

Asia Developer Academy

Introduction to NoSQL (MongoDB)

Introduction to 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.

Introduction to MongoDB (2)

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.

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.

Example of document

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.asiadev.academy',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'user2',
      message: 'My second comments',
      dateCreated: new Date(2017,5,25,7,45),
      like: 5
    }
  ]
}
```

Advantage of MongoDB

- **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 is 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.

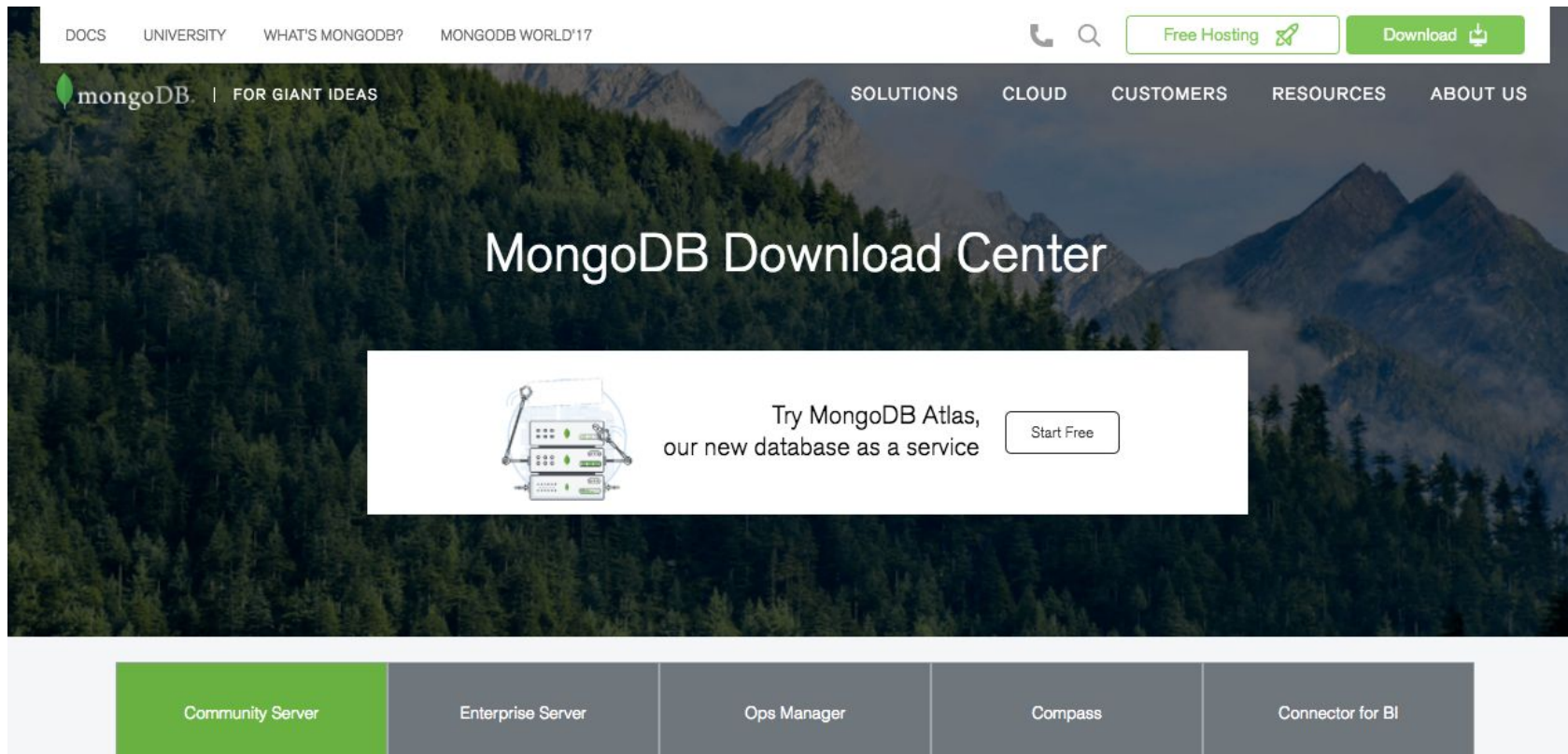
Why use MongoDB

- **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

Installing MongoDB locally



The screenshot shows the MongoDB Download Center website. The background is a scenic image of a forested mountain range. At the top, there is a navigation bar with links: DOCS, UNIVERSITY, WHAT'S MONGODB?, and MONGODB WORLD'17. On the right side of the navigation bar, there are icons for a phone and a magnifying glass, followed by two green buttons: "Free Hosting" with a rocket icon and "Download" with a download icon. Below the navigation bar, the MongoDB logo is followed by the tagline "FOR GIANT IDEAS". To the right of the logo, there are links: SOLUTIONS, CLOUD, CUSTOMERS, RESOURCES, and ABOUT US. In the center of the page, the text "MongoDB Download Center" is displayed in a large, white, sans-serif font. Below this text, there is a white rectangular box containing an illustration of a server rack with three servers, each labeled "MONGODB". To the right of the illustration, the text "Try MongoDB Atlas, our new database as a service" is displayed, followed by a "Start Free" button. At the bottom of the page, there is a horizontal row of five buttons: "Community Server" (highlighted in green), "Enterprise Server", "Ops Manager", "Compass", and "Connector for BI".

DOCS UNIVERSITY WHAT'S MONGODB? MONGODB WORLD'17

Free Hosting Download

mongoDB | FOR GIANT IDEAS SOLUTIONS CLOUD CUSTOMERS RESOURCES ABOUT US

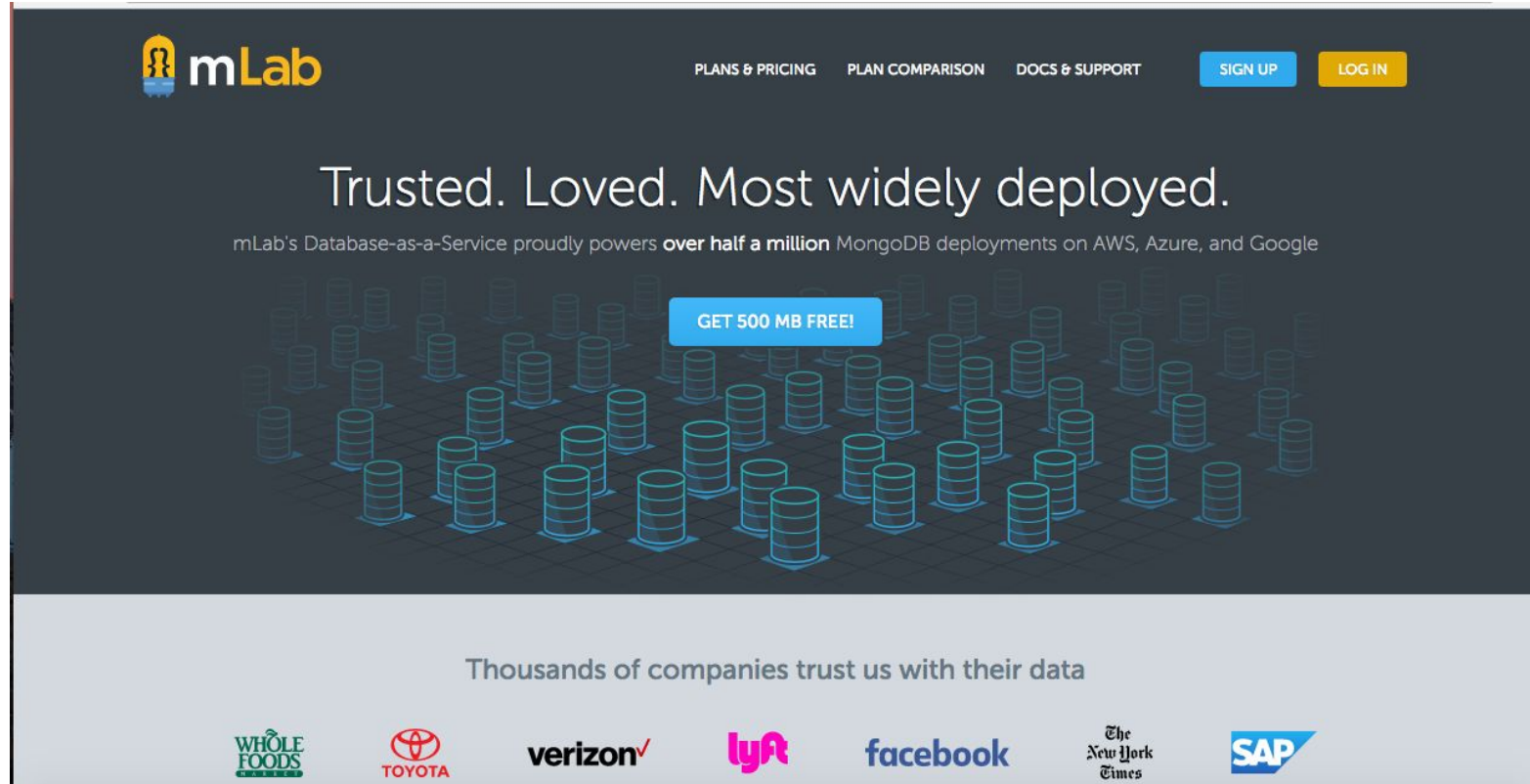
MongoDB Download Center

Try MongoDB Atlas, our new database as a service

Start Free

Community Server Enterprise Server Ops Manager Compass Connector for BI

MLab - MongoDB Cloud



The image shows a website banner for MLab, a MongoDB Cloud provider. The banner has a dark blue background with a grid of glowing blue database cylinder icons. At the top left is the MLab logo, and at the top right are navigation links for 'PLANS & PRICING', 'PLAN COMPARISON', 'DOCS & SUPPORT', 'SIGN UP', and 'LOG IN'. The main headline reads 'Trusted. Loved. Most widely deployed.' followed by a sub-headline stating that MLab's Database-as-a-Service powers over half a million MongoDB deployments on AWS, Azure, and Google. A central blue button says 'GET 500 MB FREE!'. The bottom section features the text 'Thousands of companies trust us with their data' and a row of logos for partner companies: Whole Foods Market, Toyota, Verizon, Lyft, Facebook, The New York Times, and SAP.

mLab

PLANS & PRICING PLAN COMPARISON DOCS & SUPPORT SIGN UP LOG IN

Trusted. Loved. Most widely deployed.

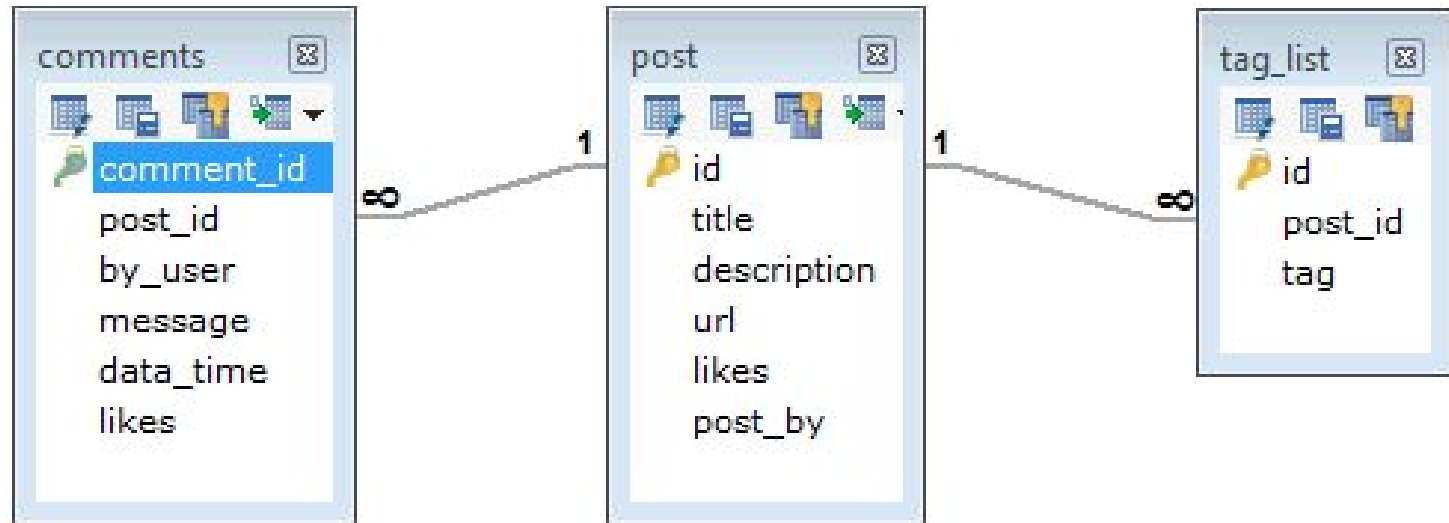
mLab's Database-as-a-Service proudly powers **over half a million** MongoDB deployments on AWS, Azure, and Google

GET 500 MB FREE!

Thousands of companies trust us with their data

WHOLE FOODS MARKET TOYOTA verizon lyft facebook The New York Times SAP

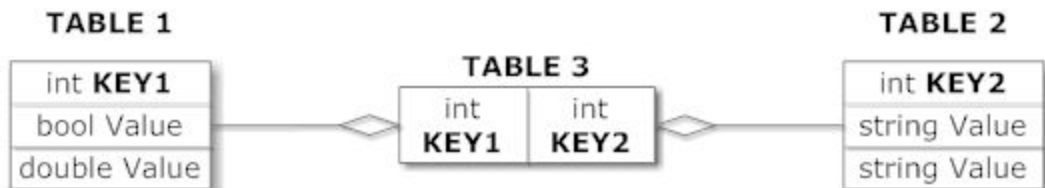
RDBMS



MongoDB

```
{
  _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
    }
  ]
}
```

Relational Model



Document Model

Collection ("Things")



Creating Database

use command to create a database. Example:

```
use mydb
```

db command to check what is the current database.

```
db
```

show dbs command to check the database list

```
show dbs
```

Drop Database

You will use `db.dropdatabase()` to **drop/delete** the current database.

```
db.dropDatabase()
```

Try It : Play with database

- 1) Connect to mongodb.
- 2) Create a new database call it mydb.
- 3) Insert one of the item inside the db:

```
db.joke.insert({"name": "Knock knock, who's there?"})
```

- 4) List down all available dbs.
- 5) Delete the db.

MongoDB Collections

- A MongoDB collection is a list of MongoDB documents and is the equivalent of a relational database table.
- A collection is created when the first document is being inserted.
- Unlike a table, a collection doesn't enforce any type of schema and can host different structured documents.
- You may use **show collections** to see the available collections at the moment.

Inserting data in MongoDB Collection

To insert data into MongoDB collection, we will use MongoDB's **insert()** or **save()** method.

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

You may also add more than one data at the time, by sending it as an **array** separated by comma.

Inserting Data in MongoDB Collection

```
db.users.insertOne(  ← collection
{
  name: "sue",        ← field: value
  age: 26,             ← field: value
  status: "pending"   ← field: value
}
)                    } document
```

Try It: Inserting a collection

```
db.places.insert({ name:'Paris',  
description:'City of Love. The most popular place in the  
world',  
country:'France',  
tags:['love','historical','shopping'], likes:100 })
```

Try It: Inserting collections

In addition to previous collection, add three collections following these criterias.

- a) One of the places will have 'France' as countries.
- b) One of the places will have new items which are comments, which have **user**, **message** and **rating** as items.
- c) Give different number of likes, eg: some will have more likes and some will have less like.

Query Document

We use **find()** method to query data from MongoDB collection.

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

You may use **pretty()** method to format the JSON nicely.

```
db.COLLECTION_NAME.find().pretty()
```

You may also use `findOne()` method if you only need to return 1 result.

```
db.COLLECTION_NAME.findOne()
```

Query

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

Filtering results.

Operation	Syntax	Example
Equality	<code>{"<key>":"<value>"}</code>	<code>db.places.find({"country":"France"})</code>
Less Than	<code>{"<key>":{"lt":<value>}}</code>	<code>db.places.find({"likes":{"lt":120}})</code>
Less than equals	<code>{"<key>":{"lte":<value>}}</code>	<code>db.places.find({"likes":{"lte":120}})</code>
Greater than	<code>{"<key>":{"gt":<value>}}</code>	<code>db.places.find({"likes":{"gt":100}})</code>
Greater than equal	<code>{"<key>":{"gte":<value>}}</code>	<code>db.places.find({"likes":{"gte":100}})</code>
Not Equal	<code>{"<key>":{"ne":<value>}}</code>	<code>db.places.find({"likes":{"ne":100}})</code>

AND and OR query in MongoDB

In **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition. Eg:

```
db.places.find({"country":"France","likes":{"$gt:100}})
```

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

```
db.places.find({$or:[{"country":"France","likes":{"$gte:100}}]})
```


Limiting records

You use **limit()** method to limit the number of documents to be displayed:

```
db.COLLECTION_NAME.find().limit(NUMBER)
```

Eg: db.places.find().limit(2)

You use **skip()** method to skip the number of documents to be displayed:

Eg: db.places.find({}, {"name":1, _id:0}).limit(1).skip(1)

Sorting records

We use **sort()** method to specify the documents in a specific sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

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

Eg: db.places.find().sort({"name":1})

Updating a document

We use `update()` or `save()` methods to update a document into collection. You will use:

Update - Update the current document with the updated data.

```
db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

```
db.places.update({'name': 'Lille'}, {$set: {'likes': 200}})
```

Save - Replace the current document with new data.

```
db.COLLECTION_NAME.save({_id:ObjectId(), NEW_DATA})
```

Deleting a document

We use **remove()** method is used to remove a document from the collection. `remove()` method accepts two parameters. One is deletion criteria and second is `justOne` flag.

Eg:

```
db.places.remove({'name':'Lille'})
```

```
db.places.remove() -> Remove all
```

Projection

Projection means finding the necessary data rather than selecting whole data of document. For example, if a data has 5 fields and you only need to show 3, then you may only set to show 3 of them.

```
db.COLLECTION_NAME.find({}, {KEY:1})
```

Eg:

```
db.mycol.find({}, {"title":1, _id:0})
```

Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. We created index to speed up the query process .

```
db.COLLECTION_NAME.ensureIndex({KEY:1})
```

Eg: db.places.ensureIndex({name:1})

Aggregation

Aggregations operations process data records and return computed results.

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

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

Eg:

```
db.places.aggregate([{$group : {_id: "$country", num_cities :  
                                {$sum : 1}}}] )
```

Try It: Aggregation

Expression	Description	Example
\$sum	Sums up the defined value from all documents in the collection.	<pre>db.places.aggregate([{\$group : {_id: "\$country", average_like : {\$sum : "\$likes"}}}])</pre>
\$avg	Calculates the average of all given values from all documents in the collection.	<pre>db.places.aggregate([{\$group : {_id: "\$country", average_like : {\$avg : "\$likes"}}}])</pre>
\$min	Gets the minimum of the corresponding values from all documents in the collection.	<pre>db.places.aggregate([{\$group : {_id: "\$country", average_like : {\$min : "\$likes"}}}])</pre>
\$max	Gets the maximum of the corresponding values from all documents in the collection.	<pre>db.places.aggregate([{\$group : {_id: "\$country", average_like : {\$max : "\$likes"}}}])</pre>