****

**MongoDB**

**Submitted By:**

**Shaikh Sameer**

**Introduction**

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

## 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.



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)](https://docs.mongodb.com/manual/crud/) as well as:

* [Data Aggregation](https://docs.mongodb.com/manual/core/aggregation-pipeline/)
* [Text Search](https://docs.mongodb.com/manual/text-search/) and [Geospatial Queries](https://docs.mongodb.com/manual/tutorial/geospatial-tutorial/).

### High Availability

MongoDB’s replication facility, called [replica set](https://docs.mongodb.com/manual/replication/), provides:

* automatic failover and
* data redundancy.

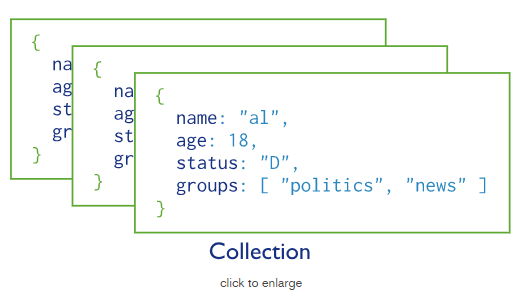
A [replica set](https://docs.mongodb.com/manual/replication/) is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

### Horizontal Scalability

MongoDB provides horizontal scalability as part of its core functionality:

* [Sharding](https://docs.mongodb.com/manual/sharding/#sharding-introduction) distributes data across a cluster of machines.
* Starting in 3.4, MongoDB supports creating [zones](https://docs.mongodb.com/manual/core/zone-sharding/#zone-sharding) of data based on the [shard key](https://docs.mongodb.com/manual/reference/glossary/#term-shard-key). In a balanced cluster, MongoDB directs reads and writes covered by a zone only to those shards inside the zone. See the [Zones](https://docs.mongodb.com/manual/core/zone-sharding/" \l "zone-sharding)manual page for more information.

MongoDB stores [BSON documents](https://docs.mongodb.com/manual/core/document/#bson-document-format), i.e. data records, in [collections](https://docs.mongodb.com/manual/reference/glossary/#term-collection); the collections in databases.



## Databases

In MongoDB, databases hold collections of documents.

To select a database to use, in the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell, issue the use <db> statement, as in the following example:

use myDB

### Create a Database

If a database does not exist, MongoDB creates the database when you first store data for that database. As such, you can switch to a non-existent database and perform the following operation in the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell:

use myNewDB

db.myNewCollection1.insertOne( { x: 1 } )

The [insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/" \l "db.collection.insertOne" \o "db.collection.insertOne()) operation creates both the database myNewDB and the collection myNewCollection1 if they do not already exist.

For a list of restrictions on database names, see [Naming Restrictions](https://docs.mongodb.com/manual/reference/limits/#restrictions-on-db-names).

## Collections

MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

### Create a Collection

If a collection does not exist, MongoDB creates the collection when you first store data for that collection.

db.myNewCollection2.insertOne( { x: 1 } )

db.myNewCollection3.createIndex( { y: 1 } )

Both the [insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/" \l "db.collection.insertOne" \o "db.collection.insertOne()) and the [createIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/" \l "db.collection.createIndex" \o "db.collection.createIndex()) operations create their respective collection if they do not already exist.

## Document Structure

MongoDB documents are composed of field-and-value pairs and have the following structure:

{

field1: value1,

field2: value2,

field3: value3,

...

fieldN: valueN

}

The value of a field can be any of the BSON [data types](https://docs.mongodb.com/manual/reference/bson-types/), including other documents, arrays, and arrays of documents. For example, the following document contains values of varying types:

**var** mydoc = {

\_id: ObjectId("5099803df3f4948bd2f98391"),

name: { first: "Alan", last: "Turing" },

birth: **new** Date('Jun 23, 1912'),

death: **new** Date('Jun 07, 1954'),

contribs: [ "Turing machine", "Turing test", "Turingery" ],

views : NumberLong(1250000)

}

The above fields have the following data types:

* \_id holds an [ObjectId](https://docs.mongodb.com/manual/reference/bson-types/" \l "objectid).
* name holds an *embedded document* that contains the fields first and last.
* birth and death hold values of the *Date* type.
* contribs holds an *array of strings*.
* views holds a value of the *NumberLong* type.

# MongoDB CRUD Operations

* [Create Operations](https://docs.mongodb.com/manual/crud/#create-operations)
* [Read Operations](https://docs.mongodb.com/manual/crud/#read-operations)
* [Update Operations](https://docs.mongodb.com/manual/crud/#update-operations)
* [Delete Operations](https://docs.mongodb.com/manual/crud/#delete-operations)

CRUD operations create, read, update, and delete [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format).

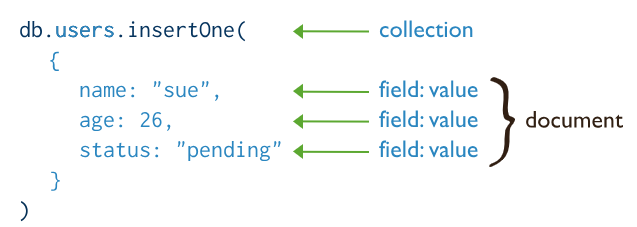
## Create Operations

Create or insert operations add new [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format) to a [collection](https://docs.mongodb.com/manual/core/databases-and-collections/#collections). If the collection does not currently exist, insert operations will create the collection.

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

* [db.collection.insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/#db.collection.insertOne) New in version 3.2
* [db.collection.insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/#db.collection.insertMany) New in version 3.2

In MongoDB, insert operations target a single [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection). All write operations in MongoDB are [atomic](https://docs.mongodb.com/manual/core/write-operations-atomicity/) on the level of a single [document](https://docs.mongodb.com/manual/core/document/).



## Read Operations

Read operations retrieves [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format) from a [collection](https://docs.mongodb.com/manual/core/databases-and-collections/#collections); i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

* [db.collection.find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find)

You can specify [query filters or criteria](https://docs.mongodb.com/manual/tutorial/query-documents/#read-operations-query-argument) that identify the documents to return.



## Update Operations

Update operations modify existing [documents](https://docs.mongodb.com/manual/core/document/#bson-document-format) in a [collection](https://docs.mongodb.com/manual/core/databases-and-collections/#collections). MongoDB provides the following methods to update documents of a collection:

* [db.collection.updateOne()](https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne) New in version 3.2
* [db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany) New in version 3.2
* [db.collection.replaceOne()](https://docs.mongodb.com/manual/reference/method/db.collection.replaceOne/#db.collection.replaceOne) New in version 3.2

In MongoDB, update operations target a single collection. All write operations in MongoDB are [atomic](https://docs.mongodb.com/manual/core/write-operations-atomicity/) on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These [filters](https://docs.mongodb.com/manual/core/document/#document-query-filter) use the same syntax as read operations.



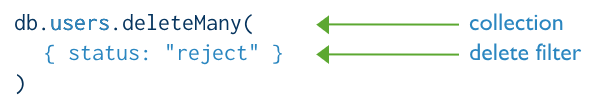
## Delete Operations

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

* [db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/#db.collection.deleteOne) New in version 3.2
* [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/#db.collection.deleteMany) New in version 3.2

In MongoDB, delete operations target a single [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection). All write operations in MongoDB are [atomic](https://docs.mongodb.com/manual/core/write-operations-atomicity/) on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These [filters](https://docs.mongodb.com/manual/core/document/#document-query-filter) use the same syntax as read operations.



# MongoDB CRUD Concepts

This section contains information on additional concepts related to CRUD operations in MongoDB.

**Atomicity, consistency, and distributed operations**

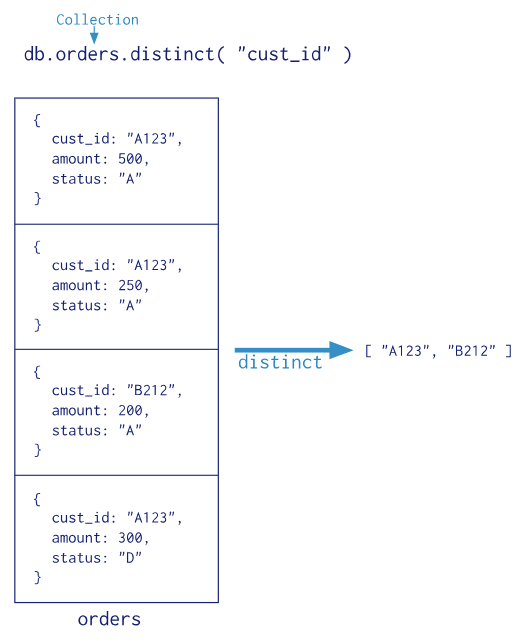
* [Atomicity and Transactions](https://docs.mongodb.com/manual/core/write-operations-atomicity/)
* [Read Isolation, Consistency, and Recency](https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/)
* [Distributed Queries](https://docs.mongodb.com/manual/core/distributed-queries/)
* [Linearizable Reads via findAndModify](https://docs.mongodb.com/manual/tutorial/perform-findAndModify-linearizable-reads/)

**Query Plan, Performance, and Analysis**

* [Query Plans](https://docs.mongodb.com/manual/core/query-plans/)
* [Query Optimization](https://docs.mongodb.com/manual/core/query-optimization/)
* [Analyze Query Performance](https://docs.mongodb.com/manual/tutorial/analyze-query-plan/)
* [Write Operation Performance](https://docs.mongodb.com/manual/core/write-performance/)

## Single Purpose Aggregation Operations

MongoDB also provides [db.collection.estimatedDocumentCount()](https://docs.mongodb.com/manual/reference/method/db.collection.estimatedDocumentCount/#db.collection.estimatedDocumentCount), [db.collection.count()](https://docs.mongodb.com/manual/reference/method/db.collection.count/#db.collection.count) and [db.collection.distinct()](https://docs.mongodb.com/manual/reference/method/db.collection.distinct/#db.collection.distinct).



All of these operations aggregate documents from a single collection. While these operations provide simple access to common aggregation processes, they lack the flexibility and capabilities of the aggregation pipeline and map-reduce.

## Examples

The following examples use the collection orders that contains the following documents:

{ \_id: 1, cust\_id: "abc1", ord\_date: ISODate("2012-11-02T17:04:11.102Z"), status: "A", amount: 50 }

{ \_id: 2, cust\_id: "xyz1", ord\_date: ISODate("2013-10-01T17:04:11.102Z"), status: "A", amount: 100 }

{ \_id: 3, cust\_id: "xyz1", ord\_date: ISODate("2013-10-12T17:04:11.102Z"), status: "D", amount: 25 }

{ \_id: 4, cust\_id: "xyz1", ord\_date: ISODate("2013-10-11T17:04:11.102Z"), status: "D", amount: 125 }

{ \_id: 5, cust\_id: "abc1", ord\_date: ISODate("2013-11-12T17:04:11.102Z"), status: "A", amount: 25 }

### Group by and Calculate a Sum

The following aggregation operation selects documents with status equal to "A", groups the matching documents by the cust\_id field and calculates the total for each cust\_id field from the sum of the amount field, and sorts the results by the total field in descending order:

db.orders.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } },

{ $sort: { total: -1 } }

])

The operation returns a cursor with the following documents:

{ "\_id" : "xyz1", "total" : 100 }

{ "\_id" : "abc1", "total" : 75 }

The [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell iterates the returned cursor automatically to print the results. See [Iterate a Cursor in the mongo Shell](https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/) for handling cursors manually in the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell.

### Return Information on Aggregation Pipeline Operation

The following aggregation operation sets the option explain to true to return information about the aggregation operation.

db.orders.aggregate(

[

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } },

{ $sort: { total: -1 } }

],

{

explain: **true**

}

)

**Specify an Initial Batch Size**

To specify an initial batch size for the cursor, use the following syntax for the cursor option:

copy

copied

cursor: { batchSize: <**int**> }

For example, the following aggregation operation specifies the *initial* batch size of 0 for the cursor:

copy

copied

db.orders.aggregate(

[

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } },

{ $sort: { total: -1 } },

{ $limit: 2 }

],

{

cursor: { batchSize: 0 }

}

)

**Specify a Collation**

*New in version 3.4.*

[Collation](https://docs.mongodb.com/manual/reference/collation/) allows users to specify language-specific rules for string comparison, such as rules for lettercase and accent marks.

A collection myColl has the following documents:

{ \_id: 1, category: "café", status: "A" }

{ \_id: 2, category: "cafe", status: "a" }

{ \_id: 3, category: "cafE", status: "a" }

The following aggregation operation includes the [collation](https://docs.mongodb.com/manual/reference/bson-type-comparison-order/#collation) option:

db.myColl.aggregate(

[ { $match: { status: "A" } }, { $group: { \_id: "$category", count: { $sum: 1 } } } ],

{ collation: { locale: "fr", strength: 1 } }

);

## Default \_id Index

MongoDB creates a [unique index](https://docs.mongodb.com/manual/core/index-unique/#index-type-unique) on the [\_id](https://docs.mongodb.com/manual/core/document/#document-id-field) field during the creation of a collection. The \_id index prevents clients from inserting two documents with the same value for the \_id field. You cannot drop this index on the \_idfield.

To create an index in the [Mongo Shell](https://docs.mongodb.com/manual/tutorial/getting-started/), use [db.collection.createIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/" \l "db.collection.createIndex" \o "db.collection.createIndex()).

db.collection.createIndex( <key and index type specification>, <options> )

The following example creates a single key descending index on the name field:

db.collection.createIndex( { name: -1 } )

The [db.collection.createIndex](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/" \l "db.collection.createIndex" \o "db.collection.createIndex) method only creates an index if an index of the same specification does not already exist.