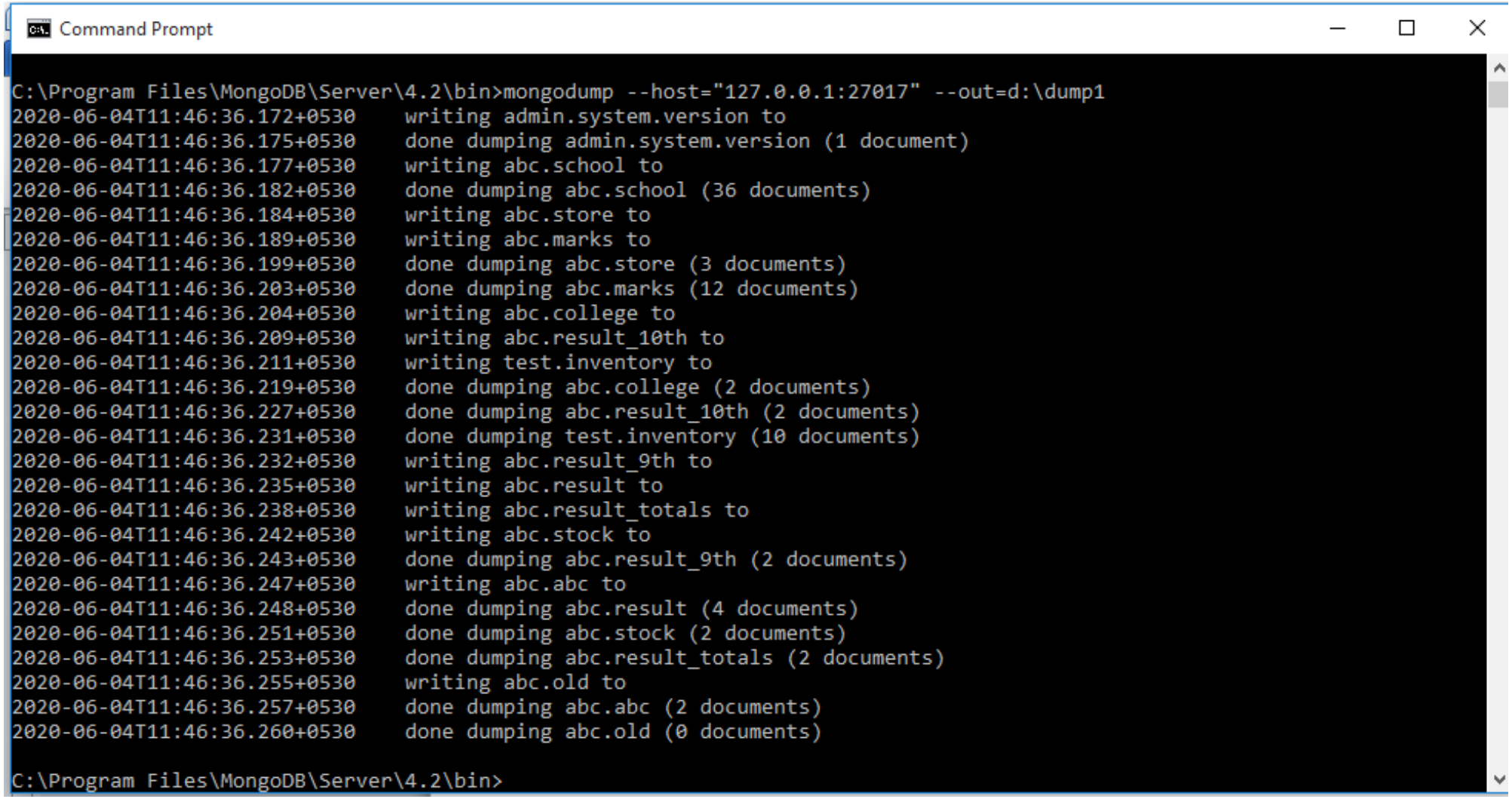

abc
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New item
Easy access
New folder
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selectio
Select
This PC › work (D:) › dump › abc
Quick access
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
work (D:)
dump
abc
admin
test
indu
manual
mongodb
SOFTWARE
TRAINING
Local Disk (E:)
22 items
Name
Date modified
Type
Size
abc 6/4/2020 11:37 AM BSON File 1 KB
abc.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
college 6/4/2020 11:37 AM BSON File 1 KB
college.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
marks 6/4/2020 11:37 AM BSON File 2 KB
marks.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
old 6/4/2020 11:37 AM BSON File 0 KB
old.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
result 6/4/2020 11:37 AM BSON File 1 KB
result.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
result_9th 6/4/2020 11:37 AM BSON File 1 KB
result_9th.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
result_10th 6/4/2020 11:37 AM BSON File 1 KB
result_10th.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
result_totals 6/4/2020 11:37 AM BSON File 1 KB
result_totals.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
school 6/4/2020 11:37 AM BSON File 3 KB
school.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
stock 6/4/2020 11:37 AM BSON File 1 KB
stock.metadata.json 6/4/2020 11:37 AM JSON File 1 KB
store 6/4/2020 11:37 AM BSON File 1 KB
store.metadata.json 6/4/2020 11:37 AM JSON File 1 KB

Creating Backup of Data

- Here, we can see that database folders abc, admin and test has been created in dump directory and abc(database folder) has the collections created under abc database.
-
- mongoDB provide various options to take customized backup.

Creating Backup of Data

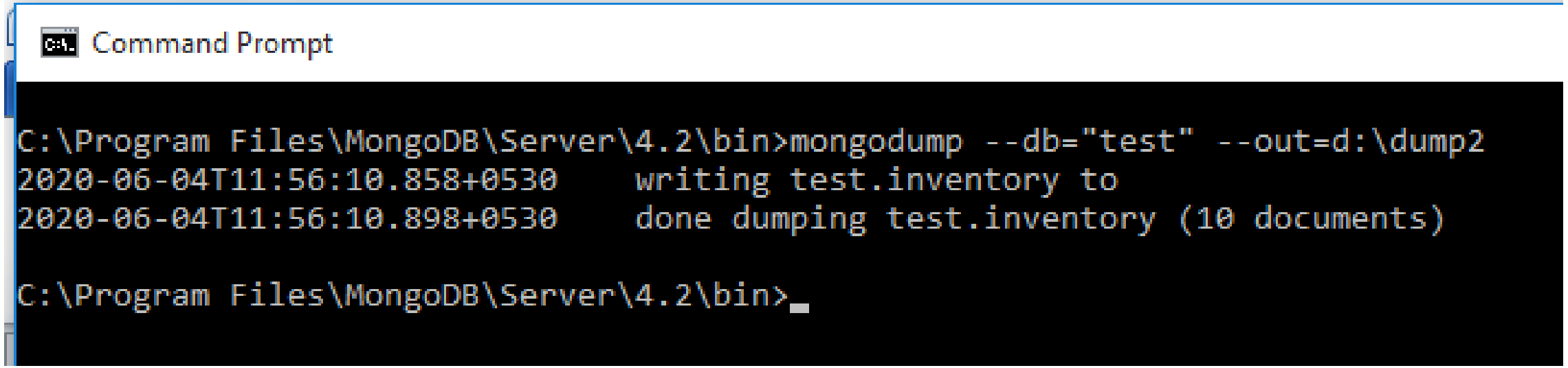
- **mongodump –host=“HOST_NAME:PORT_NUMBER” --out=BACKUP_DIRECTORY**
  - This command is used to take backup of any particular Mongod Instance all databases of specified mongod instance. In case of replicaset, it shall be replica set name followed by the port number. --out is optional.
- **Example**
- **mongodump --host="127.0.0.1:27017" --out=d:\dump1**


Command Prompt
C:\Program Files\MongoDB\Server\4.2\bin>mongodump --host="127.0.0.1:27017" --out=d:\dump1
2020-06-04T11:46:36.172+0530 writing admin.system.version to
2020-06-04T11:46:36.175+0530 done dumping admin.system.version (1 document)
2020-06-04T11:46:36.177+0530 writing abc.school to
2020-06-04T11:46:36.182+0530 done dumping abc.school (36 documents)
2020-06-04T11:46:36.184+0530 writing abc.store to
2020-06-04T11:46:36.189+0530 writing abc.marks to
2020-06-04T11:46:36.199+0530 done dumping abc.store (3 documents)
2020-06-04T11:46:36.203+0530 done dumping abc.marks (12 documents)
2020-06-04T11:46:36.204+0530 writing abc.college to
2020-06-04T11:46:36.209+0530 writing abc.result_10th to
2020-06-04T11:46:36.211+0530 writing test.inventory to
2020-06-04T11:46:36.219+0530 done dumping abc.college (2 documents)
2020-06-04T11:46:36.227+0530 done dumping abc.result_10th (2 documents)
2020-06-04T11:46:36.231+0530 done dumping test.inventory (10 documents)
2020-06-04T11:46:36.232+0530 writing abc.result_9th to
2020-06-04T11:46:36.235+0530 writing abc.result to
2020-06-04T11:46:36.238+0530 writing abc.result_totals to
2020-06-04T11:46:36.242+0530 writing abc.stock to
2020-06-04T11:46:36.243+0530 done dumping abc.result_9th (2 documents)
2020-06-04T11:46:36.247+0530 writing abc.abc to
2020-06-04T11:46:36.248+0530 done dumping abc.result (4 documents)
2020-06-04T11:46:36.251+0530 done dumping abc.stock (2 documents)
2020-06-04T11:46:36.253+0530 done dumping abc.result_totals (2 documents)
2020-06-04T11:46:36.255+0530 writing abc.old to
2020-06-04T11:46:36.257+0530 done dumping abc.abc (2 documents)
2020-06-04T11:46:36.260+0530 done dumping abc.old (0 documents)
C:\Program Files\MongoDB\Server\4.2\bin>

Creating Backup of particular database

- **mongodump --db=DB_name --out=BACKUP_DIRECTORY**
  - We may take backup of specific database instead of all the databases, by specifying the database name with --db option. For example only abc database.
- **Example**
- **mongodump --db="test" --out=d:\dump2**

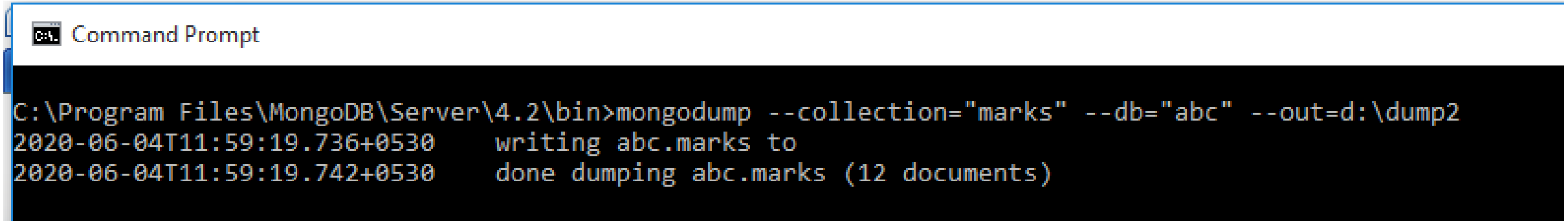
Creating Backup of particular database



Creating Backup of particular collection

- **mongodump --collection= “COLLECTION_name” --db =DB_NAME --out=BACKUP_DIRECTORY**
  - This command can be used if we want to take the backup of specific collection of a particular database.
- **Example**
  - **mongodump --collection="marks“ --db="abc“ --out=d:\dump2**

Creating Backup of Data



- In all the cases, if out path is not specified, it will take backup to dump folder if permitted. It is advisable to take backup on a separate drive / folder

Restore data

- MongoDB's **mongorestore** command is used to restore binary backup created by mongodump. The mongorestore can restore either an entire database backup or a subset of the backup.
- Restore can be performed on the same system or on another system at same location or at different location. It will restore from all the .bson files from dump directory by default. We may copy the data backup to the dump directory and system will restore from that.

Restore data

- **Syntax**

  mongorestore

  Will restore all databases from the dump folder by default

mongorestore <path-of-dumped-database>

  – command is used to store from specific path

- **Example : Restoring from d:\dump folder**

  mongorestore d:\dump

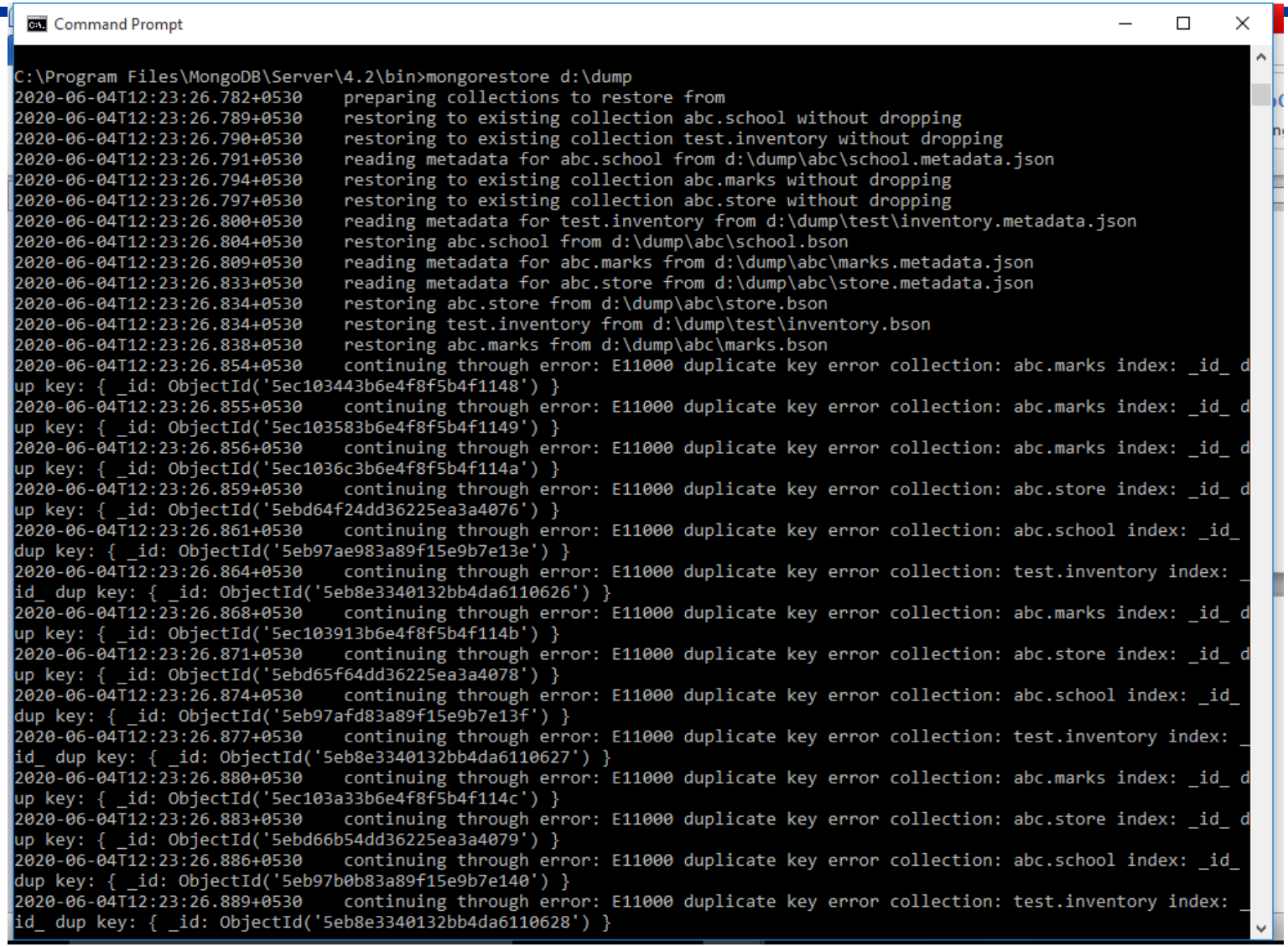
Command Prompt

```
C:\Program Files\MongoDB\Server\4.2\bin>mongorestore
2020-06-04T12:07:53.014+0530    using default 'dump' directory
2020-06-04T12:07:53.016+0530    try 'mongorestore --help' for more information
2020-06-04T12:07:53.017+0530    Failed: mongorestore target 'dump' invalid: CreateFile dump: The system cannot find the
file specified.
2020-06-04T12:07:53.017+0530    0 document(s) restored successfully. 0 document(s) failed to restore.

C:\Program Files\MongoDB\Server\4.2\bin>mongorestore d:\dump
2020-06-04T12:08:14.762+0530    preparing collections to restore from
2020-06-04T12:08:14.768+0530    restoring to existing collection abc.school without dropping
2020-06-04T12:08:14.769+0530    reading metadata for abc.school from d:\dump\abc\school.metadata.json
2020-06-04T12:08:14.770+0530    restoring to existing collection abc.marks without dropping
2020-06-04T12:08:14.771+0530    restoring to existing collection test.inventory without dropping
2020-06-04T12:08:14.771+0530    restoring to existing collection abc.store without dropping
2020-06-04T12:08:14.772+0530    reading metadata for abc.marks from d:\dump\abc\marks.metadata.json
2020-06-04T12:08:14.772+0530    restoring abc.school from d:\dump\abc\school.bson
2020-06-04T12:08:14.772+0530    reading metadata for test.inventory from d:\dump\test\inventory.metadata.json
2020-06-04T12:08:14.775+0530    reading metadata for abc.store from d:\dump\abc\store.metadata.json
2020-06-04T12:08:14.775+0530    restoring test.inventory from d:\dump\test\inventory.bson
2020-06-04T12:08:14.777+0530    restoring abc.marks from d:\dump\abc\marks.bson
2020-06-04T12:08:14.777+0530    restoring abc.store from d:\dump\abc\store.bson
2020-06-04T12:08:14.830+0530    continuing through error: E11000 duplicate key error collection: abc.store index: _id_ d
up key: { _id: ObjectId('5ebd64f24dd36225ea3a4076') }
2020-06-04T12:08:14.831+0530    continuing through error: E11000 duplicate key error collection: abc.store index: _id_ d
up key: { _id: ObjectId('5ebd65f64dd36225ea3a4078') }
2020-06-04T12:08:14.831+0530    continuing through error: E11000 duplicate key error collection: abc.store index: _id_ d
up key: { _id: ObjectId('5ebd66b54dd36225ea3a4079') }
2020-06-04T12:08:14.834+0530    restoring indexes for collection abc.store from metadata
2020-06-04T12:08:14.842+0530    finished restoring abc.store (0 documents, 3 failures)
2020-06-04T12:08:14.842+0530    restoring to existing collection abc.result without dropping
2020-06-04T12:08:14.843+0530    reading metadata for abc.result from d:\dump\abc\result.metadata.json
2020-06-04T12:08:14.846+0530    restoring abc.result from d:\dump\abc\result.bson
2020-06-04T12:08:14.879+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebced72c8f756b82a99743e') }
2020-06-04T12:08:14.880+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebced91c8f756b82a99743f') }
2020-06-04T12:08:14.881+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebceda6c8f756b82a997440') }
2020-06-04T12:08:14.883+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebcf231c8f756b82a99744a') }
2020-06-04T12:08:14.886+0530    restoring indexes for collection abc.result from metadata
```

3 items 1 item selected

Restore data

- **Drop option**
  - **If we have already restored or database is existing, then duplicate key error will occur as shown below.**


Command Prompt
C:\Program Files\MongoDB\Server\4.2\bin>mongorestore d:\dump
2020-06-04T12:23:26.782+0530 preparing collections to restore from
2020-06-04T12:23:26.789+0530 restoring to existing collection abc.school without dropping
2020-06-04T12:23:26.790+0530 restoring to existing collection test.inventory without dropping
2020-06-04T12:23:26.791+0530 reading metadata for abc.school from d:\dump\abc\school.metadata.json
2020-06-04T12:23:26.794+0530 restoring to existing collection abc.marks without dropping
2020-06-04T12:23:26.797+0530 restoring to existing collection abc.store without dropping
2020-06-04T12:23:26.800+0530 reading metadata for test.inventory from d:\dump\test\inventory.metadata.json
2020-06-04T12:23:26.804+0530 restoring abc.school from d:\dump\abc\school.bson
2020-06-04T12:23:26.809+0530 reading metadata for abc.marks from d:\dump\abc\marks.metadata.json
2020-06-04T12:23:26.833+0530 reading metadata for abc.store from d:\dump\abc\store.metadata.json
2020-06-04T12:23:26.834+0530 restoring abc.store from d:\dump\abc\store.bson
2020-06-04T12:23:26.834+0530 restoring test.inventory from d:\dump\test\inventory.bson
2020-06-04T12:23:26.838+0530 restoring abc.marks from d:\dump\abc\marks.bson
2020-06-04T12:23:26.854+0530 continuing through error: E11000 duplicate key error collection: abc.marks index: _id_ d
up key: { _id: ObjectId('5ec103443b6e4f8f5b4f1148') }
2020-06-04T12:23:26.855+0530 continuing through error: E11000 duplicate key error collection: abc.marks index: _id_ d
up key: { _id: ObjectId('5ec103583b6e4f8f5b4f1149') }
2020-06-04T12:23:26.856+0530 continuing through error: E11000 duplicate key error collection: abc.marks index: _id_ d
up key: { _id: ObjectId('5ec1036c3b6e4f8f5b4f114a') }
2020-06-04T12:23:26.859+0530 continuing through error: E11000 duplicate key error collection: abc.store index: _id_ d
up key: { _id: ObjectId('5ebd64f24dd36225ea3a4076') }
2020-06-04T12:23:26.861+0530 continuing through error: E11000 duplicate key error collection: abc.school index: _id_
dup key: { _id: ObjectId('5eb97ae983a89f15e9b7e13e') }
2020-06-04T12:23:26.864+0530 continuing through error: E11000 duplicate key error collection: test.inventory index: _
id_ dup key: { _id: ObjectId('5eb8e3340132bb4da6110626') }
2020-06-04T12:23:26.868+0530 continuing through error: E11000 duplicate key error collection: abc.marks index: _id_ d
up key: { _id: ObjectId('5ec103913b6e4f8f5b4f114b') }
2020-06-04T12:23:26.871+0530 continuing through error: E11000 duplicate key error collection: abc.store index: _id_ d
up key: { _id: ObjectId('5ebd65f64dd36225ea3a4078') }
2020-06-04T12:23:26.874+0530 continuing through error: E11000 duplicate key error collection: abc.school index: _id_
dup key: { _id: ObjectId('5eb97afd83a89f15e9b7e13f') }
2020-06-04T12:23:26.877+0530 continuing through error: E11000 duplicate key error collection: test.inventory index: _
id_ dup key: { _id: ObjectId('5eb8e3340132bb4da6110627') }
2020-06-04T12:23:26.880+0530 continuing through error: E11000 duplicate key error collection: abc.marks index: _id_ d
up key: { _id: ObjectId('5ec103a33b6e4f8f5b4f114c') }
2020-06-04T12:23:26.883+0530 continuing through error: E11000 duplicate key error collection: abc.store index: _id_ d
up key: { _id: ObjectId('5ebd66b54dd36225ea3a4079') }
2020-06-04T12:23:26.886+0530 continuing through error: E11000 duplicate key error collection: abc.school index: _id_
dup key: { _id: ObjectId('5eb97b0b83a89f15e9b7e140') }
2020-06-04T12:23:26.889+0530 continuing through error: E11000 duplicate key error collection: test.inventory index: _
id_ dup key: { _id: ObjectId('5eb8e3340132bb4da6110628') }

Restore data

- To overcome this issue, drop option is used. Drop is necessary if we are replacing an existing database.
- **Syntax**
  - **mongorestore --drop <path-of-dumped-database>**
- **Example:**
  - **mongorestore --drop d:\dump**

Command Prompt - mongorestore --drop d:\dump

```
C:\Program Files\MongoDB\Server\4.2\bin>mongorestore --drop  d:\dump
2020-06-04T12:26:38.227+0530    preparing collections to restore from
2020-06-04T12:26:38.365+0530    reading metadata for abc.school from d:\dump\abc\school.metadata.json
2020-06-04T12:26:38.575+0530    restoring abc.school from d:\dump\abc\school.bson
2020-06-04T12:26:38.765+0530    reading metadata for test.inventory from d:\dump\test\inventory.metadata.json
2020-06-04T12:26:38.809+0530    reading metadata for abc.marks from d:\dump\abc\marks.metadata.json
2020-06-04T12:26:38.894+0530    no indexes to restore
2020-06-04T12:26:38.895+0530    finished restoring abc.school (36 documents, 0 failures)
2020-06-04T12:26:39.044+0530    restoring test.inventory from d:\dump\test\inventory.bson
2020-06-04T12:26:39.049+0530    no indexes to restore
2020-06-04T12:26:39.049+0530    finished restoring test.inventory (10 documents, 0 failures)
2020-06-04T12:26:39.203+0530    restoring abc.marks from d:\dump\abc\marks.bson
2020-06-04T12:26:39.221+0530    no indexes to restore
2020-06-04T12:26:39.221+0530    finished restoring abc.marks (12 documents, 0 failures)
2020-06-04T12:26:39.353+0530    reading metadata for abc.stock from d:\dump\abc\stock.metadata.json
2020-06-04T12:26:39.587+0530    reading metadata for abc.college from d:\dump\abc\college.metadata.json
2020-06-04T12:26:39.623+0530    reading metadata for abc.store from d:\dump\abc\store.metadata.json
2020-06-04T12:26:39.672+0530    reading metadata for abc.result from d:\dump\abc\result.metadata.json
2020-06-04T12:26:39.747+0530    restoring abc.stock from d:\dump\abc\stock.bson
2020-06-04T12:26:39.766+0530    no indexes to restore
2020-06-04T12:26:39.766+0530    finished restoring abc.stock (2 documents, 0 failures)
2020-06-04T12:26:40.069+0530    restoring abc.college from d:\dump\abc\college.bson
2020-06-04T12:26:40.077+0530    no indexes to restore
2020-06-04T12:26:40.078+0530    finished restoring abc.college (2 documents, 0 failures)
2020-06-04T12:26:40.147+0530    restoring abc.store from d:\dump\abc\store.bson
2020-06-04T12:26:40.157+0530    restoring indexes for collection abc.store from metadata
2020-06-04T12:26:40.227+0530    restoring abc.result from d:\dump\abc\result.bson
2020-06-04T12:26:40.236+0530    restoring indexes for collection abc.result from metadata
2020-06-04T12:26:40.252+0530    reading metadata for abc.abc from d:\dump\abc\abc.metadata.json
2020-06-04T12:26:40.568+0530    reading metadata for abc.result_9th from d:\dump\abc\result_9th.metadata.json
2020-06-04T12:26:40.791+0530    restoring abc.abc from d:\dump\abc\abc.bson
2020-06-04T12:26:40.795+0530    no indexes to restore
2020-06-04T12:26:40.795+0530    finished restoring abc.abc (2 documents, 0 failures)
2020-06-04T12:26:41.112+0530    restoring abc.result_9th from d:\dump\abc\result_9th.bson
2020-06-04T12:26:41.117+0530    no indexes to restore
2020-06-04T12:26:41.117+0530    finished restoring abc.result_9th (2 documents, 0 failures)
2020-06-04T12:26:41.302+0530    reading metadata for abc.result_totals from d:\dump\abc\result_totals.metadata.json
2020-06-04T12:26:41.489+0530    reading metadata for abc.result_10th from d:\dump\abc\result_10th.metadata.json
```

Restoring Specific Database from a .bson file

- **Mongorestore –db=<name-your-database-want-to-restore-as> <Name-and-path-of-dumped-database-bson-file/>**
  - The command is used to store any specific database from the dump bson file. Drop is optional.

Example

mongorestore --drop --db=abc d:\dump\abc\marks.bson

In new version,

mongorestore –drop –nsInclude=abc.student

Command Prompt

```
C:\Program Files\MongoDB\Server\4.2\bin>mongorestore --drop --db=abc  d:\dump\abc\marks.bson
2020-06-04T12:39:31.143+0530    checking for collection data in d:\dump\abc\marks.bson
2020-06-04T12:39:31.204+0530    reading metadata for abc.marks from d:\dump\abc\marks.metadata.json
2020-06-04T12:39:31.341+0530    restoring abc.marks from d:\dump\abc\marks.bson
2020-06-04T12:39:31.655+0530    no indexes to restore
2020-06-04T12:39:31.655+0530    finished restoring abc.marks (12 documents, 0 failures)
2020-06-04T12:39:31.656+0530    12 document(s) restored successfully. 0 document(s) failed to restore.

C:\Program Files\MongoDB\Server\4.2\bin>
```

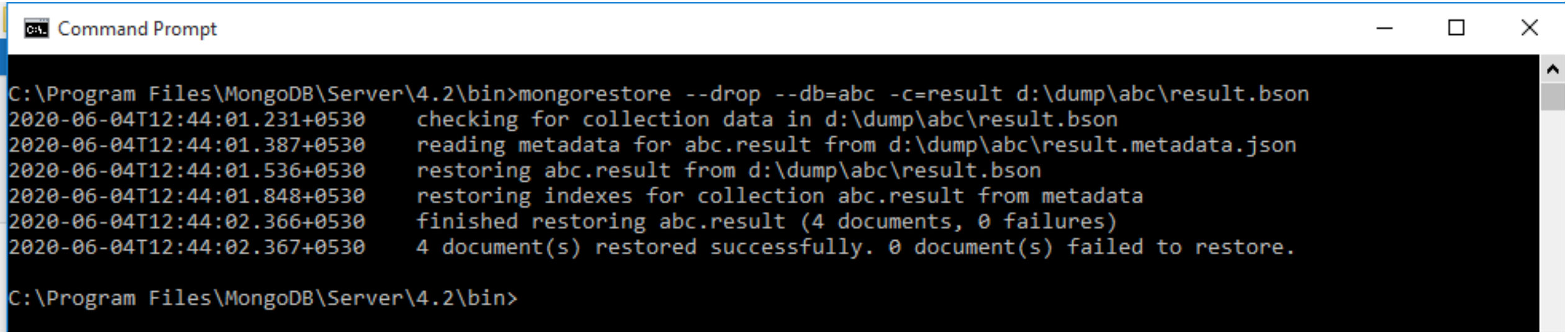
Restoring Specific collection Database from a .bson file

- **Mongorestore --db=<name-your-database-want-to-restore-as> --c collection_name <Name-and-path-of-dumped-database-bson-file/>**
  - The command is used to store any specific collection from database from the dump bson file. Drop is optional.

Example

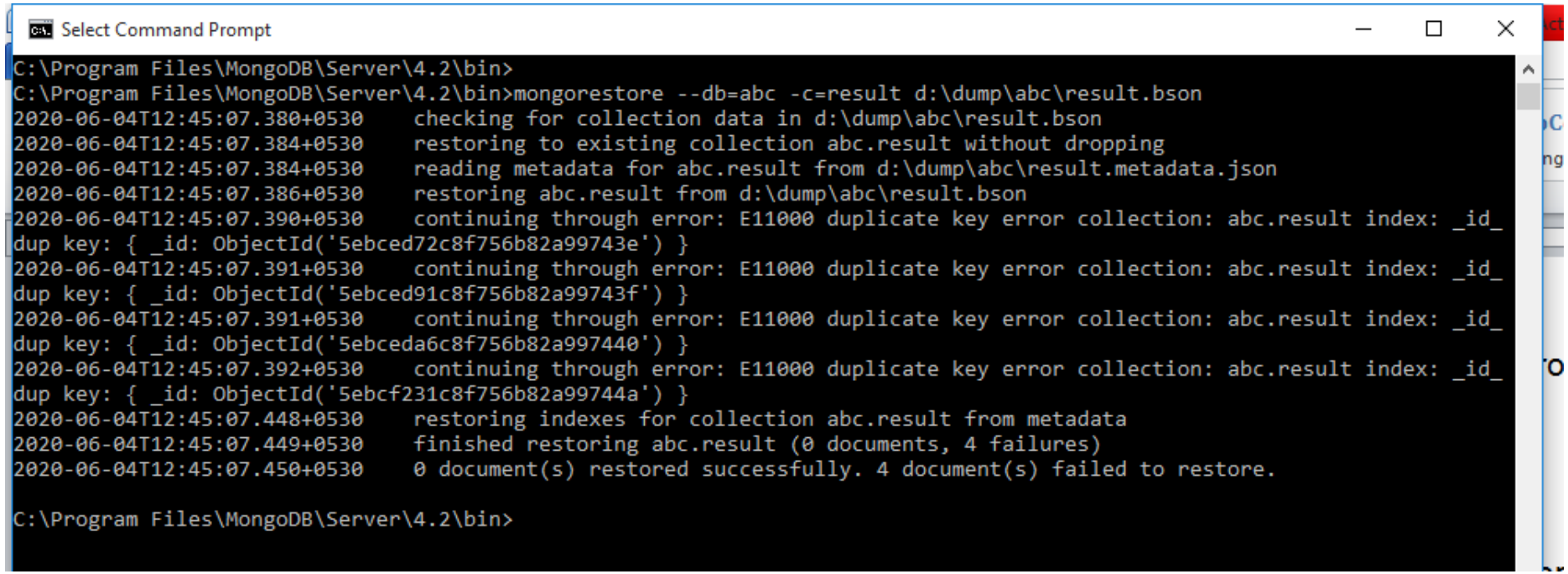
- mongorestore --drop --db=abc -c=result d:\dump\abc\result.bson

Restoring Specific collection Database from a .bson file



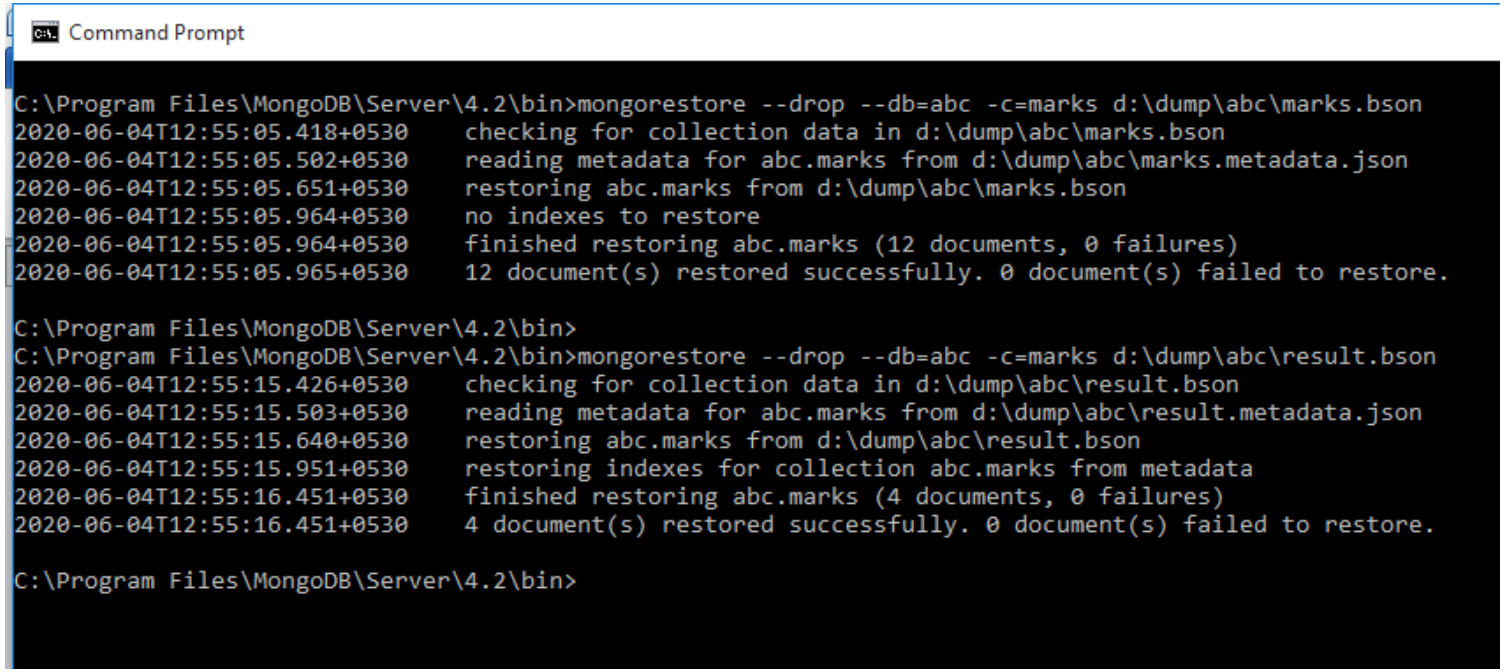
Restoring Specific collection Database from a .bson file

- **And if – drop option is not specified, it will show duplicate error.**



Restoring Specific collection Database from a .bson file

- **And specifying the collection, we may restore data from any other collection to the specified collection**



Export and Import in MongoDB

- We may export the full collection or specific field of collection as a CSV or JSON file. And vice versa may import a CSV or JSON file into a collection.
- **mongoexport command to Export Data from a Collection**
  - **mongoexport** command is used to export MongoDB Collection data to a CSV or JSON file. By default, the mongoexport command connects to mongod instance running on the localhost port number 27017.
- **Syntax**
  - **mongoexport --db DB_NAME --collection COLLECTION_name --fields Field_name(s) --type=[csv or JSON} --out=Name-Path-Output-File**

Export and Import in MongoDB

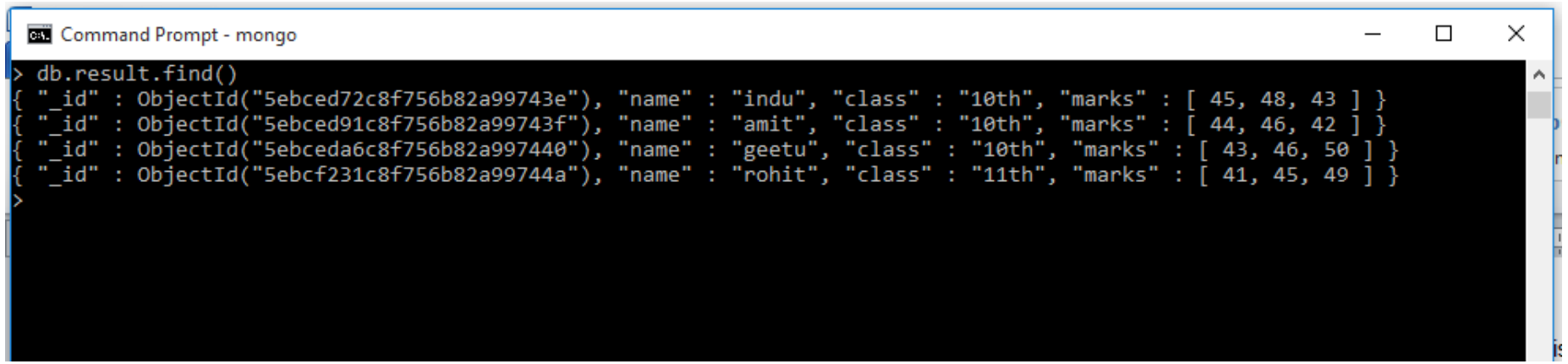
- **where,**
  - **DB_NAME** – Name of the Database of the Collection to be exported
  - **COLLECTION_name** - Name of Collection of DB_NAME to be exported
  - **Field_name(s)** - Name of the Field (or multiple fields separated by comma (,)) to be exported.If not specified, all the fields of the collection will be exported to JSON file.

Type – CSV or JSON format

- **Name-Path-Output-File** – Name and path of the output (exported) file. It is also optional. But it is suggested to properly specify the name and path of the exported file.
- When exporting to CSV format, must specify the fields in the documents to be exported otherwise it will display error.
- When exporting to JSON format, _id field will also be exported by default. While the same will not be exported if exporting as CSV file until not specified in field list

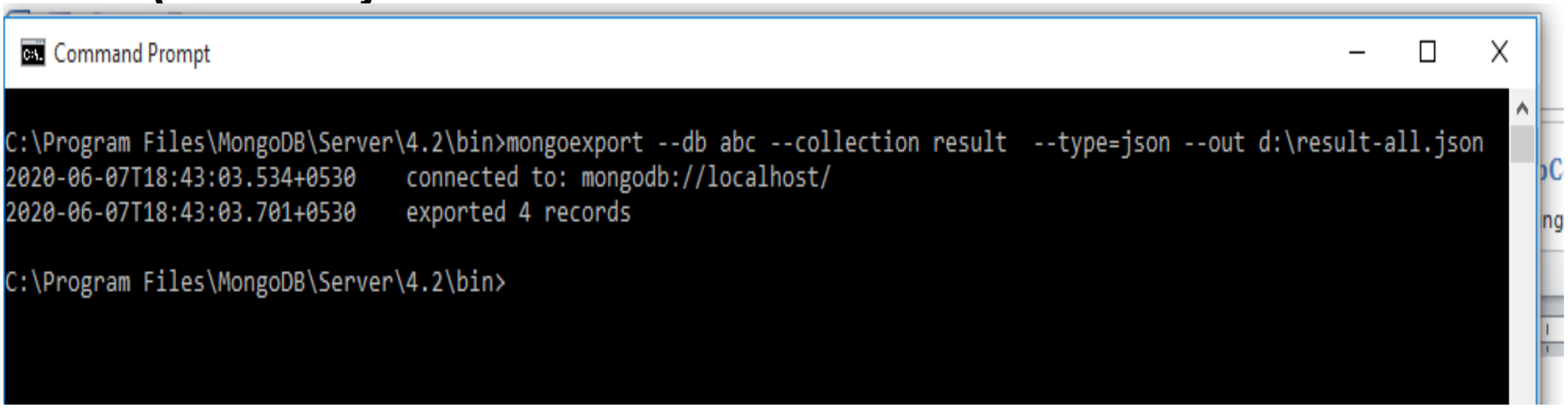
Example

- We have a result collection in abc database having multiple fields as displayed here:



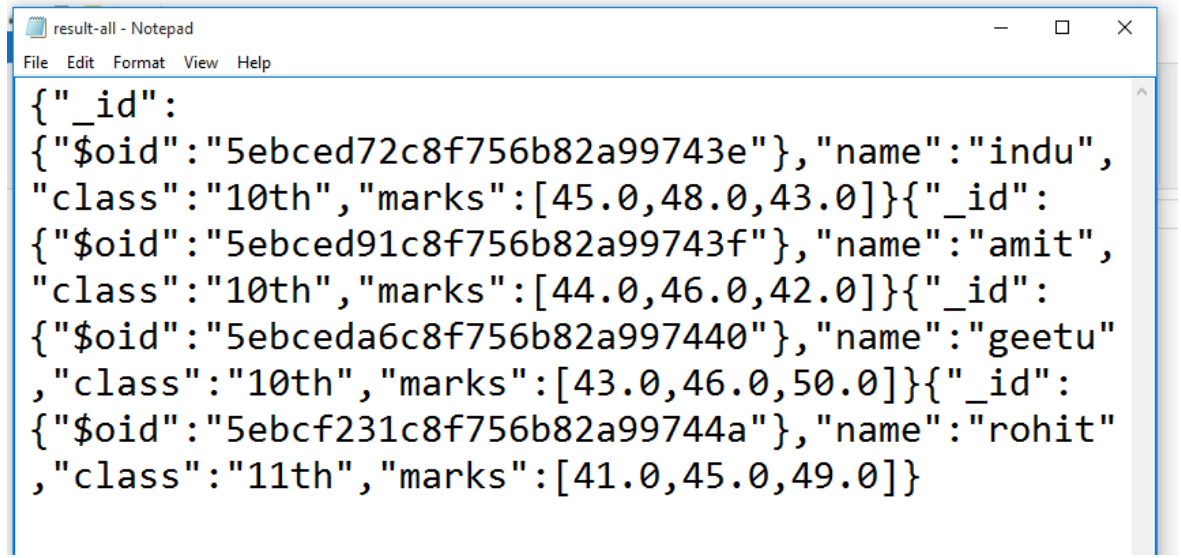
Example

- Lets export the result collection as JSON file result-all.json, means all the fields
  - **mongoexport --db abc --collection result --type=json --out d:\result-all.json**



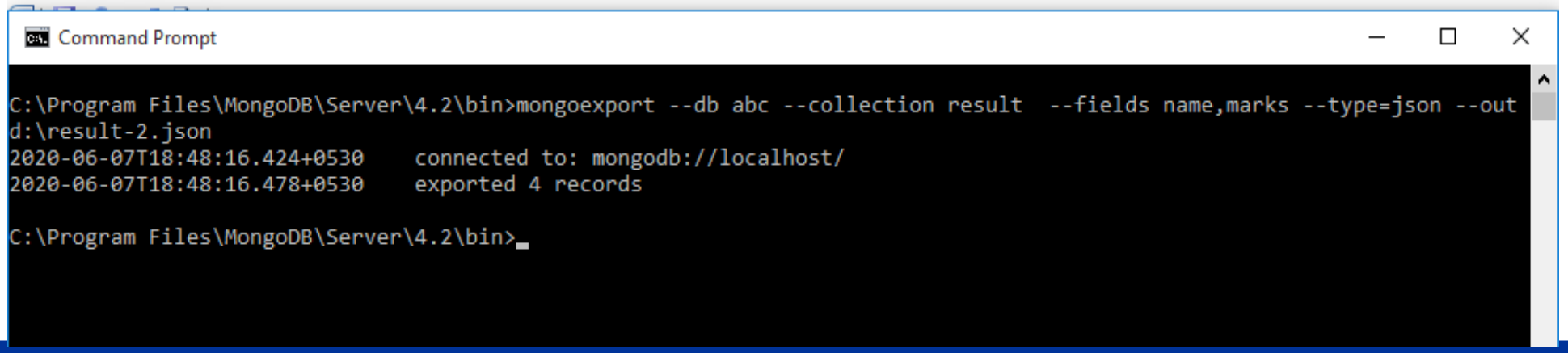
Example

- Shows that 4 records have been exported. And if we open, the created json file result-all.json, it will show like



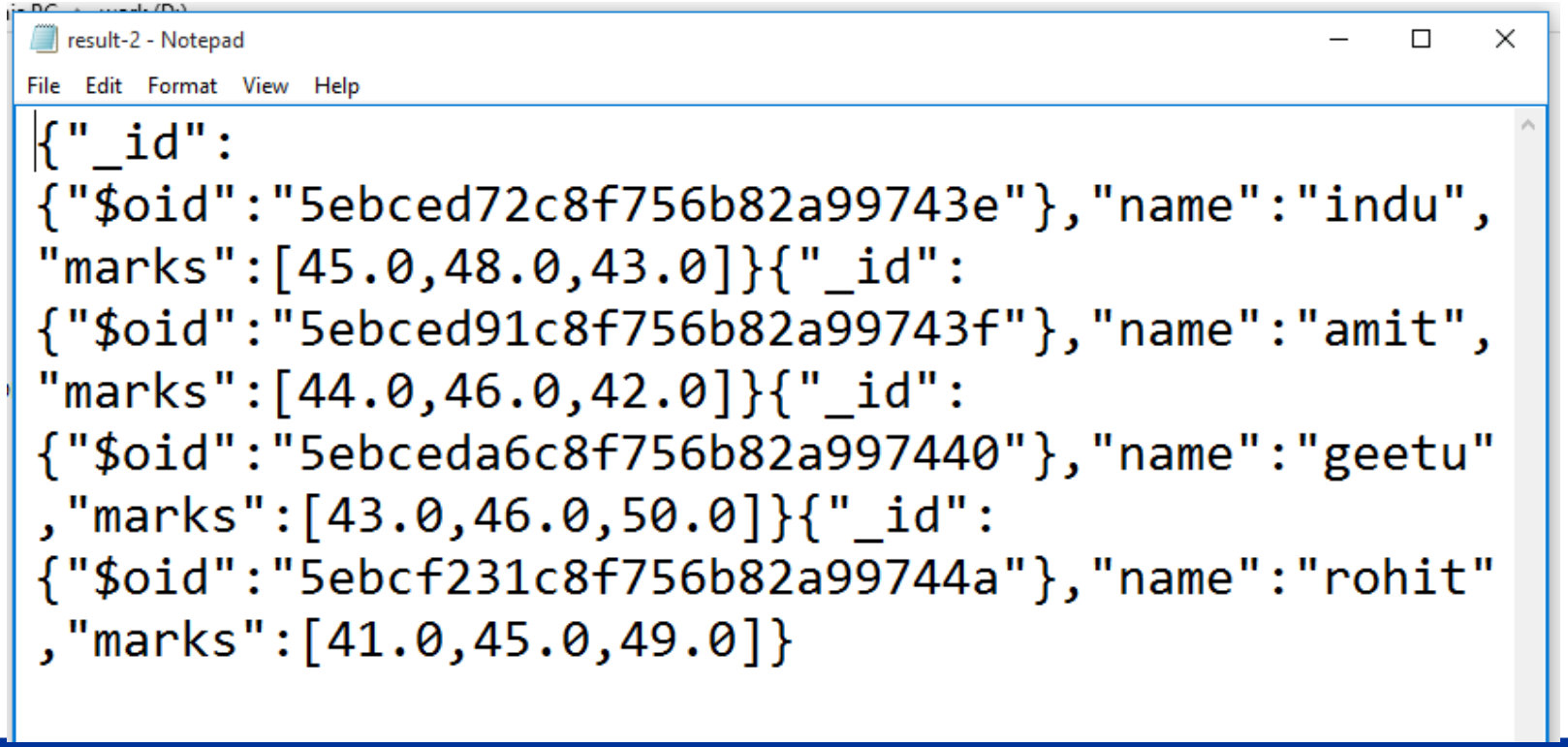
Example

- Lets export the only two fields, name and marks of result collection as JSON file result-2.json:
  - **mongoexport --db abc --collection result --fields name,marks --type=json --out d:\result-2.json**



Example

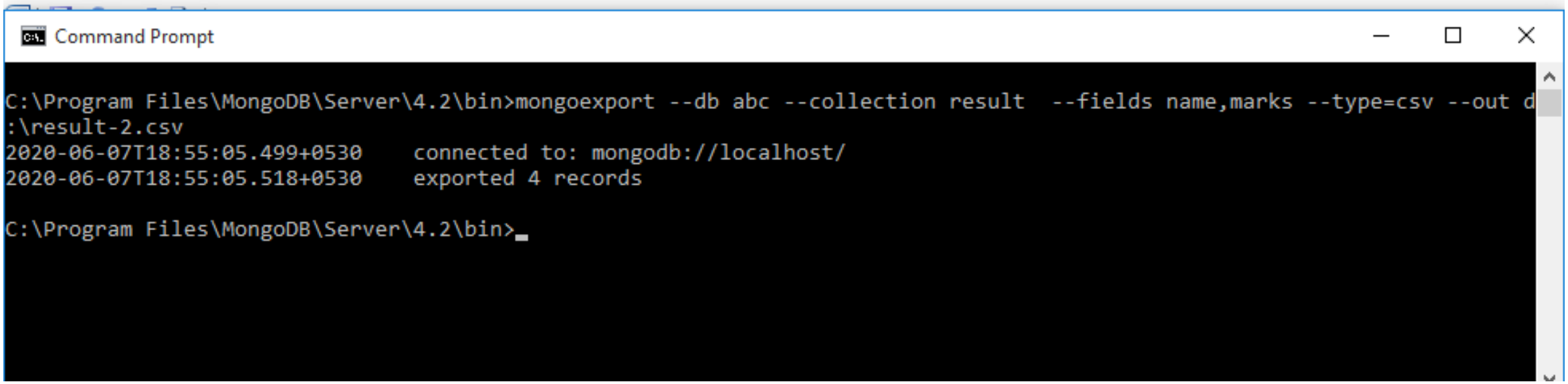
- Shows that 4 records have been exported but here only 2 fields name and marks, will be exported which may be seen in result-2.json



Example

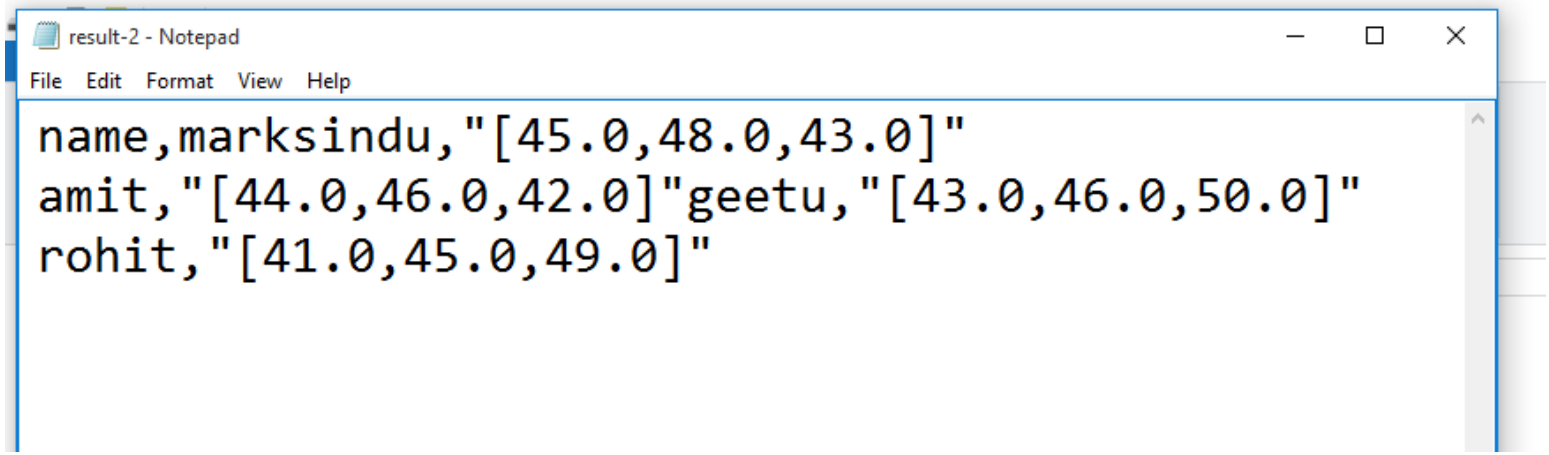
- **_id field will be exported by default irrespective of the number of fields of collection specified for export.**
- Lets export the two fields, name and marks of result collection as CSV file result-2.csv :
  - **mongoexport --db abc --collection result --fields name,marks --type=csv --out d:\result-2.csv**

Example


Command Prompt
C:\Program Files\MongoDB\Server\4.2\bin>mongoexport --db abc --collection result --fields name,marks --type=csv --out d
:\result-2.csv
2020-06-07T18:55:05.499+0530 connected to: mongodb://localhost/
2020-06-07T18:55:05.518+0530 exported 4 records
C:\Program Files\MongoDB\Server\4.2\bin>

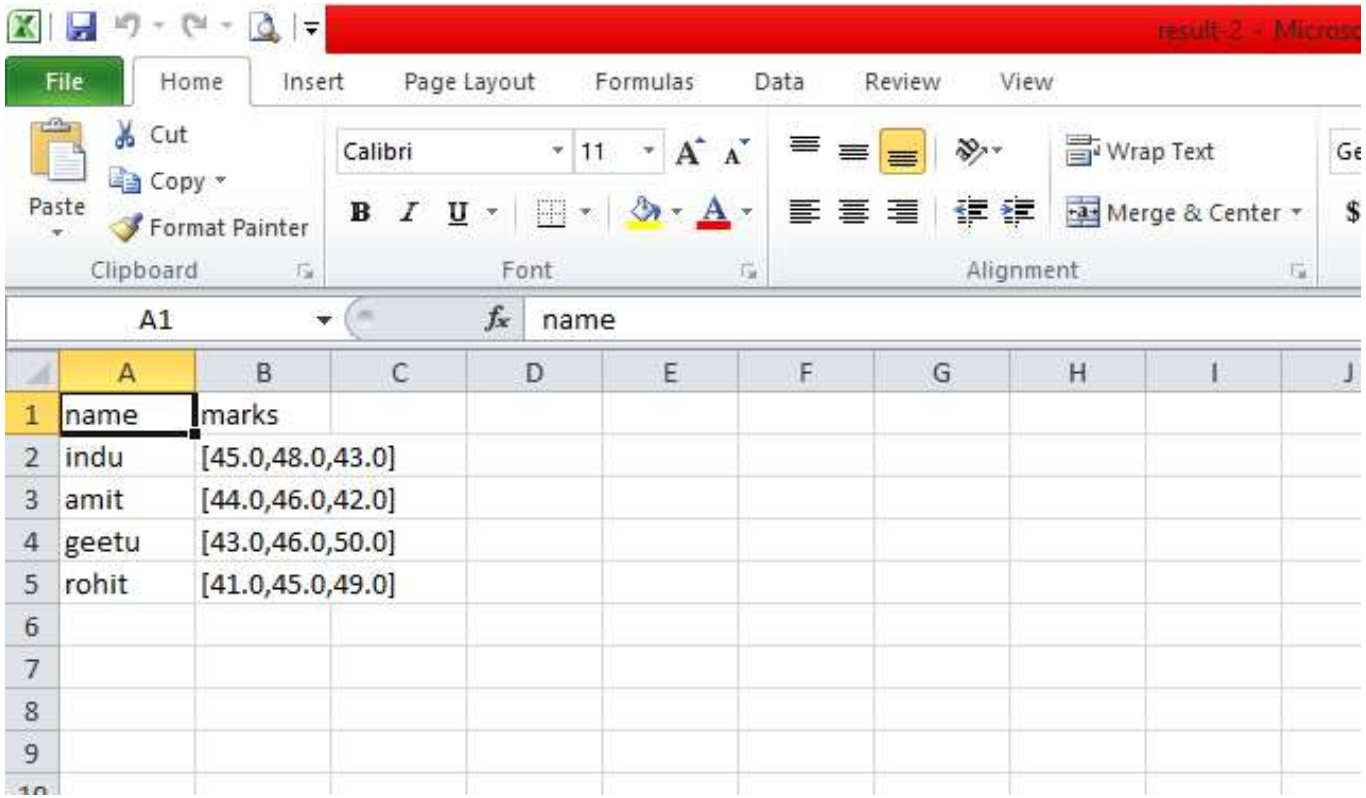
Example

- Shows that 4 records have been exported which may be seen in result-2.csv:
- If opened in notepad



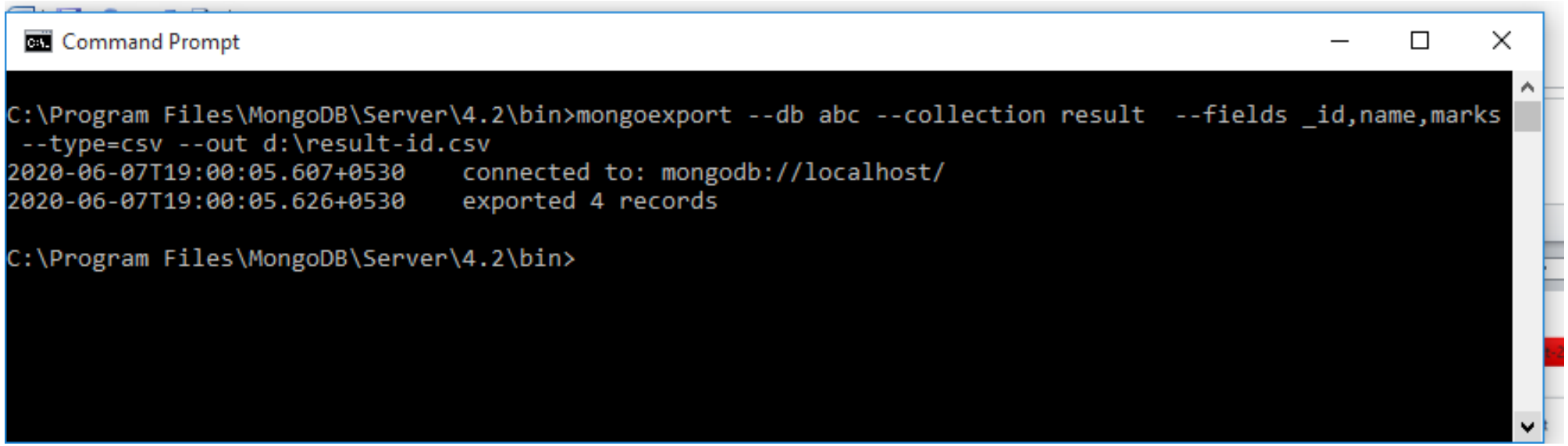
Example

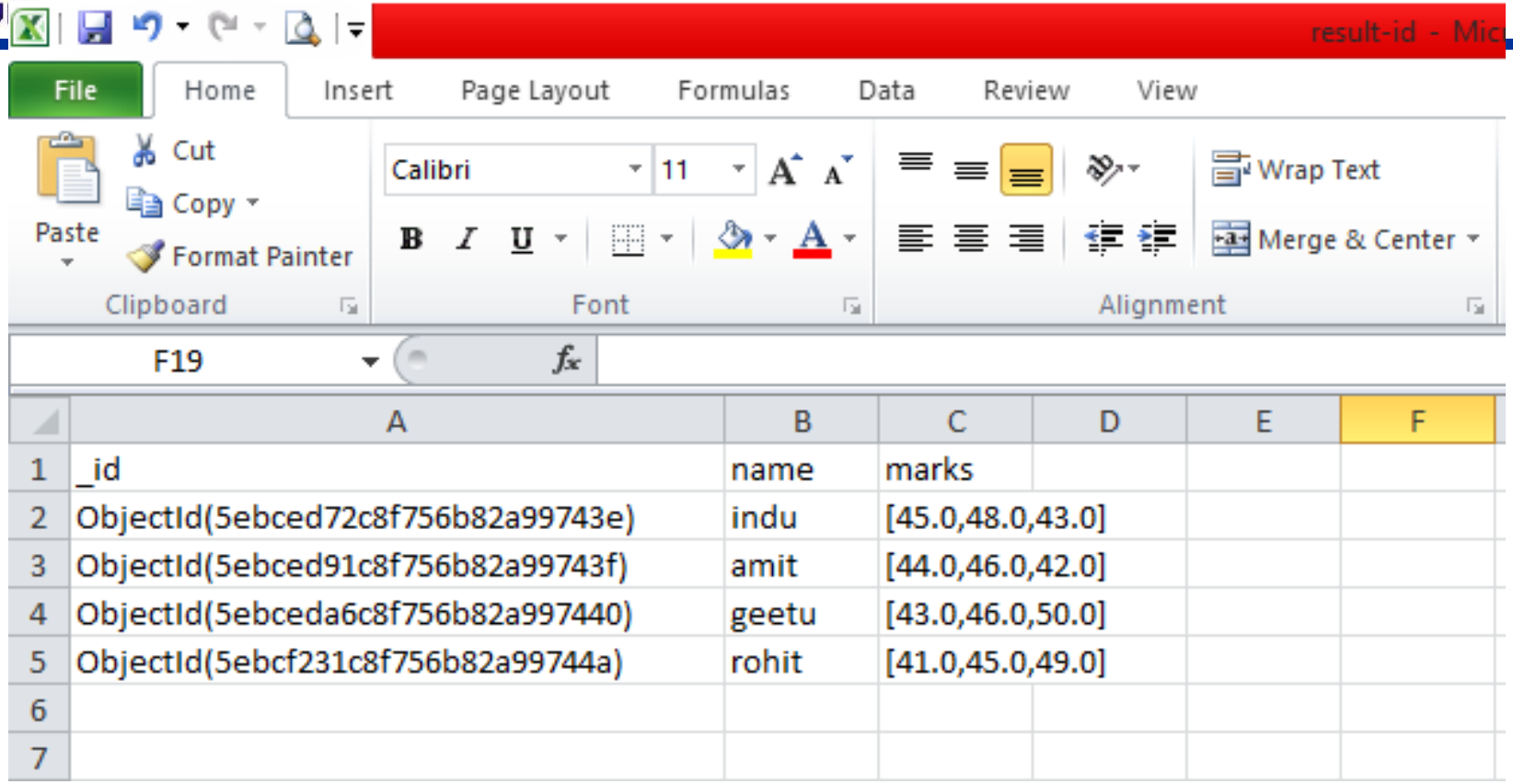
- If opened in excel



Example

- **Note : _id field will not be exported by default until not specified.**





	A	B	C	D	E	F
1	_id	name	marks			
2	ObjectId(5ebced72c8f756b82a99743e)	indu	[45.0,48.0,43.0]			
3	ObjectId(5ebced91c8f756b82a99743f)	amit	[44.0,46.0,42.0]			
4	ObjectId(5ebceda6c8f756b82a997440)	geetu	[43.0,46.0,50.0]			
5	ObjectId(5ebcf231c8f756b82a99744a)	rohit	[41.0,45.0,49.0]			
6						
7						

Contents to be covered

- Importing from JSON file
- Replication,
- Advantages of replication

mongoimport command to Import Collection

- **mongoimport** command is used to restore (import) a database from a backup(export) taken with mongoexport command.
- **Importing from JSON File**
  - To Import a collection from a JSON file, the syntax is as under
- **Syntax**

  mongoimport --db DB_NAME --collection COLLECTION_name --type=json --file=Name-of-file-to-import

where,

- **DB_NAME** – Name of the Database of the Collection to be imported

mongoimport command to Import Collection

- **COLLECTION_name** - Name of Collection of DB_NAME to be imported
- **Type** –JSON, it is optional. System by default take it as a JSON
- **Name-of-file-to-import** – Name and path of the JSON file to be imported (restore).
- **Note:** We may specify a new collection name as well as a new database name while importing CSV or JSON file to the collection file.

mongoimport command to Import Collection

- **Example :**
  - Let's assume that our backup file name is result-all.JSON which is in D:\ and the db name is abc and collection name is result.

mongoimport --db abc --collection result --type=json --file d:\result-all.json

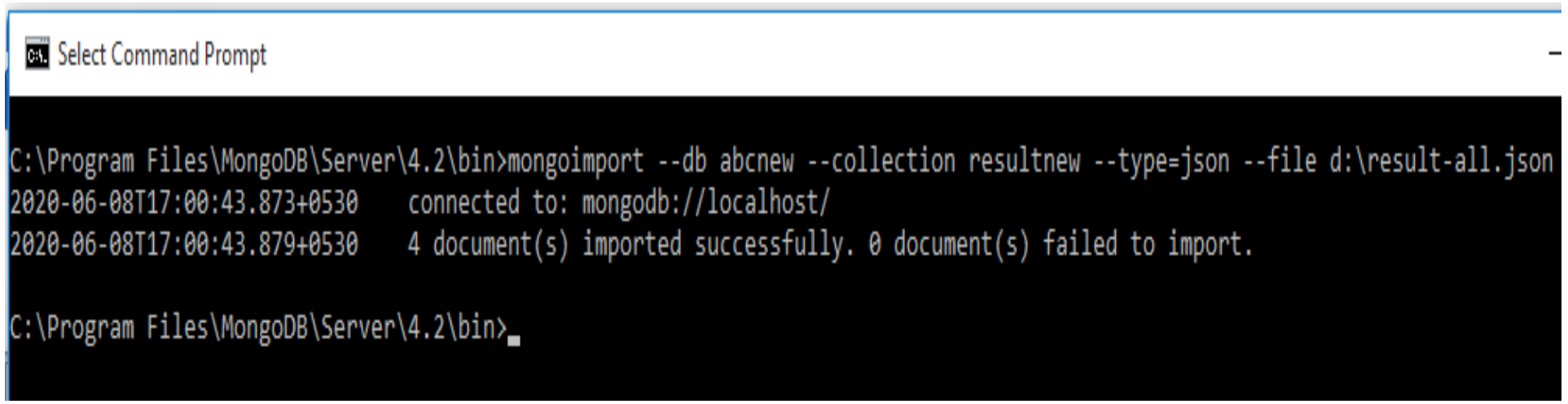
mongoimport command

```
C:\Program Files\MongoDB\Server\4.2\bin>mongoimport --db abc --collection result --type=json --file d:\result-all.json
2020-06-08T16:52:22.297+0530    connected to: mongodb://localhost/
2020-06-08T16:52:22.572+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebced72c8f756b82a99743e') }
2020-06-08T16:52:22.573+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebceda6c8f756b82a997440') }
2020-06-08T16:52:22.573+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebced91c8f756b82a99743f') }
2020-06-08T16:52:22.576+0530    continuing through error: E11000 duplicate key error collection: abc.result index: _id_
dup key: { _id: ObjectId('5ebcf231c8f756b82a99744a') }
2020-06-08T16:52:22.578+0530    0 document(s) imported successfully. 4 document(s) failed to import.
```

mongoimport command

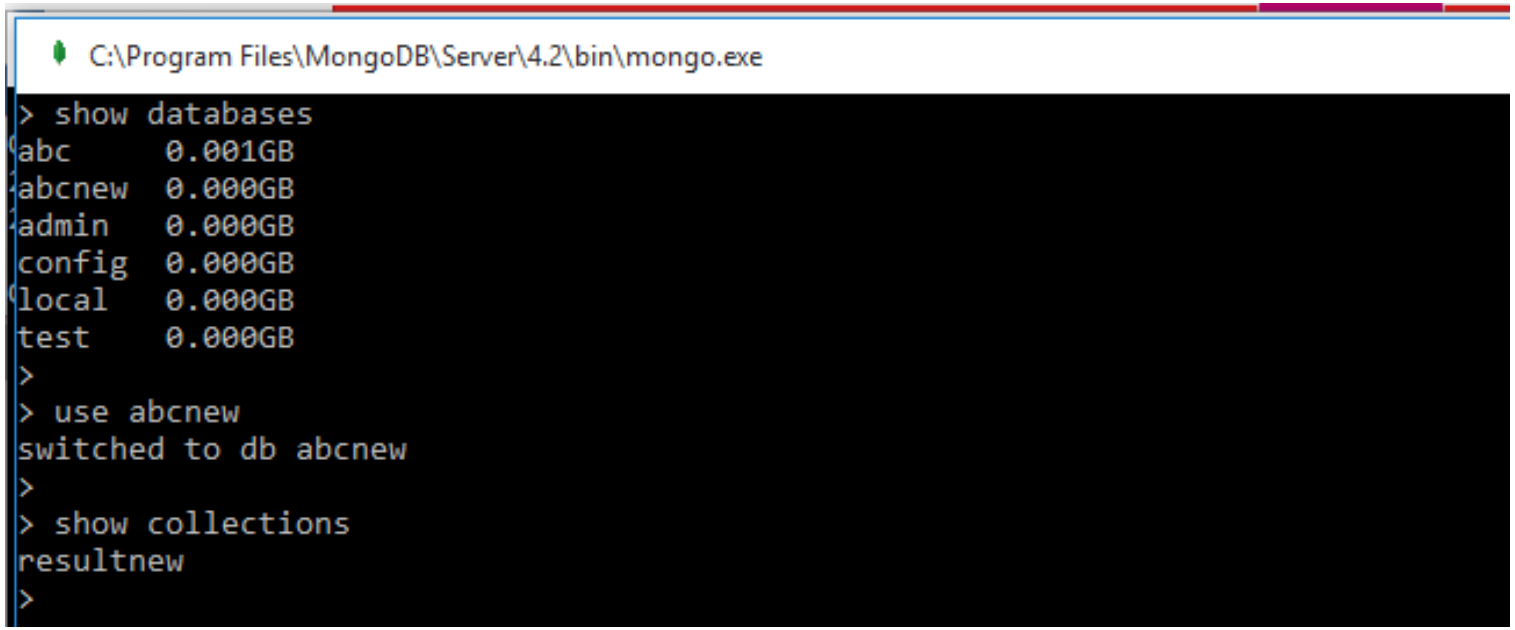
- In the above example, mongoimport imports the data in the JSON data from the result-all.json file into the collection result in the abc database. The result is showing duplicate key error because the same records are existing in the collection specified.
- We can give a new name to the collection as well as to database to which a JSON data file is imported. In the code below, the same JSON file of above example is used but imported to abcnew database as resultnew collection.
- **mongoimport --db abcnew --collection resultnew --type=json --file d:\result-all.json**

mongoimport command



mongoimport command

- Now list the databases in MongoDB and collection in the abcnew database



Importing from CSV File

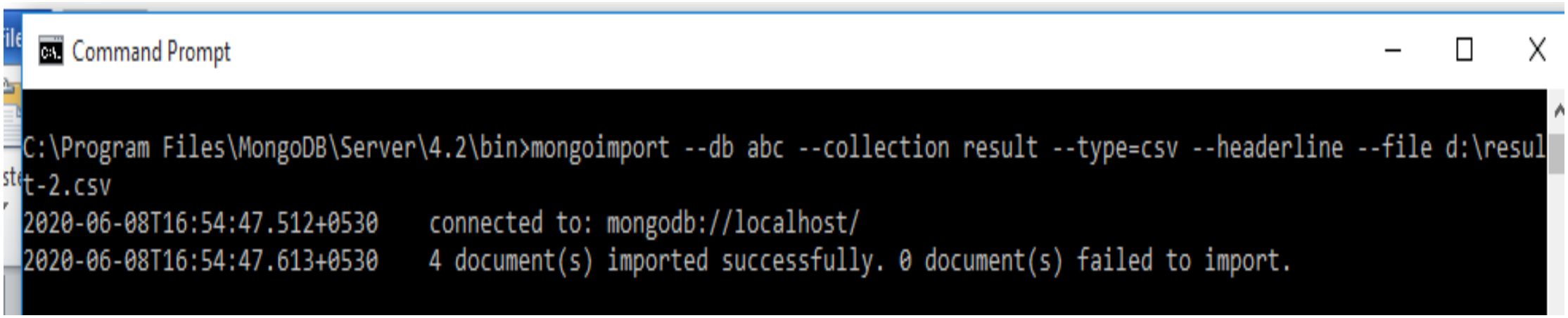
- mongoimport may also be used to import the csv formatted data into the collection. **–headerline option** is used to instructs mongoimport to determine the name of the fields using the first line in the CSV file. **If the CSV file don't have _id field, then while importing MongoDB automatically insert _id field to all the documents**. To Import a collection from a CSV file, the syntax is as under
- **Syntax**
  - **mongoimport --db DB_NAME --collection COLLECTION_name --type=csv --headerline --file=Name-of-file-to-import**
- **where,**

Importing from CSV File

- **DB_NAME** – Name of the Database of the Collection to be exported
- **COLLECTION_name** - Name of Collection of DB_NAME to be exported
- **Type** –csv
- **headerline** - to take 1st record of CSV file as field names.
- **Name-of-file-to-import** – Name and path of the JSON file to be imported (restore).

Example:

- If our backup file name is result-2.csv which is in D:\ and the db name is abc and collection name is result.
- **mongoimport --db abc --collection result --type=csv --headerline --file d:\result-2.csv**

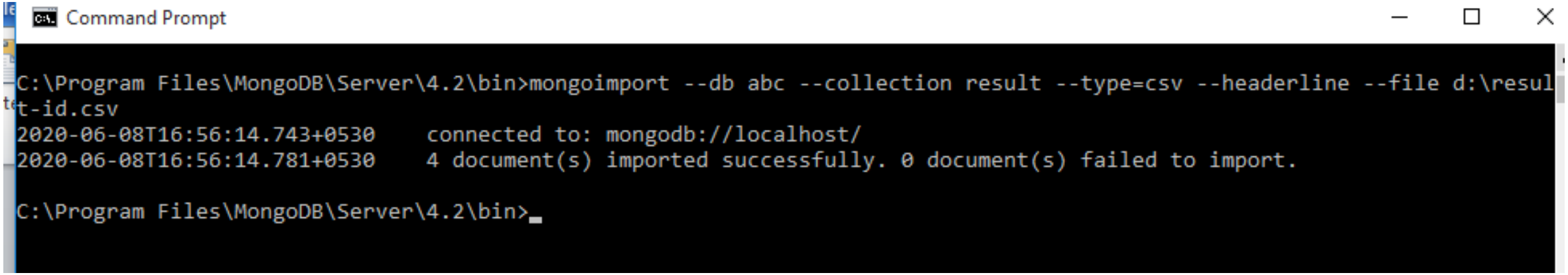


Importing from CSV File

- The result shows the records imported to the collection. As the out CSV file don't have any _id field but while importing MongoDB automatically insert _id field to all the documents. We may see that

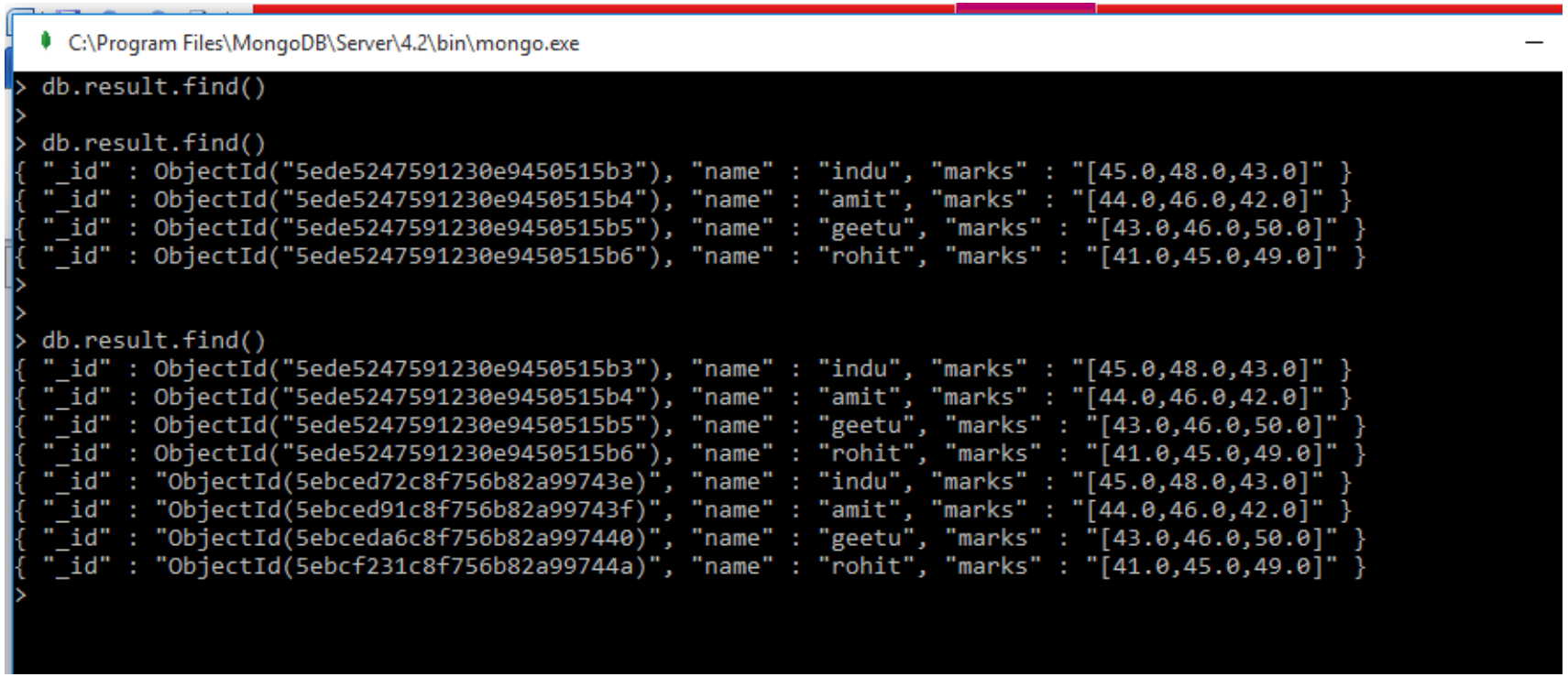
Importing from CSV File

- Now we are again importing a backup file result-id.csv having _id filed from D:\ and the db name is abc and collection name is result.
- **mongoimport --db abc --collection result --type=csv --headerline --file d:\result-id.csv**



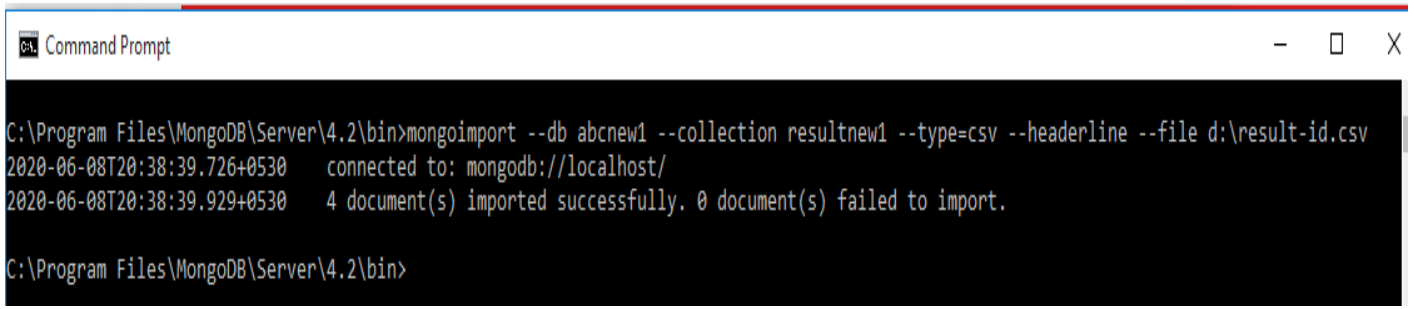
Importing from CSV File

- Now if we display the collection result, it will have the records as



Importing from CSV File

- Shows total 8 records, 4 imported in example-3 (system assigned _id) and 4 imported in example-4.
- Now, we have taken database name as abcnew1 and collection name as resultnew1. In this case the collection will be imported as a new collection name and in a different database.
- **mongoimport --db abcnew1 --collection resultnew1 --type=csv --headerline --file d:\result-id.csv**



Importing from CSV File

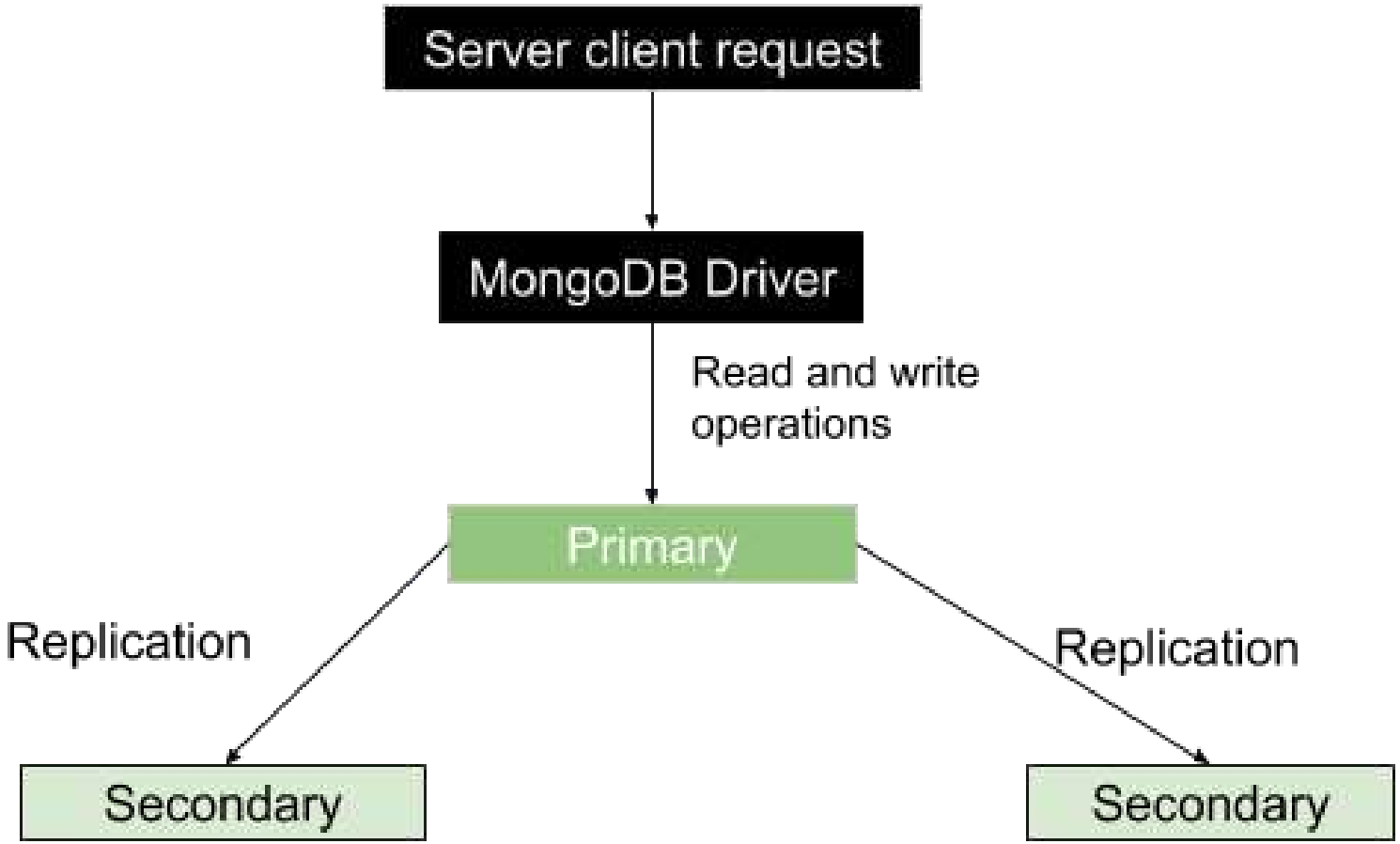
- And if we see the new database and collection, it will be like

Replication

- Replication is the process of synchronizing data or simply duplication (replication) of data across multiple servers. Replication increases data availability with multiple copies of data on different database servers to meet any unforeseen circumstances.
- Replication protects a database from the loss of a single server. Replication also allows user to recover from hardware failure and service interruptions.
- With additional copies of the data, we may dedicate servers to disaster recovery, reporting, or backup.

Replication

- Replication is simply the process of ensuring that the same data is available on more than one Mongo DB Server.
- This is sometimes required for the purpose of increasing data availability and to meet any disaster also.
- If there is only one MongoDB Server, and it goes down for any reason, there will be no access to the data. But if data has been replicated to another server or to multiple servers, the data from another server(s) will be accessible even if the primary server fails


Server client request
MongoDB Driver
Read and write operations
Primary
Replication
Replication
Secondary
Secondary

Replication

- A very important aspect of replication is the possibility of load balancing. If there are many users connecting to the system, instead of having everyone connect to one system, users can be connected to multiple servers so that there is an equal distribution of the load.
- In MongoDB, multiple MongDB Servers are grouped in sets called **Replica sets**. The Replica set will have a primary server which will accept all the write operation from clients. All other instances added to the set after this will be called the secondary instances which can be used primarily for all read operations

Advantages of Replication

- The data is safe
- There is availability of data for 24 by 7 (24 x 7)
- There is help in disaster recovery and backup of data.
- There is load balancing
- There is no downtime for maintenance (like backups, index rebuilds, compaction)
- There is read scaling (extra copies to read from)
- Replica set is transparent to the application
- Minimizes downtime for maintenance

Disadvantages of Data Replication

- There is high Cost for Hardware and other infrastructure
- There is redundant data is stored, so more space and server processing required
- It needs connectivity with all the replicated servers

Contents to be covered

- Implementation of replication
- Managing Configuration File in MongoDB
- Setting up replica set in MongoDB,
- Replication configuration ,
- Checking the Configuration
- Adding members

Implementation of Replication in MongoDB

- MongoDB implement replication by the use of replica set. A replica set is a group of **Mongod** instances that host the same data set.
- In a replica, one node is primary node that receives all write operations and all other instances called secondary instances, apply operations from the primary so that they have the same data set.
- Replica set can have only one primary node and rest all are the secondary nodes. In short,
  - Replica set is a group of two or more nodes. A minimum of 3 nodes are recommended.
  - In a replica set, one node is primary node and remaining nodes are secondary.

Implementation of Replication in MongoDB

- All the data replicates from primary to secondary nodes.
- There are new primary nodes which get elected in case there is automatic maintenance or failover
- Once the failed node is recovered, it again joins the replica set and works as a secondary node.

Features of Replica Set

- There is a cluster of N nodes
- Any one node can be primary
- All the write operations go to primary node
- There is automatic recovery
- There is consensus election of primary

Managing Configuration File in MongoDB

- Before we start creating the Replica Set in MongoDB, it is desired to create separate configuration file for each replica set i.e. for each new port. These replicas will be treated as separate instance of MongoDB Server.
- These server instances can be on the different machines for regular use and it is suggested that the replication servers must be kept apart also for safety and to meet disaster recovery. However, for demo purposes we may create multiple instances of the MongoDB server on the same system also.
- For each case, we have to at least specify the ReplicaSet Name, the server Port number and Bind IP (as applicable), database and logfile in each of the configuration file.

Managing Configuration File in MongoDB

- The configuration file of MongoDB resides in the Bin folder of MongoDB server. If we have installed the MongoDB on a Windows Machine, the configuration file “mongod.cfg” will be available in c:\program files\mongodb\server\4.2\bin directory. This file can be edited in Notepad++ easily.
- Suppose, we want to create Three (3) Replica’s, then we have to first create directories for database and log files for each of the Secondary(replicated) servers, in this case three.
- Create directories
  - c:\data1, c:\data2, c:\data3 for database and
  - c:\data1\log, c:\data2\log,c:\data3\log for log files has been created.
  - C:\data1\config , c:\data2\config c:\data3\config

Managing Configuration File in MongoDB

- Create mongo.cfg files in folders data1,data2,data3 config folder
- Edit Configuration files and specify dbpath logpath and port number
- Lets edit the configuration files: (here mongo.cfg)

```
dbPath C:\data1
logpath  C:\data1\log\mongod.log
port  27020
```

Managing Configuration File in MongoDB

- We have to edit all the configuration files in data1,data2 and data3
- For data2

dbPath C:\data2

logpath C:\data2\log\mongod.log

port 27021

Foe data3

dbPath C:\data3

logpath C:\data3\log\mongod.log

port 27022

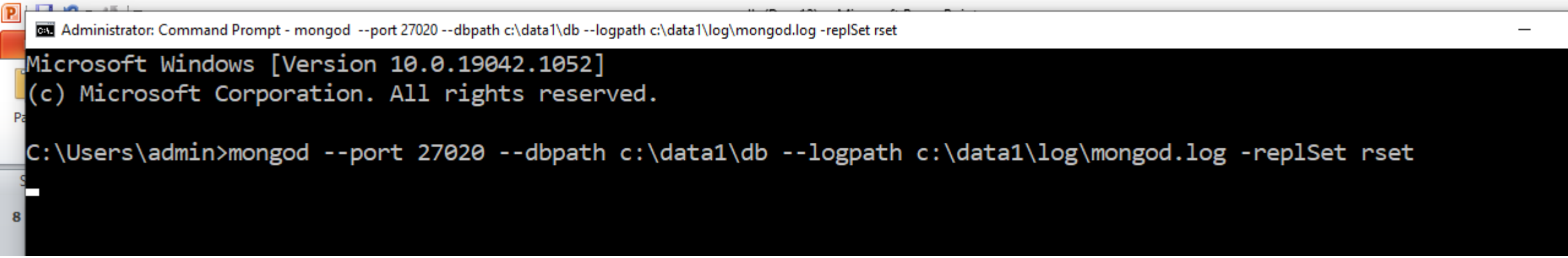
Setting up a Replica Set in MongoDB

- We may convert standalone MongoDB instance to a replica set. Following are the steps to convert to replica set,
  - Start the MongoDB server by specifying -- replSet option.
  - **Syntax**
    - **mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH”-- logpath “logpath”** --replSet **REPLICA_SET**

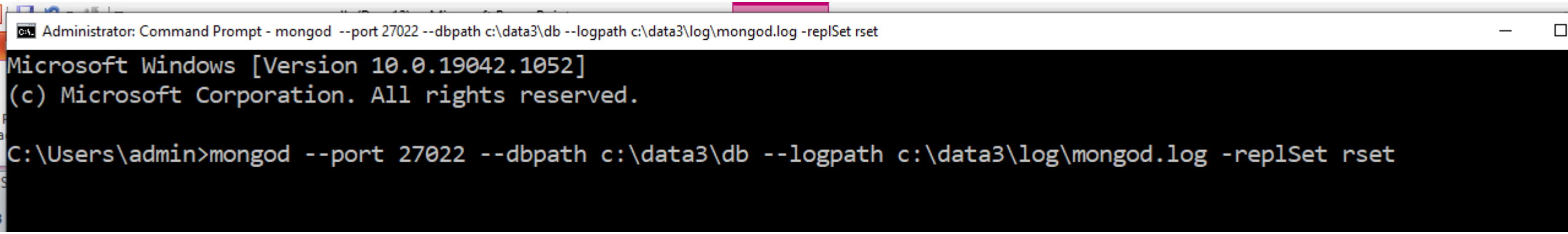
      Where,
      - Port – is the Port Number
      - YOUR_DB_DATA_PATH – Is the complete path of the DatabaseName
      - REPLICA_SET – is the Name of the Replica Set instance
      - YOUR_log_File_PATH – is the file name and path of Log file.

Setting up a Replica Set in MongoDB

- **Example**
- mongod --port 27020 --dbpath “c:\data1" --logpath “c:\data1\log\mongod.log" --replSet rset
- mongod --port 27021 --dbpath “c:\data2" --logpath “c:\data2\log\mongod.log" --replSet rset
- mongod --port 27022 --dbpath “c:\data3" --logpath “c:\data3\log\mongod.log" --replSet rset


Administrator: Command Prompt - mongod --port 27020 --dbpath c:\data1\db --logpath c:\data1\log\mongod.log -replSet rset
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
C:\Users\admin>mongod --port 27020 --dbpath c:\data1\db --logpath c:\data1\log\mongod.log -replSet rset


Administrator: Command Prompt - mongod --port 27021 --dbpath c:\data2\db --logpath c:\data2\log\mongod.log -replSet rset
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
C:\Users\admin>mongod --port 27021 --dbpath c:\data2\db --logpath c:\data2\log\mongod.log -replSet rset


Administrator: Command Prompt - mongod --port 27022 --dbpath c:\data3\db --logpath c:\data3\log\mongod.log -replSet rset
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
C:\Users\admin>mongod --port 27022 --dbpath c:\data3\db --logpath c:\data3\log\mongod.log -replSet rset

Setting up a Replica Set in MongoDB

- This will convert MongoDB instance to replica set with name “rset” and on port 27020, 27021, 27022.
- For the replication all these three instances should be running
- Now start the command prompt and connect to this mongod instance, from the windows command prompt.

mongo --port 27020

– it opens a mongo shell

Initiate a replication

- **Setting replication :**Initiate a replication by running below configuration

  rconf={"_id":”rset",members:[{_id:0,host:"127.0.0.1:27020"}] }

```
> rconf={"_id":"rset",members:[{_id:0,host:"127.0.0.1:27020"}] }
{
        "_id" : "rset",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "127.0.0.1:27020"
                }
        ]
}
>
```

- Now, run the command,

  rs.initiate(rconf)

  – This will initiate a new replica set.

```
> rs.initate(rconf)
uncaught exception: TypeError: rs.initate is not a function :
@(shell):1:1
> rs.initiate(rconf)
{ "ok" : 1 }
rset:SECONDARY>
rset:PRIMARY>
```

- As we run this command, we can see that Prompt shows as **rset.SECONDARY>**

- You can see secondary connection
- Now enter ,it will turn to primary

Add Members to Replica Set

- **rs.add()** commands adds members to the replica set. You can add mongod instance to replica set only if are connected to primary node.
- **db.isMaster()** command is to check whether you are connected to primary or not.
- **Syntax**

  >rs.add(HOST_NAME:PORT)

Add Members to Replica Set

Example

- Suppose your mongod instance name (HOST Name) is localhost (or 127.0.0.1) and it is running on port **27021**. To add this instance to replica set, command will be

```
>rs.add("localhost:27021")
```

Or

```
>rs.add("127.0.0.1:27021")
```

```
}
rset:PRIMARY> rs.add("localhost:27021")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1624339462, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1624339462, 1)
}
rset:PRIMARY>
```

Add Members to Replica Set

- Lets add another member,
- >rs.add("127.0.0.1:27022")

Add Members to Replica Set

```
rset:PRIMARY> rs.add("localhost:27022")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1624339835, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1624339835, 1)
}
rset:PRIMARY>
```

Checking the Configuration

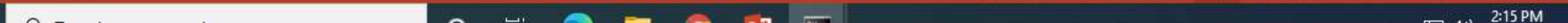
- To check the replica set configuration, the command is

  rs.conf()

  – rs.conf() method (**rs.config()** may also be used) without any parameter is used to check the present configuration of the application

```
et:SECONDARY>
et:PRIMARY>
et:PRIMARY> rs.config()

        "_id" : "rset",
        "version" : 7,
        "term" : 4,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "127.0.0.1:27020",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "secondaryDelaySecs" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 1,
```

```
        },
        {
                "_id" : 1,
                "host" : "127.0.0.1:27021",
                "arbiterOnly" : false,
                "buildIndexes" : true,
                "hidden" : false,
                "priority" : 1,
                "tags" : {

                },
                "secondaryDelaySecs" : NumberLong(0),
                "votes" : 1
        },
        {
                "_id" : 2,
                "host" : "127.0.0.1:27022",
                "arbiterOnly" : false,
                "buildIndexes" : true,
                "hidden" : false,
                "priority" : 1,
                "tags" : {

                },
```

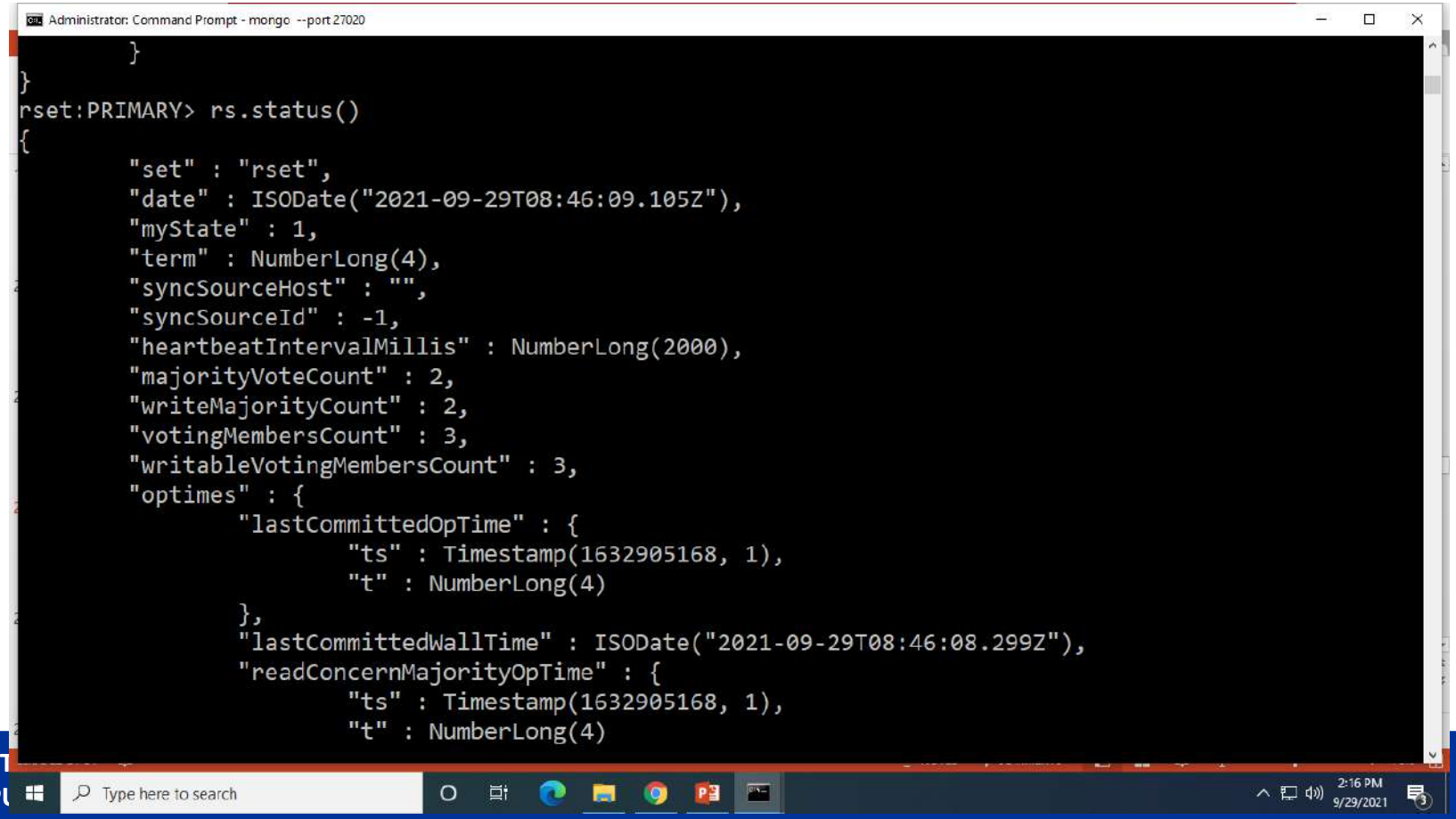


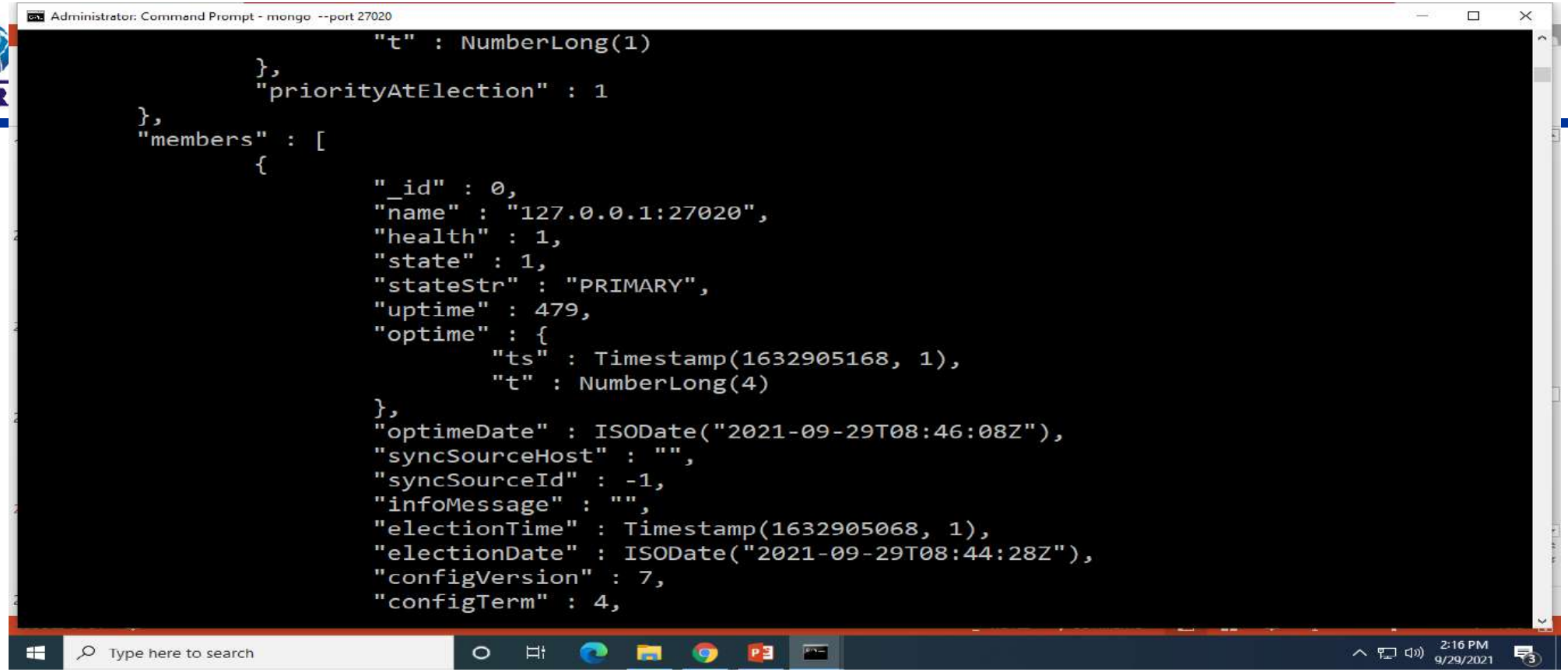
```
                "tags" : {

                },
                "secondaryDelaySecs" : NumberLong(0),
                "votes" : 1
        }
],
"protocolVersion" : NumberLong(1),
"writeConcernMajorityJournalDefault" : true,
"settings" : {
        "chainingAllowed" : true,
        "heartbeatIntervalMillis" : 2000,
        "heartbeatTimeoutSecs" : 10,
        "electionTimeoutMillis" : 10000,
        "catchUpTimeoutMillis" : -1,
        "catchUpTakeoverDelayMillis" : 30000,
        "getLastErrorModes" : {

        },
        "getLastErrorDefaults" : {
                "w" : 1,
                "wtimeout" : 0
        },
        "replicaSetId" : ObjectId("6153f3ef2f30b89a0b923317")
```

Checking Status of the Replica Set

- To check the status of replica set, command is

  rs.status()





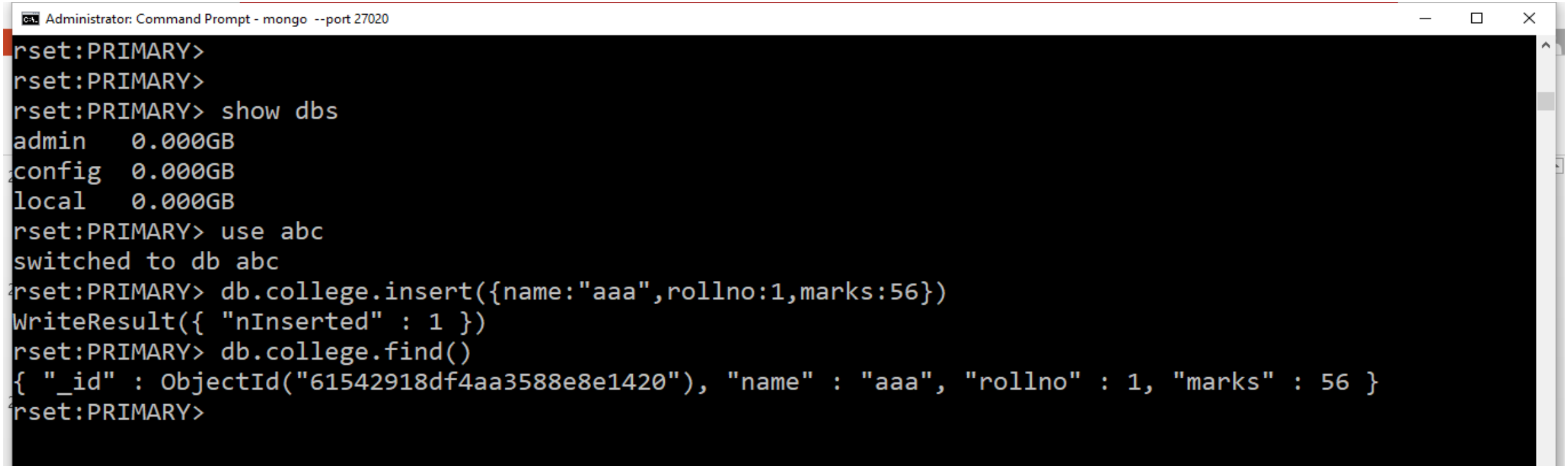
- Where,

  state: 1->primary, 2->secondary for backup

  health : 1->normal, 0->exception(means down)

- It shows a lot of other details also.

- Now create some database in this primary



- Then connect to secondary and then this data should replicate in that

```
rset:SECONDARY> rs.secondaryOk()
rset:SECONDARY> show dbs
abc     0.000GB
admin   0.000GB
config  0.000GB
local   0.000GB
rset:SECONDARY>
```

Contents to be covered

- replication configuration
- Replication status
- Remove members
- Adding new port in replica set

Replication Configuration

Now if we check the configuration again, after adding the member

> rs.config()

{
"_id" : "indu",
"version" : 7,
"protocolVersion" : NumberLong(1),
"writeConcernMajorityJournalDefault" : true,

```
"members" : [
        {
            "_id" : 0,
            "host" : "127.0.0.1:27020",
            "arbiterOnly" : false,
            "buildIndexes" : true,
            "hidden" : false,
            "priority" : 1,
            "tags" : {
            },
              "slaveDelay" : NumberLong(0),
            "votes" : 1
    },
```

```
{
        "_id" : 1,
        "host" : "127.0.0.1:27021",
          "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {
          },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
},
```

```
{
        "_id" : 2,
        "host" : "127.0.0.1:27022",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {
        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
},
```

```
{
        "_id" : 3,
        "host" : "127.0.0.1:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {
        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
},
```

```
        {
            "_id" : 4,
            "host" : "127.0.0.1:27023",
            "arbiterOnly" : false,
            "buildIndexes" : true,
            "hidden" : false,
        "priority" : 1,
            "tags" : {
             },
            "slaveDelay" : NumberLong(0),
            "votes" : 1
              }
    ],
```

```
        "settings" : {
        "chainingAllowed" : true,
        "heartbeatIntervalMillis" : 2000,
        "heartbeatTimeoutSecs" : 10,
        "electionTimeoutMillis" : 10000,
        "catchUpTimeoutMillis" : -1,
        "catchUpTakeoverDelayMillis" : 30000,
        "getLastErrorModes" : {
         },
         "getLastErrorDefaults" : {
              "w" : 1,
              "wtimeout" : 0
        },
        "replicaSetId" : ObjectId("5ecffc87e7d9c40b106bbd3a")
    }
}
```

We can see that two hosts has been added into the replica set.

Status of the Replica Set using rs.status()

rs.status() command is used to see the status of the replica set.

Syntax

rs.status()

Lets check it now as we have added two hosts.

This will show the message like this:

```
{
        "set" : "indu",
        "date" : ISODate("2020-05-29T06:17:49.792Z"),
        "myState" : 1,
        "term" : NumberLong(6),
```

```
        "syncingTo" : "",
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "heartbeatIntervalMillis" : NumberLong(2000),
    "majorityVoteCount" : 3,
    "writeMajorityCount" : 3,
    "optimes" : {
        "lastCommittedOpTime" : {
            "ts" : Timestamp(1590733061, 1),
            "t" : NumberLong(6)
        },
        "lastCommittedWallTime" : ISODate("2020-05-29T06:17:41.788Z"),
        "readConcernMajorityOpTime" : {
            "ts" : Timestamp(1590733061, 1),
```

```
                "t" : NumberLong(6)
            },
            "readConcernMajorityWallTime":ISODate("2020-05-29T06:17:41.788Z"),
                "appliedOpTime" : {
                    "ts" : Timestamp(1590733061, 1),
                    "t" : NumberLong(6)
            },
            "durableOpTime" : {
                    "ts" : Timestamp(1590733061, 1),
                    "t" : NumberLong(6)
        },
        "lastAppliedWallTime" : ISODate("2020-05-29T06:17:41.788Z"),
        "lastDurableWallTime" : ISODate("2020-05-29T06:17:41.788Z")
},
```

"lastStableRecoveryTimestamp" : Timestamp(1590733061, 1),
"lastStableCheckpointTimestamp" : Timestamp(1590733061, 1),
"electionCandidateMetrics" : {
"lastElectionReason" : "electionTimeout",
"lastElectionDate" : ISODate("2020-05-29T05:48:49.859Z"),
"electionTerm" : NumberLong(6),
"lastCommittedOpTimeAtElection" : {
"ts" : Timestamp(0, 0),
"t" : NumberLong(-1)
},
"lastSeenOpTimeAtElection" : {
"ts" : Timestamp(1590692225, 1),
"t" : NumberLong(4)
},

```
        "numVotesNeeded" : 2,
        "priorityAtElection" : 1,
        "electionTimeoutMillis" : NumberLong(10000),
        "numCatchUpOps" : NumberLong(0),
        "newTermStartDate" : ISODate("2020-05-29T05:48:51.633Z"),
        "wMajorityWriteAvailabilityDate" : ISODate("2020-05-29T05:48:52.045Z")
    },
```

```
"members" : [
        {       "_id" : 0,
                "name" : "127.0.0.1:27020",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 1755,
                  "optime" : {
                        "ts" : Timestamp(1590733061, 1),
                        "t" : NumberLong(6)
                },
```

```
                "optimeDate" : ISODate("2020-05-29T06:17:41Z"),
                "syncingTo" : "",
                    "syncSourceHost" : "",
                 "syncSourceId" : -1,
                infoMessage" : "",
        "electionTime" : Timestamp(1590731330, 1),
        "electionDate" : ISODate("2020-05-29T05:48:50Z"),
        "configVersion" : 3,
                "self" : true,
        "lastHeartbeatMessage" : ""
},
```

```
{
    "_id" : 1,
    "name" : "127.0.0.1:27021",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 1742,
    "optime" : {
        "ts" : Timestamp(1590733061, 1),
        "t" : NumberLong(6)
    },
        "optimeDurable" : {
        "ts" : Timestamp(1590733061, 1),
        "t" : NumberLong(6)
        },
```

```
        "optimeDate" : ISODate("2020-05-29T06:17:41Z"),
        "optimeDurableDate" : ISODate("2020-05-29T06:17:41Z"),
        "lastHeartbeat" : ISODate("2020-05-29T06:17:49.537Z"),
        "lastHeartbeatRecv" : ISODate("2020-05-29T06:17:49.077Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "127.0.0.1:27020",
        "syncSourceHost" : "127.0.0.1:27020",
        "syncSourceId" : 0,
        "infoMessage" : "",
        "configVersion" : 3
},
```

{ "_id" : 2,
"name" : "127.0.0.1:27022",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 1728,
"optime" : {
"ts" : Timestamp(1590733061, 1),
"t" : NumberLong(6)
},
"optimeDurable" : {
"ts" : Timestamp(1590733061, 1),
"t" : NumberLong(6)
},

"optimeDate" : ISODate("2020-05-29T06:17:41Z"),
"optimeDurableDate" : ISODate("2020-05-29T06:17:41Z"),
"lastHeartbeat" : ISODate("2020-05-29T06:17:49.541Z"),
"lastHeartbeatRecv" : ISODate("2020-05-29T06:17:49.077Z"),
"pingMs" : NumberLong(0),
"lastHeartbeatMessage" : "",
"syncingTo" : "127.0.0.1:27020",
"syncSourceHost" : "127.0.0.1:27020",
"syncSourceId" : 0,

```
{
        "_id" : 3,
        "name" : "127.0.0.1:27017",
        "health" : 0,
        "state" : 8,
        "stateStr" : "(not reachable/healthy)",
        "uptime" : 0,
        "optime" : {
                "ts" : Timestamp(0, 0),
                "t" : NumberLong(-1)
        },
        "optimeDurable" : {
                "ts" : Timestamp(0, 0),
                "t" : NumberLong(-1)
        },
```

`

```
"optimeDate" : ISODate("1970-01-01T00:00:00Z"),
        "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
        "lastHeartbeat" : ISODate("2020-05-29T06:17:49.629Z"),
        "lastHeartbeatRecv" : ISODate("1970-01-01T00:00:00Z"),
              "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "not running with --replSet",
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "configVersion" : -1
},
```

{
"_id" : 4,
"name" : "127.0.0.1:27023",
"health" : 0,
"state" : 8,
"stateStr" : "(not reachable/healthy)",
"uptime" : 0,
"optime" : {
"ts" : Timestamp(0, 0),
"t" : NumberLong(-1)
},
"optimeDurable" : {
"ts" : Timestamp(0, 0),
"t" : NumberLong(-1)
},

```
        "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
        "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
        "lastHeartbeat" : ISODate("2020-05-29T06:17:46.600Z"),
        "lastHeartbeatRecv" : ISODate("1970-01-01T00:00:00Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "Error connecting to 127.0.0.1:27023 :: caused by ::
      No connection could be made because the target machine actively refused it.",
            "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
       "configVersion" : -1
      }
],
```

```
        "ok" : 1,
"$clusterTime" : {
        "clusterTime" : Timestamp(1590733061, 1),
 "signature" : {
                "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                "keyId" : NumberLong(0)
        }
    },
    "operationTime" : Timestamp(1590733061, 1)
}
```

- See the message for Port 27017 which was not started with replica Set –
  - "stateStr" : "(not reachable/healthy)",
  - "lastHeartbeatMessage" : "not running with --replSet",

```
        {
                "_id" : 3,
                "name" : "127.0.0.1:27017",
                "health" : 0,
                "state" : 8,
                "stateStr" : "(not reachable/healthy)",
                "uptime" : 0,
                "optime" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "optimeDurable" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
                "lastHeartbeat" : ISODate("2020-05-29T06:17:49.629Z"),
                "lastHeartbeatRecv" : ISODate("1970-01-01T00:00:00Z"),
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "not running with --replSet",
                "syncingTo" : "",
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "",
                "configVersion" : -1
        },
```

- If instance has not started even not created with replica Set, the messages will be like
- stateStr" : "(not reachable/healthy)"
- "lastHeartbeatMessage" : "Error connecting to 127.0.0.1:27023 :: caused by :: No connection could be made because the target machine actively refused it.",

Command Prompt - mongo -port 27020

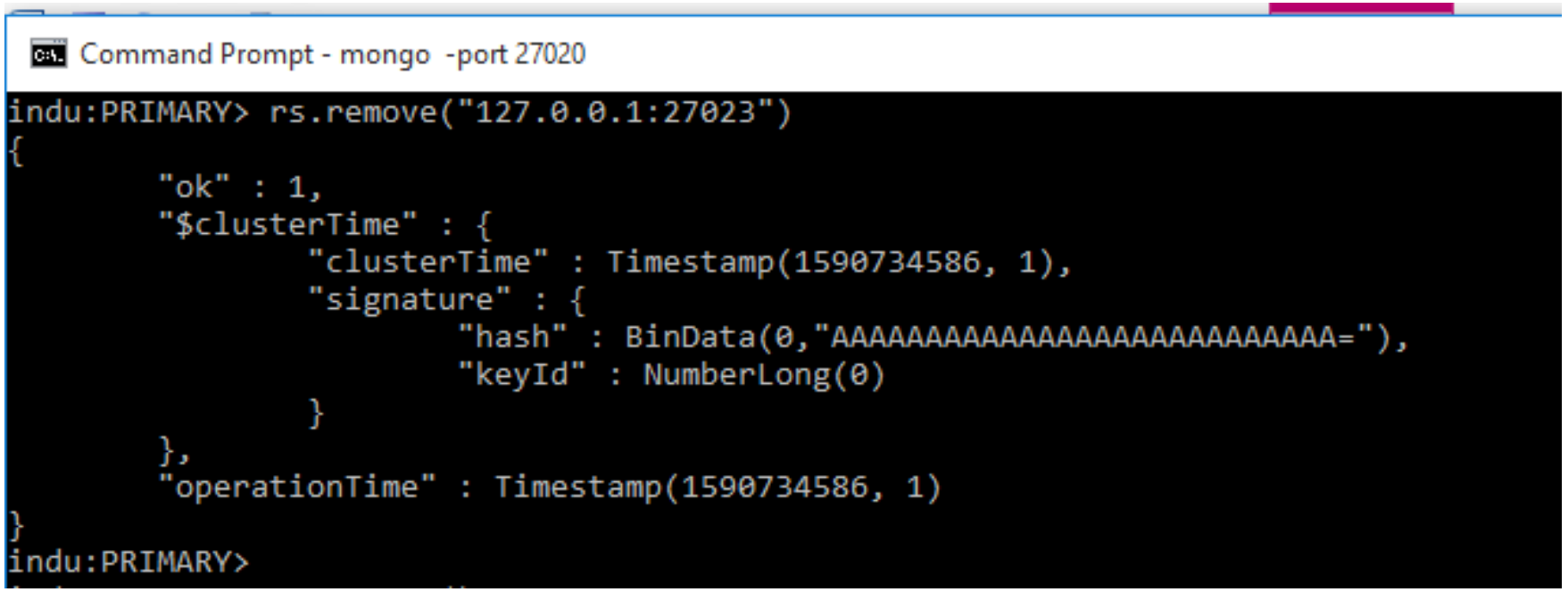

```
                },
                {
                        "_id" : 4,
                        "name" : "127.0.0.1:27023",
                        "health" : 0,
                        "state" : 8,
                        "stateStr" : "(not reachable/healthy)",
                        "uptime" : 0,
                        "optime" : {
                                "ts" : Timestamp(0, 0),
                                "t" : NumberLong(-1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(0, 0),
                                "t" : NumberLong(-1)
                        },
                        "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                        "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
                        "lastHeartbeat" : ISODate("2020-05-29T06:17:46.600Z"),
                        "lastHeartbeatRecv" : ISODate("1970-01-01T00:00:00Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "Error connecting to 127.0.0.1:27023 :: caused by :: No connection
could be made because the target machine actively refused it.",
                        "syncingTo" : "",
                        "syncSourceHost" : "",
                        "syncSourceId" : -1,
                        "infoMessage" : "",
                        "configVersion" : -1
                }
        ],
```

Remove Members

- we can remove any member(s) from the existing replica set using the rs.remove() command.
- **Syntax**

  >rs.remove(“HOST_NAME:PORT”)
- **Example,** lets remove the member

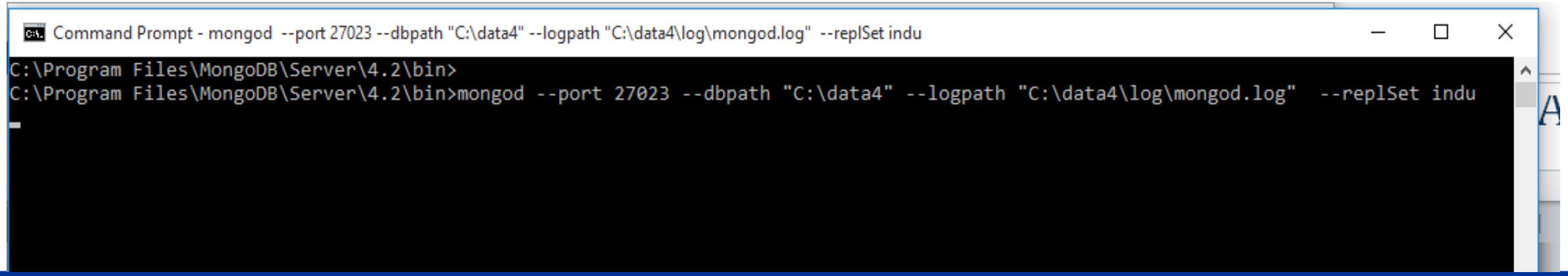
  >rs.remove("127.0.0.1:27023")



- If we run **rs.config()** command again, we will see that the particular port has been removed from the replica set.

Adding a new Port / instance in replica set: Steps and Cautions

- First create the folders for the database and log files
- Copy and Rename the configuration file
- Edit the configuration file with the db & log Path, Port number and bind IP , Replica name and size
- Start the MongoDB instance with Replica Set and Port no like



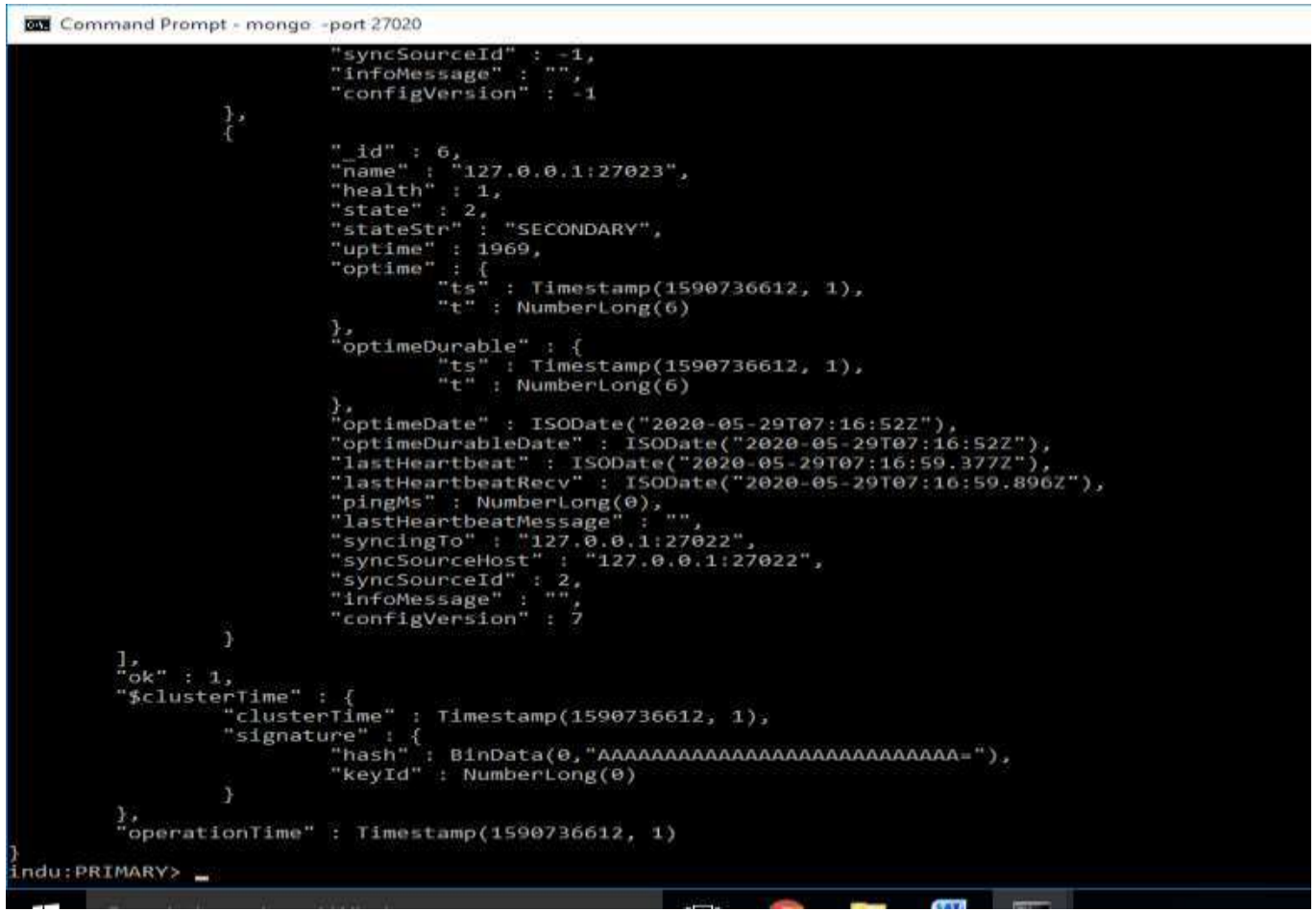
Adding a new Port / instance in replica set: Steps and Cautions

- Now Add the Host to the replica Set using rs.add () command

  >rs.add("127.0.0.1:27023")

- If desired, check the replica set configuration using rs.conf()
- Check the status of Replica Set with rs.status() again after adding the member properly. Now it shows that
  - **"stateStr" : "SECONDARY" -- Means that it is Secondary**
  - **"lastHeartbeatMessage" : "" -- means no error and is active**

Adding a new Port / instance in replica set: Steps and Cautions



Contents to be covered

- Terminologies used in replication,
- Help on replication
- Freezing a member
- Stepping down a member
- Printing the oplog of the replica set

Terminologies used in Replication

- **Heartbeat**
  - Replication is used to keep multiple instances of the database simultaneously running usually on different systems of the same replica set.
  - Heartbeat is the process to identify the current status of such servers (also called node servers) within the replica set. To do so, members of replica set send heartbeats (technically pings) to each other at every **two seconds.**
  - In case, if a heartbeat does not return (i.e. no response) within 10 seconds, then the other members of replica set mark such delinquent member as an inaccessible member.

Terminologies used in Replication

- A heartbeat request is a short message that checks everyone's current state. Heartbeat is the process to know the other member's state, like who’s primary, from which member they need to sync from or which node is down.
- The most important activity of heartbeat is to check that the primary server live and reachable by all secondary servers. In case, if the most of the secondary servers can’t reach to the primary server, then the process automatically demotes the primary server as a secondary server.

Terminologies used in Replication

- **Elections**
  - Replica set members uses election to determine which set member will become primary as the Primary node is not accessible. If a primary becomes unavailable, elections allow the set to recover normal operations without manual intervention. Elections may also be called as the part of the failover process.
  - Elections occur anytime when the primary becomes unavailable. It is important to know that the primary is the only member in the replica set which can accept write operations. And while an election is in process, the replica set has no primary and cannot accept writes. MongoDB avoids elections unless necessary.

Terminologies used in Replication

- **Election Process**
  - During the heartbeat check, if a member can’t reach the primary server then that particular member raises the **election flag** to the other members of the replica set so that other members of the replica set can’t raise the same election flag within the process.

Terminologies used in Replication

- If there is no objection against the election request, the other members will vote for the member seeking election.
- And if the member receives the majority votes from other members, then the election is successful and it will be promoted as the primary node.
- In case, if the member did not receive the majority of votes then it will remain as secondary node and maybe try to become a primary node in the future.

Terminologies used in Replication

- Elections take time to complete .Elections are essential for independent operation of a replica set; however, while an election is in process, the replica set has no primary and cannot accept writes.
- MongoDB avoids elections unless necessary. If the network condition is healthy and most of the servers are up, then the entire elections process should happen very quickly.
- Two seconds is required to notify all the members that primary is down as heartbeat response has not been received yet and the election process starts immediately.

Terminologies used in Replication

- **Oplog**
  - The oplog, i.e. operations log, is a special capped collection in MongoDB that keeps a rolling record of all operations that modify the data stored in your databases.
  - All the database operations are applied to the primary and operations are recorded on the oplog of the primary. All the secondary members then copy and apply these operations in an asynchronous process.
  - To maintain the current state of the database, all replica set members contain a copy of the oplog in the local.oplog.rs collection.
    Any secondary member may import oplog entries from any other member.

Terminologies used in Replication

- **Syncing**
  - A log of operations called oplog is maintained by MongoDB having every write (transaction/ operation) information onto the primary server. As the main aim of the replication process is to keep the same or an identical set of data on multiple servers of the Replica Set, the oplog helps in achieving this goal.
  - Oplog is a capped collection and is there in the local database on the primary server. The secondary servers query this collection(oplog) to obtain the operation details so that they can replicate that data.

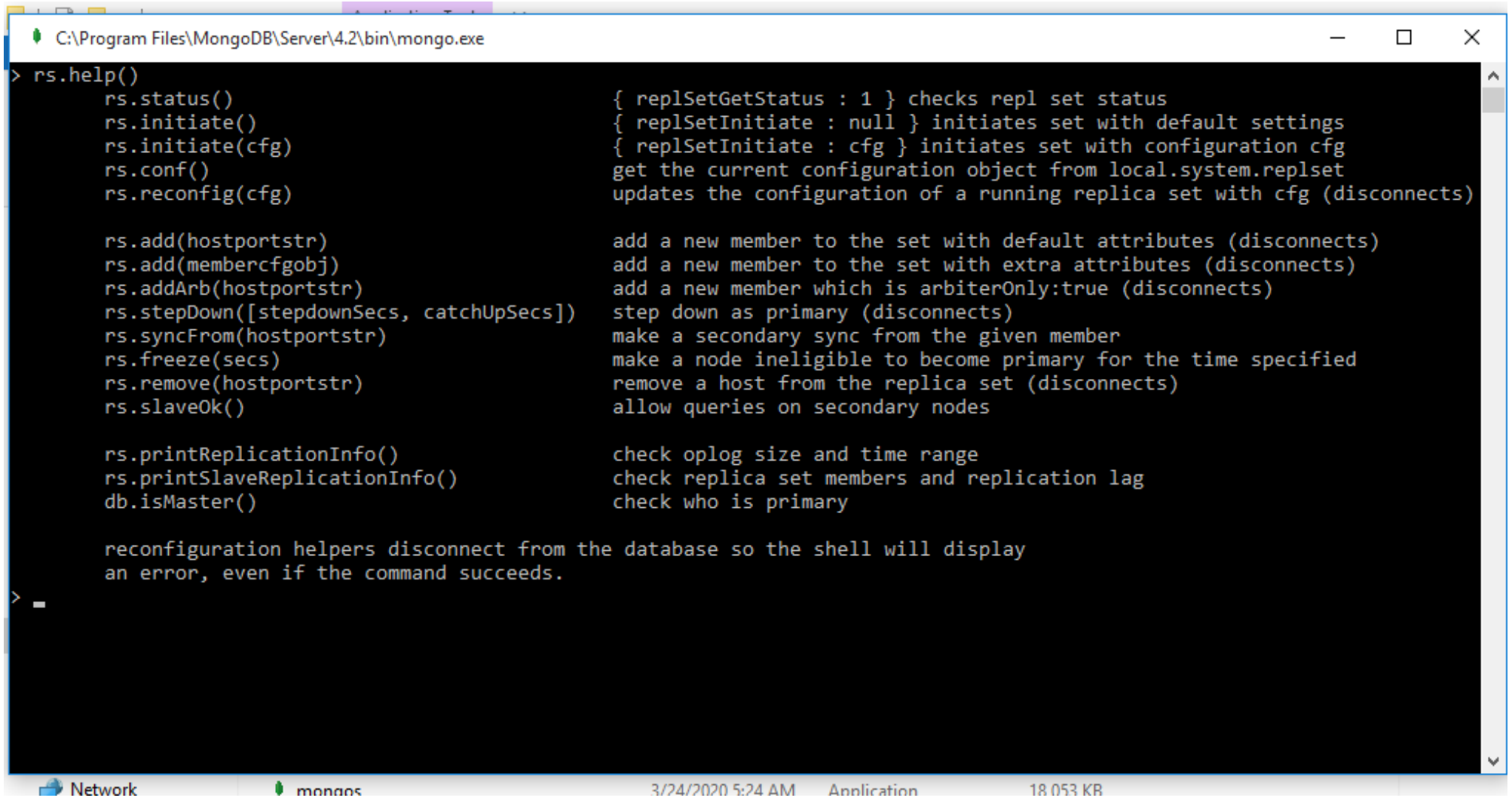
Terminologies used in Replication

- Oplog is also maintained at every secondary server in which MongoDB stores each operation related to the replication process from the primary server.
- These log files allow any replica set members to use as a sync source for other members. Every time, the secondary server first fetch the information related to the pending operations from the primary member, then apply that operation to their own data set and then write down the logs about that operation into the oplog.
- In case, if the secondary server goes down and is going up after some time, then it starts the syncing process the last operation is done by itself according to its oplog file.
- As the operation first applied to the data and then it writes to the oplog, the secondary server may redo the operations that it has already applied to its data.

Help on Replication

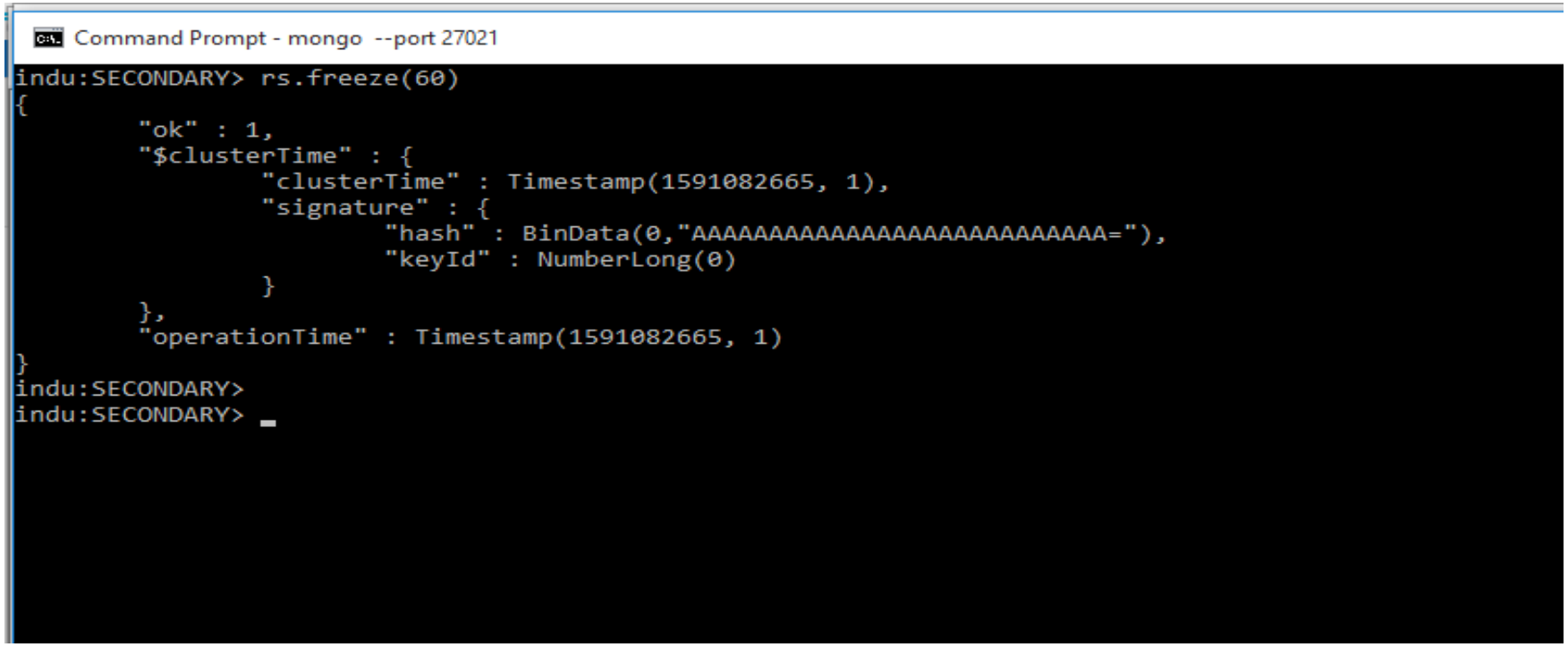
- rs.help() command provides the basic help for various replica set function/ methods.

Help on Replication



Freezing a member

– rs.freeze() method prevents the current member from seeking election as primary for a period of time specified in seconds.



Stepping Down a Member

- The method runs only if the current member is a primary node and produce an error if runs on a non-primary member. rs.stepDown() instructs the current Primary node of the replica set to become a secondary which forces an election. After the primary steps down, eligible secondary nodes hold an election for primary.
- **Syntax**
  - **rs.stepDown(stepDownPeriod, secondaryCatchUpPeriod)**

Stepping Down a Member

where,

- **stepDownPeriod-** Specifies the number of seconds to step down the primary, during this time the stepdown member is ineligible for becoming primary. By default it is 60 seconds. It must be greater than the secondaryCatchUpPeriod.

Stepping Down a Member

- **secondaryCatchUpPeriod** – specifies the number of seconds that mongod will wait for an electable secondary to catch up to the primary. This is Optional. The default wait time is 10 seconds.
- rs.stepDown() method will not immediately step down the primary. If no electable secondary node is up to date with the primary, the primary waits up to secondaryCatchUpPeriod for a secondary node to catch up. Once an electable secondary is available, the method steps down the primary. Once stepped down, the original primary becomes a secondary and is ineligible from becoming primary again for the remainder of time specified by stepDownPeriod.

Command Prompt - mongo --port 27020

```
indu:PRIMARY>
indu:PRIMARY> rs.stepDown(90,20)
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1591082825, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1591082825, 1)
}
indu:SECONDARY>
```

Read Operation on Secondary Member

- rs.secondaryOk() method allows the current connection to allow read operations to run on ***secondary*** members. i.e. allow users to run commands in the secondary or arbiter node.
- **Syntax**
  - rs.secondaryOk()

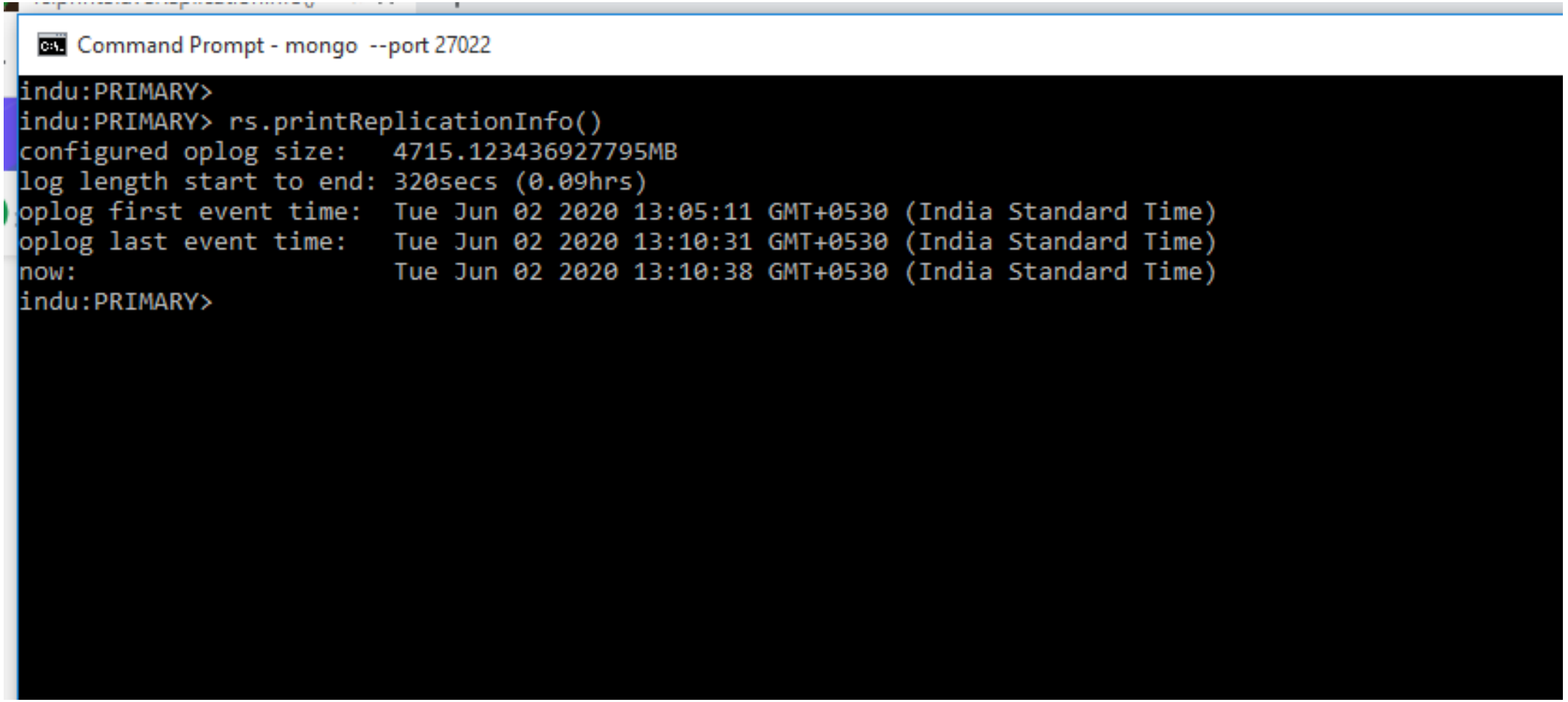
Read Operation on Secondary Member

```
rset:SECONDARY> rs.secondaryOk()
rset:SECONDARY> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
student  0.000GB
rset:SECONDARY> use student
switched to db student
rset:SECONDARY> db.school.find()
{ "_id" : ObjectId("60d174b5a2e8fa78069d751b"), "name" : "sarita", "rollno" : 1, "marks" : 45 }
{ "_id" : ObjectId("60d17528a2e8fa78069d751c"), "name" : "geeta", "rollno" : 2, "marks" : 56 }
rset:SECONDARY>
```

Printing the oplog of the replica set

- rs.printReplicationInfo() method prints the report of the replica set member's oplog from the perspective of the Primary Member.
- **Syntax**
  - rs.printReplicationInfo()

rs.printReplicationInfo()



rs.printReplicationInfo()

Command Prompt - mongo --port 27021

```
indu:SECONDARY> rs.printReplicationInfo()
configured oplog size:   4730.51523399353MB
log length start to end: 1494secs (0.42hrs)
oplog first event time:  Tue Jun 02 2020 12:46:13 GMT+0530 (India Standard Time)
oplog last event time:   Tue Jun 02 2020 13:11:07 GMT+0530 (India Standard Time)
now:                     Tue Jun 02 2020 13:11:08 GMT+0530 (India Standard Time)
indu:PRIMARY>
```

rs.printSlaveReplicationInfo()

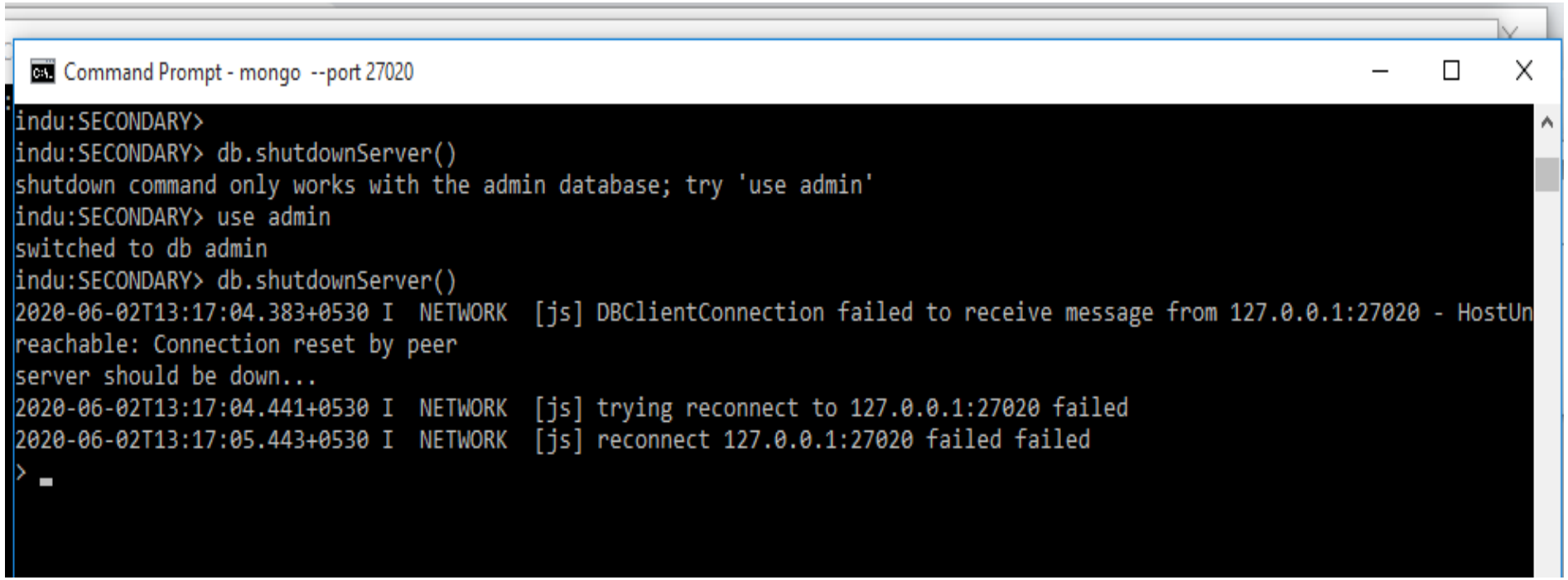
- rs.printSlaveReplicationInfo() method prints the report of the replica set member's oplog from the perspective of the Secondary Members of the Set.

- **Syntax**
  - **rs.printSlaveReplicationInfo()**

Stopping a Member

- db.shutdownServer() method is used to shut down or Stopping a member. It may be a secondary member. This command will run only with admin database.
- **Syntax**
  - **db.shutdownServer()**

Stopping a Member



choosing the right type of database for your enterprise

- There are many databases to choose from but selecting the right kind of database is very essential. All have their strengths and weaknesses

key factors to consider when evaluating your database solutions.

1. **What's the structure of the data?**
   - The data's structure will dictate how it's stored and retrieved. As most apps deal with data in a variety of formats, the process of selecting the database should include the appropriate data structures for storing and retrieving the data.
   - If you fail to do this, your mobile application will be slow to retrieve the data from the database. Furthermore, it will also require more development time to work around the data issues that will certainly come up.

key factors to consider when evaluating your database solutions.

- Other data related elements that can play an important role are as follows:
  - Accessibility to the data
  - Size of the data you wish to store
  - Scope of multiple databases
  - Speed and scalability

key factors to consider when evaluating your database solutions.

2. **Will you require a flexible data modeling solution?**
   - The flexibility of data modeling will dictate if you can appropriately and effectively modify your data model requirements for your mobile app.
   - As mobile apps today evolve rapidly, model flexibly quickly becomes an important factor to consider. In this scenario, relational databases can be a good choice if you require strong data consistency. The same is true if the data will be highly relational.
   - However, if the requirements are relaxed, NoSQL databases are the way forward as they offer enhanced flexibility.

key factors to consider when evaluating your database solutions.

3. **What client platforms does it support?**
   - Are you planning on supporting iOS, Android, or both? Are looking to go beyond and support the Windows Phone as well? What about IoT devices and wearables?
   - Even if you plan to do support more platforms at a later date, you have to take that into consideration now
   - If you plan on going down the same path, it will be important to evaluate cloud options and databases based on the required platform support during the lifecycle of the application

key factors to consider when evaluating your database solutions.

4. How much data security will you need?

- Whether at rest or in motion, a high level of security must be maintained consistently. If the storage is decentralized and synchronized, it's also important to enable secure access and transmission.
- As a result, you will need to look at the following:
  - Authentication
  - Data in motion
  - Data at rest
  - Read/write access

key factors to consider when evaluating your database solutions.

- As far as authentication goes, it has to be flexible enough to allow the following authentication providers:
  - Custom
  - Public
  - Standard

key factors to consider when evaluating your database solutions.

- At the same time, anonymous access is also important, so for the data to both rest on the server and the client, you will need to be able to support both file system encryption and data-level encryption.
- For the data that’s in motion, all communication should be conducted on secure channels like TLS or SSL. Furthermore, for data read/write access, the database will need to provide granular control over what information can be accessed and modified by users.

key factors to consider when evaluating your database solutions.

5. **How will it resolve data conflicts?**
   - Mobile platforms or platforms that generally use decentralized data writes can quickly experience conflicts as the same data can be simultaneously modified by multiple devices. As a result, you will need a robust support mechanism to resolve conflicts.
   - What's important here is the flexibility of the conflict resolution mechanism. The one you choose should be able to seamlessly enable conflict resolution on the device, by a human, by an external system, in the cloud, automatically.

key factors to consider when evaluating your database solutions.

6. What are your partition requirements?

- To meet your partition requirements, you will need configurable synch topology support (like star). This will enable certain parts to function offline.
- Star topology is quite common because it enables devices to connect to a central hub utilizing a point-to-point connection (which also allows devices to function offline). There are other common topologies like tree and mesh which allow different parts of the system and devices to operate offline

key factors to consider when evaluating your database solutions.

7. Do you want to build or buy your synch capabilities?

- When it comes to adding synch to your mobile app, you will need to make a decision on whether you want to build your own synch solution or buy it from an established provider. For most apps, you're better off buying as building your own synch will be extremely difficult and expensive (because you have to deal with complexities associated with distributed computing).

key factors to consider when evaluating your database solutions.

- What’s important here is to find a solution that’s highly flexible. However, if you’re determined to build your own synch, then you will need to be prepared to expand a significant amount of time and money to achieve it.
- When evaluating mobile synch and storage providers, be sure to go over everything listed above as it will be critical to developing a flexible, secure, manageable, and dependable mobile application.

Database Era

- Collecting and storing data is a very old ancient concept. Even after the evolution of computers and computing devices, data has been stored in various ways.
- Earlier databases had rather limited functionalities starting with plain text files called as **flat databases**. This means the data has to be of a textual format. Every new field is marked with a user defined delimiter – a special character, a comma or a colon etc.
- As there are no relations between the fields, a flat database is hard to search and navigate. Still, this works well for a small amount of data that only needs to be read and or slight modification. An example of such databases can be CSV (Comma Separated Values) files.

Database Era

- IBM introduced **hierarchical databases 60's in which** the records are connected by a tree structure, based on parent-child relationships. One item can only have one parent, while one parent can have multiple children.
- This can be said as the first step toward relational databases. It has its own disadvantages and does not work well for all types of records. N**etwork databases come after this and have a better organization of data.** These have a tree structure, but children node may have multiple parents as well. Later the era of **relational database** has arrived followed by **non-relational** (**NoSQL**) databases.

Taking Decisions on Databases

- If compliance of ACID (Atomicity, Consistency, integrity and Durability) properties is essential, RDBMS is the best solution. For example, databases for financial transactions.
- If your input data is particularly heterogeneous and difficult to encapsulate according to a normalization schema, consider using a NoSQL DBMS. For example, data from Social Media.
- If your goal is to scale database vertically, a RDBMS is suggested and if you want to scale horizontally, a NoSQL DBMS may be preferred.
- If you have a massively distributed system and can settle for eventual consistency on some nodes/partitions, you might consider a wide column store

Normalized(SQL) Vs De-normalized Data(NoSQL)

- Normalization is a database business process to break up data into the smallest possible parts. Instead of storing first and last name in one bucket, or field, normalization requires that you store the first name separately from the last name.

Normalized(SQL) Vs De-normalized Data(NoSQL)

- This is helpful if you want to sort the data by last name or by first name. Relational databases are the most common database systems and require that data is **normalized**. These data is stored in columns and rows, which in turn make up tables like a spreadsheet.
- A set of tables makes up a schema. A number of schemas create a database. Many databases can be created on a single server. RDBMS system feature much better performance for managing data over desktop database programs. RDBMS allow multiple users (even thousands!) to work with the data at the same time, creating advanced security for access to the data.

Normalized(SQL) Vs De-normalized Data(NoSQL)

- Databases like NoSQL and object-oriented do not follow the table/row/column approach of RDBMS. Instead, they build bookshelves of elements and allow access per bookshelf.
- So, instead of tracking individual words in books, these databases narrow down the data being searched for by pointing to the bookshelf, then a mechanical assistant works with the books to identify the exact word you are looking for.
- NoSQL specifically attempts to simplify bookshelves by storing data in a **denormalized** way; this means storing it in large chunks

Contents to be covered

- Strengths and weaknesses of databases
- Free and open source database

Strengths and weaknesses of databases

- There are many databases to choose from but selecting the right kind of database is very essential. All databases are not equal and each of them has specific strengths and weaknesses. Let's have a view of some of the databases:
- ***Relational database management systems***
  - It is over 4 decades when RDBMS were first come into the real world to handle the increasing flood of data being produced. And since then, RDBMS have a solid foundational theory and have influenced nearly every database system in use today.

Strengths and weaknesses of databases

  - RDBMS store normalized data in form of data sets called **relations** i.e. tables with rows and columns where all information is stored as a value of a specific cell. SQL is used to manage data in an RDBMS system. SQL is standardized and provides a level of predictability and utility.

- **Strengths**
  - Relational databases have expertise in handling highly structured data and provide support for ACID (Atomicity, Consistency, Isolation, and Durability) transactions.
  - The structure can be scaled up quickly because adding data without modifying existing data is simple.
  - Data can be easily stored, modified and retrieved using SQL queries.

Strengths and weaknesses of databases

- Access rights can be assigned to certain users, like to only view the data, modify the data / data structure or full access etc. Due to these, relational databases are well-suited to applications that require tiered access.

- **Weaknesses**
  - RDBMS are expensive to set up and grow. Horizontal scaling, or scaling by adding more servers, is usually both faster and more economical than vertical scaling, which involves adding more resources to a server is supported by RDBMS. Further, the structure of relational databases complicates the process.

Strengths and weaknesses of databases

- RDBMS are good as they are handling structured data, but difficult to handle unstructured data. Representing real world entities in context is difficult in the bounds of an RDBMS. Now a day, a lot of unstructured data is playing in the real world.

- Name:aaa,designation:”manager”,salary:13000,
- Name:abb,designation:”clerk”,

Strengths and weaknesses of databases

- For better readability, the data has to be reorganized from tables into a more readable form, and this also impact the speed. The fixed schema of these databases doesn’t support such changes in a positive way.
- Sharding is necessary to scale out a relational database. In sharding, data is horizontally partitioned and distributed across a collection of machines. But maintaining ACID compliance while Sharding in RDBMS is again a big issue.

Strengths and weaknesses of databases

- **Final verdict**
- Based on the various parameters, strengths and weaknesses, a relational database may be used for
  - Situations where data integrity is absolutely necessary for example financial applications, banking application, military defense and security applications, private health information systems etc.
  - Applications having Highly structured data
  - Applications having automation of internal processes
  - ***Some example of RDBMS are*** *Oracle, MySQL, MS SQL Server, Sybase, Informix, PostgreSQL.*

Strengths and weaknesses of databases

- ***NoSQL - Document store Databases***
  - Document-oriented databases, also known as document stores, are used to store and manage **semi-structured or unstructured data**. A document store is a nonrelational database that stores data in JSON, BSON, or XML documents.
  - These databases supports a flexible schema i.e. the data does not adhere to a fixed structure, instead it forms its own structure. Due to the lack of a defined structure i.e. no schema of table is declared before inserting data, document stores don't enforce document structure.
  - Documents can contain any data desired. Since the information of Document Stores cannot be arranged in tables, this data is not suitable for relational databases

Strengths and weaknesses of databases

- ***NoSQL - Document store Databases***
  - A document database creates a key-value pair: A **key** is assigned to a specific document. The actual information is then located within this document, which may be formatted as an XML, JSON or YAML(YAML ain't markup language) file. Documents have key-value pairs but also embed attribute metadata (fields) to make querying easier.
  - Since the document does not require a specific schema, different types of documents can also be integrated together in a document store. Further, any changes or modifications to the documents need not to be communicated to the database.

Strengths and weaknesses of databases

- ***NoSQL - Document store Databases***
- **Strengths**
  - Document stores are very flexible and can store any type of data. They can easily handle semi structured and unstructured data. Even large volumes of unstructured data can be accommodated in the database.
  - Users don’t need specify what data will be stored, so this is a good choice when it isn’t clear in advance what types of data will be stored. For example: Social Media data.

Strengths and weaknesses of databases

- Easier to integrate **new information**
- Each record may have different structure, these records are called documents. Users can create their desired structure in a particular document without affecting all documents and subsequently modify it again to a new structure. Schema can be modified without causing downtime, which leads to high availability. Write speed is generally fast, as well.
- The information is not distributed over multiple linked tables. Everything is contained in a single location, and this can result in better performance.
- Document stores can be scaled horizontally easily. The sharding necessary for horizontal scaling is much more intuitive than with relational databases, so document stores scale out fast and efficiently.

Strengths and weaknesses of databases

NoSQL - Document store Databases

- **Weaknesses**
  - Document databases sacrifice ACID compliance for flexibility.
  - Query can be done in a particular document not across multiple / different documents (not like multiple tables of RDBMS). So, not suitable for highly networked data volumes.
  - No restrictions on data and any type of data can be feed into documents, this raises the potential for accidents & threats (both unintentionally or intentionally)

Strengths and weaknesses of databases

NoSQL - Document store Databases

- **Final verdict**
- Based on the various parameters, strengths and weaknesses, a Document Store database may be used for
  - Unstructured or semi structured data
  - Content management
  - In-depth data analysis
  - Some of the examples of Document Store databases are MongoDB, CouchDB, BaseX, Elasticsearch etc.

Strengths and weaknesses of databases

NoSQL-Key-value store databases

- A **key-value store database** is a type of nonrelational **database** that uses a simple **key-value** method to **store** data. In key-value, each value is associated with a specific **key**. A **key-value database stores** data as a collection of **key-value** pairs where the **key** serves as a unique identifier. Both **keys** and **values** can be anything, ranging from simple objects to complex compound objects. It just has a one-way mapping from the key to the value to store data.

- Student
- Name
- Rollno
- marks

Strengths and weaknesses of databases

NoSQL-Key-value store databases

- A key-value store uses an array of keys where each key is associated with only one value in a collection. It is quite similar to a dictionary or a map data structure. It's also known as an associative array.
- Key1:value1,key2:value2,key3:value3
- "key" is a unique identifier associated only with the value. Keys can be anything allowed by the DBMS.

Strengths and weaknesses of databases

NoSQL-Key-value store databases

- "Values" are stored as blobs and don't need predefined schema. They can take nearly any form: numbers, strings, counters, JSON, XML, HTML, PHP, binaries, images, short videos, lists, and even another key-value pair encapsulated in an object.

- **Strengths**
  - Are very flexible and able to handle a very wide array of data types easily.
  - Have high performance as Keys are used to go straight to the value with no index searching or joins.
  - High performance because the integrated caching feature allows users to store and retrieve data in the shortest time possible.

Strengths and weaknesses of databases

NoSQL-Key-value store databases

- Can be moved from one system to another without rewriting code. i.e. easily Portable
- Easy to scale without disrupting operations. Users can add and remove servers depending on their needs without causing undesirable disruptions.
- users can simply add new features when the need arises

- **Weaknesses**
  - No query language to retrieve data. Only have some simple operations such as get, put and delete.
  - Almost impossible to query values, because they’re stored as a blob and can only be returned as such.

Strengths and weaknesses of databases

NoSQL-Key-value store databases

- Difficult to do reporting or edit parts of values.
- Data querying (retrieving) usually handled manually at the application level.

- **Final Verdict**
- Based on the various parameters, strengths and weaknesses, a Key-value Store database may be used for
  - Session management at high scale
  - User preference, profiles and settings
  - Unstructured data such as product reviews or blog comments
  - Product recommendations; latest items viewed on a retailer website drive future customer product recommendations

Strengths and weaknesses of databases

NoSQL-Key-value store databases

- Ad servicing; customer shopping habits result in customized ads, coupons, etc. for each customer in real-time
- Can effectively work as a cache for heavily accessed but rarely updated data i.e. Data that will be accessed frequently but not often updated
- Some of the examples of Key-Value Store databases are Amazon DynamoDB, MemcacheDB, InfnityDB, Redis, Aerospike, Oracle Berkeley DB, Riak KV, Voldemort etc

Strengths and weaknesses of databases

NoSQL-*Wide-column store*

- Wide-column stores are dynamic column-oriented nonrelational databases. Wide column stores are database management systems that organize related facts into columns. Groups of these columns, called “column families”, have multiple rows and rows may not have same number of columns.

Name	Details				
Aaa	Phone number	Eng	Hindi	Science	
	56456456456	52	62	52	
Bbb	address	Eng	Hindi	Science	Maths
	House no.23,bldg. no.34	85	95	95	45

Strengths and weaknesses of databases

NoSQL-*Wide-column store*

- Wide-column stores use the concept of a keyspace instead of schemas. A keyspace encompasses column families (similar to tables but more flexible in structure), each of which contains multiple rows with distinct columns.
- Each row doesn't need to have the same number or type of column. The columns name as well as record keys are not fixed in these databases.

Student			Employee			Account		
Id:1	Name:aaa		Id:1	Name:ppp	Salary:333	Id:78	Designation:”asdsda”	
Id:3	Name:bbb	Marks:34	Id:4	Salary:555		Id:56	Designation:clerk	Bonus:4000

Strengths and weaknesses of databases

NoSQL-*Wide-column store*

- **Strengths**
  - This type of database has some benefits of both relational and nonrelational databases. It deals better with both structured and semi structured data than other nonrelational databases, and it's easier to update. Columnar databases compress better than row-based systems. Also, large data sets are simple to explore.

Strengths and weaknesses of databases

NoSQL- Columnar databases

- Columnar databases are highly scalable horizontally and faster at scale because they store data in columns rather than rows. The columns are easy to scale so they can store large volumes of data. This feature also allows users to spread data across many computing nodes and data stores.
- Column stores are very efficient at data compression. Columnar databases are highly compressed compared to conventional relational databases that store data by row. They allow users to optimize storage size.
- Due to their structure, columnar databases perform particularly well with aggregation queries such as SUM, COUNT, AVG, etc.

NoSQL Columnar databases

- Database users can sort and manipulate data directly from a columnar database and there is no need to rely on the application.
- Columnar stores can be loaded extremely fast. A billion row table could be loaded within a few seconds. You can start analyzing almost immediately.
- **Weaknesses**
  - Wide-column stores are slower than relational databases when handling transactions.
  - Good for bulk write and updates but it's a costly affair in lower quantity like individual records.
  - Not suitable for Incremental data loading & Queries against only a few rows

NoSQL Columnar databases

- **Final Verdict**
- Based on the various parameters, strengths and weaknesses, a wide-column Store database may be used for
  - Big data analytics where speed is important
  - Reporting systems
  - Time Series Data

NoSQL Columnar databases

- Data warehousing on big data
- Extreme write speeds with relatively less velocity reads like Logging
- Sensor Logs [Internet of Things (IOT)]
- User preferences
- Geographic information
- *Some of the examples of wide-column Store databases are Cassandra, HBase, Microsoft Azure Cosmos, Druid, Hypertable etc.*

NoSQL-Search engine

- It may seem strange to include search engines in an article about database types. However, Elasticsearch has seen increased popularity in this sphere as developers look for innovative ways to cut down search lag.
- Elastisearch is a nonrelational, document-based data storage and retrieval solution specifically arranged and optimized for the storage and rapid retrieval of data.

NoSQL-Search engine

- **Strengths**
  - Elastisearch is very scalable. It features flexible schema and fast retrieval of records, with advanced search options including full text search, suggestions, and complex search expressions.
  - One of the most interesting search features is stemming. Stemming analyzes the root form of a word to find relevant records even when another form is used. For example, a user searching an employment database for “paying jobs” would also find positions tagged as “paid” and “pay.”

NoSQL-Search engine

- **Weaknesses**
  - Elastisearch is used more as an intermediary or supplementary store than a primary database. It has low durability and poor security. Also, Elastisearch doesn't support transactions.
  - Use a search engine like Elastisearch for:
    - Improving user experience with faster search results
    - Logging

A few open source database solutions available in the market

- **MySQL**
  - MySQL has been around since 1995 and is now owned by Oracle. Apart from its open source version, there are also different paid editions available that offer some additional features, like automatic scaling and cluster geo-replication.
  - We know that MySQL is an industry standard now, as it's compatible with just about every operating system and is written in both C and C++.
  - This database solution is a great option for different international users, as the server can provide different error messages to clients in multiple languages, encompassing support for several different character sets.

MySQL

- **Pros**
  - It can be used even when there is no network available.
  - It has a flexible privilege and password system.
  - It uses host-based verification.
  - It has security encryption for all the password traffic.
  - It consists of libraries that can be embedded into different standalone applications.
  - It provides the server as a separate program for a client/server networked environment.

MySQL

- **Cons**
  - Different members are unable to fix bugs and craft patches.
  - Users feel that MySQL no longer falls under the category of a free OS.
  - It’s no longer community driven.
  - It lags behind others due to its slow updates.

SQLite

- **SQLite**
  - SQLite is supposedly one of the most widely deployed databases in the world. It was developed in 2000 and, since then, it has been used by companies like Facebook, Apple, Microsoft and Google. Each of its releases is carefully tested in order to ensure reliability. Even if there are any bugs, the developers of SQLite are quite honest about the potential shortcomings by providing bug lists and the chronologies of different code changes for every release.
- **Pros**
  - It has no separate server process.

SQLite

- The file format used is cross-platform.
- It has a compact library, which runs faster even with more memory.
- All its transactions are ACID compliant.
- Professional support is also available for this database.

SQLite

- **Cons**
- It’s not recommended for:
  - Different client/server applications.
  - All high-volume websites.
  - High concurrency.
  - Large datasets.

MongoDB

- MongoDB was developed in 2007 and is well-known as the ‘database for giant ideas.’ It was developed by the people behind ShopWiki, DoubleClick, and Gilt Group. MongoDB is also backed by a large group of popular investors such as The Goldman Sachs Group Inc., Fidelity Investments, and Intel Capital.
- Since its inception, MongoDB has been downloaded over 15 million times and is supported by more than 1,000 partners. All its partners are dedicated to keeping this free and open source solution’s code and database simple and natural.

MongoDB

- **Pros**
  - It has an encrypted storage engine.
  - It enables validation of documents.
  - Common use cases are mobile apps, catalogues, etc.
  - It has real-time apps with an in-memory storage engine (beta).
  - It reduces the time between primary failure and recovery.

MongoDB

- **Cons**
  - It doesn’t fit applications which need complex transactions.
  - It’s a young solution—its software changes and evolves quickly

MariaDB

- MariaDB has been developed by the original developers of MySQL. It is widely used by tech giants like Facebook, Wikipedia and even Google. It’s a database server that offers drop-in replacement functionality for MySQL.
- Security is one of the topmost concerns and priorities for MariaDB developers and in each of its releases, the developers also merge in all of MySQL’s security patches, even enhancing them if required.

MariaDB

- **Pros**
  - It has high scalability with easier integration.
  - It provides real-time access to data.
  - It has the maximum core functionalities of MySQL (MariaDB is an alternative for MySQL).
  - It has alternate storage engines, patches and server optimisations.

MariaDB

- **Cons**
  - Password complexity plugin is missing.

1) What is MongoDB?

A. It is a data growth

B.It is a adminCommand

C.It is a combine objects

D.It is a document database

2) MongoDB was Initial released in .....

A. August 2009

B.August 2008

C.August 2010

D.None of the above

3) In mongodb, data is represented as a collection of _____

A. Tables

B.Images

C.Files

D.None of the above

4) A collection and a document in mongodb is equivalent to which of the sql concepts respectively?

A. Column and Row

B.Table and Row

C.Table and Column

D.Database and Table

5) Which of the following statements is correct about mongoose in mongodb?

A. it is Java library to connect with MongoDB.

B.It is a PHP library to connect with MongoDB.

C.It is Python library to connect with MongoDB

D.It is used for modeling your application data in node.js

6) A collection in mongodb is a group of .....

A. Rows

B.Schema

C.Databases

D.Related documents

7) Please choose the correct option?

A. MongoDB is a NoSQL database

B.MongoDB is column oriented data store

C.MongoDB uses XML more in comparison with JSON

D.None of the above

8) The concatenation of the collection name and database name is called .....

A. Replica

B.Sharding

C.MongoDB

D.Namespace

9) What kind of database MongoDB is?

A. NoSQL Database

B.SQL Database

C.Operational database

D.None of the above

10) Does MongoDB supports query joins between collections ?

A. No

B.Yes

11) Mongodb supports search by field range queries regular expression searches.

A. True

B.False

12) Which among following is not a supported index type in mongodb?

A. Unique

B.TTL Index

C.Neospatial

D.None of the above

13) Which of the following format is supported by mongodb?

A. SQL

B.BSON

C.XML

D.None of the above

14) Which of the following language is mongodb written in?

A. C

B.C++

C.Javascript

D.All of the Above

15) _____ member is used to support dedicated functions, such as backup or reporting.

A. Primary

B.Hidden

C.ViewState

D.None of the above

16) In mongodb ..... sorting is not supported.

A. Heap

B.Collation

C.Collection

D.None of the above

17) mongodb is an example of _____.

A. Graph Database

B.Key-Value pair Database

C.Column-based Database

D.Document-based Database

18) MongoDB stores all documents in .....

A. Rows

B.Tables

C.Collections

D.None of the above

19) Secondary indices are not available in MongoDB.

A. True

B.False

20) MongoDB supports fixed-size collections called _____ collections.

A. Capped

B.Primary

C.Secondary

D.None of the above

21) Which of the following is true about sharding?

A. Sharding is enabled at the database level

B.We cannot change a shard key directly/automatically once it is set up

C.Creating a sharded key automatically creates an index on the collection using that key

D.All of the above

22) MongoDB has been adopted as _____ software by a number of major websites and services.

A. backend

B.frontend

C.proprietary

D.None of the above

23) Which of the following method is used to query documents in collections?

A. findOne()

B.selectOne()

C.findOne1()

D.None of the above

24) Which of the following is true about mongoDB?

A. MongoDB is a cross-platform

B.MongoDB provides high performance

C.MongoDB is a document oriented database

D.All of the above

25) Which of the following is a metapackage for enterprise?

A. mongodb-enterprise

B.mongodb-enterprise-server

C.mongodb-enterprise-mongos

D.None of the above

26) Mongodb stores all documents in

A. Tables

B.Files

C.Collections

D.Database

27. MongoDB Queries can return specific fields of documents which also include user-defined _____ functions.

A. C

B. C++

C. Javascript

D. All of the mentioned

28. Dynamic schema in MongoDB makes _____ easier for applications.

A. polymorphism

B. inheritance

C. encapsulation

D. None of the mentioned

29. With _____ MongoDB supports a complete backup solution and full deployment monitoring.

A. AMS

B. DMS

C. MMS

D. CMS

30. MongoDB provides high _____ with replica sets.

A. availability

B. performance

C. scalability

D. none of the mentioned

31. MongoDB scales horizontally using _____ for load balancing purpose.

A. Partitioning

B. Sharding

C. Replication

D. None of the mentioned

32. MongoDB supports fixed-size collections called _____ collections.

A. capped

B. secondary

C. primary

D. all of the mentioned

33. Which of the following sorting is not supported by MongoDB?

A. collection

B. heap

C. collation

D. none of the mentioned

34. Which of the following is not a NoSQL database?

A. Cassandra

B. MongoDB

C. SQL Server

D. None of the mentioned

35. Which of the following is a NoSQL Database Type

A. SQL

B. JSON

C. Document databases

D. All of the mentioned

36. Which of the following is a wide-column store?

A. Riak

B. Redis

C. Cassandra

D. MongoDB

37. Which of the following are the simplest NoSQL databases?

A. Wide-column

B. Document

C. Key-value

D. All of the mentioned

38. NoSQL databases is used mainly for handling large volumes of _____ data.

A. structured

B. semi-structured

C. unstructured

D. all of the mentioned

39. What is the interactive shell for MongoDB called?

A. mongo

B. dbmong

C. mongodb

D. none of the mentioned

40. Which of the following is web-based client software for MongoDB?

A. Database Master

B. BI Studio

C. Fang of Mongo

D. Mongo3

41. _____ provides statistics on the per-collection level.

A. mongotop

B. mongosniff

C. mongofiles

D. mongooplog

42. Which of the following network analyzer fully supports MongoDB?

A. Riakshark

B. Snort

C. Wireshark

D. Suricata

43. What is the maximum size of a MongoDB document?

A. 2 MB

B. 16 MB

C. 12 MB

D. There is no maximum size. It depends on the RAM.

44. Which of the following is a metapackage for enterprise?

A. mongodb-enterprise

B. mongodb-enterprise-mongos

C. mongodb-enterprise-server

D. one of the mentioned

45. Which of the following is not a part of mongodb-enterprise-tools?

A. mongotop

B. mongodown

C. mongodump

D. none of the mentioned

46. MongoDB only provides Enterprise packages for Ubuntu _____ LTS.

A. 12.10

B. 12.04

C. 13.04

D. 13.10

47.) Which of the following is the Ubuntu package management tool?

A. capt

B. wat

C. wapt

D. dpkg

48. mongod process is stopped by issuing which of the following command?

A. sudo service mongod restart

B. sudo service mongod start

C. sudo service mongod stop

D. none of the mentioned

49. _____ is a diagnostic tool for inspecting BSON files.

A. bsondump

B. bsondumpjson

C. jsondump

D. all of the mentioned

50.) mongo looks for a database server listening on port 27017 on the _____ interface.

A. web

B. web host

C. localhost

D. all of the mentioned

51.) _____ command display the list of databases.

A. display dbs

B. show db

C. show dbs

D. show data

52. Which of the following operation is used to switch to new database mydb?

A. use db

B. use mydbs

C. use dbs

D. use mydb

53. Which of the following also returns a list of databases?

A. show database

B. display dbs

C. show databases

D. all of the mentioned

54. Which of the following method is used to query documents in collections?

A. replace

B. shell

C. move

D. find

55. Point out the wrong statement.

A. A single MongoDB server typically has single databases

B. Database is a physical container for collections

C. Collection is the equivalent of an RDBMS table

D. None of the mentioned

56. MongoDB stores all documents in _____

A. collections

B. rows

C. tables

D. All of the mentioned

57. Which of the following pipeline is used for aggregation in MongoDB?

A. information processing

B. knowledge processing

C. data processing

D. None of the mentioned

58. In aggregation pipeline, the _____ pipeline stage provides access to MongoDB queries.

A. $match

B. $batch

C. $catch

D. All of the mentioned

59. Which of the following functionality is used for aggregation framework?

A. $projectmatch

B. $project

C. $match

D. All of the mentioned

60. Which of the following is not a projection operator?

A. $elemMatch

B. $slice

C. $

D. None of the mentioned

61 Which of the following flag can be set by mongo shell?

A. noTimeout

B. Time

C. Timeout

D. None of the mentioned

62 A query _____ consists of a combination of query, sort, and projection specifications.

A. shape

B. stats

C. plan

D. All of the mentioned

63. Which of the following is wrong statement -

A. MongoDB supports search by field, range queries, regular expression searches

B. MongoDB can store the business subject in the minimal number of documents

C. Secondary indices is not available in MongoDB

D. All of the above

64. A collection and a document in MongoDB is equivalent to..... concepts respectively.

A. Table and Column

B. Table and Row

C. Column and Row

D. Database and Table

65. in how much time the MongDB writes are written to the journal?

A. 100 s

B. 60 s

C. 1 s

D. 100 ms

66. Does MongoDB supports query joins between collections ?

A. Yes

B. No

67 what is MongoDB?

A. data growth

B. document database

C. adminCommand

D. Combine objects

68. The concatenation of the collection name and database name is called a -

A. Namespace
B. MongoDB
C. sharding
D. replica

69. What is the good alternatives to MongoDB?

A. Redis & CouchDB
B. Cassandra
C. Riak & Hbase
D. All of the mentioned

70. Which statements is correct about mongoose in MongoDB ?

A. it is Java library to connect with MongoDB
B. It is used for modeling your application data in node.js
C. It is Python library to connect with MongoDB
D. It is a PHP library to connect with MongoDB

71. Hidden member is used to support dedicated functions, such as backup or reporting.

A. True
B. False

72. A collection in MongoDB is a group of .....

A. Databases
B. Related documents
C. Schema
D. Rows

73. Which is not a supported index type in MongoDB?

A. TTL Index
B. Neospatial
C. Unique
D. None of These

74. Which of the following is correct option ?

A. MongoDB uses XML more in comparison with JSON
B. MongoDB is column oriented data store
C. MongoDB is a NoSQL database
D. None of the above

75. Is MongoDB better than other NoSQL databases?

A. Yes
B. No

76. In which format MongoDB represents document structure?

A. BSON
B. .txt

C. .Docx

D. None of these

77. Howmany byte counter in BSON is starting with a random value ?

A. 4

B. 2

C. 3

D. None of the above

78. which field is always the first field in the document.?

A _id

B Ob_id

C id

D None of these

79. The application, that communicates with application MongoDB by way of a client library, is called _____ .

A. Parent

B. Driver

C. Rank

D. None of the above

80. _____sorting is not supported by MongoDB.

A. collection
B. collation
C. heap
D. none of the mentioned

81. When a relational expression is false, it has the value _____.

A. zero
B. one

82. Which of the following is not a feature for NoSQL databases?
A. Data can be easily held across multiple servers
B. Relational Data
C. Scalability
D. Faster data access than SQL databases

83. Which of the following statement is correct with respect to mongoDB?

A. MongoDB is a NoSQL Database
B. MongoDB used XML over JSON for data exchange
C. MongoDB is not scalable
D. All of the above

84. Which of the following is/are feature of mongoDB?

A. Vertically scalable
B. Centralised database
C. Dynamic schema is/are feature of mongoDB.
D. All of the above

85. Which of the following represent column in mongoDB?

A. document
B. database
C. collection
D. field

86. The system generated _id field is?

A. A 12 byte hexadecimal value
B. A 16 byte octal value
C. A 12 byte decimal value
D. A 10 bytes binary value

87. Which of the following true about mongoDB?

A. MongoDB is a cross-platform
B. MongoDB is a document oriented database
C. MongoDB provides high performance
D. All of the above

88. Collection is a group of MongoDB _____?

A. Database
B. Document
C. Field
D. None of the above

89. A developer want to develop a database for LFC system where the data stored is mostly in similar manner. Which database should use?

A. Relational
B. NoSQL
C. Both A and B can be used
D. None of the above

90. Documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data is known as ?

A. dynamic schema
B. mongod
C. mongo
D. Embedded Documents

91. Instead of Primary Key mongoDB use?

A. Embedded Documents
B. Default key _id
C. mongod
D. mongo

92. _____ are operations that process data records and return computed results.
a) ReplicaAgg
b) SumCalculation
c) Aggregations
d) None of the mentioned

93. Point out the wrong statement.
a) Map-reduce cannot have a finalize stage to make final modifications to the result
b) Map-reduce is less efficient and more complex than the aggregation pipeline
c) Specifically, a user with the userAdmin role can grant itself any privilege in the database
d) All of the mentioned

94. Running data aggregation on the _____ instance simplifies application code and limits resource requirements.
a) document

b) mongod
c) mongos
d) all of the mentioned

95. A _____ set is a group of mongod instances that host the same data set.
a) copy
b) sorted
c) radii
d) replica

96Point out the wrong statement.
a) Replication provides redundancy and increases data availability
b) Replication allows you to recover from hardware failure and service interruptions
c) With multiple copies of data on different database servers, replication protects a database from the loss of a single server
d) None of the mentioned

97. All other instances, secondaries, apply operations from the _____ so that they have the same data set.
a) center
b) secondary
c) primary
d) none of the mentioned

98. A replica set can have only _____ primary.
a) One
b) Two
c) Three
d) All of the mentioned

99. Point out the wrong statement.
a) In all cases, you can use replication to increase read capacity
b) Clients have the ability to send read and write operations to different servers
c) You can also maintain copies in different data centers to increase the locality and availability of data for distributed applications
d) None of the mentioned

100. To support replication, the primary records all changes to its data sets in its _____
a) oplog
b) adlog
c) log
d) none of the mentioned

1	d	41	A
2	a	42	C
3	D	43	B
4	b	44	A
5	D	45	B
6	D	46	B
7	A	47	B
8	D	48	C
9	A	49	A
10	B	50	c
11	A	51	c
12	C	52	D
13	B	53	C
14	D	54	D
15	B	55	A
16	B	56	A
17	D	57	C
18	C	58	A
19	B	59	b
20	A	60	D

21	B	61	A
22	A	62	A
23	A	63	C
24	D	64	B
25	A	65	D
26	C	66	B
27	C	67	b
28	A	68	A
29	C	69	D
30	A	70	B
31	B	71	A
32	a	72	B
33	C	73	B
34	C	74	c
35	C	75	A
36	C	76	A
37	C	77	C
38	C	78	A
39	A	79	B
40	A	80	B
		81	A
		82	B
		83	A
		84	C
		85	D
		86	A
		87	B
		88	B
		89	B
		90	A
		91	B
		92	C
		93	A
		94	B
		95	D
		96	D

		97	C
		98	A
		99	A
		100	A

Assignment 1

1. Create a database employee with empno,name, designation, salary, incentive, sales, working hours

2. Show all the employees having designation manager

3. Show all the employees having salary 6000

4. Give incentive to employees having working hours greater than 9

5. Add 100 to incentive of employees doing sales greater than 1000.

6. Add 1000 to salary employee whose designation accountant.

7 create index on field empno

8.create compound index on fields empno, name

9.find sum of salaries of employee having designation clerk.

10.filter the employees having designation software engineer and find the minimum salary .

Assignment 2

create a database NIELIT.create collection student with rollno,name,class,section,marks. Input marks as embedded document like marks:{eng:45,hindi:67,science:90},grades .

1. calculate the total marks of all subjects of each student.
2. sort the records based on name in ascending order
3. filter the records based on subject English
4. increment the marks by 5 for students having total marks less than 150
5. show only two fields name,rollno

Assignment 3. Download movies.json file

Write and run the following queries:

1. Show the movies titled "Gladiator".
2. Show distinct genre values of movies.
3. Show the movies of "crime" or "drama" genre.
4. Show the list of movies directed by "Hitchcock",display only title and year and sort them by year.
5. Show the list of movies where "Cotillard" played.
6. Show the movies released between 1967 and 1995.
7. Show the list of the movies released between 1967 and 1995, by displaying only title, year, director's last name sorted by year.
8. Show the number of movies by country.
9. Show the number of movies by country and actor.

Assignment 4

Create a collection called 'games'. Add games to the database. Give each document the following properties:

name, genre, rating (out of 100)

.

1. Write a query that returns all the games
2. Write a query to find one of your games by name without using limit(). Use the findOne method.
3. Write a query that returns the 3 highest rated games.

4. Update your two favourite games to have two achievements called ‘Game Master’ and ‘Speed Demon’, each under a single key. Show two ways to do this. Do the first using update() and do the second using save().
5. Write a query that returns all the games that have both the ‘Game Maser’ and the ‘Speed Demon’ achievements.
6. Write a query that returns only games that have achievements. Not all of your games should have achievements.

Assignment 5

Download the restaurant.json file.import the json file

1. Write a MongoDB query to display all the documents in the collection restaurants.
2. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.
3. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine, but exclude the field _id for all the documents in the collection restaurant.
4. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.
5. Write a MongoDB query to display all the restaurant which is in the borough Bronx.
6. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.

7. Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx.

8. Write a MongoDB query to find the restaurants who achieved a score more than 90.

9. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

10. Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.

11. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.

12. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168.

13. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' not belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order.

14. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.

15. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronxor Brooklyn.

16. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or

Queens or Bronxor Brooklyn.

17. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10.

18. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinese'

19. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates..

20. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".

21. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52..

22. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

23. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

24. Write a MongoDB query to arrange the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

25. Write a MongoDB query to know whether all the addresses contains the street or not.