MongoDB-3

WPI, Mohamed Eltabakh

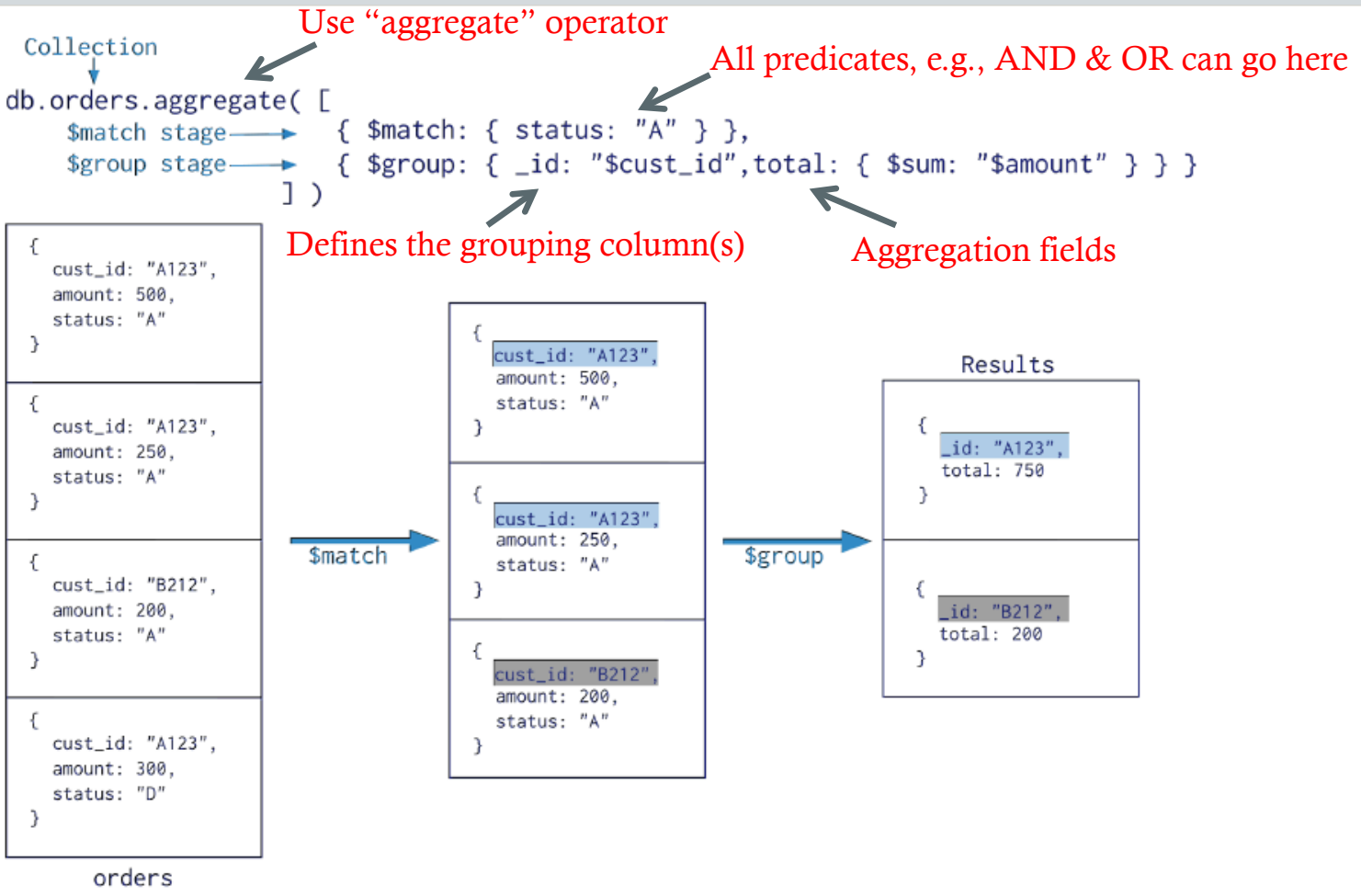
Aggregation in MongoDB (Chapter 7)

Aggregation Mechanisms

- **Aggregation Pipeline**
  - Documents go through a pipeline of operators until aggregated

- **Map-Reduce Model**

Aggregation Pipeline



Aggregation Function

Name	Description
`$sum`	Returns a sum for each group. Ignores non-numeric values.
`$avg`	Returns an average for each group. Ignores non-numeric values.
`$first`	Returns a value from the first document for each group. Order is only defined if the documents are in a defined order.
`$last`	Returns a value from the last document for each group. Order is only defined if the documents are in a defined order.
`$max`	Returns the highest expression value for each group.
`$min`	Returns the lowest expression value for each group.
`$push`	Returns an array of expression values for each group.
`$addToSet`	Returns an array of *unique* expression values for each group. Order of the array elements is undefined.

Example 1

```
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
```

For each day, get the:

- TotalPrice ← Sum (Price * Quantity)
- average quantity
- Count

```
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
```

```
db.sales.aggregate([
    {$group : {_id : {  month: { $month: "$date" },
                        day: { $dayOfMonth: "$date" },
                        year: { $year: "$date" } },
       totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] } },
       averageQuantity: { $avg: "$quantity" },
       count: { $sum: 1 }
     }}])
```

Example 2

```
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
{ "_id" : 4, "item" : "xyz", "price" : 5, "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
{ "_id" : 5, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") }
```

Get the distinct values of the items…

```
db.sales.aggregate([ {$group : {_id : "$item" } ])
```

```
db.sales.distinct("item")
```

Group By…Having

- In MongoDB ➔ $match operator after the $group

```
{ "_id": "10280",
  "country": "USA",
  "city": "NEW YORK",
  "state": "NY",
  "pop": 5574,
  "loc": [ -74.016323, 40.710537]
}
```

```
Select *
From collection
Where country = “USA”
Group By state
Having sum(pop) > 10,000,000;
```

For all documents of USA, report the states having total population > 10,000,000

Group By…Having

- In MongoDB ➔ $match operator after the $group

```
{ "_id": "10280",
  "country": "USA",
  "city": "NEW YORK",
  "state": "NY",
  "pop": 5574,
  "loc": [ -74.016323, 40.710537]
}
```

```
Select *
From collection
Where country = “USA”
Group By state
Having sum(pop) > 10,000,000;
```

```
db.zipcodes.aggregate( [
{ $match: { country: "USA" }},
{ $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
{ $match: { totalPop: { $gt: 10*1000*1000 } } }
] )
```

Example 3


```
{ "_id": "10280",
  "country": "USA",
  "city": "NEW YORK",
  "state": "NY",
  "pop": 5574,
  "loc": [ -74.016323, 40.710537]
}
```

```
{ "_id": "10290",
  "country": "USA",
  "city": "NEW YORK",
  "state": "NY",
  "pop": 87652,
  "loc": [ 43.23, 121.53]
}
```


For USA, Report for each state, the average population across its cities

Aggregation Mechanisms

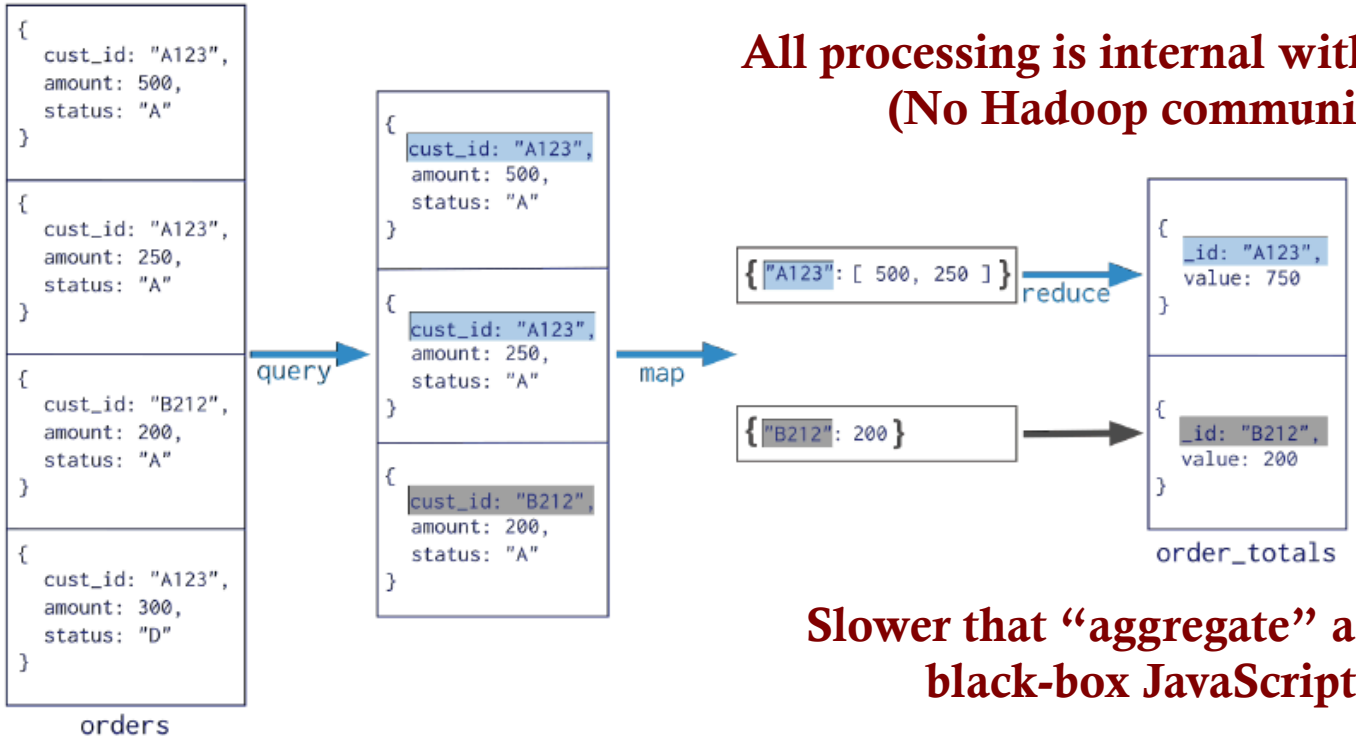
- **Aggregation Pipeline**
  - Documents go through a pipeline of operators until aggregated

- **Map-Reduce Model**
  - Similar concept as Hadoop
  - Uses JavaScript inside the functions

Map-Reduce Model

```
Collection
    ↓
db.orders.mapReduce(
        map      ──→  function() { emit( this.cust_id, this.amount ); },
        reduce   ──→  function(key, values) { return Array.sum( values ) },
                      {
        query    ──→    query: { status: "A" },
        output   ──→    out: "order_totals"
                      }
                  )
```

Map ➔ emits key-value pair



All processing is internal within MongoDB (No Hadoop communication)

Slower that "aggregate" as it involve black-box JavaScript code

Example 1

```
{
    _id: ObjectId("50a8240b927d5d8b5891743c"),
    cust_id: "abc123",
    ord_date: new Date("Oct 04, 2012"),
    status: 'A',
    price: 25,
    items: [ { sku: "mmm", qty: 5, price: 2.5 },
             { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

Group the documents by the customer id, and get the sum of Price

Example 1

```
var mapFunction1 = function() {
                       emit(this.cust_id, this.price);
                   };
```

```
var reduceFunction1 = function(keyCustId, valuesPrices) {
                          return Array.sum(valuesPrices);
                      };
```

```
db.orders.mapReduce(
                      mapFunction1,
                      reduceFunction1,
                      { out: "map_reduce_example" }
                    )
```

Example 2

```
{
     _id: ObjectId("50a8240b927d5d8b5891743c"),
     cust_id: "abc123",
     ord_date: new Date("Oct 04, 2012"),
     status: 'A',
     price: 25,
     items: [ { sku: "mmm", qty: 5, price: 2.5 },
              { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

For each item.sku, report the average quantity across all orders after 1/1/2012

Example 2: Map Function

```
var mapFunction2 = function() {
                    for (var idx = 0; idx < this.items.length; idx++) {
                        var key = this.items[idx].sku;
                        var value = {
                                        count: 1,
                                        qty: this.items[idx].qty
                                    };
                        emit(key, value);
                    }
                };
```

Example 2: Reduce Function

```
var reduceFunction2 = function(keySKU, countObjVals) {
                      reducedVal = { count: 0, qty: 0 };

                      for (var idx = 0; idx < countObjVals.length; idx++) {
                          reducedVal.count += countObjVals[idx].count;
                          reducedVal.qty += countObjVals[idx].qty;
                      }

                      return reducedVal;
                  };
```

Example 2: Final

```
db.orders.mapReduce( mapFunction2,
                     reduceFunction2,
                     {
                       out: { merge: "map_reduce_example" },
                       query: { ord_date:
                                    { $gt: new Date('01/01/2012') }}})
```