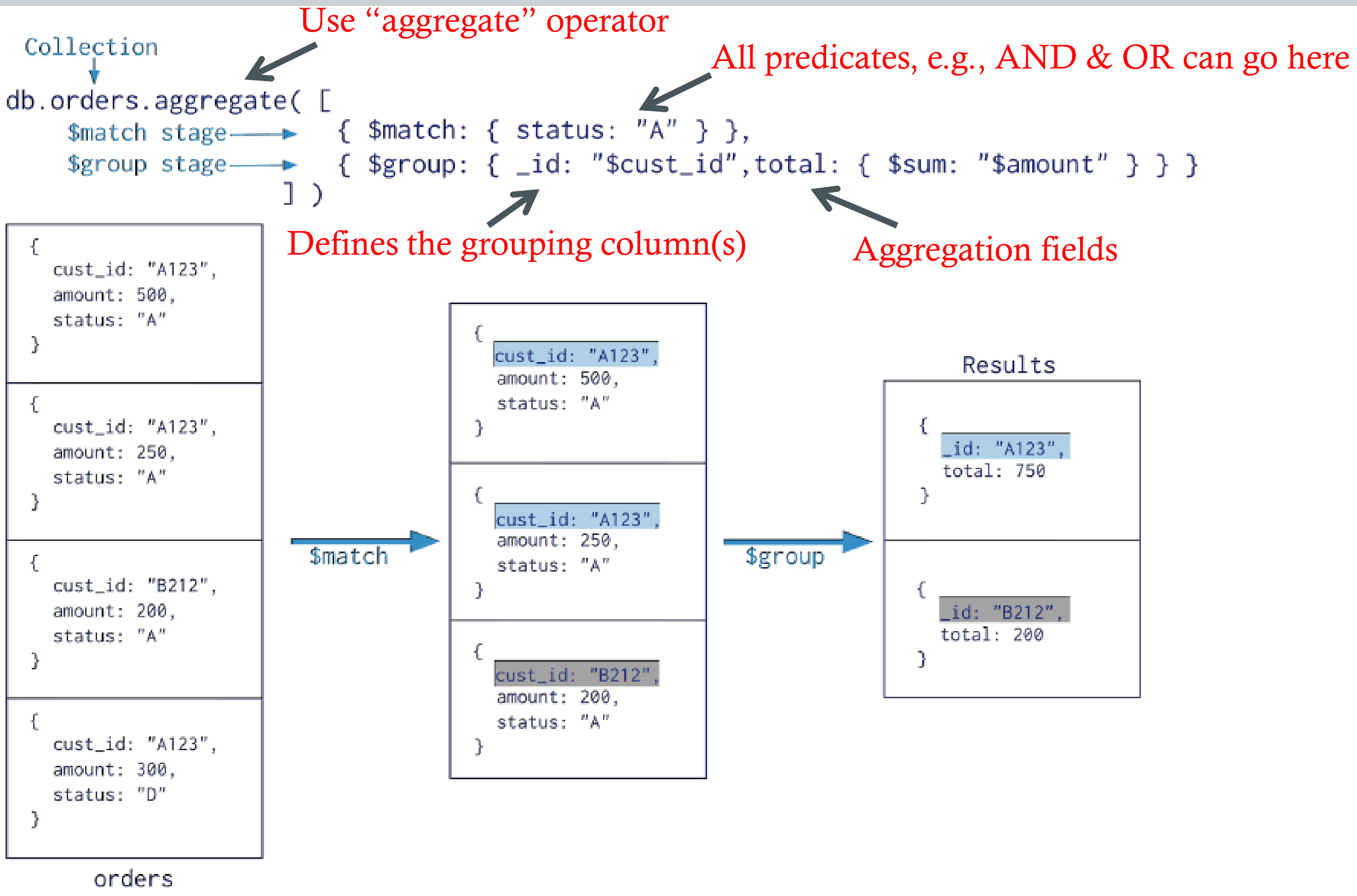
Aggregation in MongoDB

(use for project5)

Aggregation Mechanisms

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

Aggregation Pipeline



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

Use "aggregate" operator
All predicates, e.g., AND & OR can go here
Defines the grouping column(s)
Aggregation fields

```
{                        {                         Results
  cust_id: "A123",         cust_id: "A123",        {
  amount: 500,             amount: 500,              _id: "A123",
  status: "A"              status: "A"               total: 750
}                        }                         }
{                        {                         {
  cust_id: "A123",         cust_id: "A123",          _id: "B212",
  amount: 250,  $match     amount: 250,   $group     total: 200
  status: "A"              status: "A"             }
}                        }
{                        {
  cust_id: "B212",         cust_id: "B212",
  amount: 200,             amount: 200,
  status: "A"              status: "A"
}                        }
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
    orders
```

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 }
     }}])
```

Group By … Having

In MongoDB ➔ $match operator before/after the $group ?

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

SQL QUERY:

Select state, **sum(pop)**
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

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

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

SQL:

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

MongoDB:

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

Group By…Having

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

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

SQL:

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

MongoDB:

```
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.

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

```
{ "_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]
}
```

MongoDB:

```
db.zipcodes.aggregate( [
{ $match: { country: "USA" }},
{ $group: { _id: "$state", avgPop: { $avg: "pop" } } },
] )
```

Example 4

```
{ "_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 each state, return the largest and the smallest city populations.

For each state, return the largest and the smallest city populations.

```
{ "_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]
}
```

```
db.zipcodes.aggregate( [
 $sort: {  "pop"  :  -1 },
{ $group: { _id: "state",
                 minCity:  { $min: "pop" },
                 maxCity: { $max: "pop"},
 }])
```

For each state, return the largest and the smallest city along with their population.

```
{ "_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 each state, return the largest and the smallest city along with their population.

```
{ "_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]
}
```

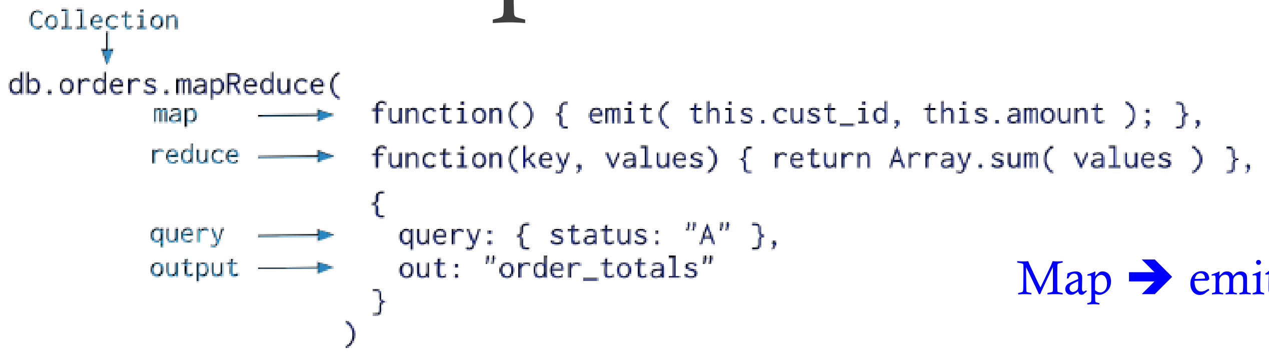
```
db.zipcodes.aggregate( [
 $sort: {  "$pop"  :  1 },
{ $group: { _id: "state",
                minPop:  { $first: "pop" },
                minCity:   { $first: "city" },
                maxPop: { $last: "pop"},
                maxCity:  { $last: "city" }}}] )
```

Aggregation Mechanisms

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

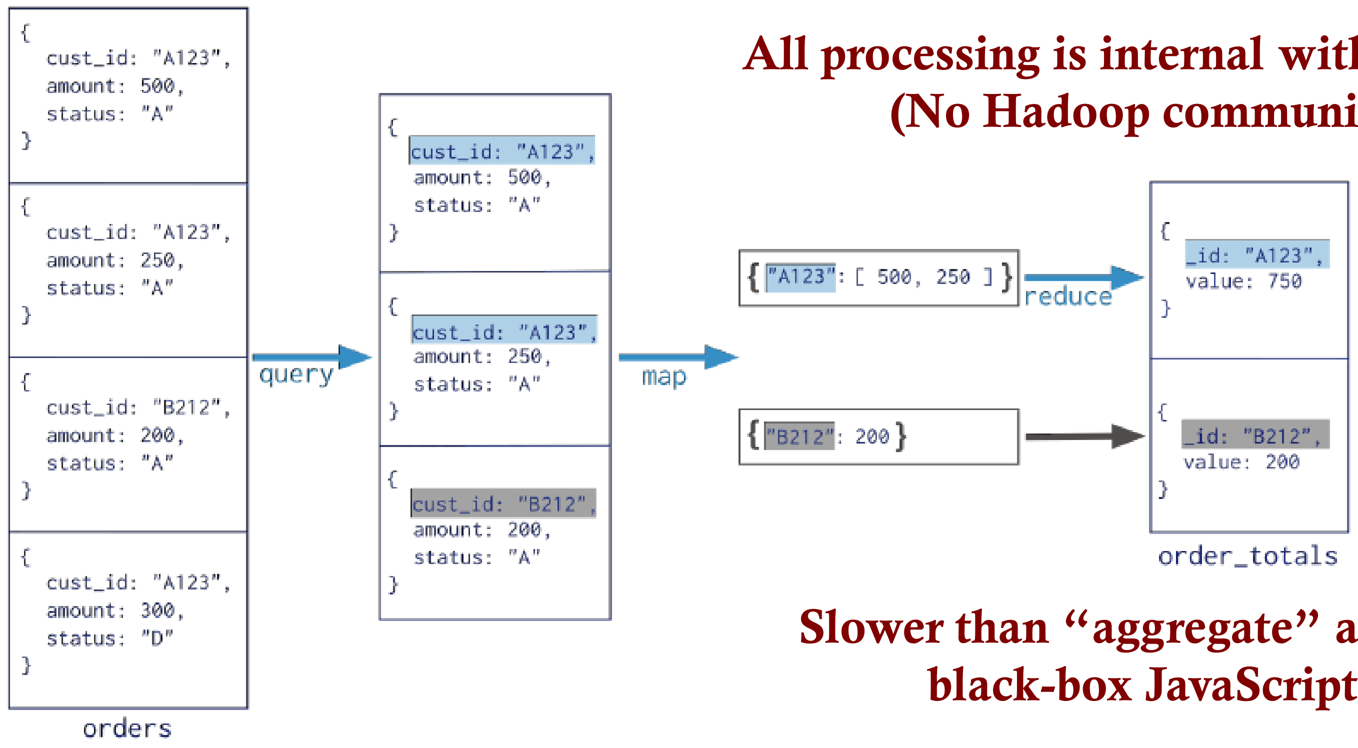
NOTE: Can be used as an alternate to aggregation.

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



{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{"A123": [500, 250]}
{"B212": 200}
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals

All processing is internal within MongoDB (No Hadoop communication)

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

Aggregation Mechanisms

- **Aggregation Pipeline (pure MongoDB query)**

- **Map-Reduce Model (with JavaScript functions)**