Database Systems Lab

Lab # 11

MongoDB Introduction

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2005

Department of Computer Science,

National University of Computer and Emerging Sciences FAST
Peshawar Campus

Contents

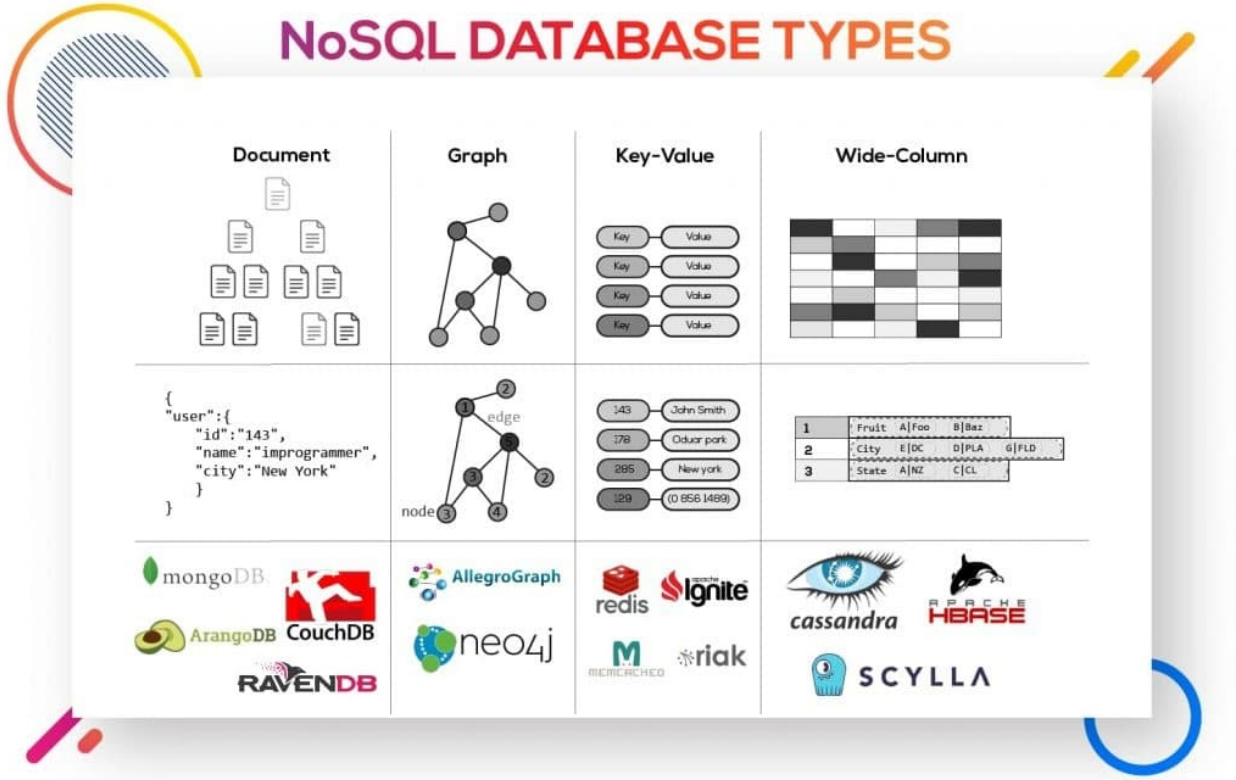
NoSQL:

NoSQL databases also known as "**not only SQL**" are **non-tabular**, and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.

NoSQL is a popular alternative to relational databases. NoSQL database does not have predefined schemas, which makes NoSQL databases a perfect candidate for rapidly changing development environments.

What are the Types of NoSQL Databases?

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



Differences between SQL and NoSQL

NoSQL (“non SQL” or “not only SQL”) databases were developed in the late 2000s with a focus on scaling, fast queries, allowing for frequent application changes, and making programming simpler for developers. Relational databases accessed with SQL (Structured Query Language) were developed in the 1970s with a focus on reducing data duplication as storage was much more costly than developer time. SQL databases tend to have rigid, complex, tabular schemas and typically require expensive vertical scaling.

The table below summarizes the main differences between SQL and NoSQL databases.

	SQL Databases	NoSQL Databases
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Development History	Developed in the 1970s with a focus on reducing data duplication	Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices.
Examples	Oracle, MySQL, Microsoft SQL Server, and PostgreSQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune
Schemas	Rigid	Flexible
Scaling	Vertical (scale-up with a larger server)	Horizontal (scale-out across commodity servers)
Joins	Typically required	Typically not required

What are the Benefits of NoSQL Databases?

NoSQL databases offer many benefits over relational databases. NoSQL databases have flexible data models, scale horizontally, have incredibly fast queries, and are easy for developers to work with.

- *Flexible data models*

NoSQL databases typically have very flexible schemas. A flexible schema allows you to easily make changes to your database as requirements change. You can iterate quickly and continuously integrate new application features to provide value to your users faster.

- *Horizontal scaling*

Most SQL databases require you to scale-up vertically (migrate to a larger, more expensive server) when you exceed the capacity requirements of your current server. Conversely, most NoSQL databases allow you to scale-out horizontally, meaning you can add cheaper, commodity servers whenever you need to.

- *Fast queries*

Queries in NoSQL databases can be faster than SQL databases. Why? Data in SQL databases is typically normalized, so queries for a single object or entity require you to join data from multiple tables. As your tables grow in size, the joins can become expensive. However, data in NoSQL databases is typically stored in a way that is optimized for queries. The rule of thumb when you use MongoDB is Data is that is accessed together should be stored together. Queries typically do not require joins, so the queries are very fast.

- Easy for developers

Some NoSQL databases like MongoDB map their data structures to those of popular programming languages. This mapping allows developers to store their data in the same way that they use it in their application code. While it may seem like a trivial advantage, this mapping can allow developers to write less code, leading to faster development time and fewer bugs.

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.

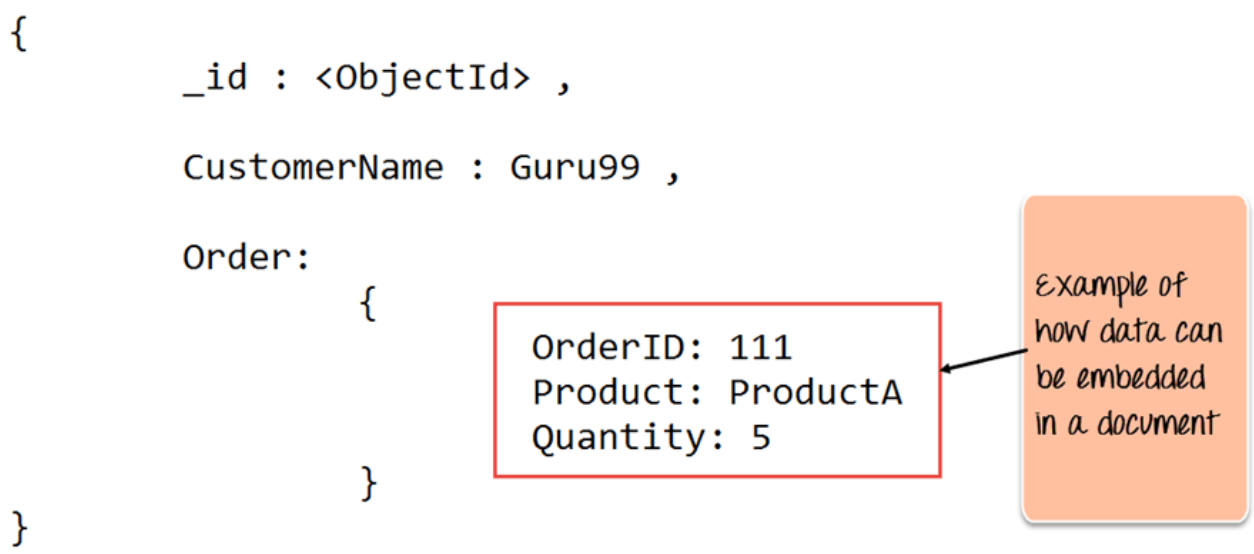
MongoDB Features

1. 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.
2. The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
3. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

MongoDB Example

The below example shows how a document can be modeled in MongoDB.

1. The _id field is added by MongoDB to uniquely identify the document in the collection.
2. What you can note is that 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.



Key Components of MongoDB Architecture

Below are a few of the common terms used in MongoDB

1. **_id** – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

_Id	CustomerID	CustomerName	OrderID

563479cc8a8a4246bd27d784	11	Guru99	111
563479cc7a8a4246bd47d784	22	Trevor Smith	222
563479cc9a8a4246bd57d784	33	Nicole	333

2. **Database** – This is a container for collections like in RDMS 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.
3. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS 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.
4. **Document** - A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
5. **Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

   The following diagram shows an example of Fields with Key value pairs. So in the example below age and 26 is one of the key value pair's defined in the document.

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)

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.

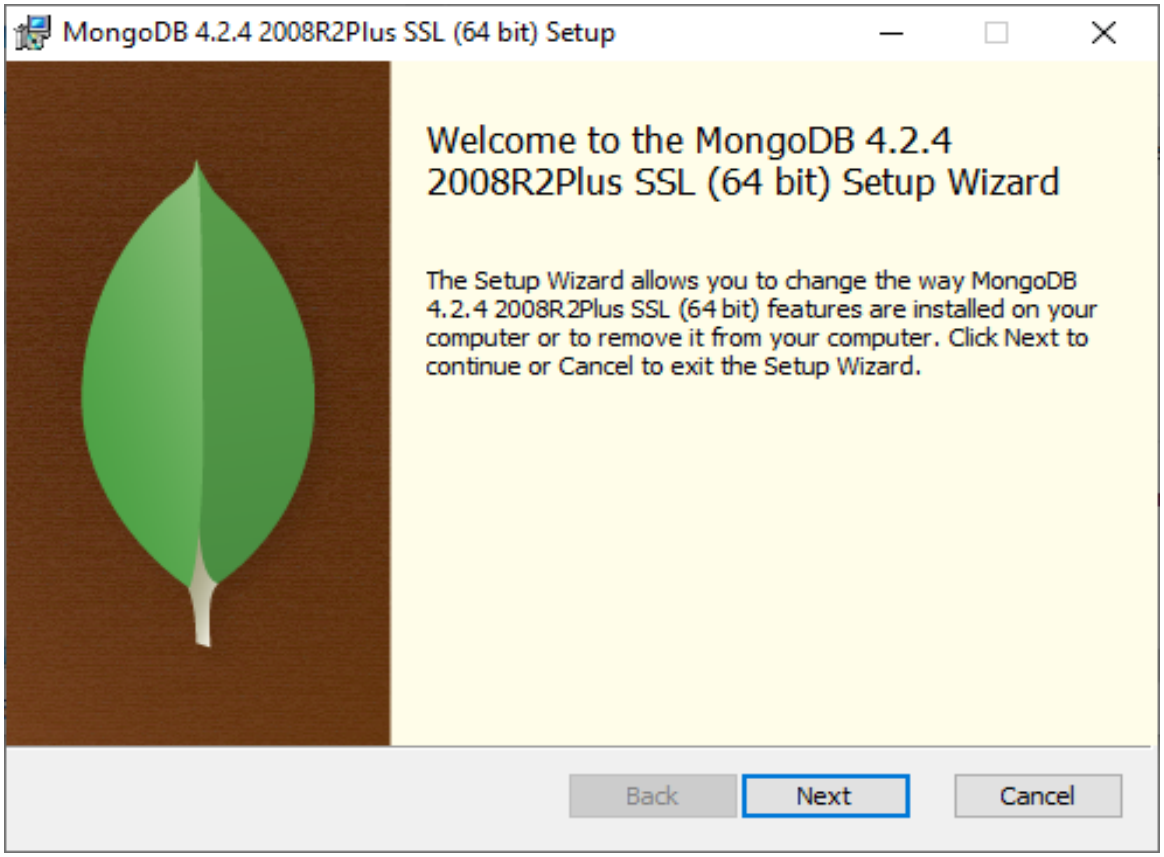
Installing Mongodb on windows

Downloading

Download from web: https://www.mongodb.com/download-center/community

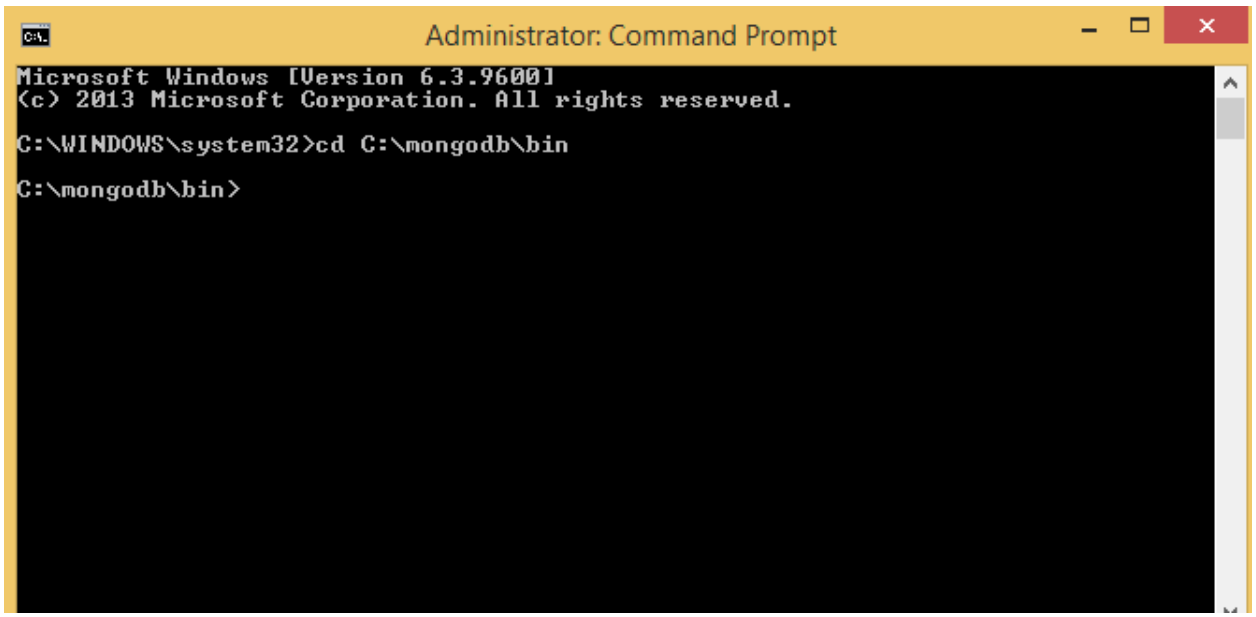
Running the setup

Double click the downloaded .msi to install the setup **[while installing chose custom option and change its location to a simpler path e.g C:\mongodb]**



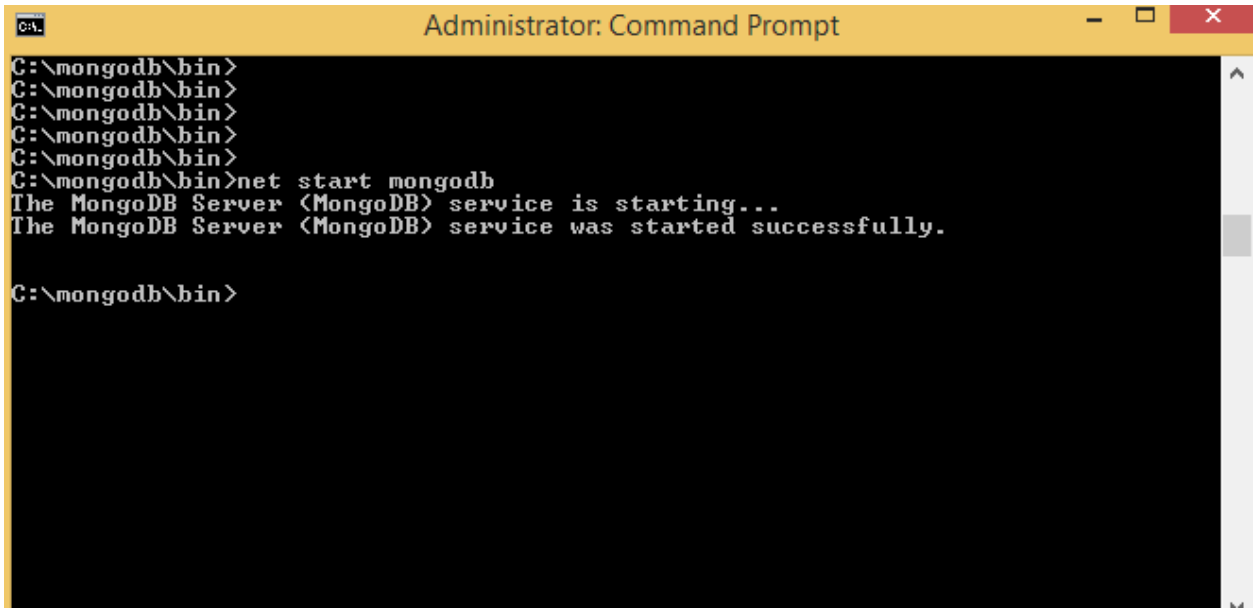
After finishing the setup. Open command prompt as an administrator

Navigate to mongo directory, into bin folder



After that to run services, run the following command:

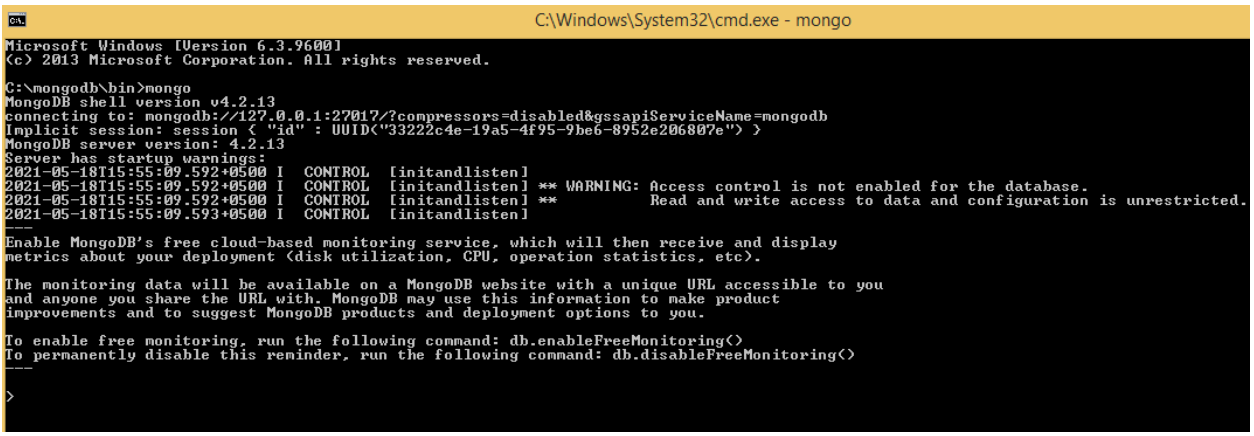
```
net start mongodb
```



Now the services are running in background

Running a mongo shell

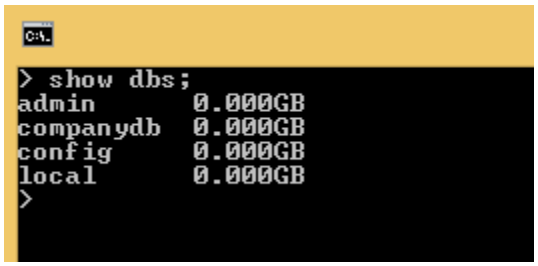
After running the services successfully we will be working on mongo shell. Open terminal and go to mongodb->bin folder and type **mongo**



To clear the screen just type “cls”, it will clear the screen

Show databases:

To see the databases, write **show dbs**



Create databases:

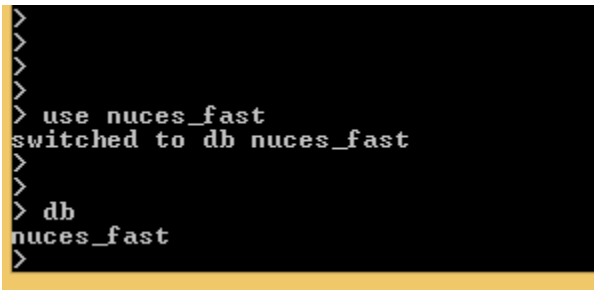
The following command is used to create a new database.

Syntax:

Use Database_name

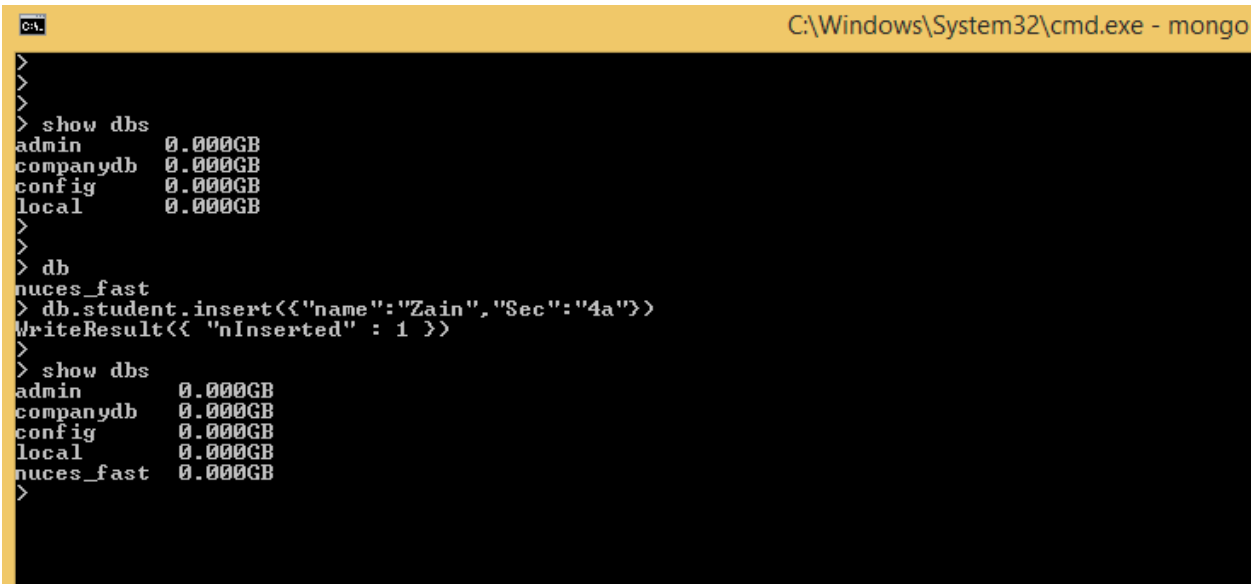
Let' take an example to demonstrate how a database is created in MongoDB. In the following example, we are going to create a database "nuces_fast".

To check the currently selected database, use the command db:



To **check the database list**, use the command show dbs:

Here, your created database "nuces_fast" is not present in the list, **insert at least one document** into it to display database:



Basic syntax for the query's. It's like json start and end with curly braces

```
{

    first_name: "Muhammad",
    last_name: "Ali",
    membership: ["abc",xyz], //can add arrays
    address:        //objects
    {

    country: "pakistan",
    city:"peshawar"
    }
}
```

Create Collections:

They are similar to tables in a relational database. Basically the hold documents or the records. To create one type

MongoDB stores BSON documents. (BSON is a binary representation of JSON documents, though it contains more data types than JSON)

Now Lets add data. But before that make a collection first

db.createCollection('collectionname')

To see all the collections in database write "**show collections**"

```
> db.createCollection("courses")
{ "ok" : 1 }
> show collections
courses
student
>
```

Insert documents:

Insert document into collection

Single document

db.customer_info.insert({write data in json format as explained earlier})

```
> db.student.insert({
... first_name:"Muhammad",
... last_name:"Ali"
... })
WriteResult({ "nInserted" : 1 })
>
```

Multiple records

Terminal Help • // To insert Multiple data at once use a • Unt

// To insert Multiple data at once use a Untitled-2 ●

```
1   // To insert Multiple data at once use arrays
2
3   db.student.insert([
4   {
5    first_name:"Khuram",
6    last_name:"Shehzad"
7   },
8   {
9    first_name:"Adnan",
10   last_name:"Hasan"
11  },
12  {
13   first_name:"Salman",
14   last_name:"Abid"
15  },
16  ])
```

```
>
>
>
> db.student.insert([
... {
...  first_name:"Khuram",
...  last_name:"Shehzad"
... },
... {
...  first_name:"Adnan",
...  last_name:"Hasan"
... },
... {
...  first_name:"Salman",
...  last_name:"Abid"
... },
... ])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
>
```

View records

To see the documents in collection “student”

db.student.find()

```
> db.student.find()
{ "_id" : ObjectId("60a4aeaf648598db366e5b1b"), "name" : "Zain", "Sec" : "4a" }
{ "_id" : ObjectId("60a4badb648598db366e5b1c"), "first_name" : "Muhammad", "last_name" : "Ali" }
{ "_id" : ObjectId("60a4be6d648598db366e5b1d"), "first_name" : "Khuram", "last_name" : "Shehzad" }
{ "_id" : ObjectId("60a4be6d648598db366e5b1e"), "first_name" : "Adnan", "last_name" : "Hasan" }
{ "_id" : ObjectId("60a4be6d648598db366e5b1f"), "first_name" : "Salman", "last_name" : "Abid" }
>
```

The data has been inserted. Here you can see the id object. It’s a unique id generated automatically now we don’t have to worry about primary key etc.

By default, db.collection.find() returns data in a dense format. By using db.collection.pretty() you can set the cursor to return data in a format that is easier for humans to parse:

db.student.find().pretty()

Installing Mongodb on Ubuntu

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

To install MongoDB Compass

1. Download MongoDB Compass

   wget https://downloads.mongodb.com/compass/mongodb-compass_1.31.1_amd64.deb

2. Install MongoDB Compass

   sudo dpkg -i mongodb-compass_1.31.1_amd64.deb

3. Start MongoDB Compass

   mongodb-compass

MongoDB Query Document using find()

The method of fetching or getting data from a MongoDB database is carried out by using queries. While performing a query operation, one can also use criteria's or conditions which can be used to retrieve specific data from the database.

MongoDB provides a function called db.collection.find () which is used for retrieval of documents from a MongoDB database.

During the course of this lab, you will see how this function is used in various ways to achieve the purpose of document retrieval.

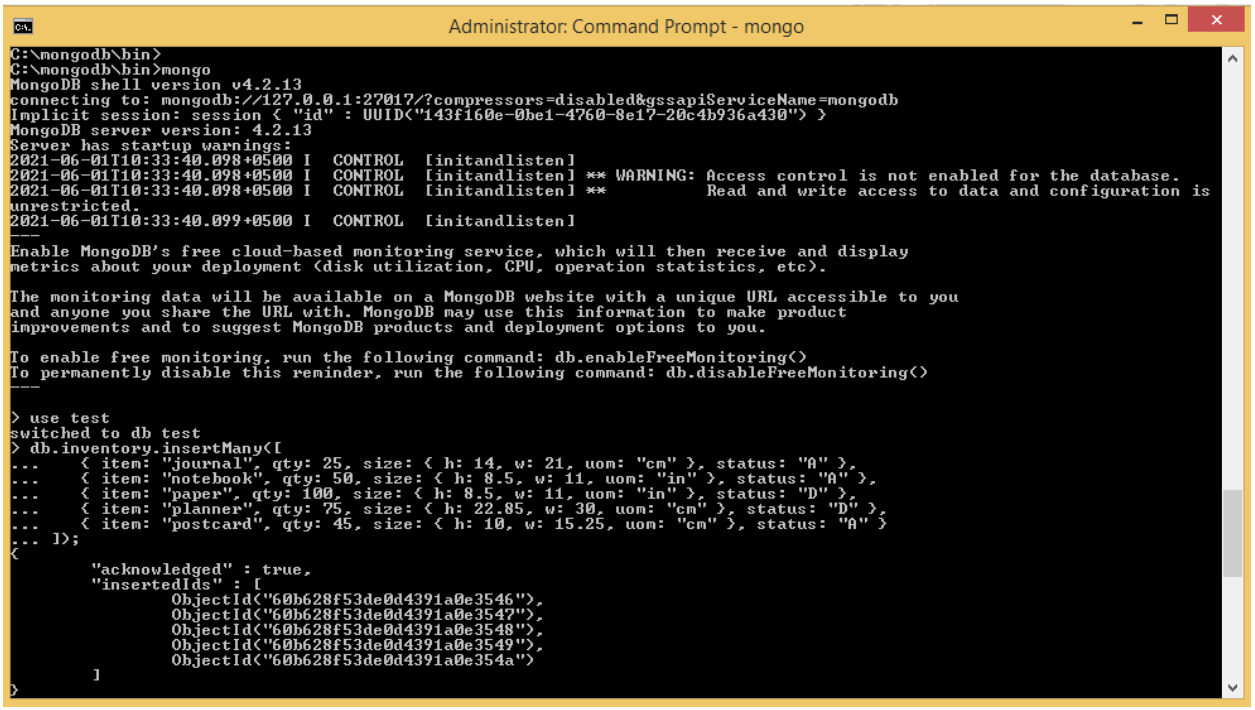
Consider that we have a collection named 'inventory' in our MongoDB database

To populate the inventory collection, run the following:

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

```
]);
```

You can run the operation in the web shell below:



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.inventory.find( {} )
```

This operation corresponds to the following SQL statement:

```
SELECT * FROM inventory
```

For more information on the syntax of the method

```
db.collection.find(query, projection)
```

Parameter	Type	Description
`query`	document	Optional. Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document (`{}`).
`projection`	document	Optional. Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter. For details, see Projection.

Specify Equality Condition

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

```
{ <field1>: <value1>, ... }
```

The following example selects from the inventory collection all documents where the status equals "D":

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

This operation corresponds to the following SQL statement:

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

Specify Conditions Using Query Operators

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```

The following example retrieves all documents from the inventory collection where status equals either "A" or "D":

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

The operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status in ("A", "D")
```

For More Details

https://www.mongodb.com/nosql-explained

https://www.mongodb.com/scale/types-of-nosql-databases