**Indexes:**

**• Indexes support the efficient resolution of queries.**

**• Without indexes, MongoDB must scan every document of a collection to**

**select those documents that match the query statement. This scan is highly**

**inefficient and require MongoDB to process a large volume of data.**

**• Indexes are special data structures, that store a small portion of the data**

**set in an easy-to-traverse form.**

**• The index stores the value of a specific field or set of fields, ordered by the**

**value of the field as specified in the index.**

**• To create an index you need to use createIndex({key:1}) method of MongoDB.**

**-----------------------------------------------------------------------------------------------------------**

**Aggregation**

**• Aggregation operations group values from multiple documents together,**

**and can perform a variety of operations on the grouped data to return a**

**single result.**

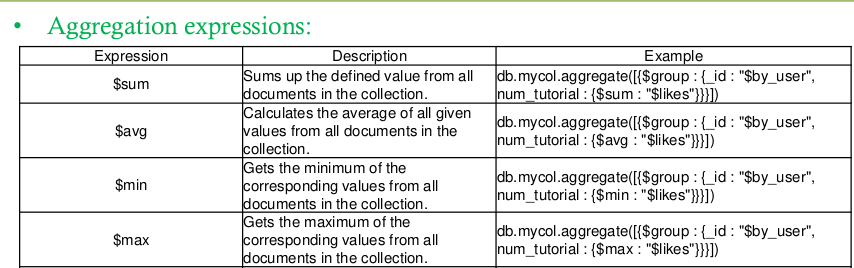
**• Syntax:**

**>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)**

# **Aggregation Pipeline**

An aggregation pipeline consists of one or more stages that process documents:

* Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.
* The documents that are output from a stage are passed to the next stage.
* An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.
* **$project**− Used to select some specific fields from a collection.
* **$match**− This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
* **$group**− This does the actual aggregation as discussed above.
* **$sort**− Sorts the documents.
* **$skip**− With this, it is possible to skip forward in the list of documents for a given amount of documents.
* **$limit**− This limits the amount of documents to look at, by the given number starting from the current positions.
* **$unwind**− This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

var pipe=[{$group:{\_id:'$name',total:{$sum:{$multiply:["$price","$quantity"]}}}

},{$sort:{total:-1}}] {$unwind:'$grades'},{$match:{'grades.grade':'A'}},

{$group:{\_id:'$borough',sum:{$sum:1}}},

{$sort:{sum:1}}

Demo:

1. Write a MongoDB query to display all the restaurant which is in the borough Bronx
2. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.
3. Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx
4. Write a MongoDB query to find the restaurants who achieved a score more than 90
   1. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and lattitude less than -65.754168.
5. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.
6. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are belonging to the borough Staten Island or Queens or Bronxor Brooklyn.
7. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronxor Brooklyn.
8. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10.
9. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees'
10. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.
11. Write a MongoDB query to know whether all the addresses contains the street or not.

// db.getCollection('Trainee').find({},{name:1,age:1,\_id:0}).skip(3).limit(2)

// gt greater than grade > 50

db.Trainee.find({grade:{$gte:50,$lte:90}},{name:1,\_id:0}).sort({grade:-1})

// db.Order.find({$and:[{name:'Cheese'},{price:{$gte:10}},{size:'small'}]},{\_id:1})

// db.getCollection('Books').find({$or:[{country:"Italy"},{country:"England"},{country:"France"}]})

//db.Books.find({country:{$in:['Italy','England','France']}})

//db.Books.find({country:{$nin:['Italy','England','France']}})

// db.Books.find({$and:[{country:'Italy'},{$or:[{pages:{$gte:300}},{year:{$gte:1925}}]}]})

db.Books.find({language:{$ne:'Italian'}})

db.getCollection('lab1').find({$or:[{year:2020},{$and:[{'cost.price':30},{'cost.currency':'EUR'}]}]})

db.getCollection('restaurants').find({$and:[

{'cuisine':{$ne:'American'}},

{'grades.score':{$gt:70}},

{'address.coord':{$lt:-65.754168}}

]}).explain('executionStats')

// info about my query

//db.getCollection('restaurants').find({"borough" : "Bronx"}).explain('executionStats')

db.getCollection('restaurants').find({"grades.score" : 2}).explain('executionStats')

aggregation

//group same name and compute sum of price

// var pipe=

// [

// {$match:{name:"Pepperoni"}},

// {$group:{\_id:'$name',price:{$sum:'$price'}}}

//

// ]

var pipe=[

{$group:{\_id:"$name",maxorder:{$sum:'$quantity'}}},{$sort:{maxorder:1}},{$limit:1}]

db.Order.aggregate(pipe)

var pip=[

{$unwind:'$grades'},

{$group:{\_id:'$borough',avgScore:{$avg:'$grades.score'}}},

{$sort:{avgScore:-1}}]

db.restaurants.aggregate(pip)

var pip=[

{$match:{year:{$gt:2017}}},

{$group:{\_id:'$year',delivery:{$sum:'$delivery\_days'}}},

{$sort:{'\_id':1}}]

db.lab1.aggregate(pip)