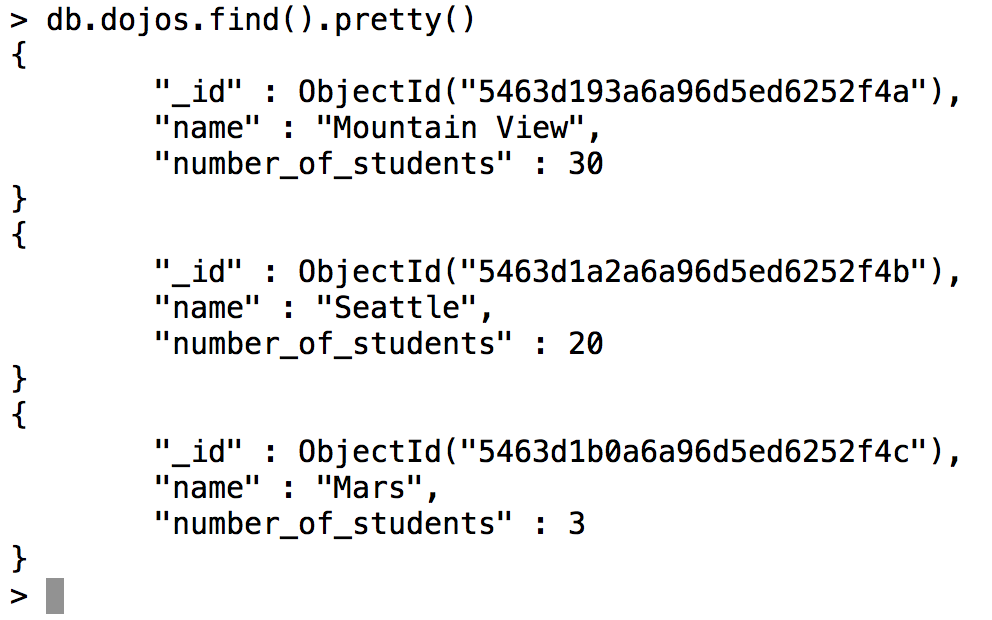
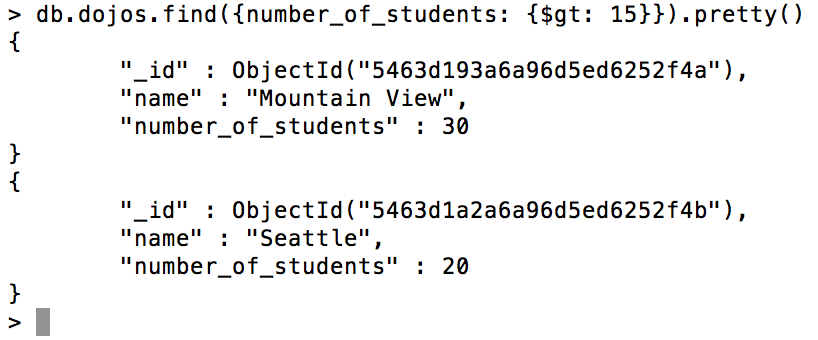
**Operators**

**Operators**are an important part of MongoDB. You've already seen the **$set** operator in the **update method**, so you should be aware of their existence. Since our queries are more **method-based** and not **typed syntax** like regular SQL, we need to pack more functionality into the documents we use to query our databases. Enter operators.  **Operators allow you to 'operate**' on the data (lame description, I know). Let's say I have the following **documents** in a **collection** called **dojos** that looked like this:



If I wanted to get all the Dojos whose **number of students is greater than 15**, I would run the following:

db.dojos.find({number\_of\_students: {$gt: 15}})



The structure of the query should make sense; the extra document that contains the operator **({$gt: 15})** is being applied to the **number of students** field of the **dojos** **collection**.  That key-value pair is the **document** we pass to the**.find() method,** and we are off and running.

**Here is a chart of the most frequently-used operators (take some time to play around with them):**

| **name** | **description** |
| --- | --- |
| $gt (greater than) | Use to query selectively on numerical-valued fields |
| $gte (greater than or equal to) | Use to query selectively on numerical-valued fields |
| $lt (less than) | Use to query selectively on numerical-valued fields |
| $lte (less than or equal to) | Use to query selectively on numerical-valued fields |
| $in (in array) | Use to find documents who have a particular value within an array. |

Check out the full lists of operators [here at MongoDB's official site](http://docs.mongodb.org/manual/reference/operator/" \t "_blank). There are tons to go through, make sure you don't spend more than 20 minutes digging around!

**Operators with arrays**

Perhaps the most important difference in data structures of NoSQL databases compared to traditional SQL is that we have the ability to have a key-value pair where the value can hold multiple pieces of information. If you can think back to the normalization rules of relational databases, one of the key rules was that no column in a table could have multiple values. This doesn't apply to a NoSQL database.Let's say we had a **students** collection with a student that looked like this:



And we wanted to add an element in the interests array. The operator we would use is called the $push operator, which should sound very familiar. To add the interest 'snowboarding' to the student document, we would run...

db.students.update({\_id: ObjectId("5463d871a6a96d5ed6252f4d")}, {$push: {interests: 'snowboarding'}})

The **$push** operator is a key of the update document, and its value is another document with a key of interests, and a value of 'snowboarding'.  Spend some time going over this if you don't understand.  Here are a couple common **array update operators**to get familiar with:

| **Name** | **Description** |
| --- | --- |
| **$push** | Push to an array contained within a document. |
| **$pop** | Removes either the first or last element from an array. EX:  db.COLLECTION.update({QUERY}, {$pop: {array\_key: (1 or -1)}})  Use 1 for the last item in the array, -1 for the first item. |
| **$addToSet** | It functions just like **$push**.  However, **$addToSet** only adds to the specified array if the value doesn't already exist (thereby preventing duplicate entries). |
| **$pull** | Removes a specified value from an array, unlike **$pop**, which removes by location. Ex:  db.COLLECTION.update({QUERY}, {$pull: {array\_key: VALUE}})  This will remove all instances of VALUE from the documents with the array specified by the array\_key that match QUERY. |

There you go! Once you understand the information contained in this section of the course, you're off and running with MongoDB. They have great documentation, but it's pretty extensive. Trust what you've learned and get ready to apply this stuff directly into your projects with Mongoose in the next section!