**What is MongoDB**

[MongoDB](https://www.javatpoint.com/mongodb-tutorial) is an open-source document database that provides high performance, high availability, and automatic scaling.

In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

MongoDB is available under General Public license for free, and it is also available under Commercial license from the manufacturer.

The manufacturing company 10gen has defined MongoDB as:

"MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen

Example of Document-Oriented Database

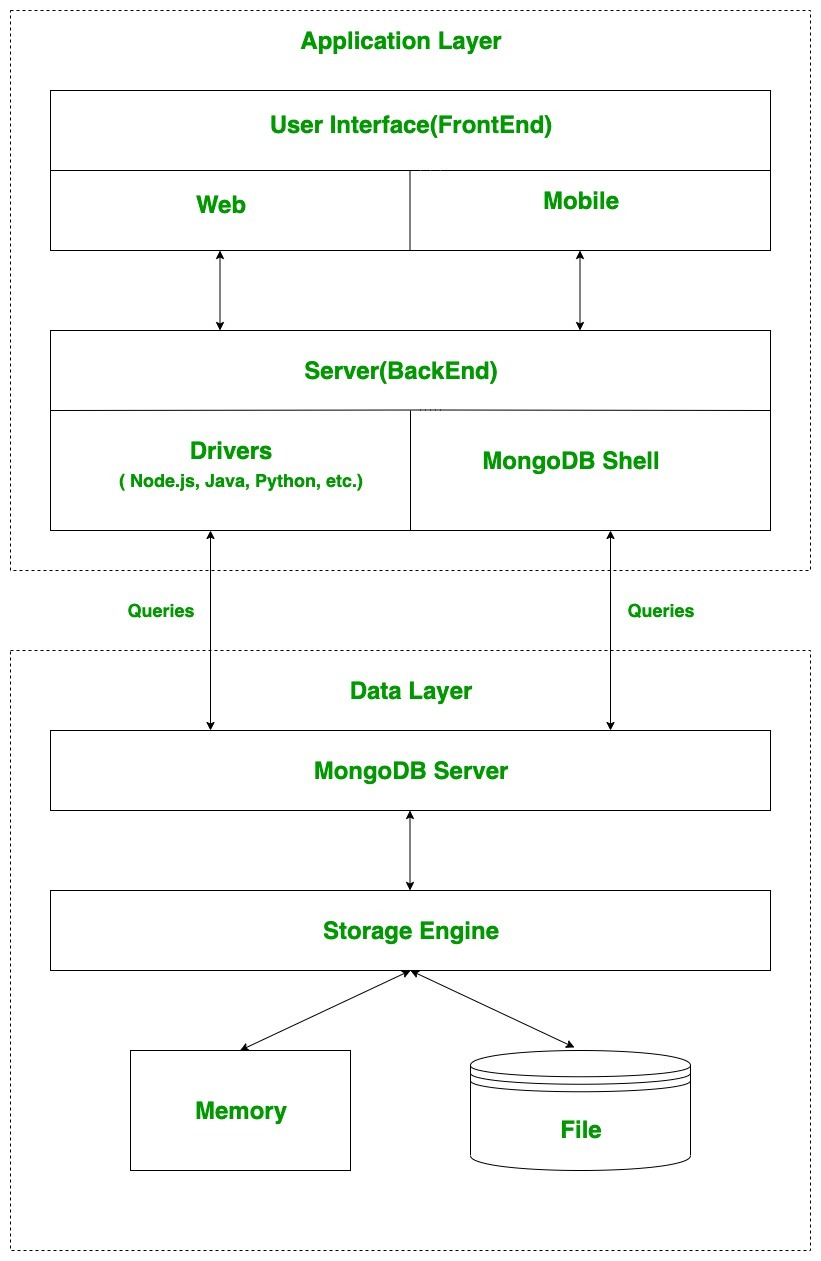
1. FirstName = "John",
2. Address = "Detroit",
3. Spouse = [{**Name**: "Angela"}].
4. FirstName ="John",
5. Address = "Wick"

**How MongoDB works ?**

MongoDB is an open-source document-oriented database. It is used to store a larger amount of data and also allows you to work with that data. MongoDB is not based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data, that’s why known as NoSQL database. Here, the term ‘NoSQL’ means ‘non-relational’. The format of storage is called BSON ( similar to JSON format). Now, let’s see how actually this MongoDB works? But before proceeding to its working, first, let’s discuss some important parts of MongoDB –

* Drivers: Drivers are present on your server that are used to communicate with MongoDB. The drivers support by the MongoDB are C, C++, C#, and .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid.
* MongoDB Shell: MongoDB Shell or mongo shell is an interactive JavaScript interface for MongoDB. It is used for queries, data updates, and it also performs administrative operations.
* Storage Engine: It is an important part of MongoDB which is generally used to manage how data is stored in the memory and on the disk. MongoDB can have multiple search engines. You are allowed to use your own search engine and if you don’t want to use your own search engine you can use the default search engine, known as*WiredTiger Storage Engine* which is an excellent storage engine, it efficiently works with your data like reading, writing, etc.

**Working of MongoDB –**



|  |  |
| --- | --- |
| RDBMS | MongoDB |
| It is a [relational database](https://www.geeksforgeeks.org/relational-model-in-dbms/). | It is a non-relational and document-oriented database. |
| Not suitable for hierarchical data storage. | Suitable for [hierarchical data storage](https://www.geeksforgeeks.org/hierarchical-model-in-dbms/#:~:text=In%20a%20hierarchical%20model%2C%20data,parent%20record%20and%20many%20children.). |
| It is vertically scalable i.e increasing RAM. | It is horizontally scalable i.e we can add more servers. |
| It has a predefined schema. | It has a dynamic schema. |
| It is quite vulnerable to SQL injection. | It is not affected by [SQL injection](https://www.geeksforgeeks.org/sql-injection-2/). |
| It centers around [ACID](https://www.geeksforgeeks.org/acid-properties-in-dbms/) properties (Atomicity, Consistency, Isolation, and Durability). | It centers around the [CAP theorem](https://www.geeksforgeeks.org/the-cap-theorem-in-dbms/) (Consistency, Availability, and Partition tolerance). |
| It is row-based. | It is document-based. |
| It is slower in comparison with MongoDB. | It is almost 100 times faster than RDBMS. |
| Supports complex joins. | No support for complex joins. |
| It is column-based. | It is field-based. |
| It does not provide JavaScript client for querying. | It provides a JavaScript client for querying. |
| It supports SQL query language only. | It supports [JSON](https://www.geeksforgeeks.org/json/) query language along with [SQL](https://www.geeksforgeeks.org/sql-tutorial/). |

**Types of NoSQL Databases**

* Document-based databases
* Key-value stores
* Column-oriented databases
* Graph-based databases

**Document-Based Database:**

The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON, or XML documents

**Key-Value Stores:**

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a key-value store. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.

**Column Oriented Databases:**

A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.

**Graph-Based databases:**

Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships.

**MongoDB Cursor**

The MongoDB cursor is a pointer that references the documents of the collection returned by the find() method.

The cursor is used to access the documents. By default, the cursor iterates automatically, but can also be iterated manually by the user.

**JSON VS BSON**

JSON, short for **JavaScript Object Notation**, makes sharing data simple and straightforward. Created by Douglas Crockford, it’s designed for easy reading and writing by humans, and easy parsing and generating by computers. Its main goal was to make a text format that’s good at showing simple data like lists and text, and really useful for websites.

JSON is special because it’s very clear and easy to use, and it uses a **“.json”** file ending to show that a file is in this format. This makes JSON great for both people and programs to work with.

**JSON Syntax:**

**1. Using ‘Objects’**

Objects in JSON are collections of key/value pairs enclosed in curly braces {}.

Each key is a string (enclosed in double quotes ") followed by a colon :, and the key/value pairs are separated by commas (,).

Example: {"firstName": "John", "lastName": "Doe", "age": 30}

**2. Using ‘Arrays’**

Arrays are ordered lists of values, enclosed in square brackets [].

Values within an array are separated by commas (,).

Example: ["apple", "banana", "cherry"]

**BSON**

BSON stands for Binary JSON. It is a binary file format that is used to store serialized JSON documents in a binary-encoded format. It was developed in 2009 by MongoDB. The MongoDB database had several scalar data formats that were of special interest only for MongoDB, hence they developed the BSON data format to be used while transferring files over the network. Although the format was developed specifically for MongoDB, it can be used anywhere as per business requirements independently.

It has several similarities with JSON for instance BSON too supports nested documents and arrays within other documents, but yet has a lot of striking differences. Refer to this post to read more about the difference between JSON and BSON.

**Sample BSON document**

Consider the following JSON document:

{

"hello" : "world"

}

It’s BSON equivalent will be:

\x16\x00\x00\x00 // Size of the Document

\x02 // 0x02 = type String

hello\x00 // field name

\x06\x00\x00\x00world\x00 // field value

\x00 // Used to represent end of object

**MongoDB Cursor**

The MongoDB cursor is a pointer that references the documents of the collection returned by the find() method.

The cursor is used to access the documents. By default, the cursor iterates automatically, but can also be iterated manually by the user.

**Using the next() Method**

We can also use the next() cursor method to access the next document. Let us discuss with the help of an example:

Count cursor:

In order to get the correct documents, we need to know how many documents are present for that collection. To get that we can use the count() method which returns the total number of documents present in the given collection.

**Cursor Limit:**

The limit() method helps to fetch limited records from a collection. Suppose we have multiple documents, but we want to have the topmost or only 2 documents, then by using the limit() method, we can achieve that.

**Cursor size:**

The cursor.size() method will be helpful to return a count of the number of documents that got as the output from the db.collection.find() query after applying any cursor.skip() and cursor.limit() methods. Hence, it is mentioned as it has applied cursor.skip() and cursor.limit() methods.

**Cursor sort:**

Usually while verifying documents, if the output is in sorted order, either in ascending or descending order, it will be easier. So we use the sort() method to sort the documents. If you want to sort the documents in ascending, then set the value of the field to 1 and in descending, then set -1.

**Cursor.toArray():**

In order to have an array that contains all documents returned by the cursor, we can use the toArray() method.

**MongoDB – Comparison Query Operators**

Comparison Query Operators in MongoDB are used to filter documents based on some specific criteria within their fields.

MongoDB uses various comparison query operators to compare the values of the documents. The following table contains the comparison query operators:

|  |  |
| --- | --- |
| **Operators** | **Description** |
| [**$eq**](https://www.geeksforgeeks.org/mongodb-equality-operator-eq/) | Matches the values of the fields that are equal to a specified value. |
| [**$ne**](https://www.geeksforgeeks.org/mongodb-inequality-operator-ne/) | Matches all values of the field that are not equal to a specified value. |
| [**$gt**](https://www.geeksforgeeks.org/mongodb-greater-than-operator-gt/) | Matches values of the fields that are greater than a specified value. |
| [**$gte**](https://www.geeksforgeeks.org/mongodb-greater-than-equals-to-operator-gte/) | Matches values of the fields that are greater than equal to the specified value. |
| [**$lt**](https://www.geeksforgeeks.org/mongodb-less-than-operator-lt/) | Matches values of the fields that are less than a specified value |
| [**$lte**](https://www.geeksforgeeks.org/mongodb-less-than-equals-to-operator-lte/) | Matches values of the fields that are less than equal to the specified value |
| [**$in**](https://www.geeksforgeeks.org/mongodb-in-operator/) | Matches any of the values specified in an array. |
| [**$nin**](https://www.geeksforgeeks.org/mongodb-nin-operator/) | Matches none of the values specified in an array. |
|  |  |

**MongoDB – Logical Query Operators**

MongoDB supports logical query operators. These operators are used for filtering the data and getting precise results based on the given conditions. The following table contains the comparison query operators:

|  |  |
| --- | --- |
| Operator | Description |
| $and | It is used to join query clauses with a logical AND and return all documents that match the given conditions of both clauses. |
| $or | It is used to join query clauses with a logical OR and return all documents that match the given conditions of either clause. |
| $not | It is used to invert the effect of the query expressions and return documents that does not match the query expression. |
| $nor | It is used to join query clauses with a logical NOR and return all documents that fail to match both clauses. |

MongoDB – Update() Method:

The update() method in MongoDB updates a document or multiple documents in the collection. When the document is updated the \_id field remains unchanged.

This method can be used for a single updating of documents as well as multiple documents. By default, the db.collection.update() method updates a single document. To update all documents that match the given query. Include the option “multi: true“.

The **updateOne()**method in MongoDB updates the first matched document within the collection based on the given query.

The value of the **\_id field** remains unchanged after updating the value. This method updates one document at a time and can also add new fields to the given document.

**The updateMany()** method updates all the documents in MongoDB collections that match the given query. When you update your document, the value of the \_id field remains unchanged. This method can also add new fields in the document. Specify an empty document({}) in the selection criteria to update all collection documents.

In MongoDB, you are allowed to replace an existing document with a new document in the collection with the help of db.collection.replaceOne() method. This method will replace the existing document with the replacement document.

**replaceOne**() is a mongo shell method, which only replaces one document at a time. The replacement document may contain different fields as compared to the original document.

Delete Operations:

deleteOne method is used to delete a single document in MongoDB. This method deletes a single existing document from the collection that matches the specified filter. This method can be used in multi-document transactions.

The **deleteOne()** method in MongoDB deletes the first document from the collection that matches the given selection criteria. It will delete/remove a single document from the collection.

db.student.deleteOne({age:17})

The **db.collection.deleteMany()** method is used to delete multiple documents from a collection in Mongo Shell. This method deletes multiple documents from the collection according to the filter.

The deleteMany() is a Mongo shell method, which can delete multiple documents. This method can be used in multi-document transactions. If you use this method in a capped collection, then it will throw an exception.

Indexing In MongoDb:

MongoDB provides a createIndex() method to create one or more indexes on collections. Using this method we can create different types of indexes like text index, 2dsphere index, 2d index, etc. It takes three parameters first one is a document that contains the field and value pairs where the field is the index key and the value describes the type of index for that field and others are optional.

If you are creating an index that is already present, then MongoDB does not recreate the existing index.

You can hide and unhide an index using hideIndex() and unhideIndex() method.

db.Collection.name.createIndex(  
    keys : {Field\_name:1/-1},  
    options : <document>,  
    commitQuorum : <string or integer>  
)

db.student.createIndex({name:1},{unique:true})

In MongoDB, the **getIndexes**() method returns an array that contains a list of documents that identify and describe the existing indexes on the specified collection. It also includes hidden indexes as well.

This method does not take any parameters.

The index information return by this method contains the keys and the options used to create an index.

In this method, the hidden index available starting from MongoDB 4.4. Only if the value is true.

Syntax:

db.Collection\_name.getIndexes()

dropIndex Method in MongoDB

The MongoDB dropIndex() method allows for the removal of specified indexes from a collection, but it does not permit the deletion of the default index of the \_id field. Additionally, hidden indexes can also be dropped using this method.

Starting from MongoDB 4.4, if the specified index is still being built, the dropIndex() method will abort the building process of the index, providing developers with greater control over index management in MongoDB collections.

Note: Starting from MongoDB 4.2, you are not allowed to remove all the non-\_id indexes using db.Collection\_Name.dropIndex(“\*”). If you want to do that then use db.Collection\_Name.dropIndexes() method.

Syntax

db.Collection\_Name.dropIndex(index : <document/string>)

**MongoDB – Index Types:**

1. Single field Index: A single field index means index on a single field of a document. This index is helpful for fetching data in ascending as well as descending order.

Syntax:

db.students.createIndex({“<fieldName>” : <1 or -1>});

2. Compound Index: We can combine multiple fields for compound indexing and that will help for searching or filtering documents in that way. Or in other words, the compound index is an index where a single index structure holds multiple references.

Syntax:

db.<collection>.createIndex( { <field1>: <type>, <field2>: <type2>, … } )

3. Multikey Index: MongoDB uses the multikey indexes to index the values stored in arrays. When we index a field that holds an array value then MongoDB automatically creates a separate index of each and every value present in that array. Using these multikey indexes we can easily find a document that contains an array by matching the items. In MongoDB, you don’t need to explicitly specify the multikey index because MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value.

Syntax:

db.<collection>.createIndex( { <field>: <type>} )

4. Geospatial Indexes: It is an important feature in MongoDB. MongoDB provides two geospatial indexes known as 2d indexes and 2d sphere indexes using these indexes we can query geospatial data. Here, the 2d indexes support queries that are used to find data that is stored in a two-dimensional plane. It only supports data that is stored in legacy coordinate pairs. Whereas 2d sphere indexes support queries that are used to find the data that is stored in spherical geometry. It supports data that is stored in legacy coordinate pairs as well as GeoJSON objects. It also supports queries like queries for inclusion, intersection, and proximity, etc.

Syntax of 2d sphere indexes:

db.<collection>.createIndex( { <Locationfield>: “2dsphere”} )

5. Text Index: MongoDB supports query operations that perform a text search of string content. Text index allows us to find the string content in the specified collection. It can include any field that contains string content or an array of string items. A collection can contain at most one text index. You are allowed to use text index in the compound index.

Syntax:

db.<collection>.createIndex( { <field>: “text”} )

6. Hash Index: To maintain the entries with hashes of the values of the indexed field(mostly \_id field in all collections), we use Hash Index. This kind of index is mainly required in the even distribution of data via sharding. Hashed keys are helpful to partition the data across the sharded cluster.

Syntax:

db.<collection>.createIndex( { \_id: “hashed” } )

7. Wildcard Index: MongoDB supports creating indexes either on a field or set of fields and if the set of fields are mentioned, it is called as Wildcard Index. Generally, the wildcard index does not include \_id field but if you what to include \_id field in the wildcard index then you have to define it explicitly. MongoDB allows you to create multiple wildcard indexes in the given collection. Wildcard indexes support queries for unknown or arbitrary fields.

Syntax:

To create a wild card index on the specified field:

db.<collection>.createIndex( { “field.$\*\*”:1 } )

To create a wild card index on all the field:

db.<collection>.createIndex( { “$\*\*”:1 } )

**Aggregation:**

Aggregation in MongoDB

Last Updated : 16 Apr, 2024

MongoDB aggregation operations process the data records/documents and return computed results. It collects values from various documents, groups them, and then performs different types of operations on that grouped data like sum , average , minimum , maximum , etc to return a computed result. It is similar to the aggregate function of SQL .

Aggregation in MongoDB allows users to transform, filter, and analyze data. They are used on multiple documents and provide an efficient way to summarize the data.

MongoDB provides three ways to perform aggregation

Aggregation Pipelines

Map Reduce Function

Single Purpose Aggregation

Aggregation Pipelines

Aggregation pipelines in MongoDB consist of stages and each stage transforms the document. It is a multi-stage pipeline and in each state, the documents are taken as input to produce the resultant set of documents.

In the next stage (ID available) the resultant documents are taken as input to produce output, this process continues till the last stage.

The basic pipeline stages provide:

filters that will operate like queries

the document transformation that modifies the resultant document

provide pipeline provides tools for grouping and sorting documents .

Stages

Each stage starts from stage operators which are:

$match: It is used for filtering the documents can reduce the amount of documents that are given as input to the next stage.

$project: It is used to select some specific fields from a collection.

$group: It is used to group documents based on some value.

$sort: It is used to sort the document that is rearranging them

$skip: It is used to skip n number of documents and passes the remaining documents

$limit: It is used to pass first n number of documents thus limiting them.

$unwind: It is used to unwind documents that are using arrays i.e. it deconstructs an array field in the documents to return documents for each element.

$out: It is used to write resulting documents to a new collection

Expressions

It refers to the name of the field in input documents for e.g. { $group : { \_id : ” $id “, total:{$sum:” $fare “}}} here $id and $fare are expressions.

Accumulators

These are basically used in the group stage

sum: It sums numeric values for the documents in each group

count: It counts total numbers of documents

avg: It calculates the average of all given values from all documents

min: It gets the minimum value from all the documents

max: It gets the maximum value from all the documents

first: It gets the first document from the grouping

last: It gets the last document from the grouping

The $limit Stage

The $limit stage is used in the aggregation pipeline.

It restricts the number of documents that pass through the pipeline.

Useful for limiting the amount of data processed or returned in an aggregation query.

It is used to reduce the number of documents processed in the pipeline, improving query performance.

$limit can be used at any stage of the pipeline but is commonly used towards the end to limit the final result set.

The $limit stage takes a single argument, which is the maximum number of documents to return.

db.orders.aggregate([  
 { $limit: 3 }  
])

Command aggregate:

1. $match

The $match command filters documents based on specified criteria, similar to the find() method. It allows users to select only those documents that match the given conditions.

Example:

To retrieve products with a price greater than $100, we can use the $match command:

db.products.aggregate([

{ $match: { price: { $gt: 100 } } }

])

2. $group

The $group command groups documents together based on a specified key and applies aggregate functions to the grouped data, such as sum, count or average.

Example

Continuing with the products collection example, let’s group products by their category and calculate the total number of products in each category:

db.products.aggregate([

{ $group: { \_id: "$category", total\_products: { $sum: 1 } } }

])

3. $project

The $project command reshapes documents by including, excluding, or renaming fields. It allows users to define the structure of the output documents.

Example:

Suppose we want to retrieve only the name and price fields of products from the products collection. We can use the $project command to include only these fields in the output:

db.products.aggregate([

{ $project: { \_id: 0, name: 1, price: 1 } }

])

4. $sort

The $sort command sorts documents based on specified fields in ascending or descending order.

Example:

To retrieve products from the products collection sorted by price in descending order, we can use the $sort command:

db.products.aggregate([

{ $sort: { price: -1 } }

])

5. $limit

The $limit command restricts the number of documents returned by an aggregation operation.

Example:

If we want to retrieve only the top 5 highest-priced products from the products collection, we can use the $limit command:

db.products.aggregate([

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

{ $limit: 3 }

])

$out Stage (aggregation)

The $out stage is a crucial component of the Aggregation Pipeline, it takes the documents returned by the aggregation pipeline and writes them to a specified collection. The returned documents can be written into a new collection by specifying the desired name of the new collection in the $out stage

The $lookup operator in MongoDB is a powerful tool for performing join-like operations between documents from two collections. It allows us to perform join operations which are defined by documents from one collection with data from another collection based on a specified matching condition.

**Syntax:**

The basic syntax of the $lookup operator is as follows:

{  
 $lookup: {  
 from: <foreignCollection>,  
 localField: <fieldInInputDocument>,  
 foreignField: <fieldInForeignDocument>,  
 as: <outputArrayField>  
 }  
}

**Arithmetic Operators:**

**MongoDB provides different types of arithmetic expression operators that are used in the aggregation pipeline stages and $add operator is one of them. This operator is used to add numbers or dates. If $add operator adds date, then it will treat other arguments as milliseconds and add to the specified date.**

**Syntax:**

**{ $add: [ <Expression1>, <Expression2>, ... <ExpressionN>] }**

**$subtract operator is one of them. This operator is used to subtract two numbers and return the difference in the numbers or to subtract two dates and return the difference in the milliseconds, or subtracts date and number in milliseconds and returns the date.**

{ $subtract: [ <expression1>, <expression2> ] }

MongoDB provides different types of arithmetic expression operators that are used in the aggregation pipeline stages and $multiply operator is one of them. This operator is used to multiply one number to another number and returns the result.

Syntax:

{ $multiply: [ <expression1>, <expression2>, ... <expressionN> ] }

MongoDB provides different types of arithmetic expression operators that are used in the aggregation pipeline stages and $divide operator is one of them. This operator is used to divide one number by another number and return the result of the division.

Syntax:

{ $divide: [ <expression1>, <expression2> ] }

**MongoDB provides different types of arithmetic expression operators that are used in the aggregation pipeline stages and $abs operator is one of them. This operator is used to find the absolute value of the specified number.**

**Syntax:**

**{ $abs: <number> }**

**Here, the number is a valid expression until it resolves to a number.**

**If the entered value is null, then this operator will return null.**

**If the entered value is NaN, then this operator will return NaN.**

**If the entered value is a missing field, then this operator will return null.**

**MongoDB provides different types of arithmetic expression operators that are used in the aggregation pipeline stages and $floor operator is one of them. This operator is used to find the largest integer less than or equal to the specified number.**

**Syntax:**

**{ $floor: <number> }**

**Here, the number is a valid expression until it resolves to a number.**

**If the entered value is null, then this operator will return null.**

**If the entered value is NaN, then this operator will return NaN.**

**If the entered value is a missing field, then this operator will return null.**

MongoDB – $inc Operator

Last Updated : 23 Apr, 2024

MongoDB $inc or increment operator is a type of field update operator. $inc operator increases the values of the fields to the specified amount or increases the field by the given value.

Important Points

This operator accepts positive and negative values.

If the given field does not exist, then this operator will create a field and set the value of that field.

This operator will generate an error, if you use this operator with a null value field.

It is an atomic operation in a single document.

Syntax

{ $inc: { field1: amount1, field2: amount2, ... } }

MongoDB $min or minimum operator is one of the field update operators. $min operator updates the field with the specified value if the specified value is less than the current value.

The $min operator will compare the values of different data types according to the BSON comparison order. This operator can also be used in embedded/nested documents using dot notation.

One can use this operator in methods like update(), updateOne(), etc. according to your requirements. If the given field does not exist, then this operator will create a field and set the value of that field.

Syntax

{ $min: { field1: value1, field2: value2 ... } }

MongoDB $max or maximum operator is one of the field update operators. $max operator updates the field with the specified value if the specified value is greater than the current value.

Important Points

This operator will compare the values of different data types according to the BSON comparison order.

You can also use this operator in embedded/nested documents using dot notation.

You can use this operator in methods like update(), updateOne(), exist, etc. according to your requirements.

If the given field does not exist, then this operator will create a field and set the value of that field.

Syntax

{ $max: { field1: value1, field2: value2 ... } }

**MongoDB Atlas**

MongoDB Atlas is a Database-as-a-Service (DBaaS). It is a fully controlled cloud-primarily based database provider. It gives a simplified and automated solution for deploying, scaling, and maintaining MongoDB databases within the cloud. It Provides capabilities that include automatic backups, protection controls, and seamless scalability.