CSCU9YQ - NoSQL Databases
Lecture 3: The Aggregation Pipeline

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
- Purpose: get more information about particular groups based on specific variables such as age, profession, or income.
- 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.

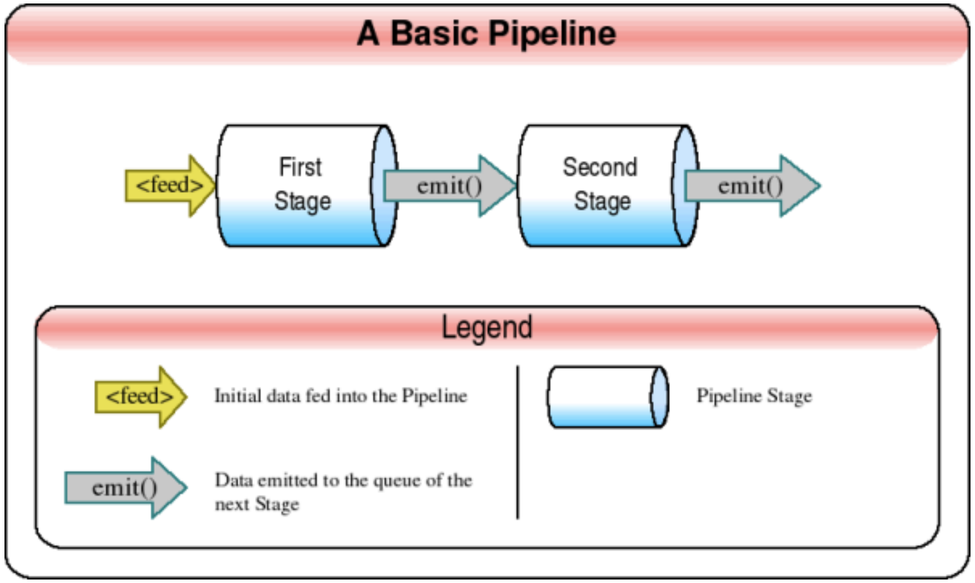
Aggregation in MongoDB

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

The Aggregation Pipeline

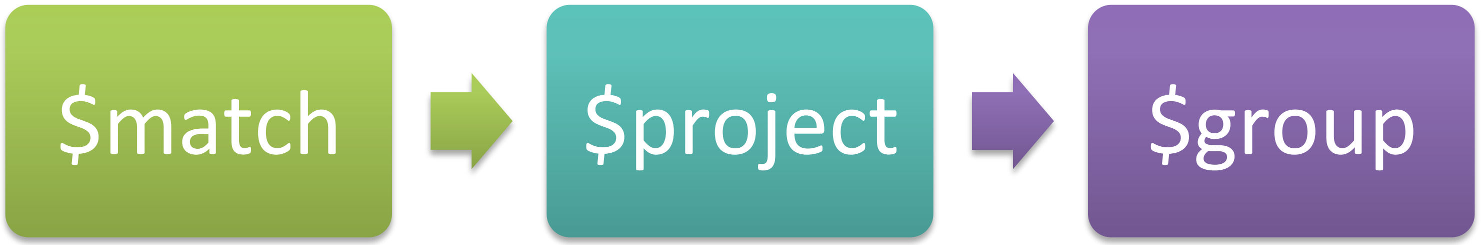
- Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.
- The most basic pipeline stages provide
  - filters that operate like queries and
  - document transformations that modify the form of the output document.



Other pipelines operators provide tools for

- Grouping and sorting documents by specific field(s)
- Aggregating the content of arrays, including arrays of documents

Aggregation Pipeline

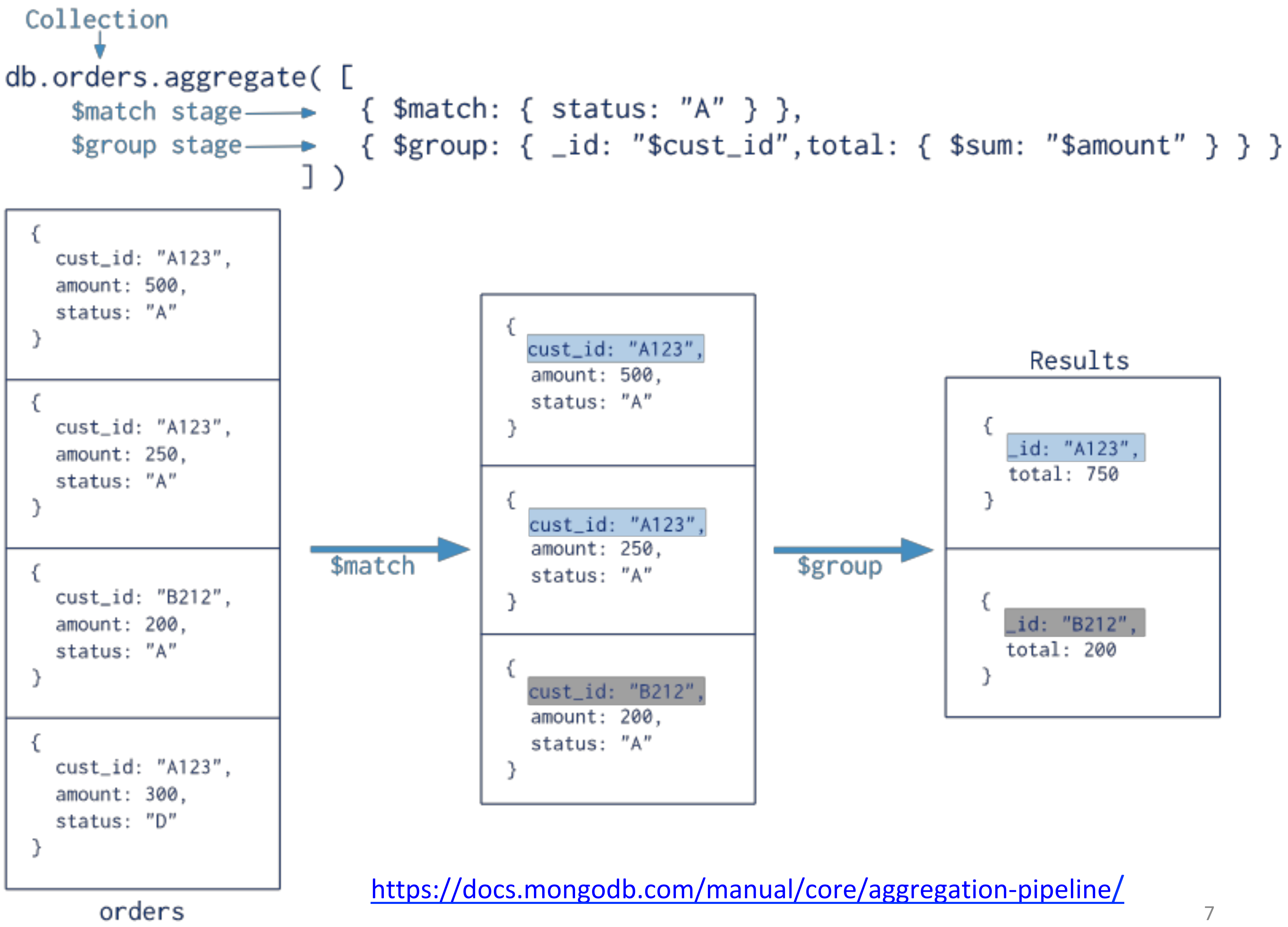


- Stages do not need to produce one output document for every input
- Some stages may generate new documents or filter out documents
- Mongo Shell method
  - `db.collection.aggregate( [ { <stage> }, ... ] )`
  - Parameters in JASON format.
  - Stages appear in an array

Some Common Stages

Stage	Description
$match:	Filter the collection according to the query parameters, and only pass through the documents matching the query, to the next stage of the pipeline.
$group	Groups documents by some specified expression and outputs to the next stage a document for each distinct grouping.
$project	Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.
$unwind	Operates on arrays. Deconstructs an array field from the input documents to output a document for each element.
$sort	Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified.

More stages:
https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#aggregation-pipeline-operator-reference



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

```
{
  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"
}
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{
  _id: "A123",
  total: 750
}
{
  _id: "B212",
  total: 200
}
```

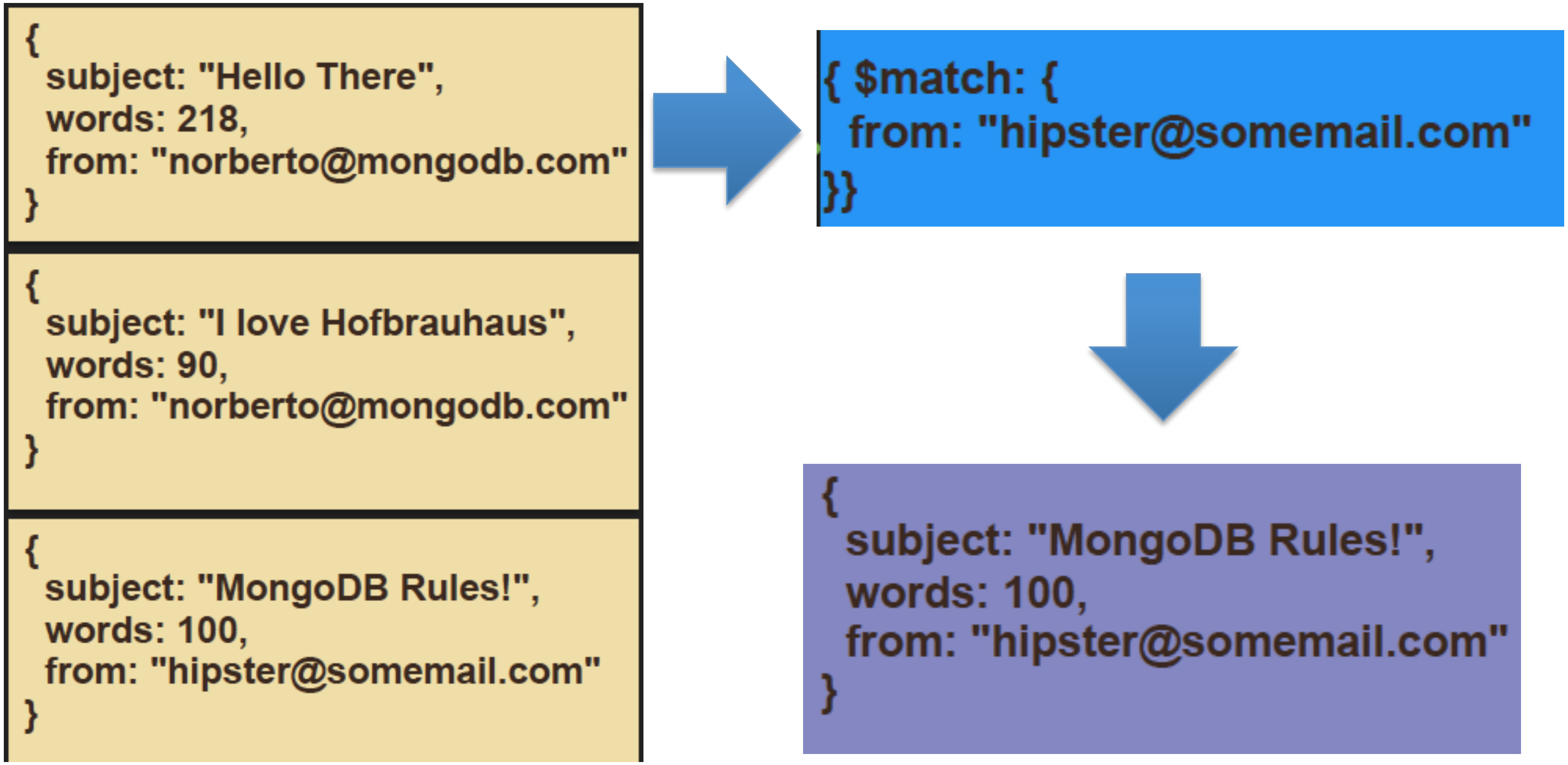
https://docs.mongodb.com/manual/core/aggregation-pipeline/

$match

- Filter documents
  - pass only the documents that match the specified condition(s) to the next pipeline stage.
- Should be placed early in the pipeline, as it limits the number of documents, and thus optimise the process
- Syntax (use existing query syntax): { $match: { <query> } }

Form & Examples	Description
{ <field1>: <value1>, ... }	Equality condition
{ <field1>: { <operator1>: <value1> }, ... }	Conditions using query operators
{ status: "A", qty: { $lt: 30 } }	Concatenation separated by ‘,’ gives an implicit AND condition
{ $or: [{ status: "A" }, { qty: { $lt: 30 } }] }	OR condition needs operator $or

$match simple match



```
{
  subject: "Hello There",
  words: 218,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "I love Hofbrauhaus",
  words: 90,
  from: "norberto@mongodb.com"
}
```

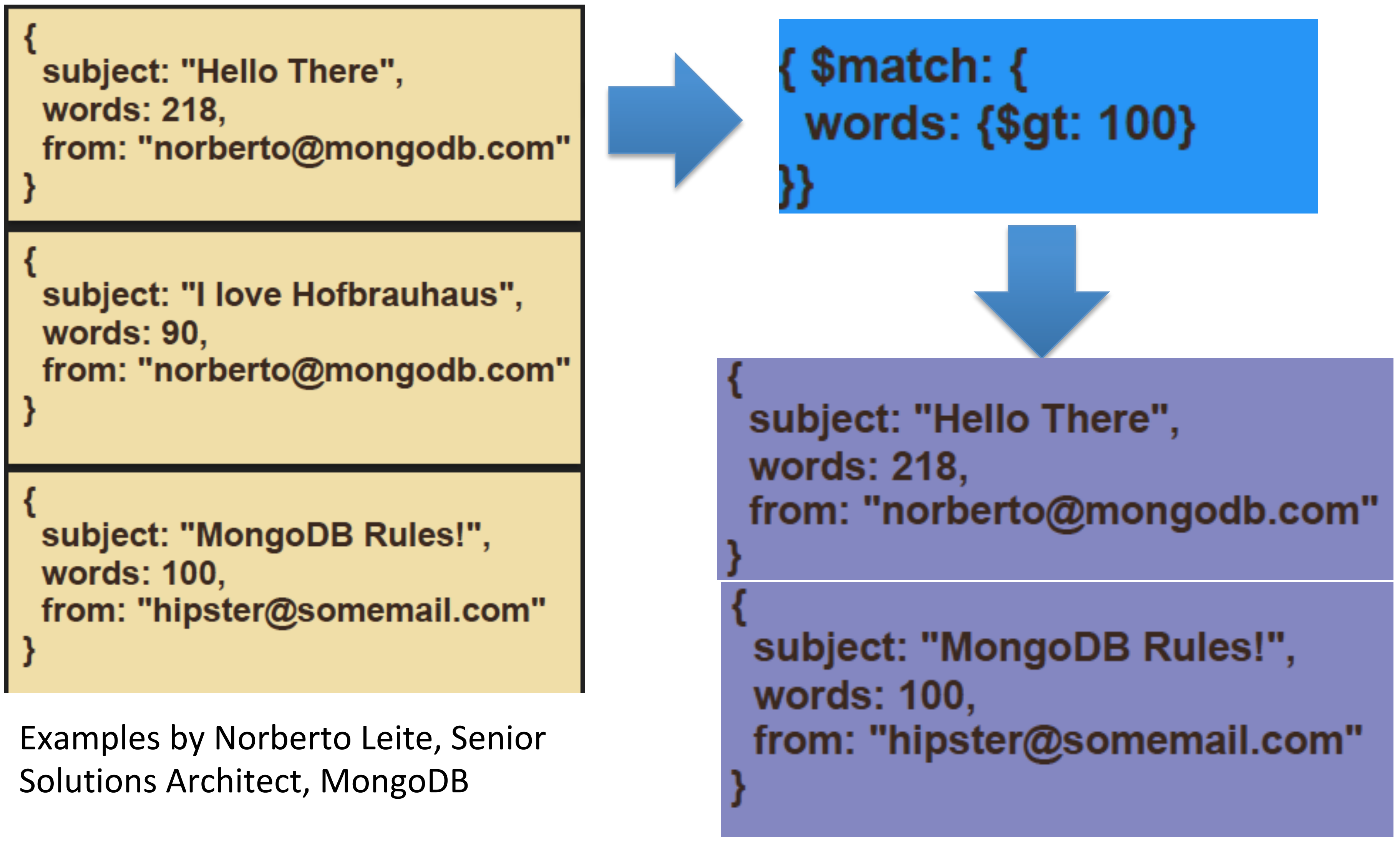
```
{
  subject: "MongoDB Rules!",
  words: 100,
  from: "hipster@somemail.com"
}
```

```
{ $match: {
  from: "hipster@somemail.com"
}}
```

```
{
  subject: "MongoDB Rules!",
  words: 100,
  from: "hipster@somemail.com"
}
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$match with Query Operations



```
{
  subject: "Hello There",
  words: 218,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "I love Hofbrauhaus",
  words: 90,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "MongoDB Rules!",
  words: 100,
  from: "hipster@somemail.com"
}
```

```
{ $match: {
  words: {$gt: 100}
}}
```

```
{
  subject: "Hello There",
  words: 218,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "MongoDB Rules!",
  words: 100,
  from: "hipster@somemail.com"
}
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$project

- Reshape Documents
  - Include, exclude or rename fields
  - Inject computed fields
  - Create sub-document fields
- Syntax: { $project: { <specification(s)> } }

Form	Description
<field>: <1 or true>	Specifies the inclusion of a field.
<field>: <0 or false>	Specifies the exclusion of a field.
_id: <0 or false>	Specifies the suppression of the _id field. By default the _id is included.
<field>: <expression>	Adds a new field or resets the value of an existing field.

$project: Including and Excluding Fields

```
{
  _id: 12345,
  subject: "Hello There",
  words: 218,
  from:"norberto@mongodb.com"
  to: [  "marc@mongodb.com",
         "sam@mongodb.com"   ],
  account: "mongodb mail",
  date: ISODate("2012-08-05"),
  replies: 3,
  folder: "Inbox",
  ...
}
```

```
{ $project: {
  _id: 0,
  subject: 1,
  from: 1
}}
```

```
{
  subject: "Hello There",
  from:"norberto@mongodb.com"
}
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$project: Renaming and Computing Fields

```
{
  _id: 12345,
  subject: "Hello There",
  words: 218,
  from:"norberto@mongodb.com"
  to: [  "marc@mongodb.com",
         "sam@mongodb.com"   ],
  account: "mongodb mail",
  date: ISODate("2012-08-05"),
  replies: 3,
  folder: "Inbox",
  ...
}
```

```
{ $project: {
   spamIndex: {
        $mul: ["$words”,"$replies"]
    },
    user: "$from"
}}
```

```
{
  _id: 12345,
  spamIndex: 72.6666 ,
  user: "norberto@mongodb.com"
}
```

$mul : operator multiply

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$project: Creating Sub-Document Fields

```
{
  _id: 12345,
  subject: "Hello There",
  words: 218,
  from:"norberto@mongodb.com"
  to: [  "marc@mongodb.com",
         "sam@mongodb.com"   ],
  account: "mongodb mail",
  date: ISODate("2012-08-05"),
  replies: 3,
  folder: "Inbox",
  ...
}
```

```
{ $project: {
   subject: 1,
   stats: {
     replies: "$replies",
     from: "$from",
    date: "$date"
}}}
```

```
{
  _id: 375,
  subject: "Hello There",
  stats: {
    replies: 3,
    from: "norberto@mongodb.com",
    date: ISODate("2012-08-05")
}}
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$group

- Groups documents by some specified expression and outputs to the next state a document for each distinct grouping
- The output documents contain
  - an _id field with the distinct groups by key
  - Computed fields that hold the values of some accumulator expression grouped by the group's _id field.
    - $max, $min, $avg, $sum
    - $addToSet, $push
    - $first, $last

$group syntax

{ $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }

- The _id field is mandatory; however, you can specify an _id value of null to calculate accumulated values for all the input documents as a whole.
- The remaining computed fields are optional and computed using the <accumulator> operators.
- The _id and the <accumulator> expressions can accept any valid expression.
- Valid Expressions
  - Field paths: "$<field>" $ followed by field name (or dotted field name)
  - Literals: of any type
  - Expression object: { <field1>: <expression1>, ... }

$group: Calculating An Average

```
{
  subject: "Hello There",
  words: 218,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "I love Hofbrauhaus",
  words: 90,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "MongoDB Rules!",
  words: 100,
  from: "hipster@somemail.com"
}
```

```
{ $group: {
  _id: "$from",
   avgWords: {$avg: "$words”}
}}
```

```
{
  _id: "norberto@mongodb.com",
  avgPages: 154
}
```

```
{
  _id: "hipster@somemail.com",
  avgPages: 100
}
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$group: Summing Fields and Counting

```
{
  subject: "Hello There",
  words: 218,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "I love Hofbrauhaus",
  words: 90,
  from: "norberto@mongodb.com"
}
```

```
{
  subject: "MongoDB Rules!",
  words: 100,
  from: "hipster@somemail.com"
}
```

```
{ $group: {
  _id: "$from",
  words: {$sum: "$words”}
  mails: {$sum: 1}
}}
```

```
{
  _id: "norberto@mongodb.com",
  words: 308,
  mails: 2
}
```

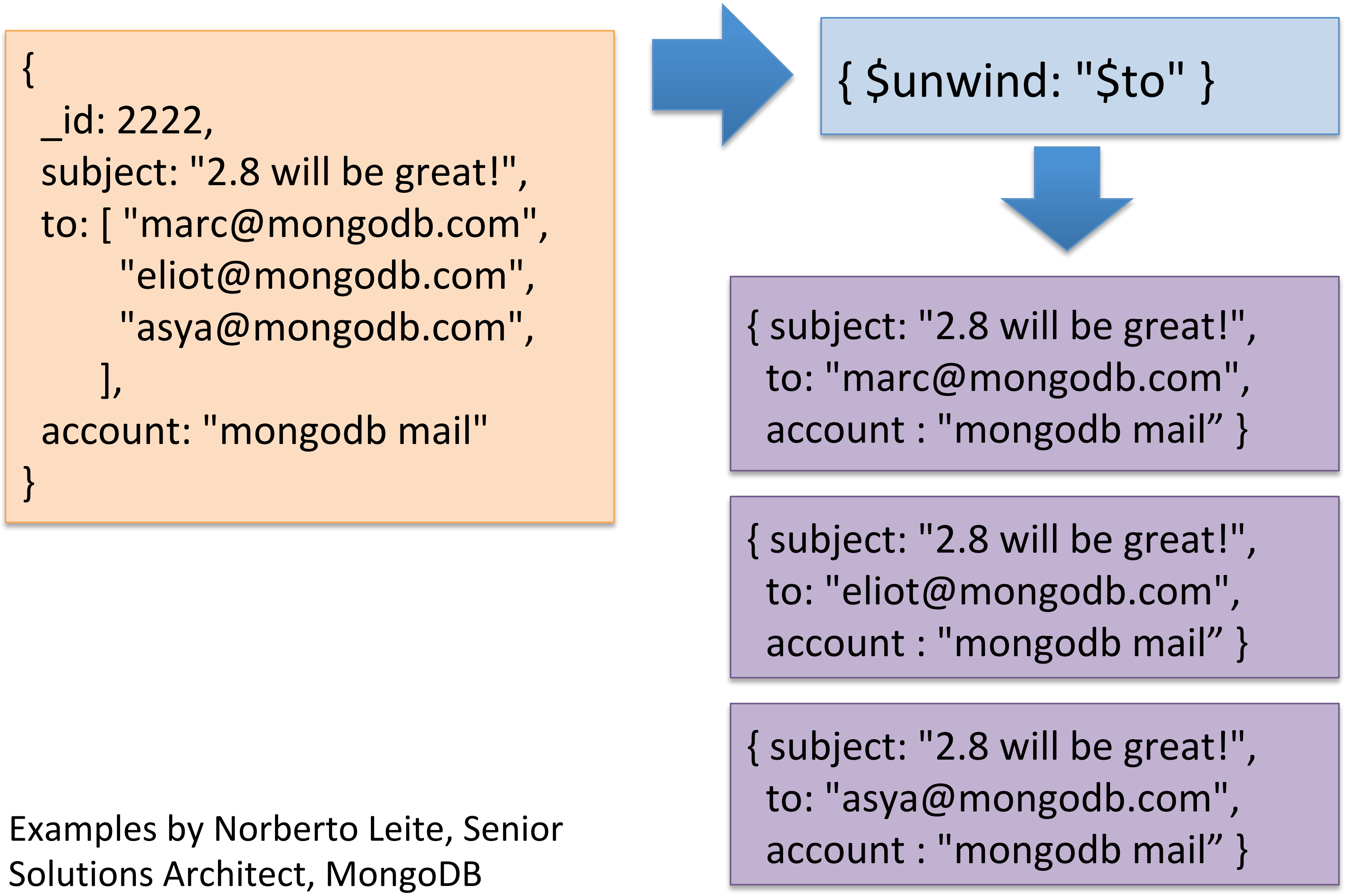
```
{
  _id: "hipster@somemail.com",
  words: 100
  mails: 1
}
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$unwind

- Operates on an array field
- Deconstructs the array from the input documents
- Creates an output a document for each array element.
- Syntax: { $unwind: <field path> }
  - <field path> is a string "$<field_name>"

$unwind: Collecting Distinct Values



```
{
  _id: 2222,
  subject: "2.8 will be great!",
  to: [ "marc@mongodb.com",
        "eliot@mongodb.com",
        "asya@mongodb.com",
      ],
  account: "mongodb mail"
}
```

```
{ $unwind: "$to" }
```

```
{ subject: "2.8 will be great!",
  to: "marc@mongodb.com",
  account : "mongodb mail” }
```

```
{ subject: "2.8 will be great!",
  to: "eliot@mongodb.com",
  account : "mongodb mail” }
```

```
{ subject: "2.8 will be great!",
  to: "asya@mongodb.com",
  account : "mongodb mail” }
```

Examples by Norberto Leite, Senior Solutions Architect, MongoDB

$sort

- Sorts all input documents and returns them to the pipeline in sorted
- Syntax:
  - { $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }
  - The sort order is a value or 1 or -1 to specify an ascending or descending sort respectively

Example combining $project and $sort: Sports Club

```
{
  _id : "jane",
  joined : ISODate("2011-02-02"),
  likes : ["golf", "racquetball"]
}
```

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

```
 user collections (more documents)
...
```

```
[
   { $project : { name:{$toUpper:"$_id"} , _id:0 } },
   { $sort : { name : 1 } }
 ]
```

```
{
 "name" : "JANE"
},
```

```
{
 "name" : "JILL"
},
```

```
{
 "name" : "JOE"
}
```

Returns user names in upper case and in alphabetical order

Example from : MongoDB Documentation
https://docs.mongodb.com/manual/tutorial/aggregation-with-user-preference-data/

Example combining $project and $sort: Sports Club

```
{
  _id : "jane",
  joined : ISODate("2011-02-02"),
  likes : ["golf", "racquetball"]
}
```

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

```
user collections (more documents)
...
```

```
[
  { $project :
    {
      month_joined : { $month : "$joined" },
      name : "$_id",
      _id : 0
    }
  },
  { $sort : { month_joined : 1 } }
 ]
```

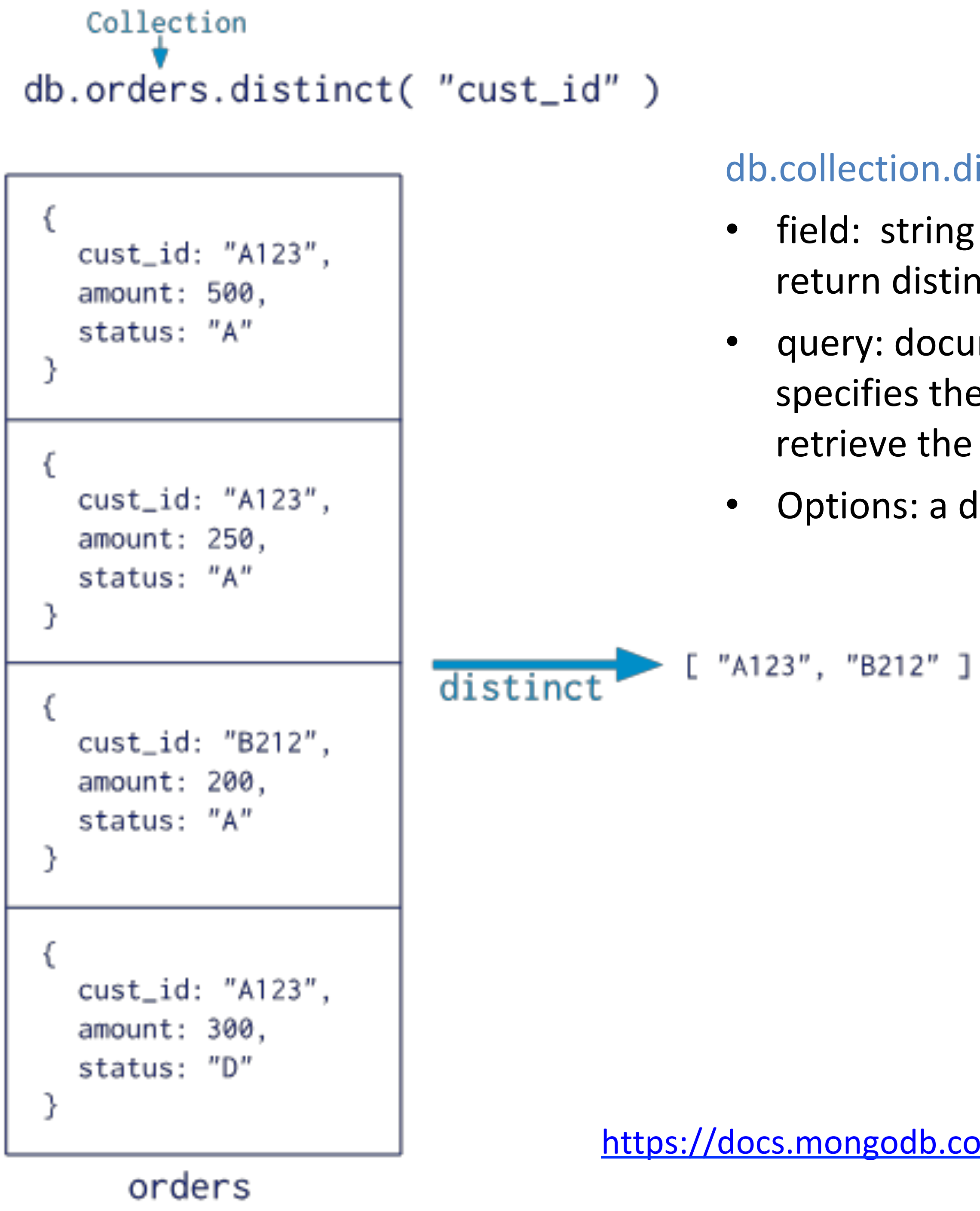
```
{
  "month_joined" : 1,
  "name" : "ruth"
},
```

```
{
  "month_joined" : 2,
  "name" : "jane"
},
```

- Returns user names sorted by the month they joined.
- $month operator converts the values of the joined field to integer representations of the month.

Single Purpose Aggregation Operations

- MongoDB also provides operations for common aggregations
  - db.collection.count(query, options): Returns the count of documents that would match a find() query for the collection or view.
  - db.collection.estimatedDocumentCount(options): returns the count of all documents in a collection. Wraps the count command.
  - db.collection.distinct(field, query, options)
- Aggregate documents from a single collection
- Provide simple access to common aggregation processes, they lack the flexibility and capabilities of the aggregation pipeline and map-reduce.



```
db.orders.distinct( "cust_id" )
```

```
{
  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"
}
```

```
[ "A123", "B212" ]
```

db.collection.distinct(field, query, options)

- field: string with the field for which to return distinct values
- query: document with a query that specifies the documents from which to retrieve the distinct values
- Options: a document that specify options

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

Aggregation Summary

- Ppowerful tool to process Data, rich library of functions
- A series of Document Transformation, concatenated stages
- MongoDB provides three ways to perform aggregation
  - Aggregation pipeline
    - Preferred method, use native code
    - More efficient than map-reduce
  - Single purpose aggregation methods
    - Limited scope
    - Easy to program common aggregations
  - Map-reduce function (Next week)
    - Harder to program, can be more flexible
    - Less efficient, uses custom JavaScript functions