

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: 'tutorialspoint',  
  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: 'tutorialspoint',  
  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 that how many tutorials are written by each user then you will use **aggregate** method as shown below:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", numTutorial : {$sum : 1}}}] )  
{  
  "result" : [  
    {  
      "_id" : "tutorialspoint",  
      "numTutorial" : 2  
    },  
    {  
      "_id" : "Neo4j",  
      "numTutorial" : 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. There is a list available aggregation expressions.

Expression	Description	Example
\$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate [{\$group: { <i>d</i> : "\$by_user", numTutorial: {\$sum: "\$likes" }}]
\$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate [{\$group: { <i>d</i> : "\$by_user", numTutorial: {\$avg: "\$likes" }}]
\$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate [{\$group: { <i>d</i> : "\$by_user", numTutorial: {\$min: "\$likes" }}]
\$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate [{\$group: { <i>d</i> : "\$by_user", numTutorial: {\$max: "\$likes" }}]
\$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate [{\$group: { <i>d</i> : "\$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: { <i>d</i> : "\$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: { <i>d</i> : "\$by_user", firstUrl: {\$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: { <i>d</i> : "\$by_user", lastUrl: {\$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 a set of documents as an input and is producing a resulting set of documents *or the final resulting JSON document at the end of the pipeline*. This can then in turn again be used for the next stage and so on.

Possible stages in aggregation framework are following:

- **\$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 position.s
- **\$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

Processing math: 100%