

Database Management Systems

MongoDB

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
and
Centre for Artificial Intelligence and Machine Learning
Indian Statistical Institute, Kolkata

May, 2022

1 Basic Features

2 Data Definition

- Data Types
- Database Creation
- Database Deletion

3 Data Manipulation

4 Data View in MongoDB

MongoDB

MongoDB is a cross-platform open-source document-oriented database program that works on NoSQL principles. It uses JSON-like documents with schemata.

A MongoDB database is a physical container for collections, a collection is a group of documents, and a document is a set of key-value pairs. The documents within a collection can have different fields. Interestingly, the collections do not enforce a schema and documents have dynamic schema.

Note: MongoDB is written in C++.

Features of MongoDB

- **Indexing:** Fields in a document can be indexed with primary and secondary indices.
- **Replication:** It provides high availability with replica sets.
- **Load balancing:** It scales horizontally using sharding and can run over multiple servers.
- **File storage:** It can be used as a file system, called GridFS, with data replication and load balancing features.
- **Aggregation:** It can adopt aggregation pipeline, map-reduce function, and single-purpose aggregation methods.
- **Server-side JavaScript execution:** JavaScript can be used in queries, aggregation functions (such as MapReduce), and sent directly to the database for execution.
- **Capped collections:** It supports fixed-size collections.
- **Ad hoc queries:** It supports field, range query, and regular expression searches.

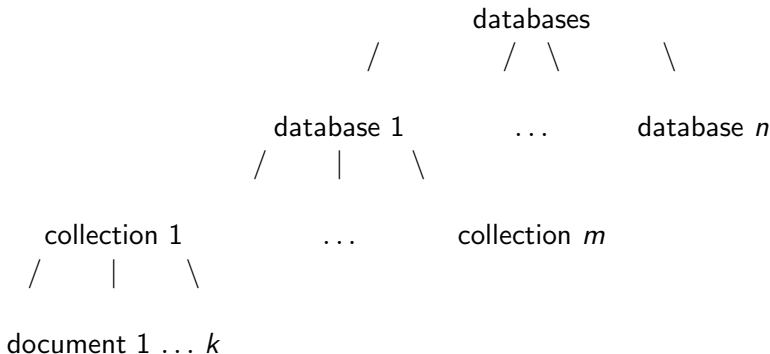
NoSQL functionalities in MongoDB

- **Data-definition language (DDL)** – provides commands for defining collections and documents, deleting collections, and modifying collections.
- **Data-manipulation language (DML)** – includes commands to work on documents, insert documents into, delete documents from, and modify documents in the database.
- **View definition** – includes commands for defining views.

Note: Support for multi-document ACID transactions has been added to MongoDB 4.0 (released in June 2018) and higher versions.

Data view through MongoDB

In practice, the databases (as a whole) comprises several separate database, each database consists of several collections, and each collection comprises several documents.



Installing MongoDB Community Server on Windows

- 1 Download the .msi version
- 2 Install anyway (not as a part of Microsoft-verified app)
- 3 Follow the Setup Wizard

Note: The latest version of MongoDB Shell is 5.0.8 (available for download at:

<https://www.mongodb.com/try/download/community>).

Installing MongoDB Compass on Windows

Compass is an interactive GUI for querying, optimizing, and analyzing MongoDB data.

- 1 Download the .zip file
- 2 Extract All files
- 3 Run MongoDBCompass

Note: The latest version of MongoDB Compass is 1.31.3 (available for download at: <https://www.mongodb.com/try/download/compass>).

Installing MongoDB Community Server and Compass together

The MongoDB Compass comes as an option (not selected by default) during the MongoDB Community Server installation so that both of them can be installed together.

Working with MongoDB Compass

MongoDB Compass - localhost:27017

Connect View Help

The screenshot shows the MongoDB Compass interface for localhost:27017. The 'Databases' tab is selected, and a 'Create database' button is highlighted with a red box. The interface displays a list of databases: admin, config, and local. Each database entry shows its storage size, the number of collections, and the number of indexes.

Database Name	Storage size	Collections	Indexes
admin	4.10 kB	1	1
config	4.10 kB	1	2
local	4.10 kB	1	1

Installing MongoDB on Linux

```
$ sudo apt-get install mongodb
[sudo] password for student:
$ sudo apt-get update
$ mongod --version /* MongoDB Server version */
db version v3.6.3
git version:
9586e557d54ef70f9ca4b43c26892cd55257e1a5
OpenSSL version:  OpenSSL 1.1.0g 2 Nov 2017
...
$ mongo -version /* MongoDB Shell version */
MongoDB shell version v3.6.3
...
```

Note: The latest version of MongoDB is 4.4 (manual available at: <https://docs.mongodb.com/manual/release-notes>).

Connecting with MongoDB

```
$ sudo service mongod start
$ mongo
connecting to:  mongodb://127.0.0.1:27017
...
> _ /* Within MongoDB Shell */
> show dbs; /* The databases present in MongoDB */
admin    0.000GB
config  0.000GB
local    0.000GB
> exit
bye
$ _ /* Out of MongoDB Shell */
$ sudo service mongod stop
```

Note: The default database is *local*.

Using a database

```
> use <database_name>
switched to db <database_name>
> _ /* Control is now on <database_name> */
> db
<database_name> /* The current database */
> show collections
<collection_name1>
<collection_name2>
...
system.indexes
> _
```

Note: The default collection is *system.indexes* (may remain hidden).

Data types in MongoDB

Type	Syntax	Details
Number	-	A signed decimal that may contain fractional part
String	"..."	A sequence of zero or more unicode characters
Boolean	true/false	A Boolean value
Array	A[..., ...]	An ordered list of zero or more elements of any type
Object	-	An unordered collection of key-value pairs where the keys are strings
null	null	An empty value

Note: A number cannot include non-numbers such as NaN.

Creating a database

```
> use <database_new>
switched to db <database_new>
> _ /* Control is now on <database_new> */
> db.<collection_new>.insert({ "Key" : "Value" });
> show dbs
admin                0.000GB
config               0.000GB
<database_new>      0.000GB /* unseen until insertion */
local                0.000GB
> _
```

Note: The *use* command creates a database if it does not exist.

Consider a JSON data

```
{
  "FirstName" : "Douglas",
  "LastName" : "Crockford",
  "Age": 64,
  "Address": {
    "State": "CA",
    "Country": "USA"
  },
  "PhoneNumbers": [
    {
      "Type": "internal",
      "Number": "1-888-221-1161"
    },
    {
      "Type": "external",
      "Number": "1-402-935-2050"
    }
  ]
}
```


Creating a collection

```
> db.<collection_name>.insert({ "FirstName" :  
"Douglas", "LastName" : "Crockford", "Age": 64,  
"Address": { "State": "CA", ... }, "PhoneNumbers":  
[ { ... }, { ... } ] })  
> show collections  
<collection_name>  
system.indexes  
> _
```

Note: Inserting a document in the collection and creating that collection can be done simultaneously.

Deleting a database

```
> use <database_new>
switched to db <database_new>
> db.dropDatabase()
> show dbs
admin                0.000GB
config               0.000GB
local                0.000GB
> _
```

Note: Alternatively, it is possible to stop MongoDB, delete the data files from the data directory, and then restart.

Deleting a collection

Syntax:

```
db. < collection_name > .drop()
```

```
> db.<collection_name>.drop()  
> show collections  
> _
```

Note: If the Collection is deleted successfully then 'true' is echoed back as acknowledgement, else 'false' would be echoed back.

Deleting all documents within a collection

Syntax:

```
db. < collection_name > .remove({ < query > })
```

```
> db.<collection_name>.remove({ })  
> show collections  
> <collection_name>  
> _
```

Note: An empty <query> will remove all the documents within a collection. The `remove()` method cannot be used on a capped collection.

Deleting selected documents within a collection

Syntax:

```
db. < collection_name > .remove({ < query >, < justOne > })
```

```
> db.<collection_name>.remove({ <query>, true })  
> show collections  
> <collection_name>  
> _
```

Note: To limit the deletion to just one document, set the value of <justOne> to **true**, otherwise keep the default value **false**.

Selecting all documents

Syntax:

```
db. < collection_name > .find({ < query > })
```

```
> db.<collection_name>.find({ })
> { "_id" : ObjectId("<Id_Number>"), "FirstName" :
  "Douglas", "LastName" : "Crockford", ... }
> db.<collection_name>.find({ }).pretty()
> {
  "_id" : ObjectId("<Id_Number>"),
  "FirstName" : "Douglas",
  "LastName" : "Crockford",
  ...
} /* Structured (pretty) output */
```

Note: An empty <query> will retrieve all the documents from a collection.

Selecting documents based on equality

Syntax:

```
db. < collection_name > .find({ < query > })
```

```
> db.<collection_name>.find({ "Age" : 64 })  
> { "_id" : ObjectId("<Id_Number>"), "FirstName" :  
"Douglas", "LastName" : "Crockford", "Age" : 64,  
... }
```

Note: This is similar to the SQL query “select * from <collection_name> where Age = 64;”.

Writing complex queries

Let us consider the following document entries in a MongoDB collection:

```
{ "_id" : ObjectId("<Id_Number>"), "Name" : "ISI",  
  "Description" : "Research institute", "YoE" :  
  "1931" }  
{ "_id" : ObjectId("<Id_Number>"), "Name" : "JU",  
  "Description" : "University", "YoE" : "1955" }  
{ "_id" : ObjectId("<Id_Number>"), "Name" :  
  "IITKGP", "Description" : "Engineering institute",  
  "YoE" : "1951" }  
{ "_id" : ObjectId("<Id_Number>"), "Name" :  
  "IIMCAL", "Description" : "Management institute",  
  "YoE" : "1961" }
```


Using regular expressions – \$regex

```
> db.<collection_name>.find({ "Description" : {
$regex : /institute$/ } }) /* SQL LIKE match */
> { "_id" : ObjectId("<Id_Number>"), "Name" : "ISI",
"Description" : "Research institute", "YoE" :
"1931" }
{ "_id" : ObjectId("<Id_Number>"), "Name" :
"IITKGP", "Description" : "Engineering institute",
"YoE" : "1951" }
{ "_id" : ObjectId("<Id_Number>"), "Name" :
"IIMCAL", "Description" : "Management institute",
"YoE" : "1961" }
```

Note: MongoDB uses Perl compatible regular expressions (i.e. PCRE) version 8.41 with UTF-8 support.

Using regular expressions – \$regex

```
> db.<collection_name>.find({ "Name": { $regex:
/^isi/i } }) /* Case-insensitive match */
> { "_id" : ObjectId("<IdNumber>"), "Name" : "ISI",
"Description" : "Research institute", "YoE" :
"1931" }
```

Using logical expressions – \$and

```

> db.<collection_name>.find({ $and: [{"Name": {
$regex: /^isi/i }}, {"Description" : { $regex :
/institute$/}}] })
> { "_id" : ObjectId("<Id Number>"), "Name" : "ISI",
"Description" : "Research institute", "YoE" :
"1931" }
> db.<collection_name>.find({ $and: [{"Name": {
$regex: /^II/ }}, {"Description" : "University"}]
})
>

```

Using logical expressions – \$or

```
> db.<collection_name>.find({ $or: [{"Name": {
$regex: /^II/ }}, {"Description" : "University"}]
})
> { "_id" : ObjectId("<Id_Number>"), "Name" : "JU",
"Description" : "University", "YoE" : "1955" }
{ "_id" : ObjectId("<Id_Number>"), "Name" :
"IITKGP", "Description" : "Engineering institute",
"YoE" : "1951" }
{ "_id" : ObjectId("<Id_Number>"), "Name" :
"IIMCAL", "Description" : "Management institute",
"YoE" : "1961" }
```

Using comparative expressions – \$gt, \$lt

```
> db.<collection_name>.find({ "YoE" : { $gt : 1960
} })
> { "_id" : ObjectId("<Id_Number>"), "Name" :
"IIMCAL", "Description" : "Management institute",
"YoE" : "1961" }
> db.<collection_name>.find({ "YoE" : { $lt : 1940
} })
> { "_id" : ObjectId("<Id_Number>"), "Name" : "ISI",
"Description" : "Research institute", "YoE" :
"1931" }
```

Aggregating collections – \$lookup

Consider the following collection “orders”:

```
db.orders.insert([
  { "_id" : 1, "item" : "mango", "price" : 50,
    "quantity" : 2 },
  { "_id" : 2, "item" : "orange", "price" : 20,
    "quantity" : 1 },
  { "_id" : 3 }
])
```

Aggregating collections – \$lookup

Consider another collection “inventory”:

```
db.inventory.insert([
  { "_id" : 1, "fruit" : "mango", description:
  "product 1", "instock" : 90 },
  { "_id" : 2, "fruit" : "guava", description:
  "product 2", "instock" : 10 },
  { "_id" : 3, "fruit" : "banana", description:
  "product 3", "instock" : 60 },
  { "_id" : 4, "fruit" : "orange", description:
  "product 4", "instock" : 70 },
  { "_id" : 5, "fruit": null, description:
  "Incomplete" }, { "_id" : 6 }
  { "_id" : 7, "fruit" : "banana", description:
  "product 3", "instock" : 10 },
])
```

Aggregating collections – \$lookup

The collections “orders” and “inventory” can be aggregated as shown below:

```
db.orders.aggregate([
{
$lookup:
{
from: "inventory",
localField: "item",
foreignField: "fruit",
as: "inventory_docs"
}
}
])
```


Aggregating collections – \$lookup

The output:

```
{
  "_id" : 1,
  "item" : "mango",
  "price" : 50,
  "quantity" : 2,
  "inventory_docs" : [
    { "_id" : 1, "fruit" : "mango", "description" :
      "product 1", "instock" : 90 }
  ]
}
{
  "_id" : 2,
  "item" : "orange",
  "price" : 20,
  ...
}
```

Creating views

```
> db.createView(<view>, <source>, <pipeline>,  
<options>)
```

Note: Views act as read-only collections, and are computed on demand during read operations.

More on MongoDB

Look into the MongoDB user manual:

<https://docs.mongodb.com/manual>