Friday, December 30, 2022 10:26 PM

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":"Krishna", field:value
"Age":24,
"Address":"BBSR"
"Lang":["Python","Java","Golang","R","Rubby"]
}

Collections/Views/On-Demand Materialized Views
MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

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

Run MongoDB with

MongoDB Atlas fully managed in the cloud,
the source available and free-to-use MongoDB Community, or
the MongoDB Enterprise Advanced subscription.

Model your data
Design your data schema to support frequent access patterns. You can update or enforce your schema at any point.

To learn more, see Data Modeling Introduction

2
Connect to MongoDB
Import data from CSV or JSON files into your MongoDB database.

To learn more, see MongoDB Shell (mongo)

The MongoDB Shell versus the Legacy mongo Shell
The new MongoDB Shell, mongosh, offers numerous advantages over the legacy mongo shell, such as:

Improved syntax highlighting.
Improved command history.
Improved logging.
Currently mongosh supports a subset of the mongo shell methods. Achieving feature parity between mongosh and the mongo shell is an ongoing effort.

To maintain backwards compatibility, the methods that mongosh supports use the same syntax as the corresponding methods in the mongo shell. To see the complete list of methods supported by mongosh, see MongoDB Shell Methods.

3
Insert, query, update, or delete documents
Use the MongoDB Query API to perform CRUD operations on your data - with or without transactions.

To learn more, see MongoDB CRUD Operations

Perform CRUD Operations
CRUD operations create, read, update, and delete documents.

Create Operations
Create or insert operations add new documents to a collection. If the collection does not exist, create operations also create the collection.

You can insert a single document or multiple documents in a single operation.

For examples, see Insert Documents.

Read Operations
Read operations retrieve documents from a collection; i.e. query a collection for documents.

You can specify criteria, or filters, that identify the documents to return.

For examples, see Query Documents.

Update Operations
Update operations modify existing documents in a collection. You can update a single document or multiple documents in a single operation.

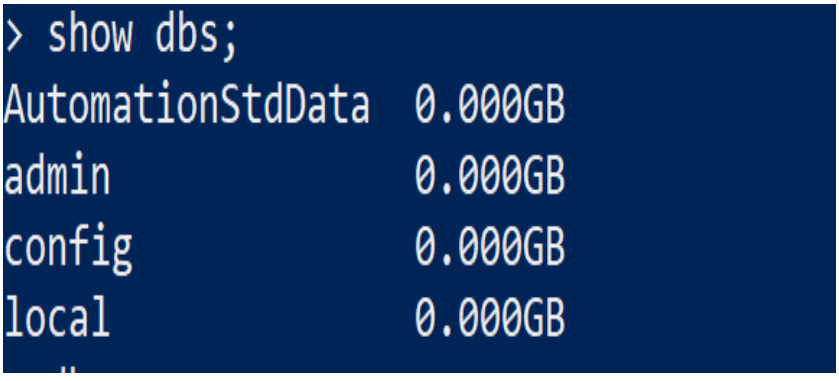
You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

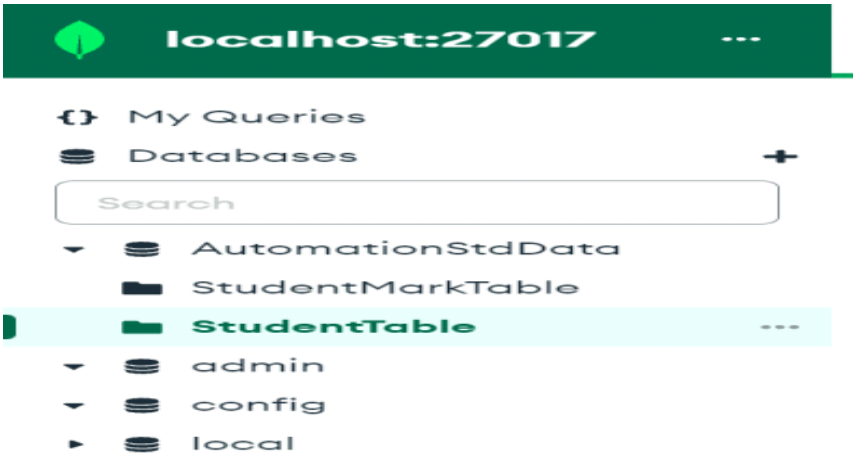
For examples, see Update Documents.

Delete Operations
Delete operations remove existing documents from a collection. You can remove a single document or multiple documents in a single operation.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

View all databases





Create a new or switch databases

```
> use AutomationStdData
switched to db AutomationStdData
```

Delete Database

```
> db.dropDatabase()
{ "dropped" : "AutomationStdData", "ok" : 1 }
```

View current Database

```
> db;
test
```

Show Collections

```
> show collections;
StudentMarkTable
StudentTable
```

Insert One Row

```
> db.StudentTable.insert({ 'Name':'Abhijit', 'Age':25, 'Salary':89000, 'Address':'Pune' })
WriteResult({ "nInserted" : 1 })
```

Insert many Rows

```
> db.StudentTable.insertMany([
... {'Name':'Abhijit',
... 'Age':25,
... 'Salary':89000,
... 'Address':'Pune'},
...
... {'Name':'Abhishek',
... 'Age':26,
... 'Salary':90000,
... 'Address':'Mumbai'},
...
... {'Name':'Abhilipsa',
... 'Age':27,
... 'Salary':87000,
... 'Address':'Chandigarh'},
... ])
```

Object Id data a Unique id

```
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("63af13922701f5267737978a"),
                ObjectId("63af13922701f5267737978b"),
                ObjectId("63af13922701f5267737978c")
        ]
}
```

Single Row Add many values like :-

"Address": [
  "Pune",
  "Mumbai"

```
> db.StudentTable.insert({ 'Name':'Abhijit', 'Age':25, 'Salary':89000, 'Address':['Pune','Mumbai'] })
WriteResult({ "nInserted" : 1 })
```


```
{
  "_id": {
    "$oid": "63af15ea2701f5267737978e"
  },
  "Name": "Sashanka",
  "Age": 34,
  "Salary": 94000,
  "Address": [
    "Pune",
    "Mumbai"
  ],
  "Award": "ETE"
}
```


New Column For Only Single Data

```
> db.StudentTable.insert({ 'Name':'Sashanka', 'Age':34, 'Salary':94000, 'Address':['Pune','Mumbai'],'Award':'ETE' })
WriteResult({ "nInserted" : 1 })
>
```

Address Mixed	**Award** String
"Pune"	No field
"Pune"	No field
"Mumbai"	No field
"Chandigarh"	No field
[] 2 elements	No field
[] 2 elements	"ETE"

```
_id: ObjectId('63af15ea2701f5267737978e')
Name: "Sashanka"
Age: 34
Salary: 94000
> Address: Array
Award: "ETE"
```

AutomationStdData.StudentTable

6 DOCUMENTS 1 INDEXES

Documents | Aggregations | Schema | Explain Plan | Indexes | Validation

Filter | Type a query: { field: 'value' } | Reset | Find | More Options

ADD DATA | EXPORT COLLECTION | 1 - 6 of 6

StudentTable

	_id ObjectId	Name String	Age Double	Salary Double	Address M:
1	ObjectId('63af12ea2701f526773…	"Abhijit"	25	89000	"Pune"
2	ObjectId('63af13922701f526773…	"Abhijit"	25	89000	"Pune"
3	ObjectId('63af13922701f526773…	"Abhishek"	26	90000	"Mumbai"
4	ObjectId('63af13922701f526773…	"Abhilipsa"	27	87000	"Chandigar
5	ObjectId('63af152a2701f526773…	"Abhijit"	25	89000	[] 2 eleme
6	ObjectId('63af15ea2701f526773…	"Sashanka"	34	94000	[] 2 eleme