**Unit-1: Concepts of NoSQL: MongoDB**

**What is MongoDb?**

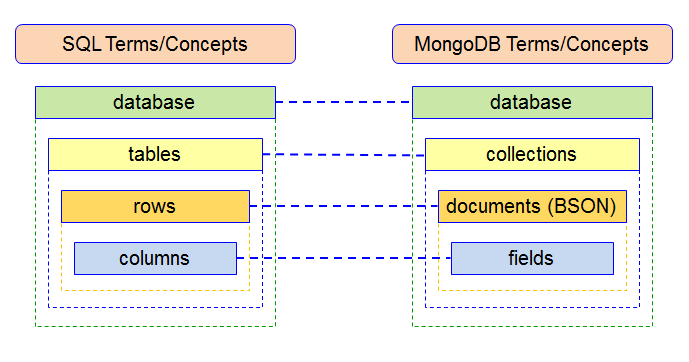
* MongoDB is an open-source, cross-platform, and distributed document-based database designed for ease of application development and scaling.
* It is a NoSQL database developed by [MongoDB Inc](https://www.mongodb.com/company).
* MongoDB name is derived from the word "Humongous" which means huge, enormous. MongoDB database is built to store a huge amount of data and also perform fast.
* [MongoDB](https://www.javatpoint.com/mongodb-tutorial) is provides high performance, high availability, and automatic scaling.

**Concepts of NoSQL**

* MongoDB is not a Relational Database Management System (RDBMS). It's called a "NoSQL" database. It is opposite to SQL based databases where it does not normalize data under schemas and tables where every table has a fixed structure. Instead, it stores data in the collections as JSON based documents and does not enforce schemas. It does not have tables, rows, and columns as other SQL (RDBMS) databases.
* The following table lists the relation between MongoDB and RDBMS terminologies.

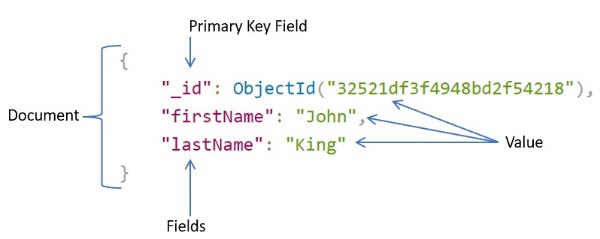
| **RDBMS (SQL Server, Oracle, etc.)** | **MongoDB (NoSQL Database)** |
| --- | --- |
| **Database** | **Database** |
| **Table** | **Collection** |
| **Row (Record)** | **Document** |
| **Column** | **Field** |

**[MongoDB vs SQL Concepts](https://www.google.com/url?sa=i&url=https%3A%2F%2Fstudio3t.com%2Facademy%2Ftopic%2Fmongodb-vs-sql-concepts%2F&psig=AOvVaw2ZDsNDXPcQXzfCtLVo2t6J&ust=1653150739383000&source=images&cd=vfe&ved=0CAoQjhxqFwoTCKiT9szA7vcCFQAAAAAdAAAAABAD" \o "MongoDB vs SQL Concepts | Studio 3T" \t "_blank)**



* In the RDBMS database, a table can have multiple rows and columns.
* Similarly in MongoDB, a collection can have multiple documents which are equivalent to the rows.
* Each document has multiple "fields" which are equivalent to the columns.
* Documents in a single collection can have different fields.

**The following is an example of JSON based document.**



# NoSQL Databases

We know that MongoDB is a NoSQL Database, so it is very necessary to know about NoSQL Database to understand MongoDB throughly.

Databases can be divided in 3 types:

1. RDBMS (Relational Database Management System)
2. OLAP (Online Analytical Processing)
3. NoSQL (recently developed database)

## What is NoSQL Database

NoSQL Database is used to refer a non-SQL or non relational database.

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

## History behind the creation of NoSQL Databases

* In the early 1970, Flat File Systems are used. Data were stored in flat files and the biggest problems with flat files are each company implement their own flat files and there are no standards.
* It is very difficult to store data in the files, retrieve data from files because there is no standard way to store data.
* Then the relational database was created by E.F. Codd and these databases answered the question of having no standard way to store data.
* But later relational database also get a problem that it could not handle big data, due to this problem there was a need of database which can handle every types of problems then NoSQL database was developed.

## Advantages of NoSQL

* It supports query language.
* It provides fast performance.
* It provides horizontal scalability.
* Can be used as Primary or Analytic Data Source
* Big Data Capability
* No Single Point of Failure
* Easy Replication
* No Need for Separate Caching Layer
* Can handle structured, semi-structured, and unstructured data with equal effect
* Object-oriented programming which is easy to use and flexible
* NoSQL databases don’t need a dedicated high-performance server
* Support Key Developer Languages and Platforms
* Simple to implement than using RDBMS
* It can serve as the primary data source for online applications.
* Handles big data which manages data velocity, variety, volume, and complexity
* Excels at distributed database and multi-data center operations
* Eliminates the need for a specific caching layer to store data.
* Offers a flexible schema design which can easily be altered without downtime or service disruption

**Disadvantages of NoSQL**

* No standardization rules
* Limited query capabilities
* [RDBMS](https://www.guru99.com/difference-dbms-vs-rdbms.html) databases and tools are comparatively mature
* It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
* When the volume of data increases it is difficult to maintain unique values as keys become difficult
* Doesn’t work as well with relational data
* The learning curve is stiff for new developers
* Open source options so not so popular for enterprises.

**Difference between mongo and mongod**

* "**mongo**" is the command-line shell that connects to a specific instance of mongod.
* "**mongod**" is the "Mongo Daemon" it's basically the host process for the database.

## History of MongoDB

* The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure.
* Window azure is a cloud computing platform and infrastructure, created by Microsoft, to build, deploy and manage applications and service through a global network.
* MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform as a Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.
* The first ready production of MongoDB has been considered from version 1.4 which was released in March 2010.

## Purpose of building MongoDB

* It may be a very genuine question that - "what was the need of MongoDB although there were many databases in action?"
* **There is a simple answer:** All the modern applications require big data, fast features development, flexible deployment, and the older database systems not competent enough, so the MongoDB was needed.

**Distinctive features of MongoDB**

* Easy to use
* Light Weight
* Extremely faster than RDBMS

**The primary purpose of building MongoDB is:**

* Scalability
* Performance
* High Availability
* Scaling from single server deployments to large, complex multi-site architectures.
* Key points of MongoDB
* Develop Faster
* Deploy Easier
* Scale Bigger

**Where is MongoDB Used?**

While MongoDB has wide application, it is used most commonly for the following:

* Building mobile and social infrastructure
* Managing and Delivering Content
* Big Data and Complex data
* Data Hub
* Managing User Data

**How Does MongoDB Work?**

* As there are no tables as such in a MongoDB NoSQL database, it is easy for developers to work with it.
* Data is stored in BSON or extended JSON format, in key-value pairs. A unique key is defined and a value assigned to it. These values are stored in documents which are then stored in a collection.
* The BSON format supports more data types for storage including Boolean, string, double, integer, object, finery data, JavaScript, array, and so on.
* You can think of the MongoDB collection as a relational dataset table. However, this collection has no pre-set structure.

### Example of document oriented database

* MongoDB is a document oriented database. It is a key feature of MongoDB. It offers a document oriented storage. It is very simple you can program it easily.
* MongoDB stores data as documents, so it is known as document-oriented database.
* **There are two different documents (separated by ".").**
* Storing data in this manner is called as document-oriented database.
* Mongo DB falls into a class of databases that calls Document Oriented Databases.
* There is also a broad category of database known as [No SQL Databases](https://www.javatpoint.com/nosql-databases).

**Features of MongoDB**

These are some important features of MongoDB:

**1. Support ad hoc queries -** In MongoDB, you can search by field, range query and it also supports regular expression searches.

**2. Indexing -** You can index any field in a document.

**3. Replication -** MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

**4. Duplication of data -** MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

**5. Load balancing -** It has an automatic load balancing configuration because of data placed in shards.

**6. Supports map reduce and aggregation tools.**

**7. Uses**[**JavaScript**](https://www.javatpoint.com/javascript-tutorial)**instead of Procedures.**

**8. It is a schema-less database written in**[**C++**](https://www.javatpoint.com/cpp-tutorial)**.**

**9. Provides high performance.**

**10. Stores files of any size easily without complicating your stack.**

**11. Easy to administer in the case of failures.**

**12. It also supports:**

JSON data model with dynamic schemas.

Auto-sharding for horizontal scalability.

Built in replication for high availability.

**Advantages of MongoDB**

There are many great features inbuilt with MongoDB. As compared to RDBMS, so let’s discuss MongoDB Benefits.

#### a. Flexible Database

We know that MongoDB is a schema-less database. That means we can have any type of data in a separate document. This thing gives us flexibility and a freedom to store data of different types.

#### b. Sharding

We can store a large data by distributing it to several servers connected to the application. If a server cannot handle such a  big data then there will be no failure condition. The term we can use here is “auto-sharding”.

#### c. High Speed

MongoDB is a document-oriented database. It is easy to access documents by indexing. Hence, it provides fast query response. The speed of MongoDB is 100 times faster than the relational database.

#### d. High Availability

MongoDB has features like replication and gridFS. These features help to increase data availability in MongoDB. Hence the performance is very high.

#### e. Scalability

A great advantage of MongoDB is that it is a horizontally scalable database. When you have to handle a large data, you can distribute it to several machines.

#### f. Ad-hoc Query Support

MongoDB has a very advanced feature for ad hoc queries. This is why we don’t need to worry about fore coming queries coming in the future.

#### g. Easy Environment Setup

It is easier to setup MongoDB then RDBMS. It also provides JavaScript client for queries.

#### h. Full Technical Support

MongoDB Inc. provides professional support to its clients. If there is any problem, you can directly reacha MongoDB client support system.

## Disadvantages Of MongoDB

Nothing is fully complete or perfect. In spite of having so many advantages, MongoDB to has some limitations. Let’s discuss some of them here.

#### a. Joins not Supported

MongoDB doesn’t support joins like a relational database. Yet one can use joins functionality by adding by coding it manually. But it may slow execution and affect performance.

#### b. High Memory Usage

MongoDB stores key names for each value pairs. Also, due to no functionality of joins, there is data redundancy. This results in increasing unnecessary usage of memory.

#### c. Limited Data Size

You can have document size, not more than 16MB.

#### d. Limited Nesting

You cannot perform nesting of documents for more than 100 levels.  
This was all about Advantages Of MongoDB Tutorial. Hope you like our explanation.

**Differences between SQL and NoSQL**

The table below summarizes the main differences between SQL and NoSQL databases.

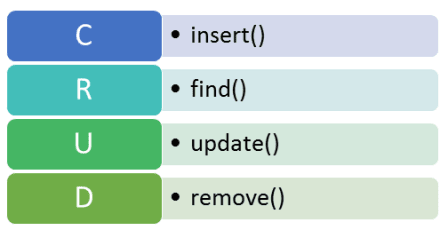
|  |  |  |
| --- | --- | --- |
| **Index** | **SQL** | **NoSQL** |
| 1) | Databases are categorized as Relational Database Management System (RDBMS). | NoSQL databases are categorized as Non-relational or distributed database system. |
| 2) | SQL databases have fixed or static or predefined schema. | NoSQL databases have dynamic schema. |
| 3) | SQL databases display data in form of tables so it is known as table-based database. | NoSQL databases display data as collection of key-value pair, documents, graph databases or wide-column stores. |
| 4) | SQL databases are vertically scalable. | NoSQL databases are horizontally scalable. |
| 5) | SQL databases use a powerful language "Structured Query Language" to define and manipulate the data. | In NoSQL databases, collection of documents are used to query the data. It is also called unstructured query language. It varies from database to database. |
| 6) | SQL databases are best suited for complex queries. | NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries. |
| 7) | SQL databases are not best suited for hierarchical data storage. | NoSQL databases are best suited for hierarchical data storage. |
| 8) | MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc. are the example of SQL database. | MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc. are the example of nosql database |

**MongoDb Datatype**

|  |  |
| --- | --- |
| **Data Types** | **Description** |
| **String** | String is the most commonly used datatype. It is used to store data. A string must be UTF 8 valid in mongodb. |
| **Integer** | Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using. |
| **Boolean** | This datatype is used to store boolean values. It just shows YES/NO values. |
| **Double** | Double datatype stores floating point values. |
| **Min/Max Keys** | This datatype compare a value against the lowest and highest bson elements. |
| **Arrays** | This datatype is used to store a list or multiple values into a single key. |
| **Object** | Object datatype is used for embedded documents. |
| **Null** | It is used to store null values. |
| **Symbol** | It is generally used for languages that use a specific type. |
| **Date** | This datatype stores the current date or time in unix time format. It makes you possible to specify your own date time by creating object of date and pass the value of date, month, year into it. |

## What are CRUD Operations with MongoDB?

CRUD Operations with MongoDB are the methods that MongoDB exposes for storage management. **CRUD** is an acronym for Create, Read, Update, and Delete. You can use these four basic methods for viewing, searching, and changing resources in your database.

[](https://res.cloudinary.com/hevo/image/upload/f_auto,q_auto/v1685935959/hevo-learn-1/crud.png?_i=AA)

The following is a brief overview of what each operation does:

* **Create**: This is the operation that is used to insert or add new documents to the database.
* **Read**: This operation is used to view or fetch documents from the database.
* **Update**: This operation should be used when modifying existing documents.
* **Delete**: As the name suggests, the delete operation is used to remove documents from the database.

Together these four CRUD operations make up the essential operations of interacting with your MongoDB server.

## CRUD Operations with MongoDB:

**Create Operations:**

* [insert()](https://hevodata.com/learn/crud-operations-with-mongodb/#10)
* [insertOne()](https://hevodata.com/learn/crud-operations-with-mongodb/#10)
* [insertMany()](https://hevodata.com/learn/crud-operations-with-mongodb/#11)

## Read Operations

* [find()](https://hevodata.com/learn/crud-operations-with-mongodb/#12)
* [findOne()](https://hevodata.com/learn/crud-operations-with-mongodb/#13)

## Update Operations

* [updateOne()](https://hevodata.com/learn/crud-operations-with-mongodb/#14)
* [updateMany()](https://hevodata.com/learn/crud-operations-with-mongodb/#15)

## Delete Operations

* [remove()](https://hevodata.com/learn/crud-operations-with-mongodb/#16)
* [deleteOne()](https://hevodata.com/learn/crud-operations-with-mongodb/#16)
* deleteMany()

**MongoDbCommands**

This list covers almost all the most used commands in MongoDB.

Let assume that you are working inside “DemoMongo” Database Where a collection named “Demo1” on a MongoDB database of your choice

**1. Database Commands**

* Create a new or switch databases

**Syntax- use databasename**

**Example- use Employee**

**use Students**

**use DemoMongo**

* View all databases

**show dbs**

* View current Database

**db**

* Delete current Database

**db.dropDatabase()**

**2. Collection Commands**

* Create a collection

**Syntax - db.createCollection('**collection name**')** //table name

**Example- db.createCollection('**Demo1**')**

**db.createCollection('**Demo2**')**

**db.createCollection('**Demo3**')**

**db.createCollection('**Demo4**')**

* Show Collections

**show collections**

* Drop a collection named 'Demo1'

**Syntax- db.collectionname.drop()**

**Example- db.Demo1.drop()**

**3. Row(Document) Commands**

**[C (create)**🡪 **insert(), insertOne(), InertMany() ]**

* Insert Row

**db. Demo1.insert({**

**“name”: “pooja”,**

**“Address”: “Adajan”,**

**“city”: “surat”**

**})**

* Insert One Row

**db. Demo1.insertOne({**

**“name”: “pooja”,**

**“Address”: “Adajan”,**

**“city”: “surat” })**

* Insert Many Row

**db.Demo1.insertMany([**

**{"name":"nisha","address":"pal","city":"Baroda"},**

**{"name":"payal","phone":123456756},**

**{"name":"komal","city":"Ahmedabad"},**

**{"name":"neha","address":"pal"},**

**{"name":"Nehal","city":"surat","age":32},**

**{"name":"Arman","phone":123456756},**

**{"name":"Dipal","age":"38","phone":9685741254},**

**{"name":"Nirav","address":"Adajan","age":28}])**

**[R (Read)🡪 find(), findOne()]**

* Show all Rows in a Collection

**Syntax- db.collectionname. find ()**

**Example- db.Demo1.find()**

* Show single Row in a Collection

**Syntax- db.collectionname. findOne ()**

**Example- db.Demo1.findOne()**

* Show all Rows in a Collection (Prettified)

**Syntax - db.collectionname. find ().pretty()**

**Example - db. Demo1.find().pretty()**

* Limit the number of rows in output

**db.Demo1.find().limit(2)**

* Count the number of rows in the output

**db.Demo1.find().count()**

* Search in a MongoDb Database

**Syntax – db. db.collection.find({query : projection})**

**Example - db.Demo1.find({"address":"surat"})**

**[U(Update)🡪 updateOne(), updateMany(), replaceOne()]**

* updateOne

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

**Example-**

**db.Demo1.updateOne({"name":"Dipal"},{$set:{"city":"adajan"}})**

* updateMany

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

**Example- db. Demo1.updateMany({"age":38},{$set: {"name":"Dipal"}})**

**db.Demo1.updateMany({},{$set:{"salary":25000}})**

* replaceOne

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

**Example- db. Demo1.** **replaceOne({"age":"38"},{"age":38})**

**[D (Delete)**🡪 **deleteOne(), deleteMany(), remove()]**

* Delete Row

**Syntax - [db.collection.remove({query : projection})](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "db.collection.deleteMany" \t "_target)**

**Example - db. Demo1.remove({"phone":123456756})**

**Syntax - [db.collection.deleteOne({query : projection})](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "db.collection.deleteMany" \t "_target)**

**Example - db. Demo1.deleteOne({"phone":123456756})**

**Syntax - [db.collection. deleteMany ({query : projection})](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "db.collection.deleteMany" \t "_target)**

**Example - db. Demo1. deleteMany ({"phone":123456756})**

**Operators**

* sort the number of rows in output

**Syntax - db.** **collection.find().sort({key:1}) //** **ascending**

**Example - db.Demo1.find().sort({key:1}) //** **ascending**

**Syntax - db.** **collection.find().sort({key:-1}) //** **descending**

**Example - db.Demo1.find().sort({key:-1}) //** **descending**

* Limit the number of rows in output

**Syntax - db.** **collection.find().limit(Number)**

**Example- db.Demo1.find().limit(2)**

* Count the number of rows in the output

**Syntax - db.** **collection.find().count()**

**Example- db.Demo1.find().count()**

* skip the number of rows in output

**Syntax - db.** **collection.find().skip(Number)**

**Example - db.** **Demo1.find().skip(3)**

# MongoDB Query and Projection Operator

1. MongoDB Comparison Operators
2. Eqals to - $eq:

**db.Demo1.find({age:{$eq:30}})**

1. Greaterthan- $gt:

**db.Demo1.find({age:{$gt:28}})**

1. Less than- $lt:

**db.Demo1.find({"age":{$lt:32}})**

1. Greater than or Eq- $gte:

**db.Demo1.find({"age":{$gte:30}})**

1. Less than or Eq - $lte:

**db.Demo1.find({"age":{$lte:30}})**

1. Or - $or:

**db.Demo1.find({$or:[{city:"surat"},{age:32}]})**

1. And- $and:

**db.Demo1.find({$and:[{city:"surat"},{age:32}]})**

1. Like - /

**db.Demo1.find({name:/Ni/})**

1. In - $in

**db.Demo1.find({age:{$in:[25,30]}})**

1. Nor - $nor

**db.Demo1.find({$nor:[{city:"surat"},{age:32}]})**

* Distinct

**db.Demo1.distinct("city")**

**db.Demo1.distinct("age")**

**db.Demo1.distinct("age",{"city":"surat"})**

* Aggregation Using **aggregate()** Method

1. $match

**db.Demo1.aggregate([{$match:{city:"surat"}}])**

1. $group

**db.Demo1.aggregate([{$group:{city:"surat"}}])**

1. $sum

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$sum:"$age"}}}])**

**db.Demo1.aggregate([{$group:{\_id:"$name",totage:{$sum:"$age"}}}])**

1. $min

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$min:"$age"}}}])**

1. $max

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$min:"$age"}}}])**

1. $avg

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$avg:"$age"}}}])**

1. $first

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$first:"$age"}}}])**

1. $last

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$last:"$age"}}}])**

1. $push

**db.Demo1.aggregate([{$group:{\_id:"name",totage:{$push:"$age"}}}])**

[**What is Aggregation in MongoDB?**](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#what-is-aggregation-in-mongodb)

**Aggregation** is a way of processing a large number of documents in a collection by means of passing them through different stages. The stages make up what is known as a pipeline. The stages in a pipeline can filter, sort, group, reshape and modify documents that pass through the pipeline.

One of the most common use cases of Aggregation is to calculate aggregate values for groups of documents. This is similar to the basic aggregation available in SQL with the GROUP BY clause and COUNT, SUM and AVG functions. MongoDB Aggregation goes further though and can also perform relational-like joins, reshape documents, create new and update existing collections, and so on.

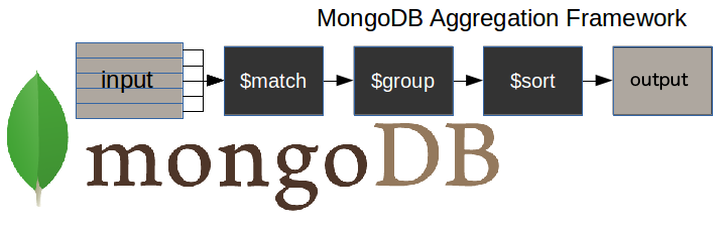
While there are other methods of obtaining aggregate data in MongoDB, the aggregation framework is the recommended approach for most work.

There are what are called **single purpose methods** like estimatedDocumentCount(),  count(), and distinct() which are appended to a find() query making them quick to use but limited in scope.

The **map-reduce framework** on MongoDB is a predecessor of the aggregation framework and much more complex to use. MongoDB have deprecated Instead of map-reduce, you should use an aggregation pipeline. Aggregation pipelines provide better performance and usability than map-reduce. You can rewrite map-reduce operations using aggregation pipeline stages, such as $group , $merge , and others.

[**How does the MongoDB aggregation pipeline work?**](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#what-is-the-mongodb-aggregation-pipeline)

Here is a diagram to illustrate a typical MongoDB aggregation pipeline.



* **$match stage** – filters those documents we need to work with, those that fit our needs
* **$group stage** – does the aggregation job
* **$sort  stage** – sorts the resulting documents the way we require (ascending or descending)

The input of the pipeline can be a single collection, where others can be merged later down the pipeline.

The pipeline then performs successive transformations on the data until our goal is achieved.

This way, we can break down a complex query into easier stages, in each of which we complete a different operation on the data. So, by the end of the query pipeline, we will have achieved all that we wanted.

This approach allows us to check whether our query is functioning properly at every stage by examining both its input and the output. The output of each stage will be the input of the next**.**

There is no limit to the number of stages used in the query, or how we combine them.

To achieve optimum query performance there are a number of best practices to take into account. We will come to those later in the article.

## [MongoDB aggregate pipeline syntax](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#mongodb-aggregate-pipeline-syntax)

This is an example of how to build an aggregation query:

**db*.*collectionName*.*aggregate(pipeline*,*options),**

* where collectionName – is the name of a collection,
* pipeline – is an array that contains the aggregation stages,
* options – optional parameters for the aggregation

This is an example of the aggregation pipeline syntax:

pipeline = [

{ $match : { … } },

{ $group : { … } },

{ $sort : { … } }

]

### [MongoDB $match](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#mongodb-match)

The $match stage allows us to choose just those documents from a collection that we want to work with. It does this by filtering out those that do not follow our requirements.

### [MongoDB $project](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#mongodb-project)

It is rare that you ever need to retrieve all the fields in your documents. It is good practice to return only those fields you need so as to avoid processing more data than is necessary.

The $project stage is used to do this and to add any calculated fields that you need.

In the code that follows, please note that:

* We must explicitly write \_id : 0 when this field is not required
* Apart from the \_id field, it is sufficient to specify only those fields we need to obtain as a result of the query

### [MongoDB $group](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#mongodb-group)

With the $group stage, we can perform all the aggregation or summary queries that we need, such as finding counts, totals, averages or maximums.

## Aggregation pipeline 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.

#### [MongoDB $group aggregation operators](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/#mongodb-aggregation-operators)

The $group stage supports certain expressions (operators) allowing users to perform arithmetic, array, boolean and other operations as part of the aggregation pipeline.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| $count | Calculates the quantity of documents in the given group. |
| $max | Displays the maximum value of a document’s field in the collection. |
| $min | Displays the minimum value of a document’s field in the collection. |
| $avg | Displays the average value of a document’s field in the collection. |
| $sum | Sums up the specified values of all documents in the collection. |
| $push | Adds extra values into the array of the resulting document. |

**SQL to MongoDB Mapping**

The table below presents the various SQL terminology and concepts, which are similar to MongoDB terminology and concepts.

|  |  |
| --- | --- |
| **SQL Terms** | **MongoDB Terms** |
| database | Database |
| table | Collection |
| row | document or BSON document |
| column | field |
| index | index |
| SQL statements | MongoDB statements |
| CREATE TABLE Students (id NOT NULL, AUTO\_INCREMENT, user\_id Varchar(20),  age Number, status char(1),PRIMARY KEY (id)) | db.createCollection ( " Students”) |
| INSERT INTO Students (user\_id,age,status)  VALUES (1, "Aman", 45, "A") | db.Students.insertOne({user\_id:"1",name:"Aman", age: 18, status: "A" }) |
| DELETE FROM Students WHERE status = "D" | db. Students.deleteMany({status:"D" } ) |
| UPDATE JavaTpoint SET status = "C"  WHERE age > 25 | db.JavaTpoint.updateMany({age:{$gt:25}},{$set:{status:”c”}}] |