MongoDB

In MongoDB, data is stored as documents. These documents are stored in MongoDB in JSON (JavaScript Object Notation) format. JSON documents support embedded fields, so related data and lists of data can be stored with the document instead of an external table as in Relational Database.

Consider the following sample document

```
{
 "name": "notebook",
 "qty": 50,
 "rating": [ { "score": 8 }, { "score": 9 } ],
 "size": { "height": 11, "width": 8.5, "unit": "in" },
 "status": "A",
 "tags": [ "college-ruled", "perforated"]
}
```

Installation & setup

Download and install "Community server" from the link https://www.mongodb.com/try/download/community .

MongoDB offers both an Enterprise and Community version of its powerful distributed document database. The community version offers the flexible document model along with ad hoc queries, indexing, and real time aggregation to provide powerful ways to access and analyze your data.

MongoDB server requires a data director to store all data. MongoDB's default data directory path is the absolute path \data\db on the drive from which you start MongoDB.

So create folder "data" in "C drive" and "db" sub folder in it which is default location.

Start MongoDB server

To start MongoDB, run mongod.exe. For example, from the **Command Prompt**:

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe"
```

This starts the main MongoDB database process. The waiting for connections message in the console output indicates that the mongod.exe process is running successfully.

Connect to MongoDB.

To connect to MongoDB through the shell, open another **Command Prompt**.

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"
```

"mongod" is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.

To check existing databases , use the following command

- **show dbs**

To create a new database, use

- **use database-name**

To see current database, use

- **db**

MongoDB CRUD operations

Create operation:

MongoDB provides the following methods to insert documents into a collection:

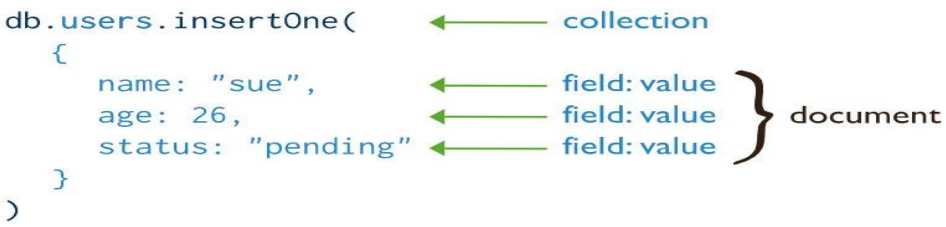
- **db.collection.insertOne()** *New in version 3.2*
- **db.collection.insertMany()** *New in version 3.2*

Syntax:

```
db.collection.insertOne(
   <document>,
   {
      writeConcern: <document>
   }
)
```

```
db.collection.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   }
)
```

Ex:



Note: By default documents are inserted in order.

Reading operation:

Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:

- **db.collection.find()**
- **db.collection.findOne()**

Update operations:

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- **db.collection.updateOne()** *New in version 3.2*

  --Updates a single document within the collection based on the filter.

- **db.collection.updateMany()** *New in version 3.2*

  --Updates all documents that match the specified filter for a collection.

- **db.collection.replaceOne()** *New in version 3.2*

  --Replaces a single document within the collection based on the filter.

syntax:

db.collection.updateOne(filter, update)

filter- the selection criteria

update-The modification to apply

Ex:

db.collection.updateOne({ eno:100},{ $set: { name :"kiran" }}

Difference between updateOne and replaceOne:

With replaceOne() you can only replace the entire document, while updateOne() allows for updating fields.

Since replaceOne() replaces the entire document - fields in the old document not contained in the new will be lost. With updateOne() new fields can be added without losing the fields in the old document.

For example if you have the following document:

```
{
   "_id" : ObjectId("0123456789abcdef01234567"),
   "my_test_key3" : 3333
}
```

Using:

replaceOne({"_id" : ObjectId("0123456789abcdef01234567")}, { "my_test_key4" : 4})
results in:

```
{
   "_id" : ObjectId("0123456789abcdef01234567"),
   "my_test_key4" : 4.0
}
```

Using:

updateOne({"_id" : ObjectId("0123456789abcdef01234567")}, {$set: { "my_test_key4" : 4}})
results in:

```
{
   "_id" : ObjectId("0123456789abcdef01234567"),
   "my_test_key3" : 3333.0,
   "my_test_key4" : 4.0
}
```

Note that with updateOne() you can use the update operators on documents.

Delete operations:

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

- **db.collection.deleteOne()** *New in version 3.2*
- **db.collection.deleteMany()** *New in version 3.2*

Query documents

To read documents from a collection, the following two methods are used.

db.collection.find()---It returns all documents of collection

db.collection.findOne()—It returns first document of collection.

To read specific document/documents from a collection, query selectors can be used.

Query selectors for comparison:

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.

Logical operators:

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.