Advanced MongoDB

Gregory S. DeLozier, Ph.D.
Kent State University

Document Structures

```
{
     first_name: “Paul”,
     surname: “Miller”,
     city: “London”,
     location: [45.123,47.232],
     cars: [
        { model: “Bentley”,
          year: 1973,
          value: 100000, ….},
        { model: “Rolls Royce”,
          year: 1965,
          value: 330000, ….},
     ]
}
```

PyMongo

```
import pymongo

client = pymongo.MongoClient()

db = client.<database_name>

db = client.stuff
```

PyMongo Insertion

```
import datetime

item = {"author": "Mike",
    "text": "My first blog post!",
    "tags": ["mongodb", "python", "pymongo"],
    "date": datetime.datetime.utcnow()
    }

db.stuff.insert(item)
```

PyMongo Find_One

```
db.stuff.find_one()

db.stuff.find_one({"author":"Mike"})

db.stuff.find_one({"_id":id}) #<---- not string
```

PyMongo Find (List)

```
db.stuff.find ()

db.stuff.find ({"author":"Mike"})

db.posts.find({"date": {"$lt": d}}).sort("author")
```

PyMongo save/update/delete

```
db.stuff.save (<item>)

db.stuff.update ({"author":"Mike"}, {"$set":{age:44}})

db.stuff.remove({"author":"Mike"})
```

Aggregation

Aggregation

We want to combine results for a large number of records

- Grouping - creating groups of records to be processed together
- Matching - processing a group of records

These are called "stages" of "aggregation"
A sequence of stages is called an "aggregation pipeline"

Simple Aggregation

* db.collection.count(query, options)
  db.collection.find(query).count()

* db.collection.distinct(field, query)

Distinct Example

```
{ "_id": 1, "dept": "A", "item": { "sku": "111", "color": "red" }, "sizes": [ "S", "M" ] }
{ "_id": 2, "dept": "A", "item": { "sku": "111", "color": "blue" }, "sizes": [ "M", "L" ] }
{ "_id": 3, "dept": "B", "item": { "sku": "222", "color": "blue" }, "sizes": "S" }
{ "_id": 4, "dept": "A", "item": { "sku": "333", "color": "black" }, "sizes": [ "S" ] }
```

* `db.inventory.distinct( "dept" )` → `[ "A", "B" ]`

General Aggregator

* General purpose aggregation machine
* Has "processing pipeline" by stages
* Details here:

https://docs.mongodb.com/manual/reference/method/db.collection.aggregate

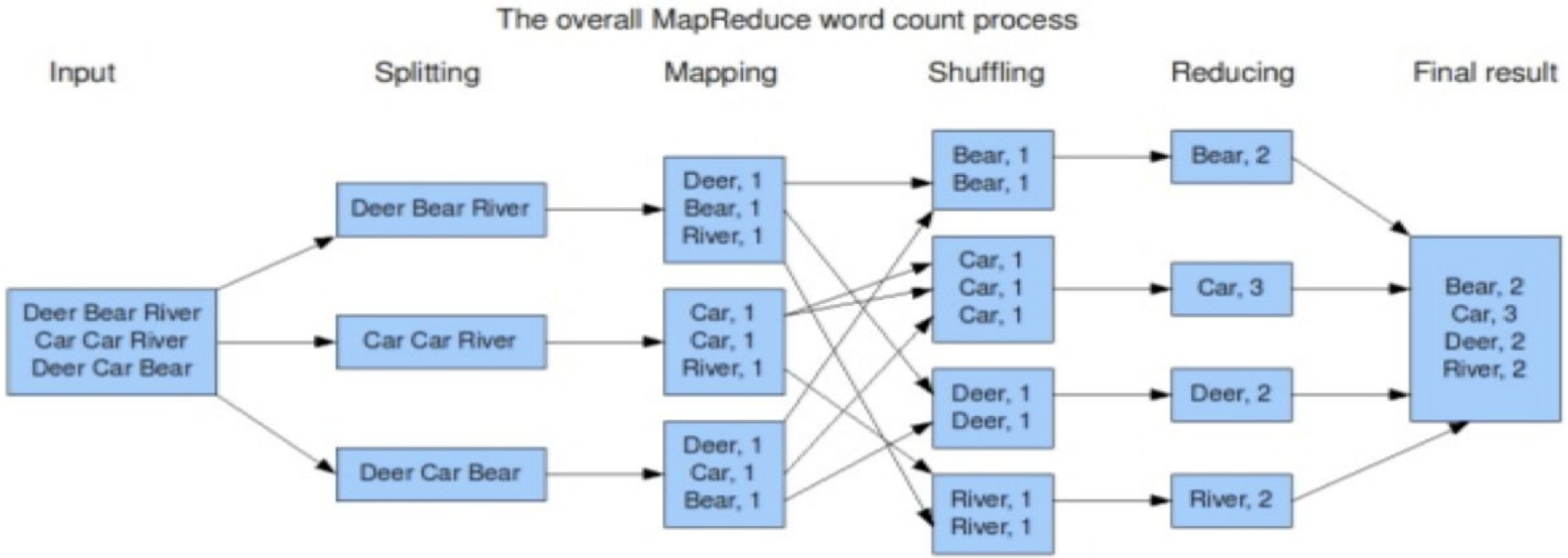
Aggregation Resources

- Based on material here:
  - https://docs.mongodb.org/manual/aggregation/
- Zip Code examples here:
  - https://docs.mongodb.org/manual/tutorial/aggregation-zip-code-data-set/

MapReduce

- Distributed query system
- Splits the whole search space into parts and distributes the parts
- *Maps* those parts onto partial solutions with keys
- Shuffles the elements of the solutions by key values
- *Reduces* all of the partial solutions with the same key into a key result
- Combines all of the key results into a final result

A MapReduce Example



Elements of Map/Reduce

Mapper

- Gets elements of a solution (lines, regions, groups, whatever)
- Emits Key/Value pairs

Reducer

- Gets Key,[Value,Value,Value] sets
- Emits Key/KeyResult pairs

Example Mapper

```
def mapper(self, _, line):
    for word in line.split():
        yield (word,1)
```

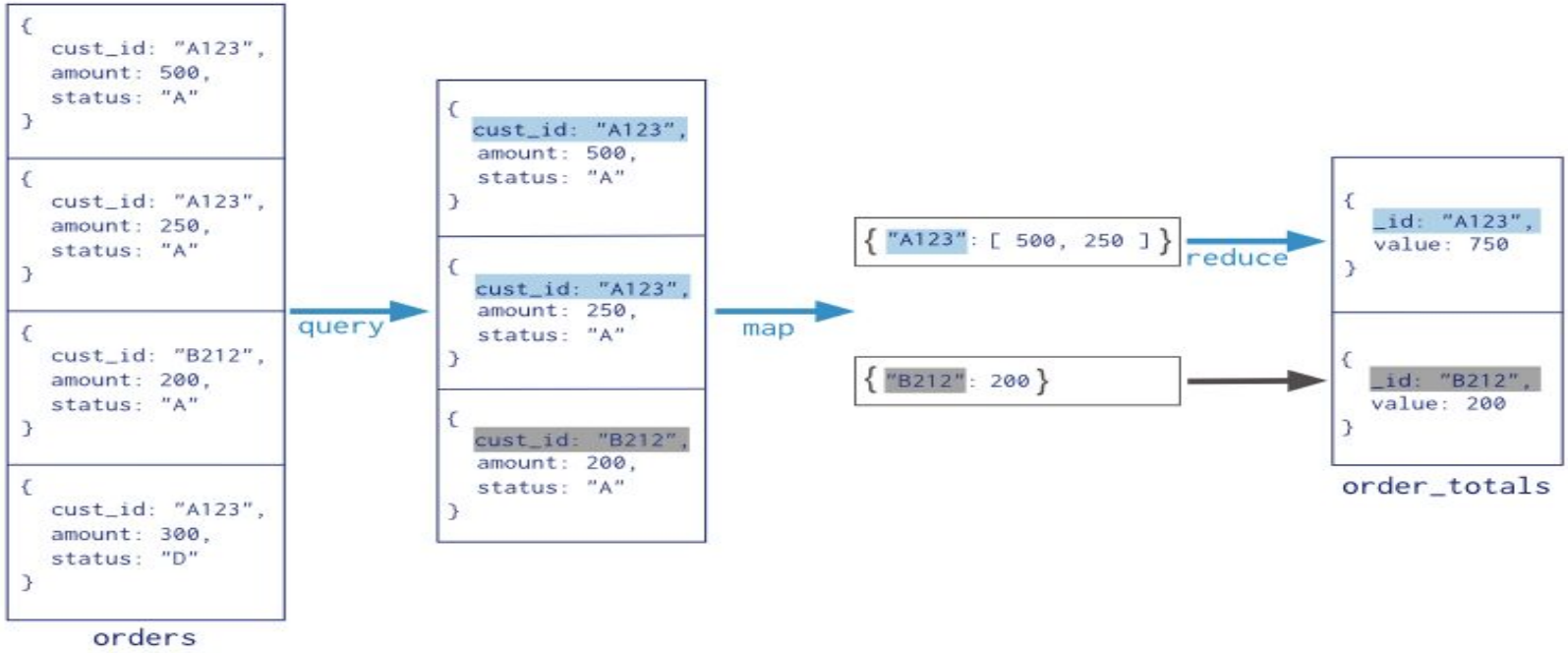
Example Reducer

```
def reducer(self, key, values):
    yield key, sum(values)
```

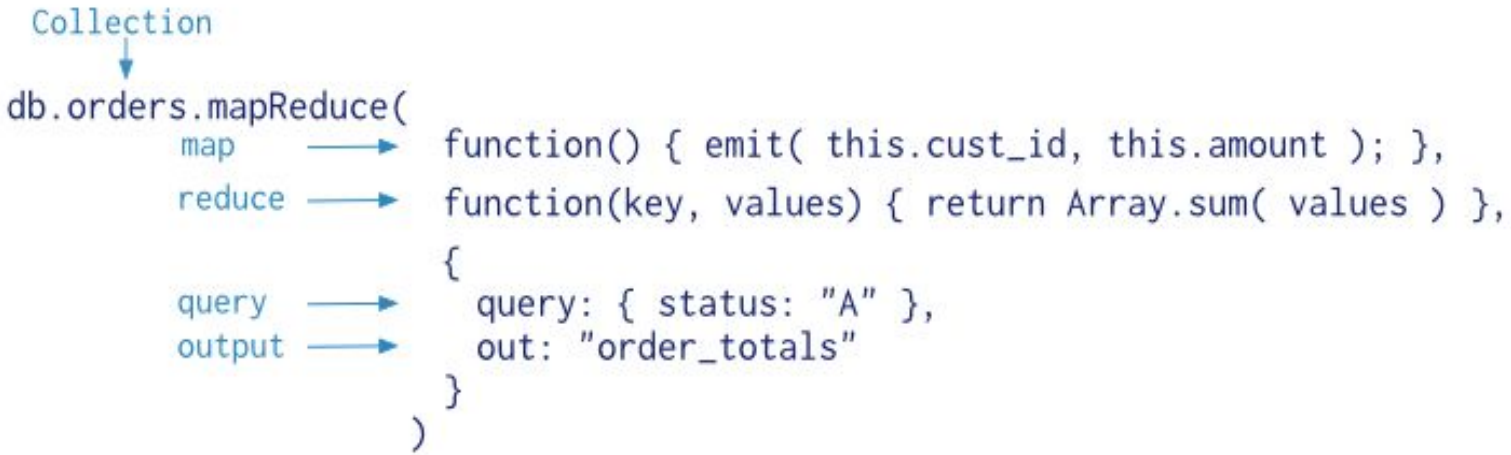
Python Demonstration

Demo Time!

MapReduce in Mongo



Mongo MapReduce Command



Mongo MapReduce in Python

```
>>> from bson.code import Code
>>> mapper = Code("""
...               function () {
...                 this.tags.forEach(function(z) {
...                   emit(z, 1);
...                 });
...               }
...               """)
```

Reducer Function in Python

```
>>> reducer = Code("""
...               function (key, values) {
...                 var total = 0;
...                 for (var i = 0; i < values.length; i++) {
...                   total += values[i];
...                 }
...                 return total;
...               }
...               """)
```

Running MapReduce

```
>>> result = db.things.map_reduce(mapper, reducer, "myresults")
>>> for doc in result.find():
...   print doc
```