**MongoDB Operations**

**Validations Supported by MongoDB**

MongoDB will only enforce a few rules, which means we’ll need to make sure data is valid client-side before saving it.

1: No other document shares same \_id

2: No syntax errors

3: Document is less than 16mb

Let’s exercise with some more recipes.

**Recipe 1: Introducing Projections**

find() takes a second parameter called a “projection” that we can use to specify the exact fields we want back by setting their value to true.

So let’s start, insert some documents in employees collection as below.

db.employees.insert(

{"name": "Dave",

"company": "TCS",

"age": 22,

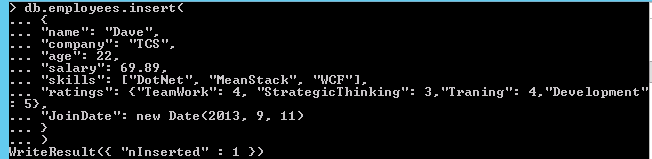
"salary": 69.89,

"skills": ["DotNet", "MeanStack", "WCF"],

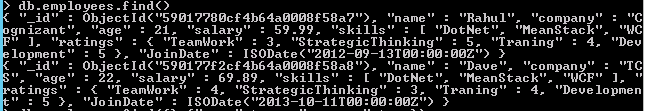
"ratings": {"TeamWork": 4, "StrategicThinking": 3,"Traning": 4,"Development": 5},

"JoinDate": new Date(2013, 9, 11)}

)



db.employees.find() // It will return all documents with all fields.



db.employees.find({},{"name":true, "company":true}) // It will only retrieve what ’s needed

C:\Users\462439\Desktop\3.PNG

db.employees.find({"age": {"$gte": 22}},{"name":true, "company":true})

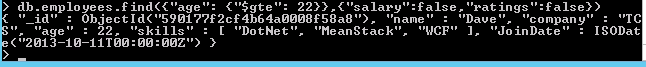
C:\Users\462439\Desktop\4.PNG

Here we can see when selecting fields, all other fields except the \_id are automatically set to false.

**Excluding Fields**

Sometimes you want all the fields except for a few. In that case, we can exclude specific fields.

db.employees.find({"age": {"$gte": 22}},{"salary":false,"ratings":false})



When excluding fields, all fields but those set to false are defaulted to true, it is great for removing sensitive data.

**Excluding the \_id**

The \_id field is always returned whenever selecting or excluding fields. It’s the only field that can be set to false when selecting other fields.

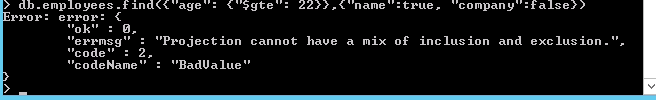
C:\Users\462439\Desktop\6.PNG

This is the only time we can mix an exclusion with selections. Removing the id is common when preparing data reports for non-developers.

**Either Select or Exclude Fields**

Whenever projecting, we either select or exclude the fields we want — we don’t do both.

db.employees.find({"age": {"$gte": 22}},{"name":true, "company":false})



**Recipe 2:** **Counting Our Documents**

Whenever we search for documents, an object is returned from the find method called a “cursor object.” And By default, the first 20 documents are printed out.

{"\_id": ObjectId(...), ... } {"\_id": ObjectId(...), ... } {"\_id": ObjectId(...), ... }….

When there are more than 20 documents, the cursor will iterate through them 20 at a time. Typing “it” will display the next 20 documents in the cursor.

**db.employees.find().count() //**Method on cursor that returns the count of matching documents.

**Recipe 3:** **Basic Pagination**

We can implement basic pagination by limiting and skipping over documents. To do this, we’ll use the skip() and limit() cursor methods.

db. employees.find().skip(3).limit(3)// It will skip first 3 documents and show next 3 documents only.

**Recipe 4:** **Introducing the Aggregation Framework**

The aggregation framework allows for advanced computations. “Aggregate” is a fancy word for combining data.

db.employees.aggregate([{"$group": {"\_id": "$company"}}])

“$group”: Stage operator that’s used to group data by any field we specify.

“$Company”: Field names that begin with a “$” are called "field paths” and are links to a field in a document.

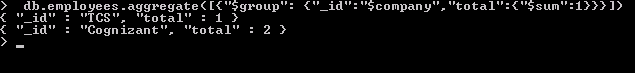
"\_id": This is known as the “group key” and is required.

**Recipe 5:** **Using Accumulators**

Anything specified after the group key is considered an accumulator. Accumulators take a single expression and compute the expression for grouped documents.

db.employees.aggregate([{"$group": {"\_id": "$company","total": {"$sum": 1}}}])

"$sum": This is Accumulator and will add 1 for each Accumulator matching document.



**Summing the salary Per company**

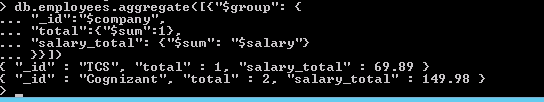
db.employees.aggregate([{"$group": {

"\_id":"$company",

"total":{"$sum":1},

"salary\_total": {"$sum": "$salary"}

}}])



**Averaging Salary Per Company**

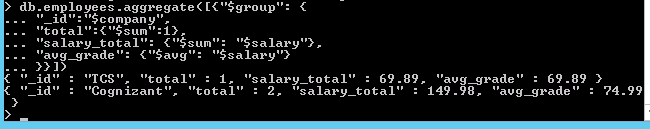
db.employees.aggregate([{"$group": {

"\_id":"$company",

"total":{"$sum":1},

"salary\_total": {"$sum": "$salary"},

"avg\_grade": {"$avg": "$salary"}

}}])

**Returning the Max Salary per Company**

db.employees.aggregate([{"$group": {

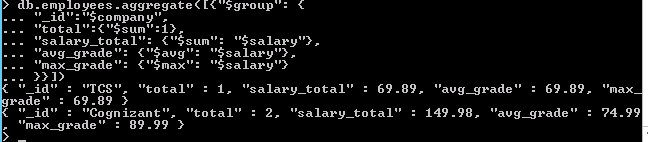
"\_id":"$company",

"total":{"$sum":1},

"salary\_total": {"$sum": "$salary"},

"avg\_grade": {"$avg": "$salary"},

"max\_grade": {"$max": "$salary"}}}])



**Using $max and $min Together**

db.employees.aggregate([{"$group": {

"\_id":"$company",

"total":{"$sum":1},

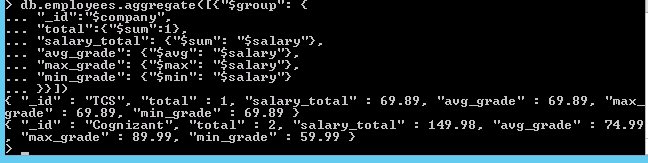
"salary\_total": {"$sum": "$salary"},

"avg\_grade": {"$avg": "$salary"},

"max\_grade": {"$max": "$salary"},

"min\_grade": {"$min": "$salary"}

}}])



**Pulling Conditional Company Information**

Using the $match Stage Operator we can filter it out. $match is just like a normal query and will only pass documents to the next stage if they meet the specified condition(s).

db.employees.aggregate([

{"$match": {"age": 29}},

{"$group": {

"\_id":"$company",

"total":{"$sum":1},

"salary\_total": {"$sum": "$salary"},

"avg\_grade": {"$avg": "$salary"},

"max\_grade": {"$max": "$salary"},

"min\_grade": {"$min": "$salary"}

}}])

