MONGODB AGGREGATE FUNCTIONS EXPLAINED

This is the second part of the tutorial on how to use NodeJS with MongoDB. Here we switch to using the regular MongoDB shell and commands to make the study of aggregate functions simpler.

To show how to use aggregate functions, we will first explain how to do basic queries. Then we will show how to do the **WordCount** program, which is what people start with when they are first learning, for example, Apache Spark.

(This article is part of our MongoDB Guide. Use the right-hand menu to navigate.)

Basically, there are two aggregate functions: **aggregate** and **MapReduce**. Aggregate functions are the same as the familiar SQL command:

```
SQL select xxx, count (xxx) from table group by xxx
```

The aggregate functions are **count, sum, average, min, max,** etc.

The data that we used in the first part was smoker survey date. It has a complex structure. Recall that the schema is this:

```
var schema = new mongoose.Schema({
cachedContents : {
largest : String,
non_nill : Number,
nill : Number,
tip :  ,
smallest : String,
format : {
```

```
displayStyle : String,
align : String
}
}
});
```

And the sample data looks like this:

```
"_id" : ObjectId("59e9a6144bde260fa07e68f1"),
"cachedContents" : {
"nill" : 0,
"non_nill" : 14069,
"largest" : "WY",
"tip" :
},
"__v" : 0
}
```

That is complicated to work with, since it has an array of embedded documents: **tip:**. So we will first show how to query this data, then we will make simpler data to show the aggregate and mapreduce functions.

Simple Queries

A simple find query will output all fields. Like this:

```
db.smokersmodels.find({ "cachedContents.largest" : "2016" }).pretty()
```

```
"_id" : ObjectId("59e9a6144bde260fa07e6fda"),
"cachedContents" : {
"nill" : 0,
"non_nill" : 14069,
"largest" : "2016",
"tip" :
```

Word Count

Now we make some simpler data to illustrate how to do the WordCount program with MongoDB.

First, create some data:

```
db.words.insertMany( );
```

They print it out to see what it looks like:

```
db.words.find().pretty()
{ "_id" : ObjectId("59eb051e878eb4d6aca74243"), "word" : "the" }
{ "_id" : ObjectId("59eb051e878eb4d6aca74244"), "word" : "rain" }
{ "_id" : ObjectId("59eb051e878eb4d6aca74245"), "word" : "in" }
{ "_id" : ObjectId("59eb051e878eb4d6aca74246"), "word" : "spain" }
```

```
{ "_id" : ObjectId("59eb051e878eb4d6aca74247"), "word" : "spain" }
```

The trick with the WordCount program has always been to make the (key,value) pairs (word, 1) and then sum the number 1:

```
db.words.aggregate()
```

```
{ "_id" : "in", "cnt" : 1 }
{ "_id" : "rain", "cnt" : 1 }
{ "_id" : "spain", "cnt" : 2 }
{ "_id" : "the", "cnt" : 1 }
```

As you can see, the word **spain** occurs 2 times.

MapReduce

Doing this with mapReduce basically does the same thing. We first create (key,value) pairs (word, 1) using the **emit** function. Then we use the JavaScript **Array.sum** function to sum the number 1.

```
db.words.mapReduce(
function() { emit(this.word,1); },

   function(key, values) {return Array.sum(values)}, {
query:{},

   } out:"total_matches"
).find()
```

```
{ "_id" : "in", "value" : 1 }
{ "_id" : "rain", "value" : 1 }
{ "_id" : "spain", "value" : 2 }
{ "_id" : "the", "value" : 1 }
```

The result is the same.

Additional Resources

Mongodb Aggregation Pipeline from zahid-mian