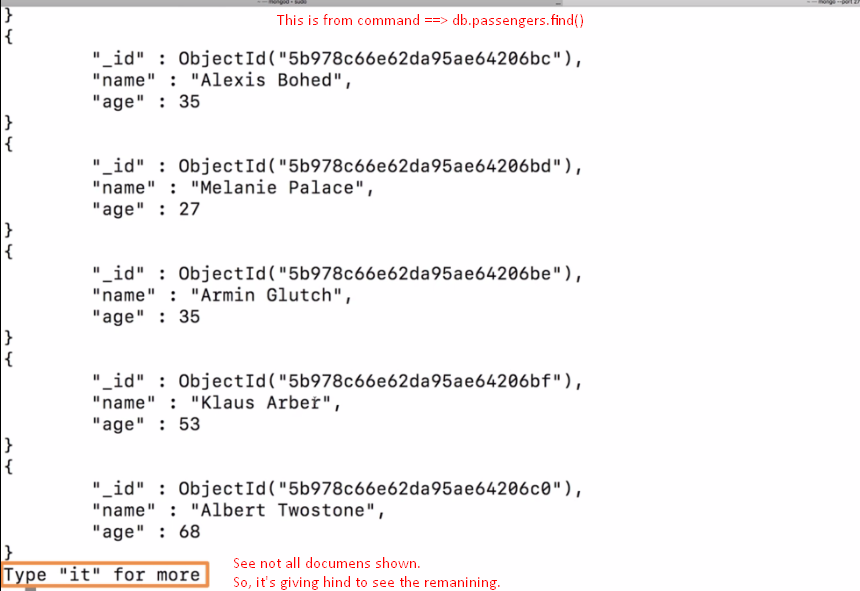
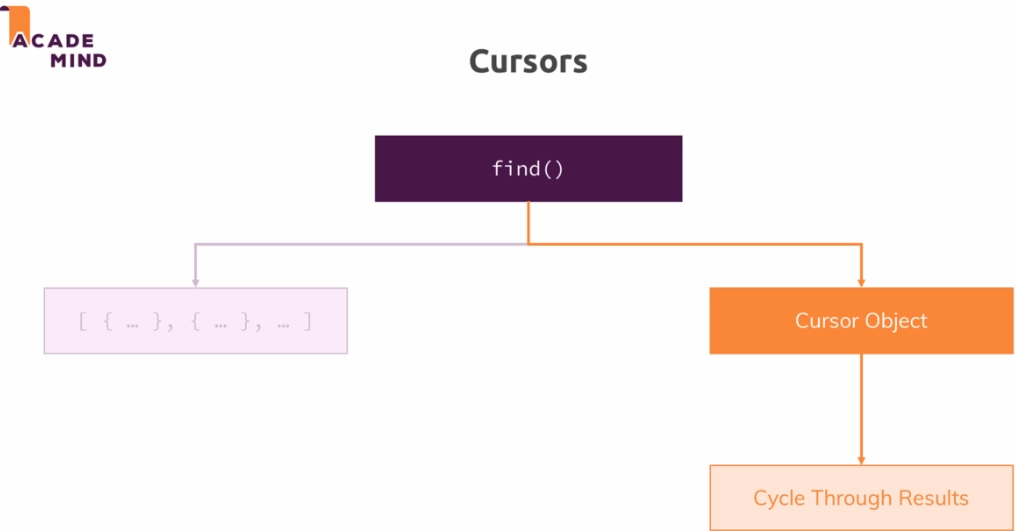
1. **Command**🡺 db.collectionName.insertMany([{d1},{d2}])
2. 
3. 

**NOTE**: This **it** is the feature shown in the shell. But there is another way of getting data if you were using a mongodb driver in an app in a second.   
**NOTE**: By default, first 20 documents are returned.

1. Db.passengers.find() returns **cursor object** not data for each collection. Find() doesn’t give us an array of all the documents in a collection and that makes sense a lot of sense because that collection could be very big and if find() would immediately send us back all the documents and you think about a collection with let’s say 20 million documents, then this would take super long but most importantly, it would send a lot of data over the wire.
2. So instead of that, it gives us back **cursor object** which is an object with a lot of **metadata** behind it that allows us to **cycle through the results** and that is what **“it”** **command** did. It basically used that cursor to fetch the next bunch of data. 
3. **Command**🡺db.passengers.find().toArray()  
   toArray() actually goes ahead and exhaust the cursor so go through the entire list and fetch all the documents and not stop after the first 20, which by the way is simply a feature by the mongodb shell. It gives you the first 20 documents automatically but then stops, toArray() simply gets them all and gives you an array.  
   But that is not optimal if you’ve got a lot of elements.
4. In your app code, you would most likely use something like **db.passengers.find().forEach(lambda)**; which allows you to write some code to do something on every element in collection. forEach syntax depends on the mongodb driver. But in javascript and the shell is based on javascript, you could pass a javascript function to it using an arrow function here where you will get an argument.  
   
5. forEach(array\_function): Doesn’t load all the document in memory but on demand. Thus saving your bandwidth and memory.
6. 