Getting Started

1. Download and install the MongoDB software
   – The build is unique per platform: Mac, Windows, Linux

2. Import the restaurants dataset (per instructions in HW # 6)

3. In your terminal, enter "mongo" to launch the MongoDB console

Querying MongoDB

In your terminal, you can see what's Mongo keeping track of:

```
show databases
```

Gives you a list of all the databases within this mongo instance.

```
use <database>
```

Tells mongo which database you want to query

```
show collections
```

Gives you a list of all the collections in this database

Querying MongoDB

```
db.stats()
```

Tells you some interesting things about the database.

```
db
```

Tells you which database you're using.

Querying MongoDB

```
db.<collection>.find()
```

The `find()` method prints out all the documents in the collection in an unformatted view

```
db.restaurants.find()
```

```
Type "it" for more
```

This means that the command you gave mongo produced more output than will print out on your console screen. "it" means "iterate".

Querying MongoDB

```
db.restaurants.find().pretty()
```

The `pretty()` method shows the command output in a more structured JSON format

```
db.restaurants.findOne()
```

The `findOne()` method prints out just one document.

(like `limit 1` in SQL)

The `findOne()` method comes out "pretty"

Querying MongoDB

```
db.restaurants.find({"name":"Riviera Caterer"})

db.restaurants.find({"name":"Riviera Caterer"}).pretty()

db.restaurants.find({"cuisine":"Italian"})
```

Adding a key/value within the find() method finds all the documents that match the criteria given.

Must be enclosed in `{ }`

```
db.restaurants.find({"cuisine":"Irish"}).count()
```

The `count()` method counts matching documents and returns just the count.

Querying MongoDB

Like the SQL WHERE clause, you can use different conditions to match

```
$lt
$lte
$gt
$gte
$ne
$in

db.restaurants.find({"cuisine":{$in:["Italian",
      "Indian", "Iranian", "Icelandic"]}}).pretty()
```

Querying MongoDB

You can "and" multiple conditions together (or you can `$or`)

```
db.restaurants.find({$and:[{"cuisine":"Italian"},
      {"borough":"Queens"}]}).pretty()
```

The `find()` method will return ALL fields in a document unless you specify one or more fields to return

```
db.restaurants.find({"borough":"Queens"},{"borough":1,
"cuisine":1})
```

If you select certain fields, it will always return the _id field unless you tell it NOT to do so.

```
db.restaurants.find({"borough":"Queens"},{"_id":0,"bor
ough":1,"cuisine":1})
```

Querying MongoDB

Finding a string within a field: use REGEX

```
db.restaurants.find({"name":"{$regex: /pattern/, $options:
'<options>' } }"})
```

`/pattern/ =` any regex string pattern

Options

`i`	Case insensitivity to match upper and lower cases.
`m`	For patterns that include anchors (i.e. ^ for the start, $ for the end)
`s`	Allows the dot character (i.e. `.`) to match all characters *including* newline

Querying MongoDB

Finding a string within a field: All restaurants with "burger" in their name

Add option for case insensitivity

```
db.restaurants.find({"name":{$regex: /burger/}}).pretty()

db.restaurants.find({"name":{$regex: /Burger/}}).pretty()

db.restaurants.find({"name":{$regex: /Burger/}},{"name":1,_id:0})
.pretty()

db.restaurants.find({"name":{$regex: /burger/, $options:"i"}},
{"name":1,_id:0}).pretty()
```

Querying MongoDB

The `$limit()` method works just like in SQL

```
db.restaurants.find({"borough":"Queens"},{"_id":0,"bor
ough":1,"cuisine":1}).limit(4)
```

The `$sort()` method works like `ORDER BY` in SQL

```
db.restaurants.find({"borough":"Queens"},{"_id":0,"bor
ough":1,"cuisine":1}).sort({"cuisine":1})
```

```
db.restaurants.find({"borough":"Queens"},{"_id":0,"bor
ough":1,"cuisine":1}).sort({"cuisine":-1})
```

Some fields have fields within:

```
"address" : {
      "building" : "4277",
      "coord" : [
            -73.8675389,
            40.8977829
      ],
      "street" : "Katonah Ave",
      "zipcode" : "10470"
},
```

Finding a field within a field:

```
db.restaurants.find({"address.zipcode":"10305"},{"_id"
:0,"borough":1,"cuisine":1})

db.restaurants.find({"address.zipcode":"10305"},{"_id"
:0,"address.coord":1,"cuisine":1})
```

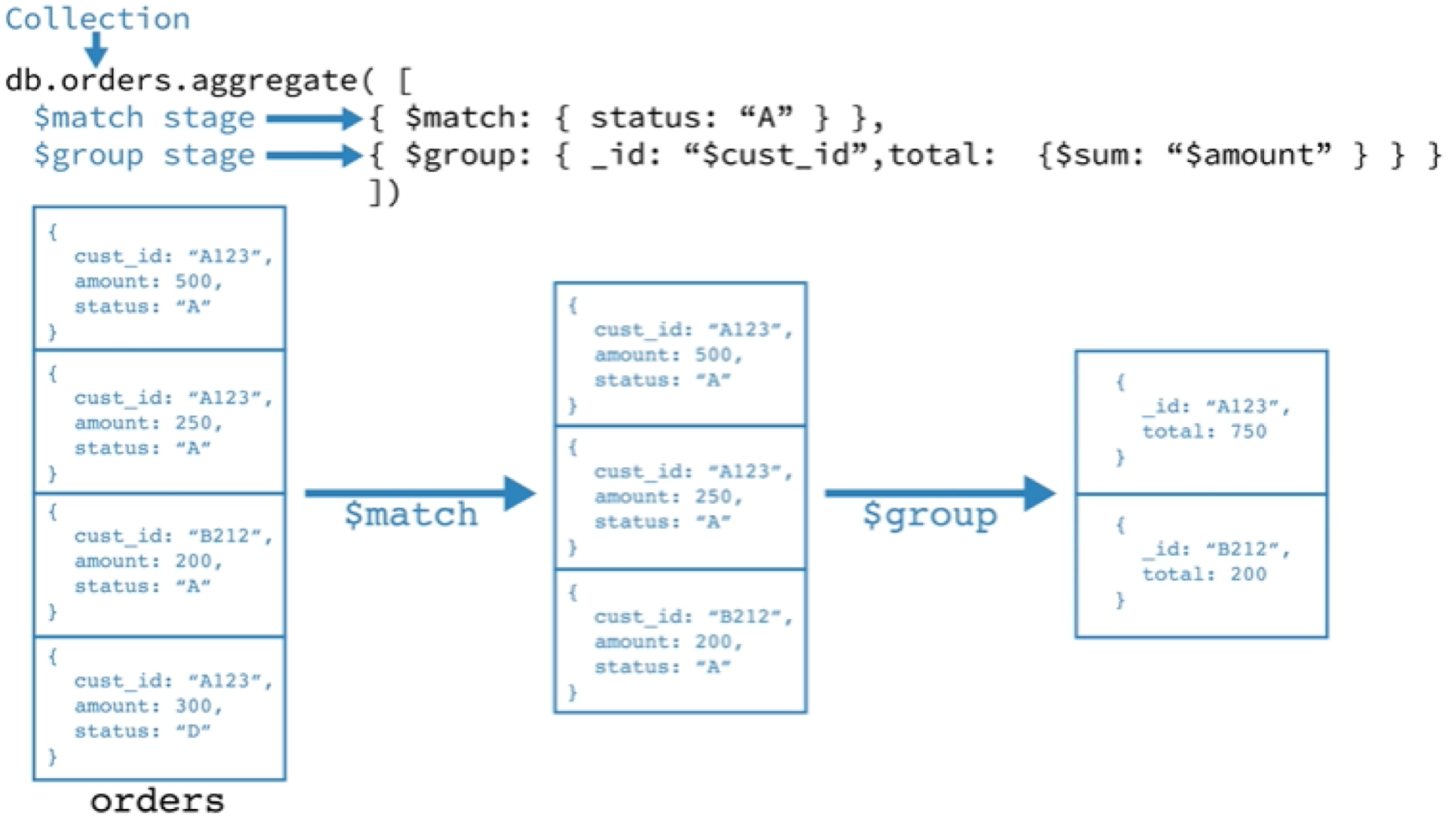
Querying MongoDB

OK. That was simple stuff. Now, it gets more complex.

The `aggregate()` method is used to pipe results.

With the `aggregate()` method the output of one mongo query step becomes the input to the next mongo query step.

Steps are chained together in a pipeline.



Aggregate pipeline operation:

```
db.orders.aggregate([
      { $match: { status: "A" } },
      { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

The `aggregate()` method offers many options for processing steps in the pipeline.

Pipe operators

Common pipes	analysis
$group	Group documents in collection, which can be used for statistics
$match	Filter the data and output only the documents matching the results
$project	Modify the structure of the input document (e.g. rename, add, delete fields, create settlement results, etc.)
$sort	Output after sorting the results
$limit	Limit the number of results for pipeline output
$skip	Skip the result of specified quantity and return the rest
$unwind	Split fields of array type

Expression operators

Common expressions	Meaning
$sum	Calculate the sum, and {& dollar; sum: 1} represents the value of the returned sum × 1 (that is, the number of sums). Using {& dollar; sum: '& dollar; specified field'} can also directly obtain the sum of the values of the specified field
$avg	Average value
$min	Seek min value
$max	Seek max value
$push	Insert values into an array in the result document
$first	Get the first document data according to the sorting of documents
$last	Similarly, get the last data

Querying MongoDB

Mongodb aggregation operation	MySQL operations / functions
$match	where
$group	group by
$match	having
$project	select
$sort	order by
$limit	limit
$sum	sum()
$lookup	join

MongoDB Query language – aggregate examples

```
db.restaurants.aggregate([{$match:{"cuisine":"Irish"}}])

db.restaurants.aggregate([{$match:{"cuisine":"Irish"}}]).pretty()

db.restaurants.aggregate( [
   { $group: { _id: "$cuisine", totalCount: { $sum: 1 } } },
   { $match: { totalCount: { $gte: 1000 } } }
] )

   db.restaurants.aggregate( [
      { $match: { "cuisine":"Italian"} },
      { $project: { _id:0, borough:1, name:1} }
   ] )
```

MongoDB Query language – aggregate example: unwinding an array

```
db.restaurants.find({"borough":"Brooklyn"}).pretty()

db.restaurants.find(
    {"borough":"Brooklyn"},
    {"_id":0,"name":1,"grades":1}
).pretty()

db.restaurants.aggregate([
    { $match: {"borough":"Brooklyn"}},
    { $project: { _id:0, name:1, grades:1} }
]).pretty()

db.restaurants.aggregate([
    { $match: {"borough":"Brooklyn"}},
    { $unwind: "$grades"},
    { $project: { _id:0, name:1, grades:1} }
]).pretty()
```

MongoDB Query language – aggregate example: unwinding an array

```
db.restaurants.aggregate([
    { $match: {"borough":"Brooklyn"}},
    { $unwind: "$grades"},
    { $group: { _id: "$name", totalScores:
       { $sum: "$grades.score" } } },
    { $sort: {totalScores:-1}}
])
```