# JSON and MongoDB

Advanced Databases SS19

Fang Wei-Kleiner

# JSON (JavaScript Object Notation)

- Very lightweight data exchange format
  - Much less verbose and easier to parse than XML
  - Increasingly used for data exchange over Web: many Web APIs use JSON to return responses/ results
- Based on JavaScript
  Conforms to JavaScript object/array syntax—you can directly manipulate JSON representations in JavaScript
- But it has gained widespread support by all programming languages

# JSON Data Model

- Two basic constructs
  - Array: comma-separated list of “things” enclosed by brackets
    - Order is important
  - Object: comma-separated set of pairs enclosed by braces; each pair consists of an attribute name (string) and a value (any “thing”)
    - Order is unimportant
      Attribute names “should” be unique within an object
- Simple types: numbers, strings (in double quotes), and special values “true”, “false”, and “null”
- Thing = a simple value or an array or an object

```
[
  { "ISBN": "ISBN-10",
    "price": 80.00,
    "title": "Foundations of Databases",
    "authors": [ "Abiteboul",  "Hull", "Vianu" ],
    "publisher": "Addison Wesley",
    "year": 1995,
    "sections": [
      { "title": "Section 1",
        "sections": [
          { "title": "Section 1.1" },
          { "title": "Section 1.2" }
        ]
      },
      { "title": "Section 2" }
    ]
  }, … …
]
```

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <section>
      <title>Section 1</title>
      <section><title>Section 1.1</title></section>
      <section><title>Section 1.2</title></section>
```

# MongoDB

- Document database designed for ease of development and scaling
- NoSQL (Non-relational) database types:
    - Key-value stores: Every single item in the database is stored as an attribute name, or key, together with its value.
    - Document databases: pair each key with a complex data structure known as a document.
    - Wide-column stores: store columns of data together, instead of rows.
    - Graph stores: store information about networks, such as social connections.
- Data model is JSON (BSON)
- Good support for indexing, partitioning, replication
- Nice integration in Web development stacks

# BSON

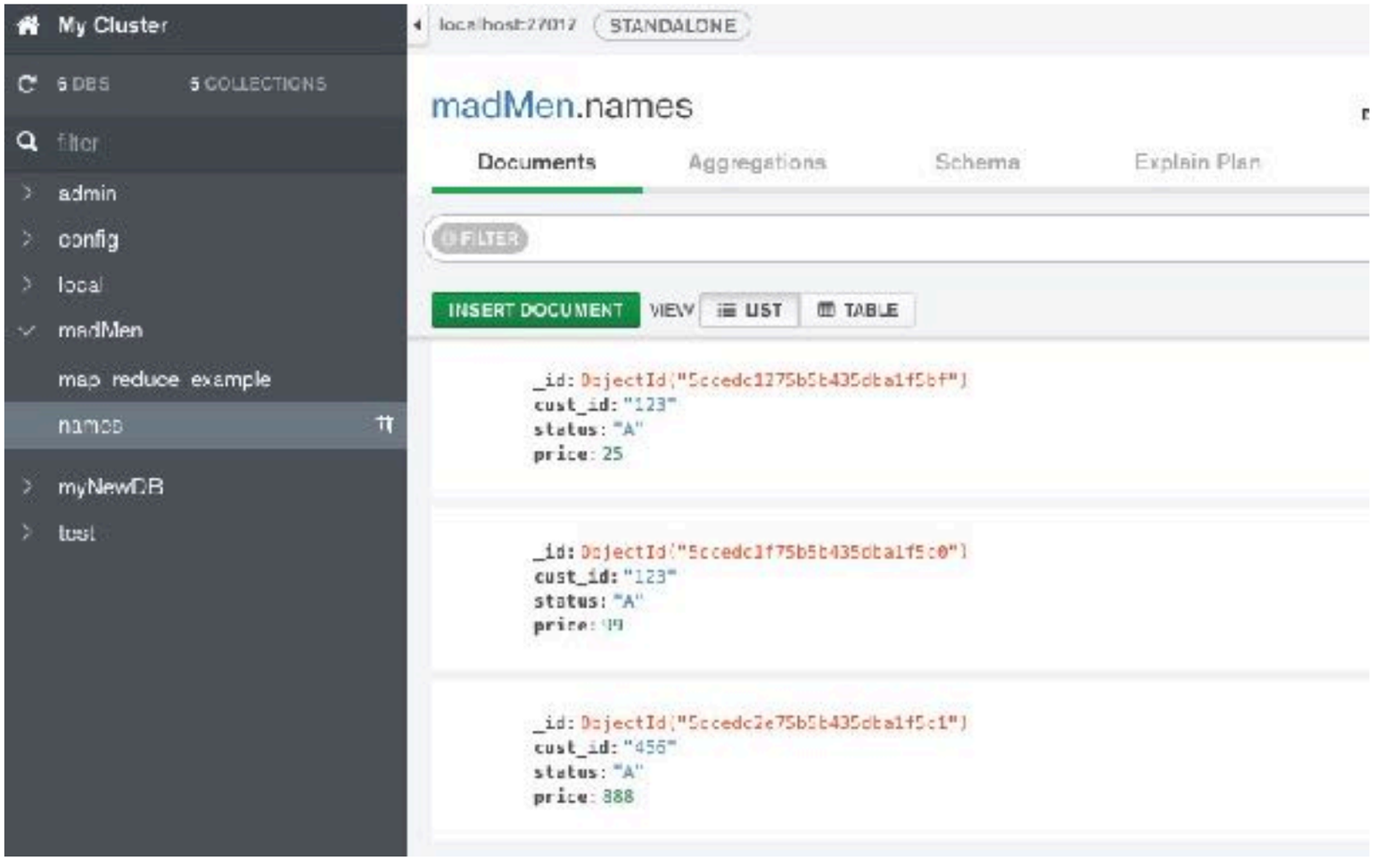
MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents.


The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
             }
```

# MongoDB databases

- Database = a number of collections
- Collection = a list of documents
- Each document stored in a collection requires a unique _id field that acts as a primary key.



# MongoDB CRUD Operations

- Create, Read, Update, Delete operations.
  - db.collection.insertOne() insert a single document into a collection
  - db.collection.insertMany() insert multiple documents into a collection

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } }
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm"
])
```

**CREATING A COLLECTION:**

If the collection does not currently exist, insert operations will create the collection.

# Query Documents

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

```
db.inventory.find( {} )
SELECT * FROM inventory
```

```
db.inventory.find( { status: "D" } )
SELECT * FROM inventory WHERE status = “D"
```

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

```
db.inventory.find( {
    status: "A",
    $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR
item LIKE "p%")
```

# Query on Embedded/Nested Documents

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )

—> Exact match, including field order

When querying using dot notation, the field and nested field must be inside quotation marks.

db.inventory.find( { "size.uom": "in" } )

db.inventory.find( { "size.h": { $lt: 15 }, "size.uom": "in", status: “D" } )

# Query an Array

```
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [ 14, 21 ] },
  { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [ 14, 21 ] },
  { item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [ 14, 21 ] },
  { item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [ 22.85, 30 ] },
  { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [ 10, 15.25 ] }
]);
```

db.inventory.find( { tags: ["red", "blank"] } )

—> tags value is an array with exactly two elements, “red” and “blank”, in the specified order

db.inventory.find( { tags: { $all: ["red", "blank"] } } )

—> array contains both “red” and “blank”, no order requirements, can contain other elements

db.inventory.find( { tags: "red" } )

—> tags is an array which contains “red”

db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )

—> for each condition, there is at least one element satisfies it

db.inventory.find( { dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } } )

—> there is at least one element satisfies both condition

db.inventory.find( { "dim_cm.1": { $gt: 25 } } )

—> the second element in the array dim_cm greater than 25.

# Project Fields to Return from Query

```
db.inventory.insertMany( [
  { item: "journal", status: "A", size: { h: 14, w: 21, uom: "cm" }, instock: [ { warehouse: "A", qty: 5 } ] },
  { item: "notebook", status: "A",  size: { h: 8.5, w: 11, uom: "in" }, instock: [ { warehouse: "C", qty: 5 } ] },
  { item: "paper", status: "D", size: { h: 8.5, w: 11, uom: "in" }, instock: [ { warehouse: "A", qty: 60 } ] },
  { item: "planner", status: "D", size: { h: 22.85, w: 30, uom: "cm" }, instock: [ { warehouse: "A", qty: 40 } ] },
  { item: "postcard", status: "A", size: { h: 10, w: 15.25, uom: "cm" }, instock: [ { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ] }
]);
```

db.inventory.find( { status: "A" }, { item: 1, status: 1 } )

SELECT _id, item, status from inventory WHERE status = "A"

db.inventory.find( { status: "A" }, { item: 1, status: 1, _id: 0 } )

SELECT item, status from inventory WHERE status = “A"

db.inventory.find( { status: "A" }, { item: 1, status: 1, "instock.qty": 1 } )

—> qty field embedded in instock array

db.inventory.find( { status: "A" }, { item: 1, status: 1, instock: { $slice: -1 } } )

—> the last element in the instock array

# MongoDB aggregation pipeline

- Idea: think of a query as performing a sequence of “stages,” each transforming an input sequence of BSON objects to an output sequence of BSON objects
- “Aggregation” is a misnomer: there are all kinds of stages
  - Selection ($match), projection ($project), sorting ($sort)
  - Computing/adding attributes with generalized projection ($project/$addFields), unnesting embedded arrays ($unwind), and restructuring output ($replaceRoot)
    - Operators to transform/filter arrays ($map/$filter)
  - Join ($lookup)
  - Grouping and aggregation ($group)
    - Operators to aggregate (e.g.,$sum) or collect into an array ($push)

# Normalise and Sort Documents

```
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : ["golf", "racquetball"]
}
{
  _id : "joe",
  joined : ISODate("2012-07-02"),
  likes : ["tennis", "golf", "swimming"]
}
```

aggregation framework.

```
db.users.aggregate(
  [
    { $project : { name:{$toUpper:"$_id"} , _id:0 } },
    { $sort : { name : 1 } }
  ]
)
```

$project operator:

->Create a new field name

->Converts the value of _id to upper case, value stored in name

->Suppresses the id field

$sort operator:

->Sort values by name

$xxx: field path in a document, tell MongoDB to interpret xxx as a field in the current object instead of just a string literal

# Return Total Number of Joins per Month

```
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : ["golf", "racquetball"]
}
{
  _id : "joe",
  joined : ISODate("2012-07-02"),
  likes : ["tennis", "golf", "swimming"]
}
```

```
db.users.aggregate(
   [
     { $project : { month_joined : { $month : "$joined" } } } ,
     { $group : { _id : {month_joined:"$month_joined"} , number : { $sum : 1 } } },
     { $sort : { "_id.month_joined" : 1 } }
   ]
)
```

$project operator:

->create a new field monthly_joined

->$month operator converts the value to integer of month

$group operator:

->collects documents with a given month_joined value and counts how many documents there are for that value. For each unique value, $group create a document with two fields: “_id”, which contains a nested document, and number, which is a generated value.

# Return five most common “Likes”

```
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : ["golf", "racquetball"]
}
{
  _id : "joe",
  joined : ISODate("2012-07-02"),
  likes : ["tennis", "golf", "swimming"]
}
```

```
db.users.aggregate(
   [
     { $unwind : "$likes" },
     { $group : { _id : "$likes" , number : { $sum : 1 } } },
     { $sort : { number : -1 } },
     { $limit : 5 }
   ]
)
```

$unwind operator: separates each value in the “likes” array, and creates a new version of the source document for every element in the array

$group operator: grouping on like and count.

$limit operator: only include the first 5 result document.

# Aggregation with Zip Code Data Set


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


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

Return states with more than 10M population.

$match stage: filters the grouped documents to those documents whose totalPop value is greater than 10M.

```
SELECT state, SUM(pop) AS totalPop
FROM zipcodes
GROUP BY state
HAVING totalPop >= (10*1000*1000)
```

# Return Largest and Smallest Cities by State


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


```
db.zipcodes.aggregate( [
    { $group:
        {
          _id: { state: "$state", city: "$city" },
          pop: { $sum: "$pop" }
        }
    },
    { $sort: { pop: 1 } },
    { $group:
        {
          _id : "$_id.state",
          biggestCity:  { $last: "$_id.city" },
          biggestPop:   { $last: "$pop" },
          smallestCity: { $first: "$_id.city" },
          smallestPop:  { $first: "$pop" }
        }
    },
] )
```

Can you rewrite the query without using $sort?

# $addFields (Aggregation)

```
{
  _id: 1,
  student: "Maya",
  homework: [ 10, 5, 10 ],
  quiz: [ 10, 8 ],
  extraCredit: 0
}
{
  _id: 2,
  student: "Ryan",
  homework: [ 5, 6, 5 ],
  quiz: [ 8, 8 ],
  extraCredit: 8
}
```

```
db.scores.aggregate( [
   {
     $addFields: {
       totalHomework: { $sum: "$homework" } ,
       totalQuiz: { $sum: "$quiz" }
     }
   },
   {
     $addFields: { totalScore:
       { $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ] } }
   }
] )
```

```
{
  "_id" : 1,
  "student" : "Maya",
  "homework" : [ 10, 5, 10 ],
  "quiz" : [ 10, 8 ],
  "extraCredit" : 0,
  "totalHomework" : 25,
  "totalQuiz" : 18,
  "totalScore" : 43
}
{
  "_id" : 2,
  "student" : "Ryan",
  "homework" : [ 5, 6, 5 ],
  "quiz" : [ 8, 8 ],
  "extraCredit" : 8,
  "totalHomework" : 16,
  "totalQuiz" : 16,
  "totalScore" : 40
}
```

# $filter (Aggregation)

Selects a subset of an array to return based on the specified condition

input —> array

as —> element of array

cond —> condition to determine the element should be in the array

“$$xxx” —> variable generated during excution

```
{
  _id: 0,
  items: [
    { item_id: 43, quantity: 2, price: 10 },
    { item_id: 2, quantity: 1, price: 240 }
  ]
}
{
  _id: 1,
  items: [
    { item_id: 23, quantity: 3, price: 110 },
    { item_id: 103, quantity: 4, price: 5 },
    { item_id: 38, quantity: 1, price: 300 }
  ]
}
{
  _id: 2,
  items: [
    { item_id: 4, quantity: 1, price: 23 }
  ]
}
```

```
db.sales.aggregate([
  {
    $project: {
      items: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gte: [ "$$item.price", 100 ] }
        }
      }
    }
  }
]}
```

```
{
  "_id" : 0,
  "items" : [
    { "item_id" : 2, "quantity" : 1, "price" : 240 }
  ]
}
{
  "_id" : 1,
  "items" : [
    { "item_id" : 23, "quantity" : 3, "price" : 110 },
    { "item_id" : 38, "quantity" : 1, "price" : 300 }
  ]
}
{ "_id" : 2, "items" : [ ] }
```

# $lookup (Join)

```
db.orders.insert([
  { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
  { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
  { "_id" : 3  }
])
db.inventory.insert([
  { "_id" : 1, "sku" : "almonds", description: "product 1", "instock" : 120 },
  { "_id" : 2, "sku" : “almonds", description: "product 2", "instock" : 80 },
  { "_id" : 3, "sku" : "cashews", description: "product 3", "instock" : 60 },
  { "_id" : 5, "sku": null, description: "Incomplete" },
  { "_id" : 6 }
])
```

```
db.orders.aggregate([
  {
    $lookup:
      {
        from: "inventory",
        localField: "item",
        foreignField: "sku",
        as: "inventory_docs"
      }
  }
])
```

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

```
{
  "_id" : 1,
  "item" : “almonds",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    { "_id" : 1, "sku" : "almonds", "description" : "product 1", "instock" : 120 },
    { "_id" : 2, "sku" : "almonds", "description" : "product 2", "instock" : 80 }
  ]
}
{
  "_id" : 2,
  "item" : "pecans",
  "price" : 20,
  "quantity" : 1,
  "inventory_docs" : [  ]
}
{
  "_id" : 3,
  "inventory_docs" : [
    { "_id" : 5, "sku" : null, "description" : "Incomplete" },
    { "_id" : 6 }
  ]
```

# $push (Aggregation)

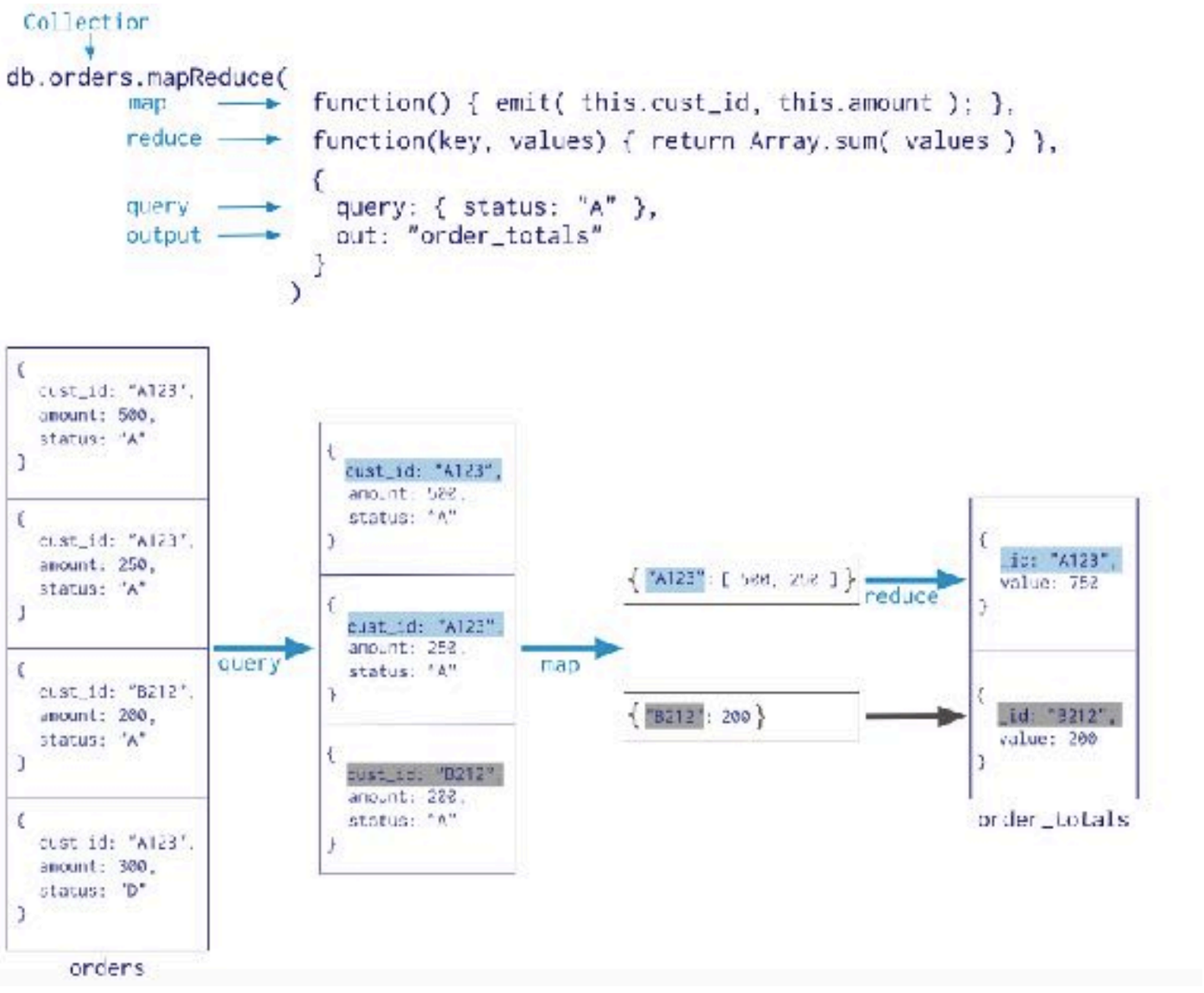
```
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-01-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-02-03T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 5, "date" : ISODate("2014-02-03T09:05:00Z") }
{ "_id" : 4, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-02-15T08:00:00Z") }
{ "_id" : 5, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-02-15T09:05:00Z") }
{ "_id" : 6, "item" : "xyz", "price" : 5, "quantity" : 5, "date" : ISODate("2014-02-15T12:05:10Z") }
{ "_id" : 7, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-02-15T14:12:12Z") }
```

```
db.sales.aggregate(
  [
    {
      $group:
      {
        _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
        itemsSold: { $push:  { item: "$item", quantity: "$quantity" } }
      }
    }
  ]
)
```

```
{
  "_id" : { "day" : 46, "year" : 2014 },
  "itemsSold" : [
    { "item" : "abc", "quantity" : 10 },
    { "item" : "xyz", "quantity" : 10 },
    { "item" : "xyz", "quantity" : 5 },
    { "item" : "xyz", "quantity" : 10 }
  ]
}
{
  "_id" : { "day" : 34, "year" : 2014 },
  "itemsSold" : [
    { "item" : "jkl", "quantity" : 1 },
    { "item" : "xyz", "quantity" : 5 }
  ]
}
{
  "_id" : { "day" : 1, "year" : 2014 },
  "itemsSold" : [ { "item" : "abc", "quantity" : 2 } ]
}
```

$push —> returns an array of all that result from applying the expression. Only available in $group stage

# $map-reduce (Aggregation)




- Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.
- 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.