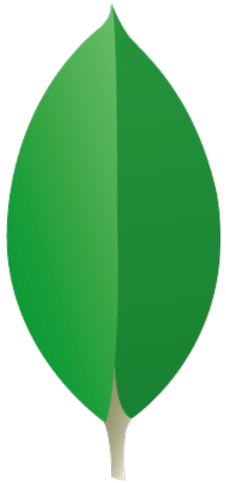


mongoDB

A Quick Start on NoSql Document Store

Altug Tanaltay - Sabancı University

Overview

- MongoDB is an open-source document database and leading NoSQL database written in C++.
- It is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- It works on concept of database collection and document.

Core Concepts - RDBMS Relation

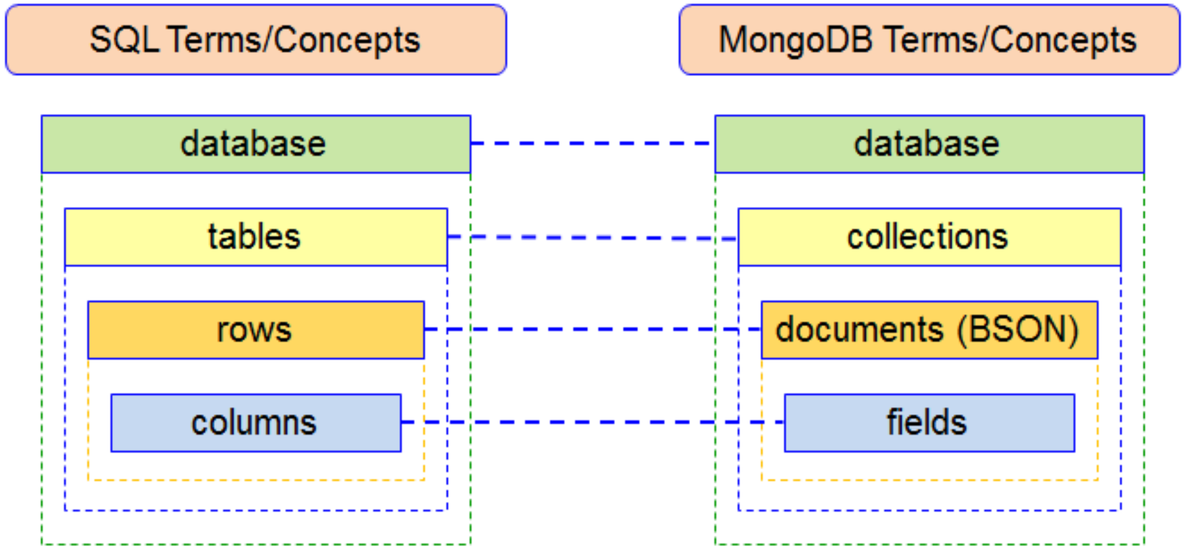
Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB

```
{
 "_id": ObjectId("....."),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```

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)



Advantages of MongoDB over RDBMS

- **Schema less** – MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- **Structure** of a single object i**s clear**.
- **No complex joins**.
- **Deep query-ability**. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- Tuning.
- Ease of scale-out – MongoDB is easy to scale.
- **Conversion/mapping of application objects** to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Documents to Java Class Relation

Documents of students collection:

MongoDB

```
{
 "_id": ObjectId("....."),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```

MongoDB Database

Student Java class:

```
class Student{
   int studentId
   String studentName;
   int age;
   String college;
}
```

Student instances:

```
new Student(1001,"Chaitanya",30,"Beginnersbook");
new Student(1002,"Steve",29,"Beginnersbook");
new Student(1003,"Megan",28,"Beginnersbook");
```

Spring Application

Why Use MongoDB?

- Document Oriented Storage – Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-Sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Docker Installation

- Use provided docker-compose.yml to download and run the official mongo image. (https://hub.docker.com/_/mongo)
- Save the compose file in a directory and create a sub directory named “data”.
- Access the directory with command prompt or terminal and call: docker-compose up
- Accesing the Mongo Shell using command prompt /b terminal:

```
docker exec -it [DOCKER CONTAINER NAME] /bin/sh
```

```
mongosh mongodb://admin:pass@localhost:27017/?authMechanism=DEFAULT
```

mongodb connection string

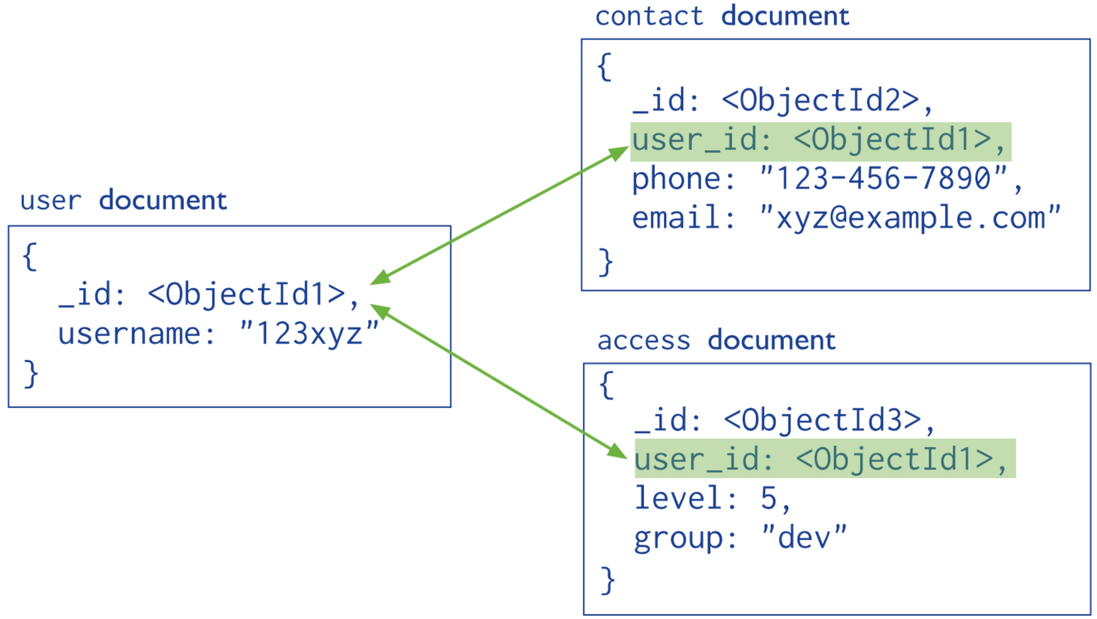
- You may use **Mongo Shell** or **MongoDB Compass** for database operations.
- For installation of Mongo Compass: https://www.mongodb.com/products/compass

Data Modeling

- MongoDB provides two types of data models: — Embedded data model and Normalized data model. Based on the requirement, you can use either of the models while preparing your document.

Embedded Model

Normalized Model

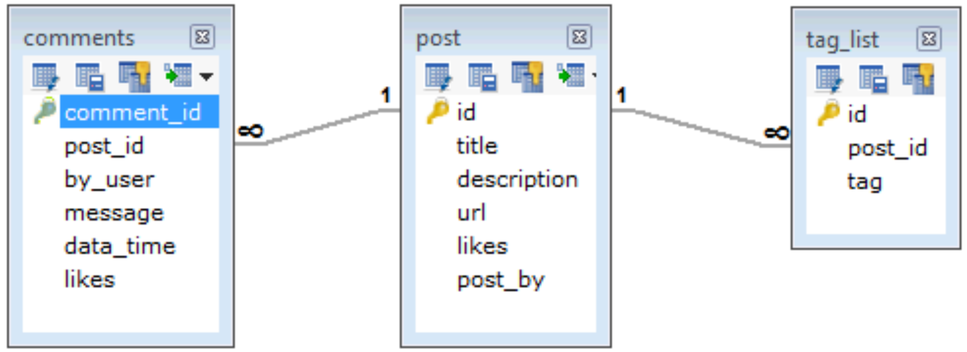


Data Modeling - Example

- Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.
  - Every post has the unique title, description and url.
  - Every post can have one or more tags.
  - Every post has the name of its publisher and total number of likes.
  - Every post has comments given by users along with their name, message, data-time and likes.
  - On each post, there can be zero or more comments.

mongoDB

RDBMS Approach



MongoDB Approach

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

Database Commands

Basic syntax of **use DATABASE** statement is as follows –

```
use DATABASE_NAME
```

```
>use mydb
switched to db mydb
```

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

```
>db
mydb
```

If you want to check your databases list, use the command **show dbs**.

```
>show dbs
local     0.78125GB
test      0.23012GB
```

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

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

Drop database

```
db.dropDatabase()
```

Basic syntax of **createCollection()** command is as follows –

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

```
db.COLLECTION_NAME.drop()
```

Insert Document

```
>db.COLLECTION_NAME.insert(document)
```

```
>db.COLLECTION_NAME.insertOne(document)
```

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

```
> db.empDetails.insertOne(
        {
                First_Name: "Radhika",
                Last_Name: "Sharma",
                Date_Of_Birth: "1995-09-26",
                e_mail: "radhika_sharma.123@gmail.com",
                phone: "9848022338"
        })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

```
> db.empDetails.insertMany(
        [
                {
                        First_Name: "Radhika",
                        Last_Name: "Sharma",
                        Date_Of_Birth: "1995-09-26",
                        e_mail: "radhika_sharma.123@gmail.com",
                        phone: "9000012345"
                },
                {
                        First_Name: "Rachel",
                        Last_Name: "Christopher",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Rachel_Christopher.123@gmail.com",
                        phone: "9000054321"
                },
                {
                        First_Name: "Fathima",
                        Last_Name: "Sheik",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Fathima_Sheik.123@gmail.com",
                        phone: "9000054321"
                }
        ]
)
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5dd631f270fb13eec3963bed"),
                ObjectId("5dd631f270fb13eec3963bee"),
                ObjectId("5dd631f270fb13eec3963bef")
        ]
}
>
```

Query Documents

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

```
>db.COLLECTION_NAME.find()
```

```
> db.mycol.find({title: "MongoDB Overview"})
{
  "_id" : ObjectId("5dd6542170fb13eec3963bf0"),
  "title" : "MongoDB Overview",
  "description" : "MongoDB is no SQL database",
  "by" : "tutorials point",
  "url" : "http://www.tutorialspoint.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

RDBMS Where Clause Equivalents in MongoDB

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eg;<value>}}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50
Values in an array	{<key>:{$in:[<value1>, <value2>,..... <valueN>]}}	db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty()	Where name matches any of the value in :["Raj", "Ram", "Raghu"]
Values not in an array	{<key>:{$nin:<value>}}	db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty()	Where name values is not in the array :["Ramu", "Raghav"] or, doesn't exist at all

Query Documents

for ignoring case

Searching for strings:

db.users.find({name" {$regex: “ja", "$options": "i"}}) Returns all users with “ja” in their names

AND in MongoDB

Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND –

```
>db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ]
```

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
> db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoD
{
        "_id" : ObjectId("5dd4e2cc0821d3b44607534c"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "by" : "tutorials point",
        "url" : "http://www.tutorialspoint.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
>
```

OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** –

```
>db.mycol.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

NOT in MongoDB

Syntax

To query documents based on the NOT condition, you need to use $not keyword following is the basic syntax of **NOT** –

```
>db.COLLECTION_NAME.find(
        {
                $NOT: [
                        {key1: value1}, {key2:value2}
                ]
        }
).pretty()
```

Example

Following example will retrieve the document(s) whose age is not greater than 25

```
> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )
{
        "_id" : ObjectId("5dd6636870fb13eec3963bf7"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Age" : "24",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

Update Documents

- Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:
  - db.collection.updateOne()
  - db.collection.updateMany()
  - db.collection.replaceOne()

```
{
  <update operator>: { <field1>: <value1>, ... },
  <update operator>: { <field2>: <value2>, ... },
  ...
}
```

Update All Matching

```
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

Update First Matching

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

Replace whole Document

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 4
)
```

Delete Documents

- MongoDB's deleteOne() and deleteMany() methods are used to remove a document from the collection.

Delete All Documents

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

Delete All Matching Documents

```
db.inventory.deleteMany({ status : "A" })
```

Delete First Matching Document

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

Limiting Records

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)
```

Example

Consider the collection myycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},
{_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}
```

Following example will display only two documents while querying the document.

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

Sorting Records

- 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})
```

Example

Consider the collection myycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"}
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"}
{_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Aggregation

- For the aggregation in MongoDB, you should use aggregate() method.

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example

In the collection you have the following data –

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

Now from the collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following **aggregate()** method –

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{ "_id" : "tutorials point", "num_tutorial" : 2 }
{ "_id" : "Neo4j", "num_tutorial" : 1 }
>
```

Available Aggregate Expressions

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum :
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg :
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min :
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max :
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

For More Info

- Tutorials Point Quick Guide
  - https://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm
- MongoDB Official Getting Started
  - https://www.mongodb.com/docs/manual/tutorial/getting-started/