# CSCU9YQ - NoSQL Databases
# Lecture 6.a: Map Reduce

Gabriela Ochoa

# What is Data Aggregation?

- Any process in which information is gathered and expressed in a summary form, for purposes such as statistical analysis
- Is a type of more sophisticated Query
- Aggregation operations
  - group values from multiple documents together
  - perform a variety of operations on the grouped data to return a single result.

# Aggregation in MongoDB

MongoDB provides three ways to perform aggregation

- Aggregation pipeline
  - Preferred method, use native code
  - More efficient than map-reduce
- Map-reduce function
  - Harder to program, can be more flexible
  - Less efficient, uses custom JavaScript functions
- Single purpose aggregation methods
  - Limited scope
  - Easy to program common aggregations

# JavaScript and MongoDB

- MongoDB uses JavaScript for
  - The command line interface
  - Server side scripts such as Map and Reduce
- A basic working knowledge of JavaScript will help you build better MongoDB queries
- JavaScript
  - OO, interpreted programming language
  - Weakly, Dynamic typed
  - Used to create interactive effects within web browsers

# JavaScript Structure

- Program blocks are delimited by `{` and `}`
- Indentation can be used for readability, but does not affect the semantics (unlike Python)
- Lines completed with semi colon `;`
- Define a function like this

```
function add(a,b){
    return a+b;
}
```

  - Call it like this: `s=add(1,3);`

# JavaScript Arrays

- Can be mixed type

```
a=["a","b",32];
```

- Numeric Index Arrays

```
a[0]="s";
["s","b",32]
```

- String indexed (JSON) objects

```
b={"key1":"value1","key2":2};
```

Initialise empty: `a=[];` or `b={};`

# JavaScript Iterating through Arrays

- Iterate over the values in a numeric indexed array

```
for(i=0;i<a.length;i++)
    print(a[i]);
```

- Or iterate over the keys

```
for(k in b) print(k);
```

- Or the values

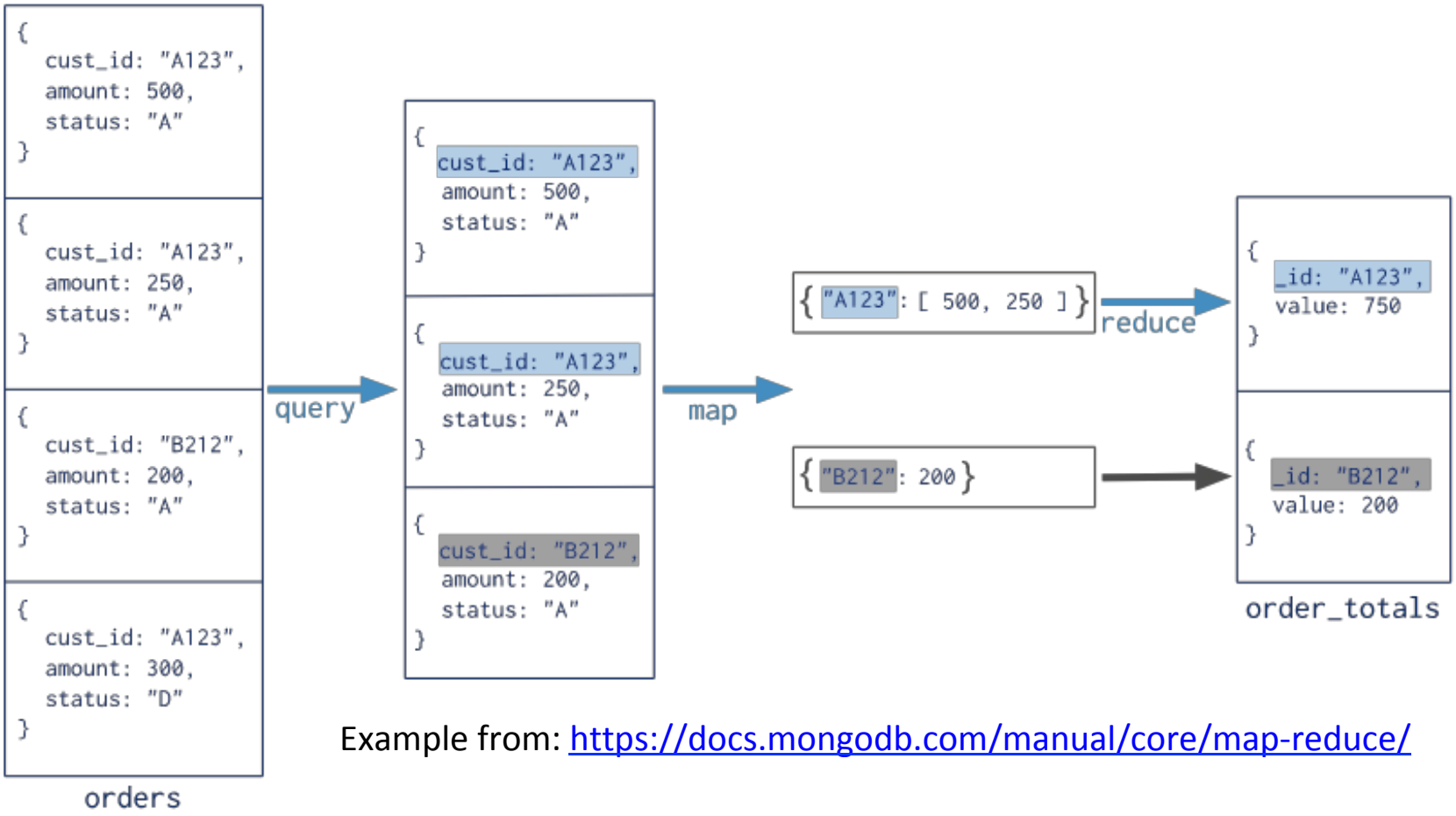
```
for(k in b) print(b[k]);
```

# Map Reduce in MongoDB

- MongoDB provides the `mapReduce` database command.
- Three parameters
  - Map: A function that defines how the queried documents will be mapped to new documents.
  - Reduce: A function that defines defines how the mapped documents are reduced.
  - A JSON Object: that contains both the query filter and the name of the output collection

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"
                       }
                   )
```




Example from: https://docs.mongodb.com/manual/core/map-reduce/

# How does it work

- Map phase
  - It is applied to each input document (i.e. the documents in the collection that match the query condition).
  - Emits key-value pairs.
- Reduce phase
  - Collects and condenses the aggregated data, for those keys that have multiple values.
- Output
  - Stores the results in a collection.
  - Optionally, the output of the reduce function may pass through a finalize function to further condense or process the results of the aggregation.

# The Map Function

- No parameters
- Is called with every document in the query result set
- Accesses each document via a variable called `this`
- Produces new documents by calling `emit()`
- Emitted documents are: a key and an array of values, matching the key.

# The Reduce Function

- Takes two parameters – A key and an array of values (documents output from Map)
- Outputs an object
- The calculation is on all of the values in the array (i.e all the values with the given key)
- Might be simple like sum
- Or something more complex

# Another Example

- Assume a collection with exam scores
- Let us analyse the result of the following

```
db.testscores.mapReduce(
        function() { emit(this.grade, this.score ); },
        function(key, values) { return Array.avg(values) },
        {
          query: { type: "reading" },
          out: "scores_by_grade”
        }
)
```

- First, the operation will query the documents and return the subset of documents that include a type property with a value of "reading":

```
[
   {"student":"phyllis-tuffield","grade":2,"type":"reading","score":500},
   {"student":"kevin-kennedy","grade":2,"type":"reading","score":400},
    {"student":"maxwell-amland","grade":3,"type":"reading","score":300}
]
```

- Next it will use grouping to map the documents to individual groups based on the mapping logic. The mapping logic is grouping documents by the grade property and creating a collection of score values:

```
[
  { 2: [ 500, 400 ] },
  { 3: [ 300 ]}
]
```

# Result

- Finally the operation **reduces** the documents by averaging the values found in the array. These documents are stored in the collection referenced in the operation (**scores_by_grade**):

```
db.scores_by_grade.find()
                    [
                       { "_id" : 2, "value" : 450 }
                       { "_id" : 3, "value" : 300 }
                    ]
```

# Summary

- MongoDB provides the `mapReduce` database command.
- Three parameters
  - Map: A function that defines how the queried documents will be mapped to new documents.
  - Reduce: A function that defines defines how the mapped documents are reduced.
  - A JSON Object: that contains both the query filter and the name of the output collection