# MONGODB - AGGREGATION

 

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count * and with group by is an equivalent of mongodb aggregation.

## The aggregate Method

For the aggregation in MongoDB, you should use **aggregate** method.

## Syntax

Basic syntax of **aggregate** method is as follows –

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

## Example

In the collection you have the following data –

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

Now from the above collection, if you want to display a list stating how many tutorials are written by each user,

then you will use the following **aggregate** method –

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "tutorials point",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
>
```

Sql equivalent query for the above use case will be **select by_user, count * from mycol group by by_user**.

In the above example, we have grouped documents by field **by_user** and on each occurrence of by_user previous value of sum is incremented. Following is a list of available aggregation expressions.

| Expression | Description | Example |
|---|---|---|
| $sum | Sums up the defined value from all documents in the collection. | db.mycol.aggregate [$group : _id :" $by_user " , num_tutorial : $sum :" $likes " ] |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate [$group : _id :" $by_user " , num_tutorial : $avg :" $likes " ] |
| $min | Gets the minimum of the corresponding values from all documents in the collection. | db.mycol.aggregate [$group : _id :" $by_user " , num_tutorial : $min :" $likes " ] |
| $max | Gets the maximum of the corresponding values from all documents in the collection. | db.mycol.aggregate [$group : _id :" $by_user " , num_tutorial : $max :" $likes " ] |
| $push | Inserts the value to an array in the resulting document. | db.mycol.aggregate [$group : _id :" $by_user " , url : $push :" $url " ] |
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate [$group : _id :" $by_user " , url : $addToSet :" $url " ] |

| | | |
|---|---|---|
| $first | Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate [{$group : {_id : "$by_user", first_url : {$first : "$url"}}}] |
| $last | Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate [{$group : {_id : "$by_user", last_url : {$last : "$url"}}}] |

## Pipeline Concept

In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents *orthefinalresultingJSONdocumentattheendofthepipeline*. This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework –

- **$project** – Used to select some specific fields from a collection.
- **$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.