MongoDB-Advanced Topics

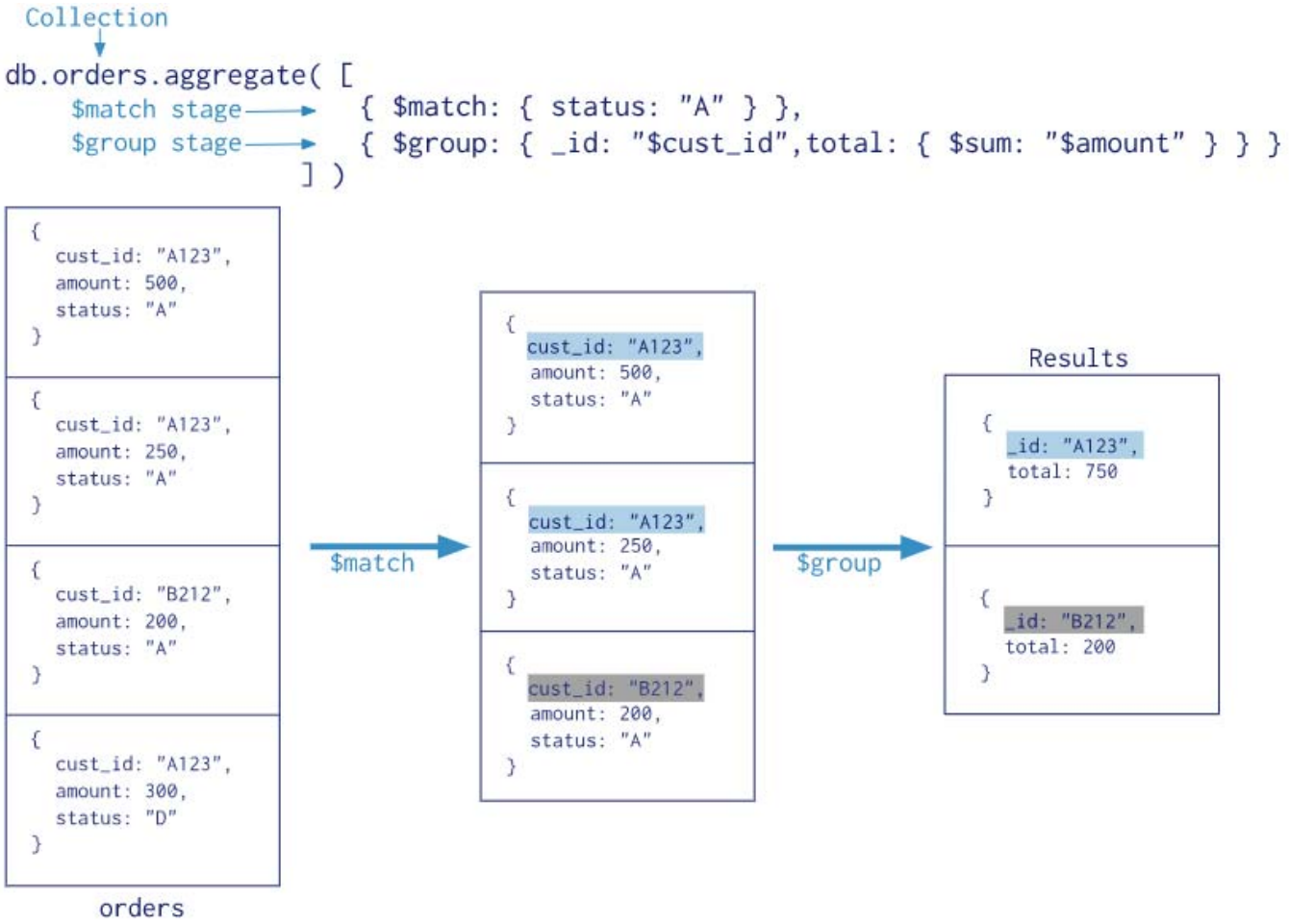
Aggregation

- For the aggregation in MongoDB, use **aggregate()** method

- Basic syntax

db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

MongoDB Aggregation Pipeline



$match stage

- Consider the **orders** collection

```
db.orders.insertMany( [
  { 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" }
] );
```

$match stage

- Run the following command

```
db.orders.aggregate ( [
        { $match : { status : "A" } }
] );
```

which is equivalent to

```
db.orders.find( {status:"A"} )
```

$match acts as a filter to reduce the amount of documents that are given as input to the next stage

$match stage

- More examples

```
db.orders.aggregate ( [
        { $match : { cust_id: “A123“, status : "A" } }
] );

db.orders.aggregate ( [
        { $match : { $or: [ {amount: {$gte: 300}} , {status : "D"}] } }
] );
```

$group stage

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

Equivalent to

```
select      sum(amount) as total
from        orders
group by    cust_id
```

Pipeline

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

Equivalent to

```
select      cust_id, sum(amount) as total
from        orders
where       status= "A"
group by    cust_id
```

Add $sort to pipeline

```
db.orders.aggregate ( [
        { $match : { status : "A" } },
        { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },
        { $sort : { _id : 1 } }
] );
```

$count

```
db.orders.aggregate ( [
        { $match : { status : "A" } },
        { $group: { _id: null, count: { $sum: 1 } } }
] );
```

The _id field is required for grouping and would normally contain fields from each document that we'd like to preserve. We make it null here to just count the number of documents

$sum: 1 count the number of documents

$avg, $min, $max

```
db.orders.aggregate ( [
        { $match : { status : "A" } },
        { $group: { _id: null, avg_amount: { $avg: "$amount" } } }
] );

db.orders.aggregate ( [
        { $group: { _id: "$cust_id", max_amount: { $max: "$amount" } } }
] );
```

Pipeline Optimization

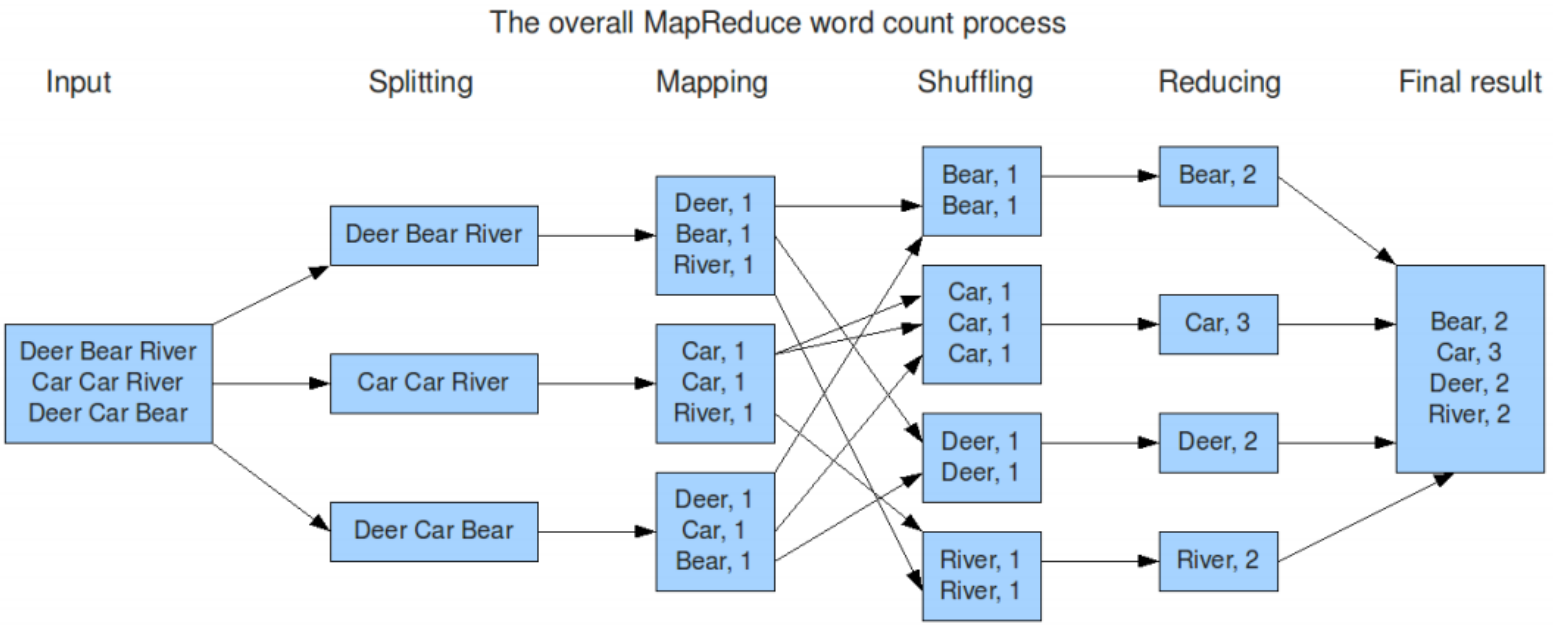
- Place the $match as early in the aggregation pipeline as possible. Because $match limits the total number of documents in the aggregation pipeline, earlier $match operations minimize the amount of processing down the pipe

MapReduce

What is MapReduce

- Origin from Google, [OSDI'04]
- A simple programming model
- For large-scale data processing
- Exploits large set of commodity computers
- Executes process in distributed manner
- Two basic operations on the input
  - Map
  - Reduce

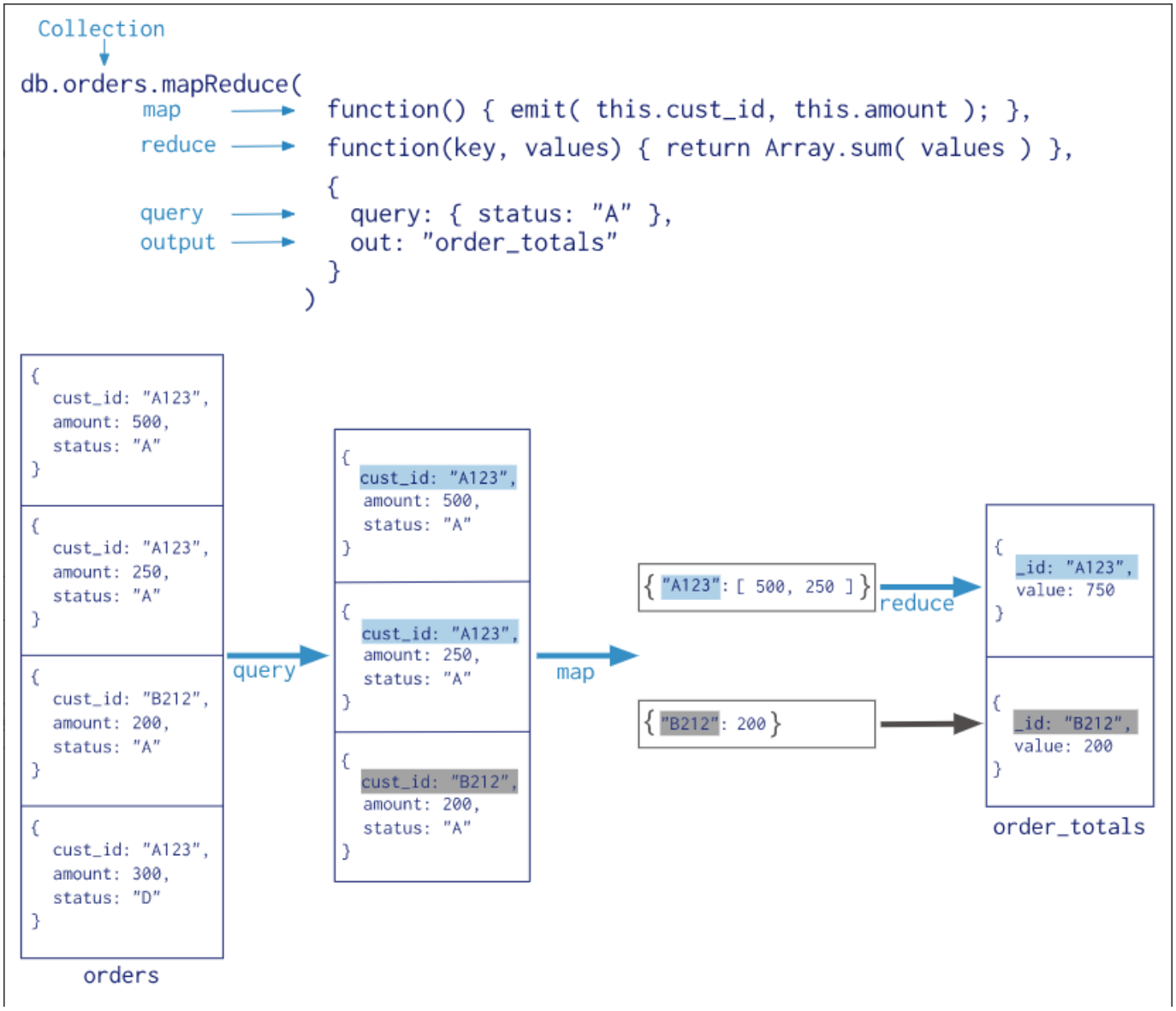
How MapReduce Works



MapReduce in MongoDB

- MongoDB also provides map-reduce operations to perform aggregation
- In the map-reduce operation
  - MongoDB applies the *map* phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits key-value pairs.
  - For those keys that have multiple values, MongoDB applies the *reduce* phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection.

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


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"
}
)
{ 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

MapReduce Output

```
{
    "result" : "order_totals",          // Result stored in collection order_totals
    "timeMillis" : 659,                 // The command execution time in milliseconds
    "counts" : {
        "input" : 3,                    // The number of input documents, which is the number of
                                           times the mapReduce command called the map function
        "emit" : 3,                     // The number of times the mapReduce command called the
                                           emit function
        "reduce" : 1,                   // The number of times the mapReduce command called the
                                           reduce function
        "output" : 2                    // The number of output values produced
    },
    "ok" : 1                            // A value of 1 indicates the mapReduce command ran
                                           successfully. A value of 0 indicates an error
}
```

MapReduce in MongoDB

- MongoDB also provides map-reduce operations to perform aggregation
- In the map-reduce operation
  - MongoDB applies the *map* phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits key-value pairs.
  - For those keys that have multiple values, MongoDB applies the *reduce* phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection.

MongoDB Join

$lookup for Join

- To perform an equality match between a field from the input documents with a field from the documents of the “joined” collection, the $lookup stage has the following syntax:

```
{
  $lookup:
    {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
    }
}
```

MongoDB Data Modeling

Design Principles

- Focus on data usage
  - Data which is accessed together should stay together
  - Need to know some data access patterns / queries before designing

- Field values can be
  - Absent
  - Set to null
  - A single value
  - An array of multiple values (e.g. phone numbers of a person)
    - [6041234567, 7781234567]

Flexible Schema

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection

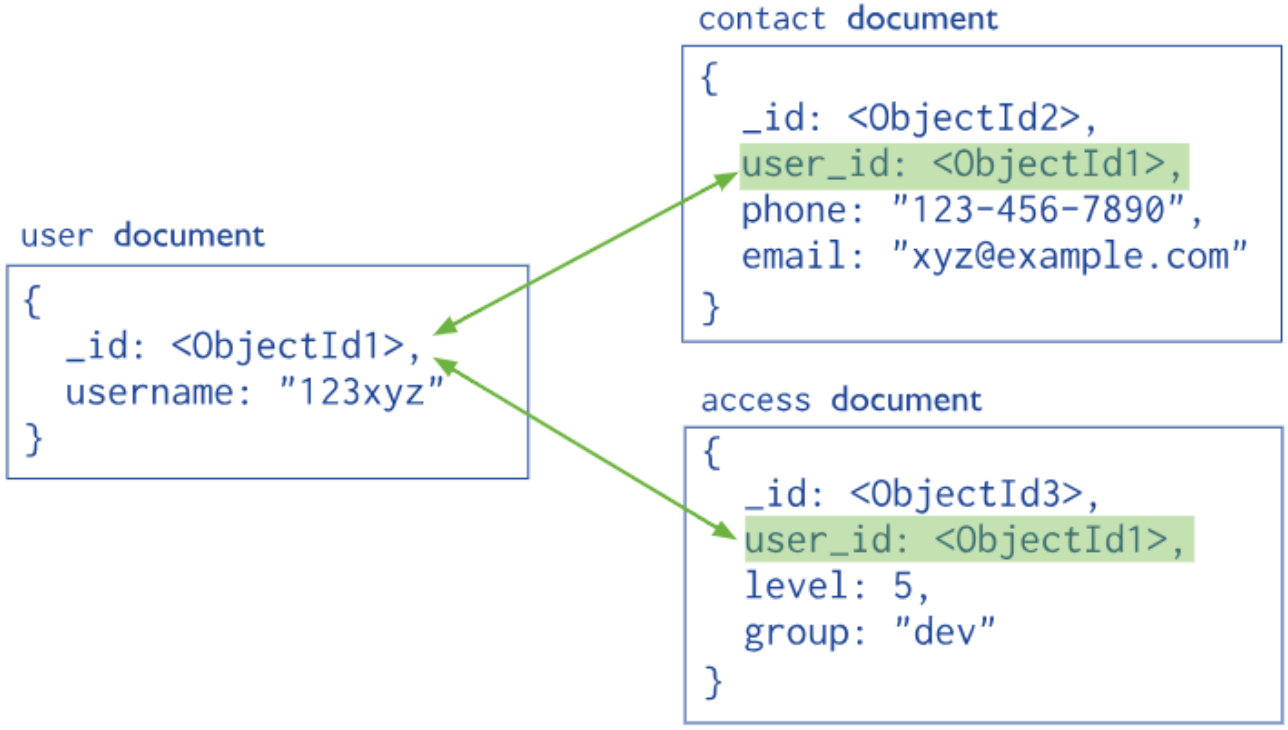
- To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

Embedded Data Model

- Embedded documents capture relationships between data by storing related data in a single document structure
- Denormalized: allow applications to retrieve and manipulate related data in a single database operation
- For many use cases in MongoDB, the denormalized data model is optimal

Normalized Data Models (Referencing)

- Normalized data models describe relationships using references between documents.



1:1 Relationship with Embedded Documents

```
{
  _id: "joe",
  name: "Joe Bookreader"
}

{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```

```
{
  _id: "joe",
  name: "Joe Bookreader",
  address: {
          street: "123 Fake Street",
          city: "Faketon",
          state: "MA",
          zip: "12345"
        }
}
```

1:M Relationship

```
{
 _id: "joe",
 name: "Joe Bookreader"
}

{
 patron_id: "joe",
 street: "123 Fake Street",
 city: "Faketon",
 state: "MA",
 zip: "12345"
}

{
 patron_id: "joe",
 street: "1 Some Other Street",
 city: "Boston",
 state: "MA",
 zip: "12345"
}
```

```
{
 _id: "joe",
 name: "Joe Bookreader",
 addresses: [
  {
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
  },
  {
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
  }
 ]
}
```

M:N Relationships

- No Intersection collection (table)
- Add array in both collection
  - Need to maintain data in both for consistency

Book {

authors: […]

}

Author {

Books: […]

}

http://learnmongodbthehardway.com/schema/schemabasics/

References

https://www.tutorialspoint.com/mongodb/mongodb_aggregation.htm

https://docs.mongodb.com/manual/

https://appdividend.com/2018/10/25/mongodb-aggregate-example-tutorial/

https://appdividend.com/2018/10/26/mongodb-mapreduce-example-tutorial/

http://learnmongodbthehardway.com/schema/schemabasics/