mongoDB

MongoDB is a document-oriented NoSQL database used for high-volume data storage. It contains the data model, which allows you to represent hierarchical relationships. It uses JSON-like documents with optional schema instead of using tables and rows in traditional relational databases. Documents containing key-value pairs are the basic units of data in MongoDB.

MongoDB is a document database designed for ease of development and scaling.

•MongoDB Community is the source available and free to use edition of MongoDB.

•MongoDB Enterprise is available as part of the MongoDB Enterprise Advanced subscription and includes comprehensive support for your MongoDB deployment. MongoDB Enterprise also adds enterprise-focused features such as LDAP and Kerberos support, on-disk encryption, and auditing.

Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

```
{
 name:"meera",
 class:"2A",
 doj:"2021-11-11“
}
```

The advantages of using documents are:

•Documents (i.e. objects) correspond to native data types in many programming languages.

•Embedded documents and arrays reduce need for expensive joins.

•Dynamic schema supports fluent polymorphism.

Key Features

High Performance

MongoDB provides high performance data persistence. In particular,

•Support for embedded data models reduces I/O activity on database system.

•Indexes support faster queries and can include keys from embedded documents and arrays.

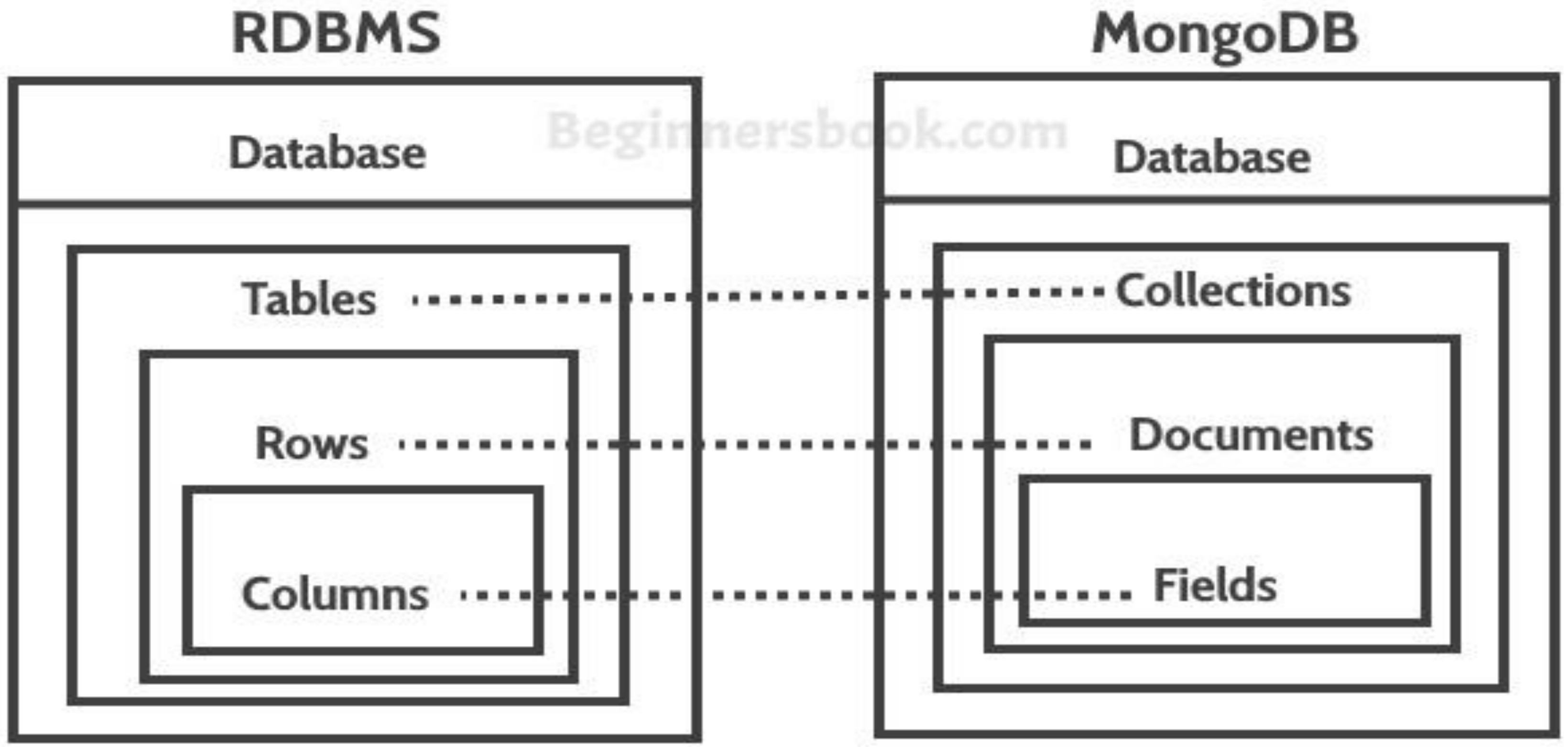
Rich Query Language

MongoDB supports a rich query language to support read and write operations (CRUD) as well as:

•Data Aggregation

•Text Search and Geospatial Queries.

Mapping relational database to MongoDB



Download & Install MongoDB on Windows

The following steps can be used to install MongoDB on Windows 10

Step 1) Go to mongo official page and Download MongoDB Community Server.

Step 2: Click Next when the MongoDB installation windows pops up.

Step 3: Accept the MongoDB user Agreement and click Next.

Step 4: When the setup asks you to choose the Setup type, choose Complete.

Step 5: Click Install to begin the installation.

Step 6: That’s it. Click Finish once the MongoDB installation is complete.

MongoDB Create Database

Once you are in the MongoDB shell, create the database in MongoDB by typing this command:

```
use database_name
```

EX: use db1

To list down all the databases, use the command **show dbs**.

```
> show dbs
```

Now we are creating a collection **user** and inserting a document in it.

```
> db.user.insert({name: “Akhil", age: 30})
```

The _id field is the unique naming convention that MongoDB uses across all of its content

- _id is the primary key on elements in a collection; with it, records can be differentiated by default.
- _id is automatically indexed. Lookups specifying { _id: <someval> } refer to the _id index as their guide.
- Architecturally, by default the _id field is an ObjectID, one of MongoDB's BSON types. Users can also override _id to something other than an ObjectID, if desired.

- "a 4-byte value representing the seconds since the Unix epoch,
- a 3-byte machine identifier,
- a 2-byte process id, and
- a 3-byte counter, starting with a random value."

Drop Database in MongoDB

We use db.dropDatabase() command to delete a database. You should be very careful when deleting a database as this will remove all the data inside that database, including collections and documents stored in the database.

```
db1.dropDatabase()
```

To verify that the database is deleted successfully. Execute the show dbs command again to see the list of databases after deletion.

Create Collection in MongoDB

Method 1: Creating the Collection in MongoDB on the fly

The cool thing about MongoDB is that you need not to create collection before you insert document in it. With a single command you can insert a document in the collection and the MongoDB creates that collection on the fly.

Syntax: **db.collection_name.insert({key:value, key:value…})**

```
use db1
switched to db1

db.db1.insert({ name: "Chithra", age: 30, web :"Chithra.com", })
```

To check whether the document is successfully inserted, type the following command. It shows all the documents in the given collection.

Syntax: **db.collection_name.find()**

Method 2: Creating collection with options before inserting the documents

We can also create collection before we actually insert data in it. This method provides you the options that you can set while creating a collection.

```
db.createCollection(name, options)
```

name is the collection name and **options** is an optional field that we can use to specify certain parameters such as size, max number of documents etc. in the collection.

```
➢ db.createCollection("students")
➢  { "ok" : 1 }
```

MongoDB Insert Document

Syntax to insert a document into the collection:

```
db.collection_name.insert()
```

```
db.beginnersbook.insert(
{
name: "Chithra",
age: 30,
email: "abc@gmail.com",
course:“Nodejs“
} )
```

Insert Multiple Documents in collection

```
var doc1 =
[
 {
 "StudentId" : 1001,
 "StudentName" : "Steve",
 "age": 30
 },
{
"StudentId" : 1002,
"StudentName" : "Negan",
 "age": 42
},
{
"StudentId" : 3333,
"StudentName" : "Rick",
"age": 35
},
 ];

db.students.insert(doc1);
```

```
db.students.find().pretty()
```

Query Document based on the criteria

Equality Criteria:

For example: I want to fetch the data of “Steve” from students collection. The command for this should be:

```
db.students.find({StudentName : "Steve"}).pretty()
```

Greater Than Criteria:

```
db.collection_name.find({"field_name":{$gt:criteria_value}}).pretty()
```

Example:

```
db.students.find({"age":{$gt:32}}).pretty()
```

Less than Criteria:

Not Equals Criteria:

```
db.collection_name.find({"field_name":{$ne:criteria_value}}).pretty()
```

Example:

```
db.students.find({"StudentId":{$ne:1002}}).pretty()
```

Greater than equals Criteria:

```
db.collection_name.find({"field_name":{$gte:criteria_value}}).pretty()
```

```
db.collection_name.find({"field_name":{$lte:criteria_value}}).pretty()
```

MongoDB – Update Document in a Collection

In MongoDB, we have two ways to update a document in a collection.

1) update()
2) save().

The update() method is used when we need to update the values of an existing document while save() method is used to replace the existing document with the document that has been passed in it.

```
db.collection_name.update(criteria, update_data)
```

```
db.got.update({"name":"Jon Snow"},{$set:{"name":"Kit Harington"}})
```

To update multiple documents with the update() method:

```
db.got.update({"name":"Jon Snow"}, {$set:{"name":"Kit Harington"}},{multi:true})
db.collection_name.save( {_id:ObjectId(), new_document} )
```

```
db.got.save({"_id" : ObjectId("59bd2e73ce524b733f14dd65"), "name":
```

A very important **point to note** is that when you do not provide the _**id** field while using save() method, it calls insert() method and the passed document is inserted into the collection as a new document

To get the _id of a document, you can either type this command:

```
db.got.find().pretty()
```

MongoDB Delete Document from a Collection

```
db.collection_name.remove(delete_criteria)
```

```
db.students.remove({"StudentId": 3333})
```

How to remove only one document

When there are more than one documents present in collection that matches the criteria then all those documents will be deleted if you run the remove command.

However there is a way to limit the deletion to only one document so that even if there are more documents matching the deletion criteria, only one document will be deleted.

```
db.collection_name.remove(delete_criteria, justOne)
```

Here justOne is a Boolean parameter that takes only 1 and 0, if you give 1 then it will limit the the document deletion to only 1 document.

```
db.student.remove({"age": 32}, 1)
```

Remove all Documents

If you want to remove all the documents from a collection but does not want to remove the collection itself then you can use remove() method like this:

```
db.collection_name.remove({})
```

MongoDB Projection

This is used when we want to get the selected fields of the documents rather than all fields.

For example, we have a collection where we have stored documents that have the fields: student_name, student_id, student_age but we want to see only the student_id of all the students ,in that case we can use projection.

Syntax:

```
db.collection_name.find({},{field_key:1 or 0})
```

To get only the student_id for all the documents:

```
> db.studentdata.find({}, {"_id": 0, "student_id": 1})
```

Value 1 means show that field and 0 means do not show that field. When we set a field to 1 in Projection other fields are automatically set to 0, except _id, so to avoid the _id we need to specifically set it to 0 in projection

The limit() method

This method limits the number of documents returned in response to a particular query.

Syntax:

```
db.collection_name.find().limit(number_of_documents)
```

Example:

```
db.studentdata.find({student_id : {$gt:2002}}).limit(1).pretty()
```

Skip() Method

The skip() method is used for skipping the given number of documents in the Query result.

```
> db.studentdata.find({student_id : {$gt:2002}}).limit(1).skip(1).pretty()
```

sort() method

Syntax :

```
db.collecttion_name.find().sort({field_key:1 or -1})
```

```
db.studentdata.find({}, {"student_id": 1, _id:0}).sort({"student_id": -1})
```