Chapter 4.
Aggregation Operations

References

- MongoDB The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition
- https://docs.mongodb.com/
- https://www.mongodb.com/docs/manual/

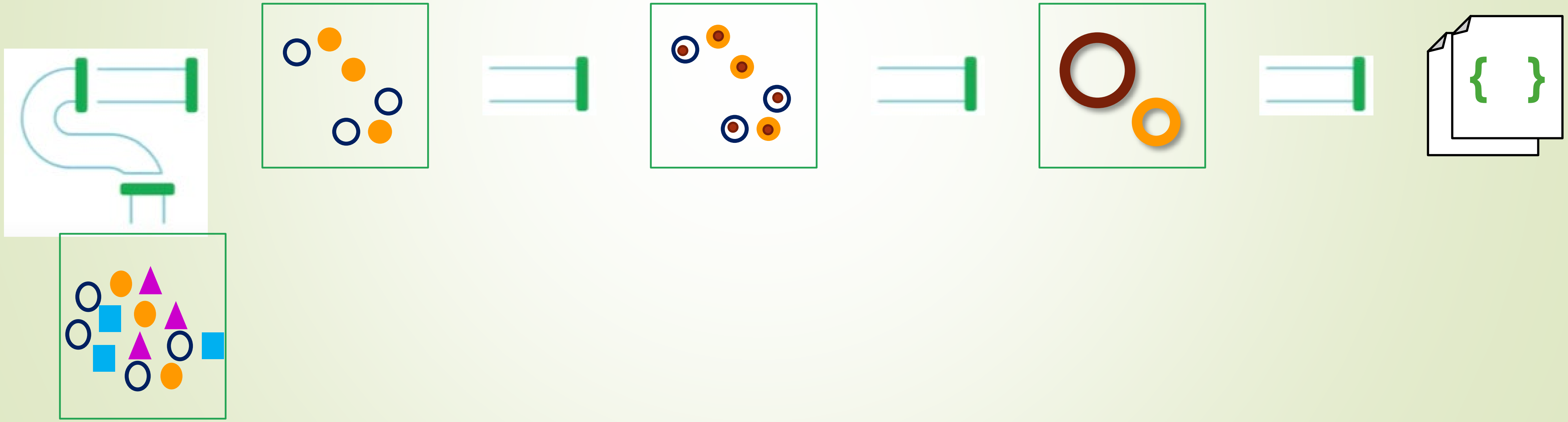
Learning objectives

- Understand Aggregation Pipeline, Aggregation Stages
- Using operators to construct expressions in the Aggregation Pipeline Stages

Contents

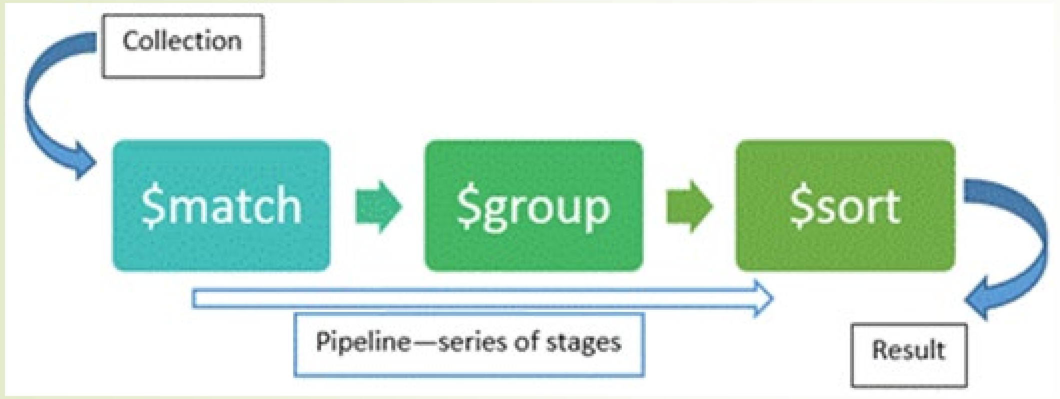
1. What is the Aggregation Pipeline in MongoDB?

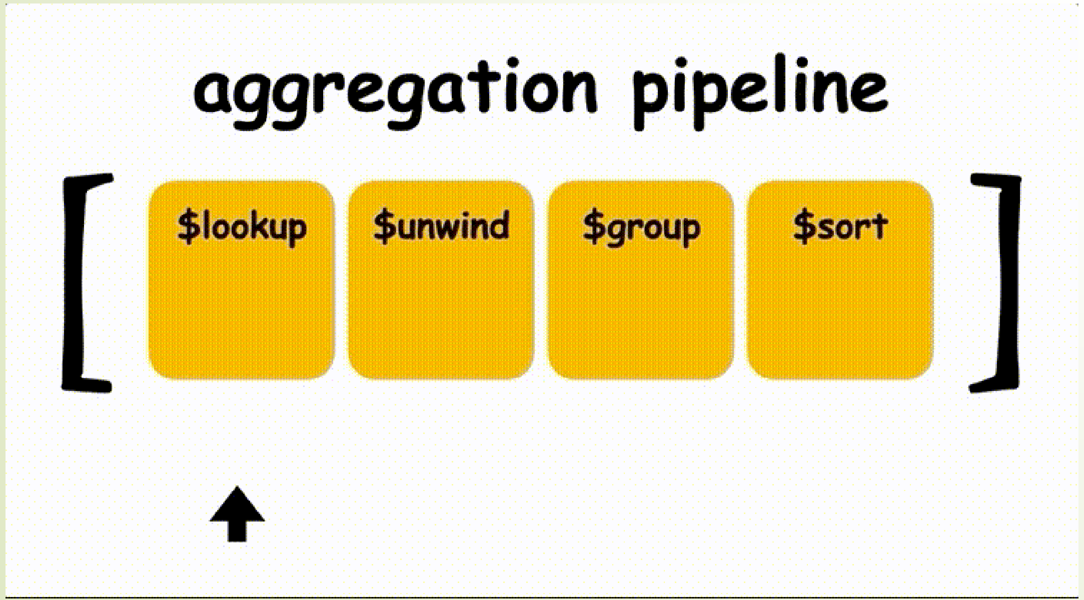
- The Aggregation Pipeline refers to a specific flow of operations that processes, transforms, and returns results. In a pipeline, successive operations are informed by the previous result.



1. What is the Aggregation Pipeline in MongoDB?

- Typical pipeline examples:





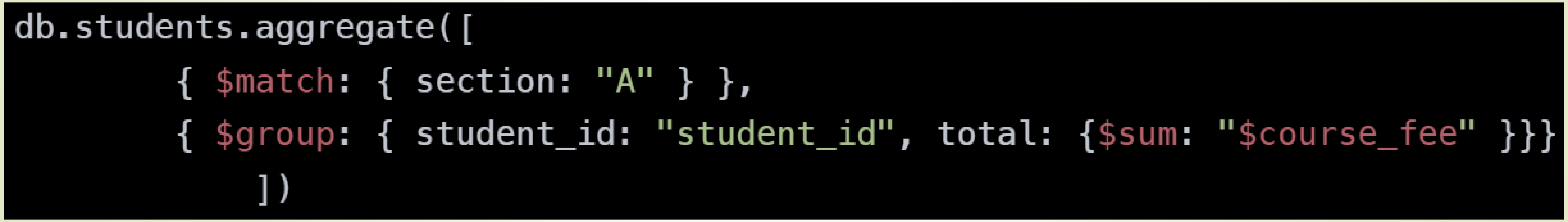
The output from the $match stage is fed into $group and then the output from the $group stage into $sort

1. What is the Aggregation Pipeline in MongoDB? Example

{ student_id: "P0001", class: 101, section: "A", course_fee: 12 },
{ student_id: "P0002", class: 102, section: "A", course_fee: 8 },
{ student_id: "P0002", class: 101, section: "A", course_fee: 12 },
{ student_id: "P0004", class: 103, section: "B", course_fee: 19 }

Using *aggregation pipeline,* calculate the total course fee of all the students in *Section A:*

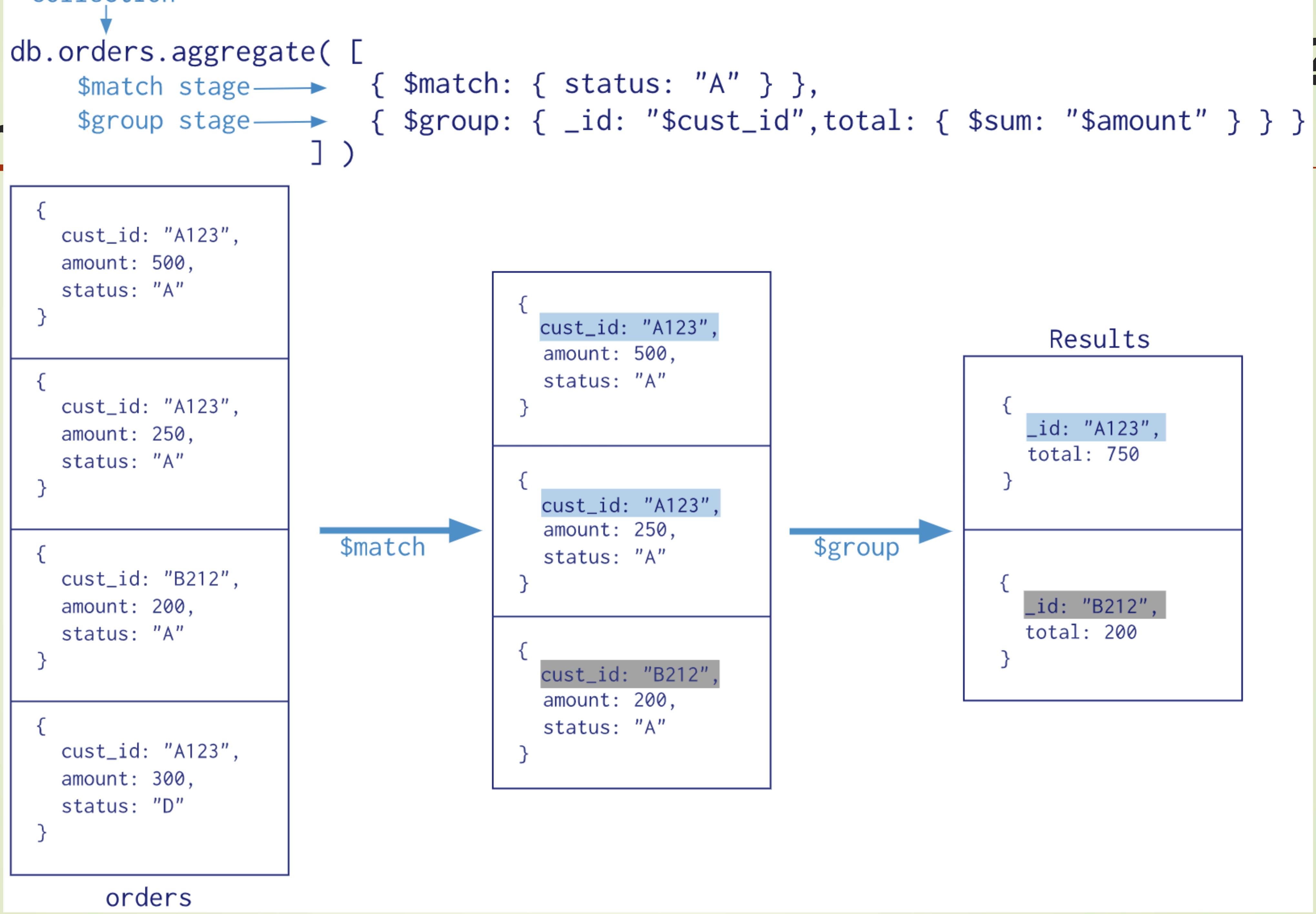
Here, the final results: { student_id : "P001", total: 12 },
{ student_id : "P002", total: 20 }



```
db.students.aggregate([
        { $match: { section: "A" } },
        { $group: { student_id: "student_id", total: {$sum: "$course_fee" }}}
            ])
```

In this query, use $match to limit the students to Section A. Then, we have grouped the students by student_id and calculated the sum total of *course_fee.*

1. W ?
Exar



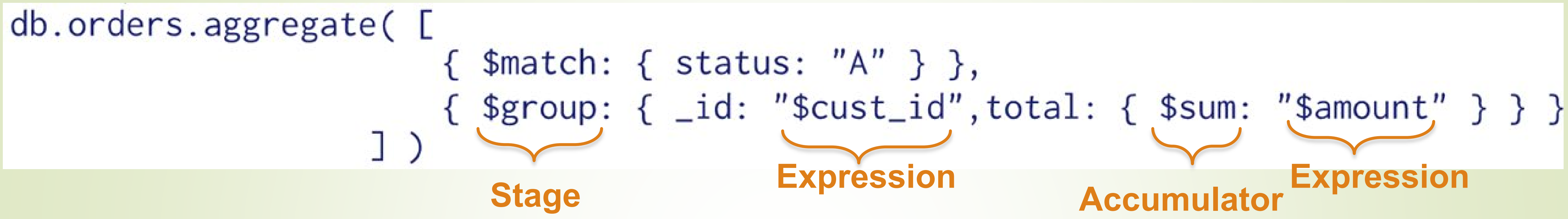
```
db.orders.aggregate( [
      { $match: { status: "A" } },
      { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
] )
```

```
{
  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
}
```

1. What is the Aggregation Pipeline in MongoDB? Example



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

Stage:
- **$project**: reshape data
- **$match**: filter data
- **$group**: aggregate data
- **$sort**: sorts data
- **$skip**: skips data
- **$limit**: limit data
- **$unwind**: normalizes data
- …

Expression: refers to the name of the field in input documents

Accumulator: used in the group stage
- **$sum:** sums numeric values for the documents in each group
- **$count:** counts total numbers of documents
- **$avg:** calculates the average
- **$min:** gets the minimum value
- **$max:** gets the maximum value
- …

1. What is the Aggregation Pipeline in MongoDB?

- An Aggregation Pipeline consists of one or more **stages** that process documents:
  - Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, calculate values…
  - The documents that are output from a stage are passed to the next stage.
  - An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

1. What is the Aggregation Pipeline in MongoDB?

- Why do we use Aggregation Pipeline?
  - To group values from multiple documents together.
  - To perform operations on the grouped data to return a single result.
  - To analyze data changes over time.

Contents

2. Aggregation Structure and Syntax

- Syntax: db.collectionName.aggregate(pipeline, <options>)

```
db.collectionName.aggregate( [ { <stage1> }, { <stage2> }, ..., { <stageN> } ], <options> )
```

  - Each stage is a JSON object of key value pairs.
  - Stages always work in sequence.
  - While there is no limit to the number of stages used in the query, it is worth noting that the order of the stages matters and there are optimizations that can help your pipeline perform better.
  - A stage can appear multiple times in a pipeline, with the exception of *$out, $merge* and *$geoNear* stages.
  - Stages are composed of one or more **aggregation operators** or **expressions**.

2. Aggregation Structure and Syntax
Common pipeline stages

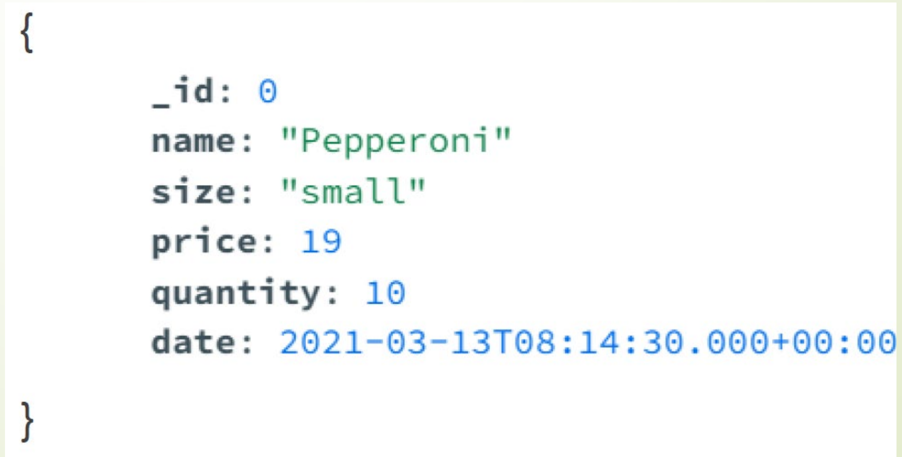
Method	Description
$match	Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.
$project	Reshapes each document in the stream, such as by adding new fields or removing existing fields.
$group	Group documents in collection, which can be used for statistics.
$unwind	Deconstructs an array field from the input documents to output a document for each element.
$lookup	Performs a left outer join to an unsharded collection in the same database to filter in documents from the "joined" collection for processing.
$redact	Restricts the contents of the documents based on information stored in the documents themselves.
$out	Takes the documents returned by the aggregation pipeline and writes them to a specified collection.
$merge	Writes the results of the aggregation pipeline to a specified collection.
self study: $sort, $limit, $skip,…	

2. Aggregation Structure and Syntax

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM()	$sum
COUNT()	$sum $sortByCount
join	$lookup/$unwind

2. Aggregation Structure and Syntax
$match stage

- Syntax: `{ $match: { <query> } }`
  - Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage.
  - Place the $match as early in the aggregation pipeline as possible.
  - $match uses standard MongoDB query operators.
  - Can’t use $where with $match.
- Example: orders collection



```
{
        _id: 0
        name: "Pepperoni"
        size: "small"
        price: 19
        quantity: 10
        date: 2021-03-13T08:14:30.000+00:00
}
```

2. Aggregation Structure and Syntax
$match stage: Example

- List the orders name is ‘Vegan’

```
db.orders.aggregate([{$match:{'name':'Vegan'}}])
```

- List the orders name is Vegan and size is medium

```
db.orders.aggregate([{$match:{$and:[{'name':'Vegan'},{size:'medium'}]}}])
db.orders.aggregate([{$match:{'name':'Vegan',size:'medium'}}])
```

- List the orders name is Cheese or size is medium

```
db.orders.aggregate([{$match:{$or:[{name:'Cheese’}, {size:'medium'}]}}]
```

- List the order that price over 20 or quantity over 30

```
db.orders.aggregate([{$match:{$or:[{price:{$gt:20}}, {quantity:{$gt:30}}]}}])
```

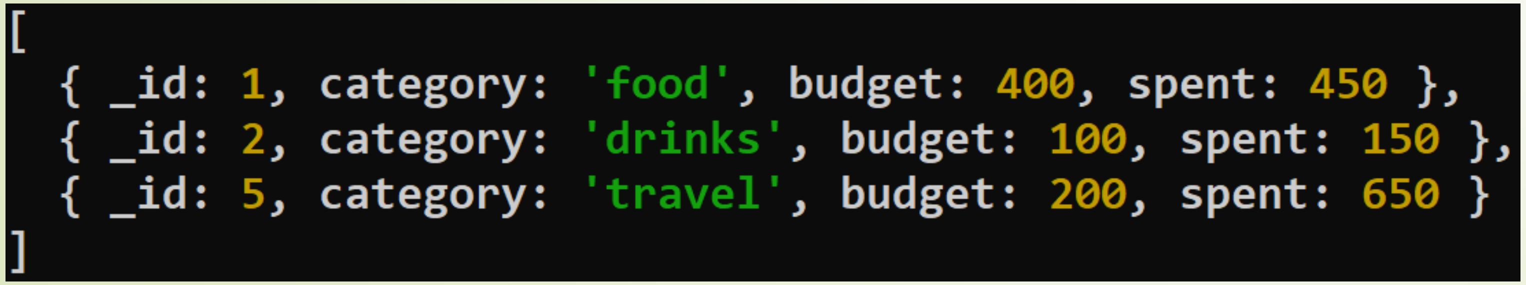
2. Aggregation Structure and Syntax
$match stage: Example

```
db.monthlyBudget.insertMany([
        { _id : 1, category : "food", budget: 400, spent: 450 },
        { _id : 2, category : "drinks", budget: 100, spent: 150 },
        { _id : 3, category : "clothes", budget: 100, spent: 50 },
        { _id : 4, category : "misc", budget: 500, spent: 300 },
        { _id : 5, category : "travel", budget: 200, spent: 650 }])
```

Find documents where the spent amount exceeds the budget:

```
db.monthlyBudget.aggregate([{ $match:{ $expr: { $gt: [ "$spent" , "$budget" ] } } }])
```

$expr can build query expressions that compare fields from the same document in a $match stage.

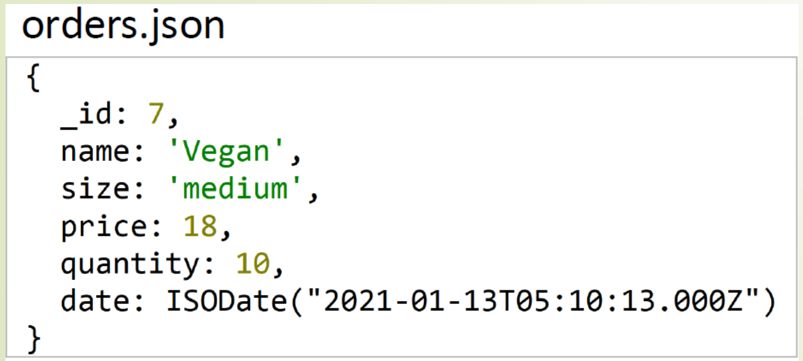


```
[
  { _id: 1, category: 'food', budget: 400, spent: 450 },
  { _id: 2, category: 'drinks', budget: 100, spent: 150 },
  { _id: 5, category: 'travel', budget: 200, spent: 650 }
]
```

2. Aggregation Structure and Syntax
$project stage

- Syntax: `{ $project: { field1: 1, field2: 0,… } }`
- With $project state we can selectively **remove** and **retain** fields and also **reassign** existing field values and **derive** entirely new fields.
- Example:
  - Remove and retain fields form embedded documents

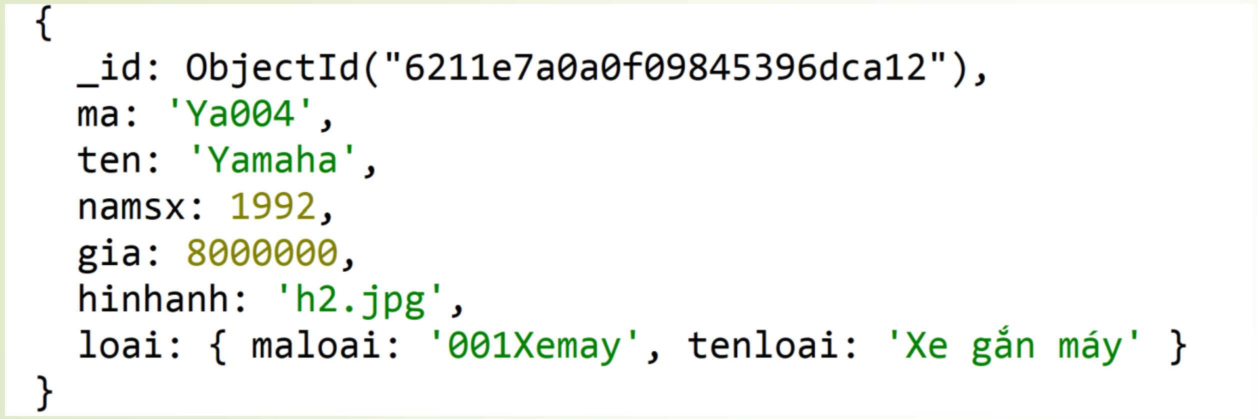
```
db.orders.aggregate([{$project:{name:1, quantity:1, _id:0}}])

db.orders.aggregate([{$match:{'name':'Vegan'}},
                     {$project:{name:1, quantity:1, _id:0}}])
```



orders.json

```
{
  _id: 7,
  name: 'Vegan',
  size: 'medium',
  price: 18,
  quantity: 10,
  date: ISODate("2021-01-13T05:10:13.000Z")
}
```

2. Aggregation Structure and Syntax
$project stage



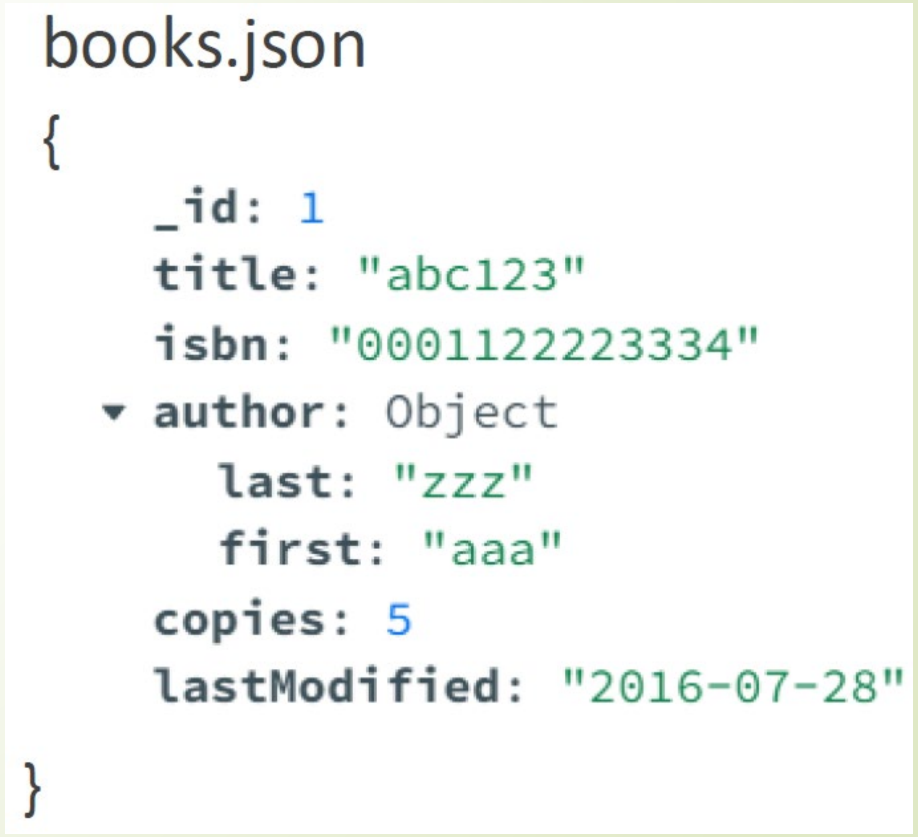
```
{
  _id: ObjectId("6211e7a0a0f09845396dca12"),
  ma: 'Ya004',
  ten: 'Yamaha',
  namsx: 1992,
  gia: 8000000,
  hinhanh: 'h2.jpg',
  loai: { maloai: '001Xemay', tenloai: 'Xe gắn máy' }
}
```

```
db.Xe.aggregate([{$project:{ma:1, ten:1,_id:0, "loai.maloai":1}}])

db.Xe.aggregate([{$project:{
                                ma:1,
                                id:"$loai.maloai",
                                type:"$loai.tenloai"}}])
```

2. Aggregation Structure and Syntax
$project stage

The following $project stage uses the REMOVE variable to excludes the author.middle field only if it equals "":

```
db.books.aggregate( [
{
    $project: {
        title: 1,
        "author.first": 1,
        "author.last": 1,
        "author.middle": {
            $cond: { if: { $eq: [ "", "$author.middle"] },
            then: "$$REMOVE",
            else: "$author.middle"} }
    }
}] )
```



books.json

```
{
    _id: 1
    title: "abc123"
    isbn: "0001122223334"
  ▾ author: Object
      last: "zzz"
      first: "aaa"
    copies: 5
    lastModified: "2016-07-28"
}
```

2. Aggregation Structure and Syntax
$project stage

```
db.books.aggregate(
[{
   $project: {
      title: 1,
      isbn: {
         prefix: { $substr: [ "$isbn", 0, 3] },
         group: { $substr: [ "$isbn", 3, 2] },
         publisher: { $substr: [ "$isbn", 5, 4] },
         title: { $substr: [ "$isbn", 9, 3] },
         checkDigit: { $substr: [ "$isbn", 12, 1] }
      },
      lastName: "$author.last",
      copiesSold: "$copies"
}} ])
```

```
{
"_id" : 1,
"title" : "abc123",
"isbn" : {
    "prefix" : "000",
    "group" : "11",
    "publisher" : "2222",
    "title" : "333",
    "checkDigit" : "4"
},
"lastName" : "zzz",
"copiesSold" : 5
}
```

2. Aggregation Structure and Syntax
$group stage

- Group input documents by the specified _id expression and for each distinct grouping, outputs a document.

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

  - _id: **Required**. If _id value is null, or any other constant value, the $group stage calculates accumulated values for all the input documents as a whole.
  - field: Optional. Computed using the **accumulator operators.**

2. Aggregation Structure and Syntax
$group stage: accumulator operators

- $addToSet: Insert value to array field.

```
{ $addToSet: <expression> }
```

- $sum: Returns a sum of numerical values. Ignores non-numeric values.

```
{ $sum: [ <expression1>, <expression2> ... ] }
```

you can omit [ ], if you have a single expression.

- $count: Returns the number of documents in a group.

```
{ $count: { } }  = { $sum:1 }
```

- $avg: Returns an average of numerical values.

```
{ $avg: [ <expression1>, <expression2> ... ] }
```

2. Aggregation Structure and Syntax
$group stage: accumulator operators

- $max: Returns the highest expression value for each group.

```
{ $max: [ <expression1>, <expression2> ... ] }
```

- $min: Returns the lowest expression value for each group.

```
{ $min: [ <expression1>, <expression2> ... ] }
```

- $first: Returns a value from the first document for each group.

```
{ $first: <expression> }
```

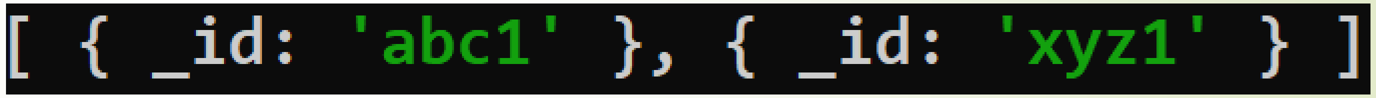
- $last: Returns a value from the last document for each group

```
{ $last: <expression> }
```

2. Aggregation Structure and Syntax
$group stage: Example

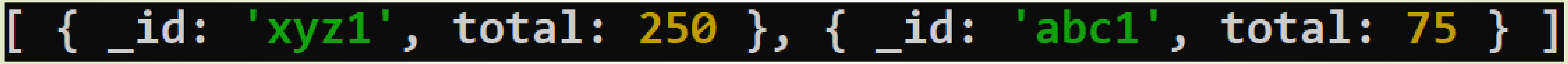
```
{ _id: 1, cust_id: "abc1", ord_date: ISODate("2012-11-02T17:04:11.102Z"), status: "A", amount: 50 }
{ _id: 2, cust_id: "xyz1", ord_date: ISODate("2013-10-01T17:04:11.102Z"), status: "A", amount: 100 }
{ _id: 3, cust_id: "xyz1", ord_date: ISODate("2013-10-12T17:04:11.102Z"), status: "D", amount: 25 }
{ _id: 4, cust_id: "xyz1", ord_date: ISODate("2013-10-11T17:04:11.102Z"), status: "D", amount: 125 }
{ _id: 5, cust_id: "abc1", ord_date: ISODate("2013-11-12T17:04:11.102Z"), status: "A", amount: 25 }
```

```
db.orders.aggregate([{ $group: {_id: "$cust_id"}}])
```



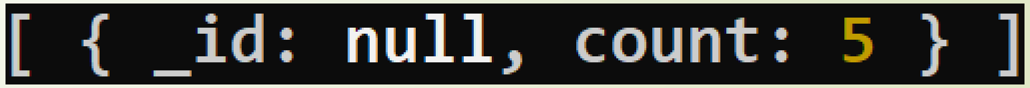
```
[ { _id: 'abc1' }, { _id: 'xyz1' } ]
```

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



```
[ { _id: 'xyz1', total: 250 }, { _id: 'abc1', total: 75 } ]
```

```
db.orders.aggregate([{ $group: {_id: null, count: { $count: {}}} }])
```

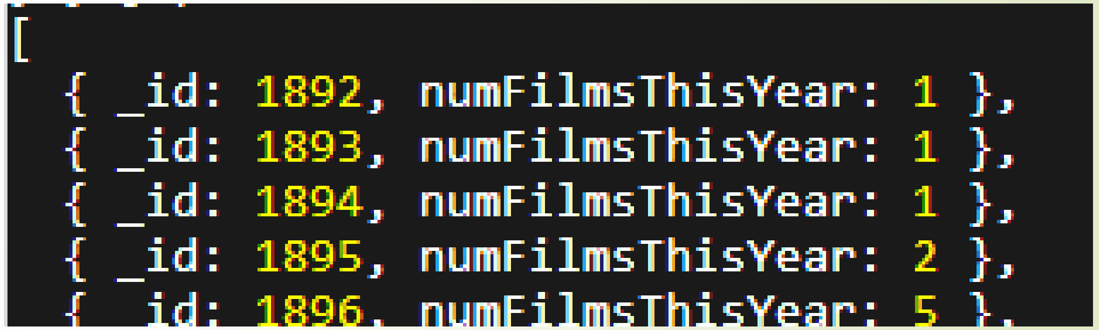


```
[ { _id: null, count: 5 } ]
```

2. Aggregation Structure and Syntax
$group stage: Example

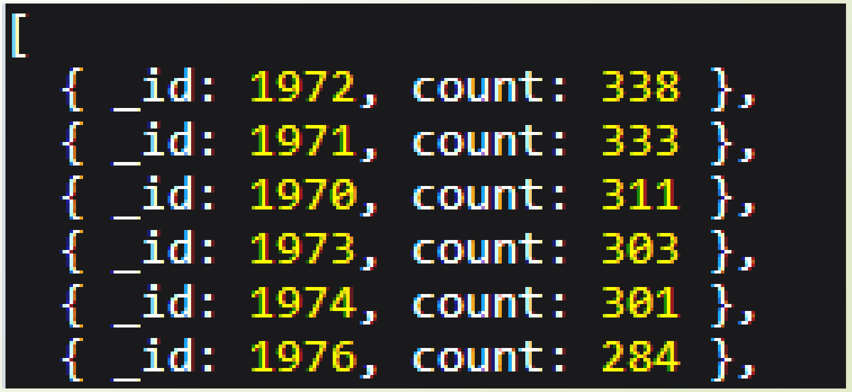
- Group and sort:

```
db.movies.aggregate( [
                { $group: { _id : '$year', 'numFilmsThisYear': { $sum: 1 } } },
                { $sort: { _id : 1} }
] )
```



```
[
  { _id: 1892, numFilmsThisYear: 1 },
  { _id: 1893, numFilmsThisYear: 1 },
  { _id: 1894, numFilmsThisYear: 1 },
  { _id: 1895, numFilmsThisYear: 2 },
  { _id: 1896, numFilmsThisYear: 5 },
```

```
db.movies.aggregate( [
                { $group : { _id : '$year', count : { $sum : 1 } } },
                { $sort : { count : -1} }
] )
```

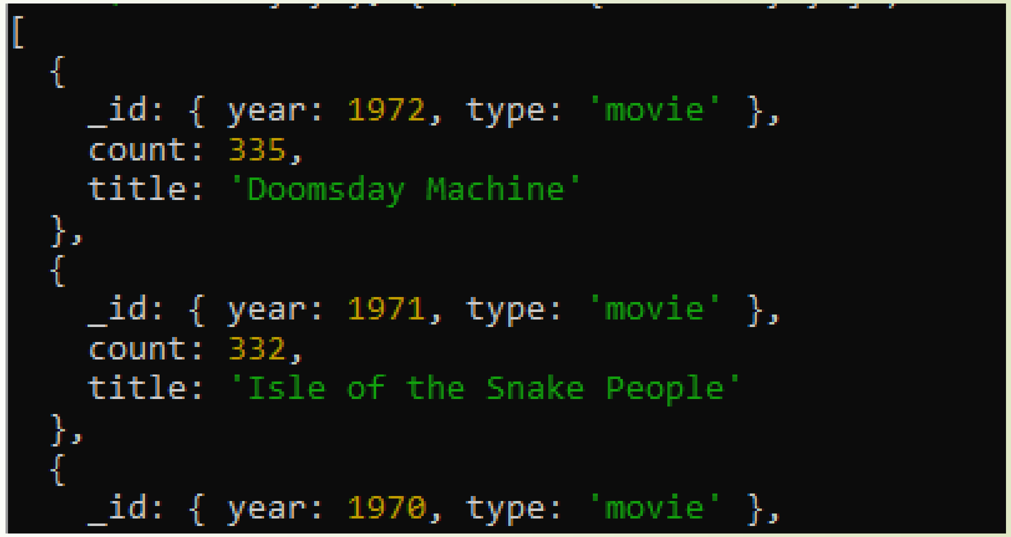


```
[
  { _id: 1972, count: 338 },
  { _id: 1971, count: 333 },
  { _id: 1970, count: 311 },
  { _id: 1973, count: 303 },
  { _id: 1974, count: 301 },
  { _id: 1976, count: 284 },
```

2. Aggregation Structure and Syntax
$group stage: Example

- Grouping on multiple columns:

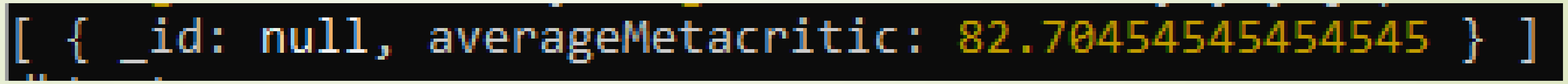
```
db.movies.aggregate( [
   { $group :
            { _id : { year : '$year', type : '$type' },
              count : { $sum : 1 }, title : { $first : '$title'}
            }
   },
   { $sort : { count : -1 } }
] )
```



```
[
  {
    _id: { year: 1972, type: 'movie' },
    count: 335,
    title: 'Doomsday Machine'
  },
  {
    _id: { year: 1971, type: 'movie' },
    count: 332,
    title: 'Isle of the Snake People'
  },
  {
    _id: { year: 1970, type: 'movie' },
```

- Filtering results to only get document with a numeric value:

```
db.movies.aggregate( [
         { $match : { metacritic : { $gte : 0 } } },
         { $group : { _id : null, averageMetacritic: { $avg: '$metacritic' } } }
] )
```



```
[ { _id: null, averageMetacritic: 82.70454545454545 } ]
```

2. Aggregation Structure and Syntax
$set stage

- Add new fields to documents. If the name of the new field is the same as an existing field name (including _id), $set overwrites the existing value of that field with the value of the specified expression.
  - Syntax:
```
{ $set: { <newField>: <expression>, ... } }
```
  - Example:
```
db.scores.insertMany([
{ _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( [
    { $set: {
        totalHomework: { $sum: "$homework" },
        totalQuiz: { $sum: "$quiz" } } },
    { $set: {
        totalScore: { $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ]} }
        }
] )
```

2. Aggregation Structure and Syntax
$count - $skip - $limit - $sort stage

- $count
  - Syntax: { $count: ‘string’ }
  - Example: db.Book.aggregate([{ $count:'Tong so document' }])
- $skip
  - Syntax: { $skip: <positive 64-bit integer> }
  - Example: db.Books.aggregate([{ $skip:3 }])
- $limit
  - Syntax: { $limit: <positive 64-bit integer> }
  - Example: db.Books.aggregate([{ $limit:3 }])
- $sort
  - Syntax: { $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }

    [1: ascending, -1: descending]
  - Example: db.Books.aggregate([{ $sort: {first_name:1, mark:-1} }])

2. Aggregation Structure and Syntax
Arithmetic expression operators

- $add `{ $add: [ <expression1>, <expression2>, ... ] }`
- $subtract
- $multiply
- $divide
- $mod
- $pow
- ….

2. Aggregation Structure and Syntax Arithmetic expression operators

- Example:

```
{ "_id" : 1, "item" : "abc", "price" : 10, "fee" : 2, date: ISODate("2014-03-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "fee" : 1, date: ISODate("2014-03-01T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "fee" : 0, date: ISODate("2014-03-15T09:00:00Z") }
```

```
db.sales.aggregate( [ { $project: { item: 1, total: { $add: [ "$price", "$fee" ] } } } ])
```

```
→   { "_id" : 1, "item" : "abc", "total" : 12 }
    { "_id" : 2, "item" : "jkl", "total" : 21 }
    { "_id" : 3, "item" : "xyz", "total" : 5 }
```

Bài Tập – Customers collection

1. Xuất các khách hàng ở city Wilmington với các thông tin address, city, state.
2. Xuất các khách hàng với các thông tin fed_id, individual, first_name, last_name của officer.
3. Xuất các khách hàng ở city là Woburn và state là MA . Kết quả sẽ được hiển thị chỉ với 2 trường address và officer.
4. Xuất 10 khách hàng đầu tiên thực hiện sắp xếp các kết quả theo thứ tự tăng dần của postal_code và giảm dần của fed_id.
5. Xuất số lượng khách hang ở city Wilmington.
6. Xuất số lượng khách hàng theo từng loại fed_id cho nhóm khách hàng ở city Salem.
7. Nhóm các khách hàng có cùng state và cho biết số lượng khách hàng trên từng state đã gom nhóm.

Bài Tập – collection Customers.json

- Xuất các khách hàng với state MA, sau đó sẽ nhóm các khách hàng vừa tìm được theo city và đếm xem có bao nhiêu khách hàng ở mỗi city. Tiếp theo thực hiện sắp xếp các kết quả vừa tìm được theo postal_code theo thứ tự tăng dần và xuất 5 khách hàng đầu tiên
  - Xuất các khách hàng với state MA .
  - Nhóm các khách hàng vừa tìm được theo city và đếm xem có bao nhiêu khách hàng ở mỗi city.
  - Sắp xếp các kết quả vừa tìm được theo postal_code theo thứ tự tăng dần
  - Xuất 5 khách hàng đầu tiên

2. Aggregation Structure and Syntax
$unwind stage

- Deconstructs an array field from the input documents to output a document for each element.
- Syntax: `{ $unwind: <field path> }`
- Example: db.movies.aggregate([{ $unwind : '$genres' }])

{ Title : *'The Martian'* , genres : [*'Action'*, *'Adventure'*, *'Sci-Fi'*] }
{ Title : *'Batman Begins'* , genres : [*'Action'* , *'Adventure'*] }

$unwind : '$genres'

{ Title : *'The Martian'* , genres : *'Action'* }
{ Title : *'The Martian'* , genres : *'Adventure'* }
{ Title : *'The Martian'* , genres : *'Sci_Fi'* }
{ Title : *'Batman Begins'* , genres : *'Action'* }
{ Title : *'Batman Begins'* , genres : *'Adventure'* }

2. Aggregation Structure and Syntax
$unwind stage

- Example: Group on year and genres of movies collection:

```
db.movies.aggregate( [
        { $unwind : '$genres' },
        { $group : { _id : {year : '$year', genre : '$genres' }, numFilms : { $sum: 1 } } },
        { $sort : { '_id.genre' : 1, numFilms: -1} }
] )
```

- Recap on a few things:
  - $unwind only works on an array of values.
  - Using $unwind on large collections with big documents may lead to performance issues.

2. Aggregation Structure and Syntax
$lookup stage

- To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the 'joined' collection.

```
db.collectionName.aggregate([
  {  $lookup:
      {  from : <collection to join>,
         localField : <field from the input documents>,
         foreignField : <field from the documents of the 'from' collection>,
         as : <output array field>
      }
  }
])
```

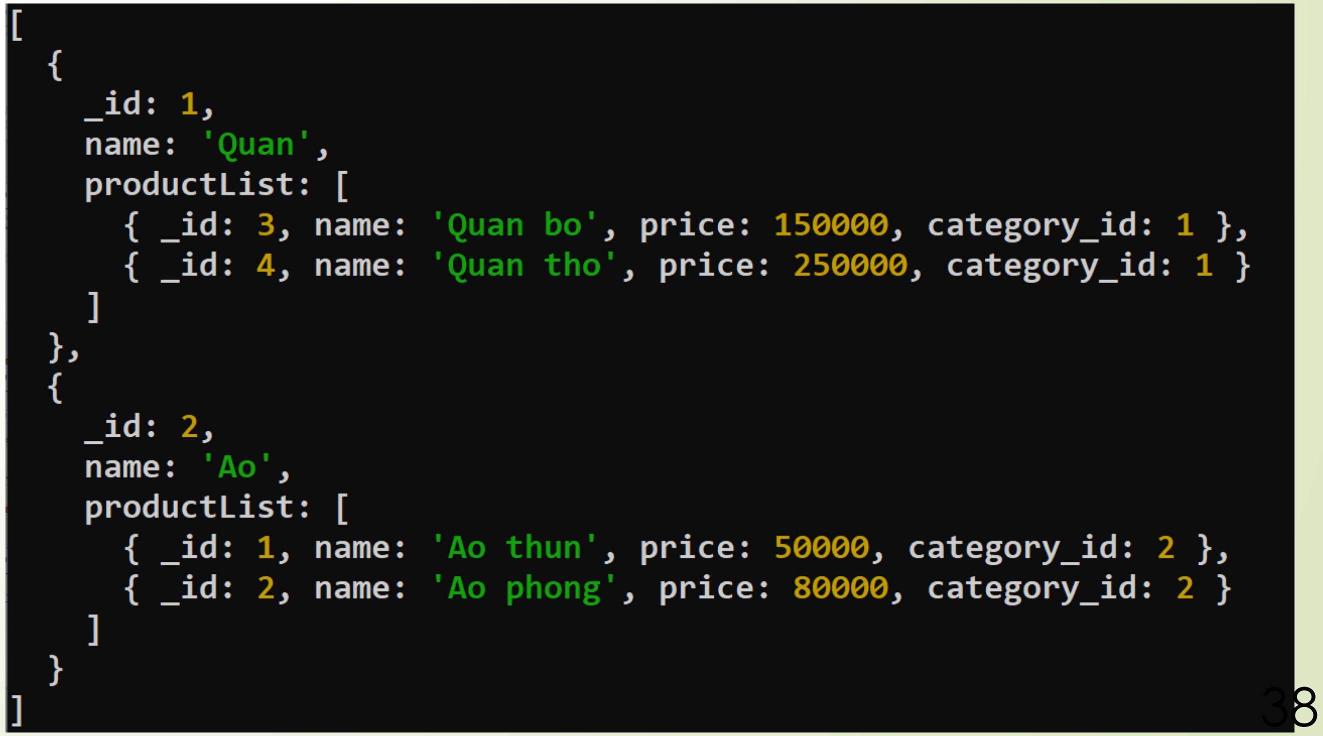
```
SELECT *, <output array field>
FROM collection
WHERE <output array field> IN (
   SELECT *
   FROM <collection to join>
   WHERE <foreignField> = <collection.localField>
);
```

2. Aggregation Structure and Syntax
$lookup stage: Example

category collection:
{_id: 1, name: "Quan"},
{_id: 2, name: "Ao"}

product collection:
{_id: 1, name: 'Ao thun', price: 50000, **category_id: 2**},
{_id: 2, name: 'Ao phong', price: 80000, **category_id: 2**},
{_id: 3, name: 'Quan bo', price: 150000, **category_id: 1**},
{_id: 4, name: 'Quan tho', price: 250000, **category_id: 1**}

```
db.category.aggregate([{
    $lookup: {
        from: 'product',
        localField: '_id',
        foreignField: 'category_id',
        as: 'productList' }
}])
```



```
[
  {
    _id: 1,
    name: 'Quan',
    productList: [
      { _id: 3, name: 'Quan bo', price: 150000, category_id: 1 },
      { _id: 4, name: 'Quan tho', price: 250000, category_id: 1 }
    ]
  },
  {
    _id: 2,
    name: 'Ao',
    productList: [
      { _id: 1, name: 'Ao thun', price: 50000, category_id: 2 },
      { _id: 2, name: 'Ao phong', price: 80000, category_id: 2 }
    ]
  }
]
```

2. Aggregation Structure and Syntax
$lookup stage: Example

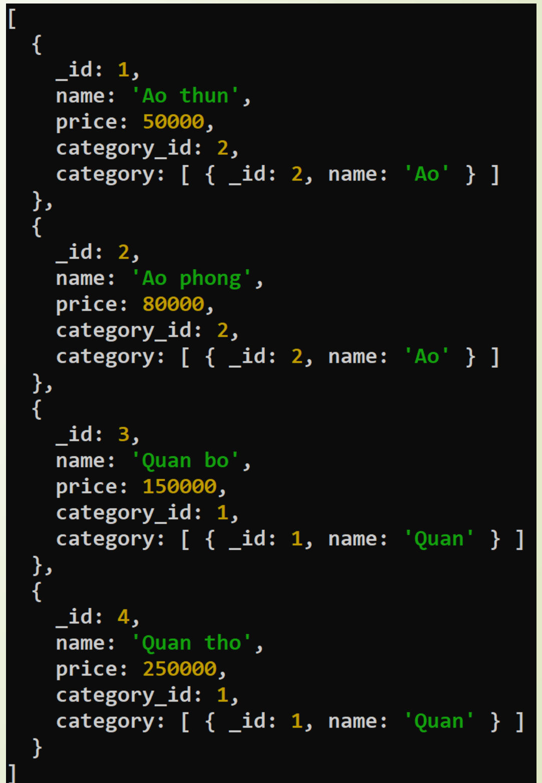
category collection:

```
{_id: 1, name: "Quan"},
{_id: 2, name: "Ao"}
```

product collection:

```
{_id: 1, name: 'Ao thun', price: 50000, category_id: 2},
{_id: 2, name: 'Ao phong', price: 80000, category_id: 2},
{_id: 3, name: 'Quan bo', price: 150000, category_id: 1},
{_id: 4, name: 'Quan tho', price: 250000, category_id: 1}
```

```
db.product.aggregate([{
    $lookup: {
        from: 'category',
        localField: 'category_id',
        foreignField: '_id',
        as: 'category' }
}])
```



```
[
  {
    _id: 1,
    name: 'Ao thun',
    price: 50000,
    category_id: 2,
    category: [ { _id: 2, name: 'Ao' } ]
  },
  {
    _id: 2,
    name: 'Ao phong',
    price: 80000,
    category_id: 2,
    category: [ { _id: 2, name: 'Ao' } ]
  },
  {
    _id: 3,
    name: 'Quan bo',
    price: 150000,
    category_id: 1,
    category: [ { _id: 1, name: 'Quan' } ]
  },
  {
    _id: 4,
    name: 'Quan tho',
    price: 250000,
    category_id: 1,
    category: [ { _id: 1, name: 'Quan' } ]
  }
]
```

2. Aggregation Structure and Syntax
$lookup stage: Correlated subquery

- Syntax: (new in version 5.0)

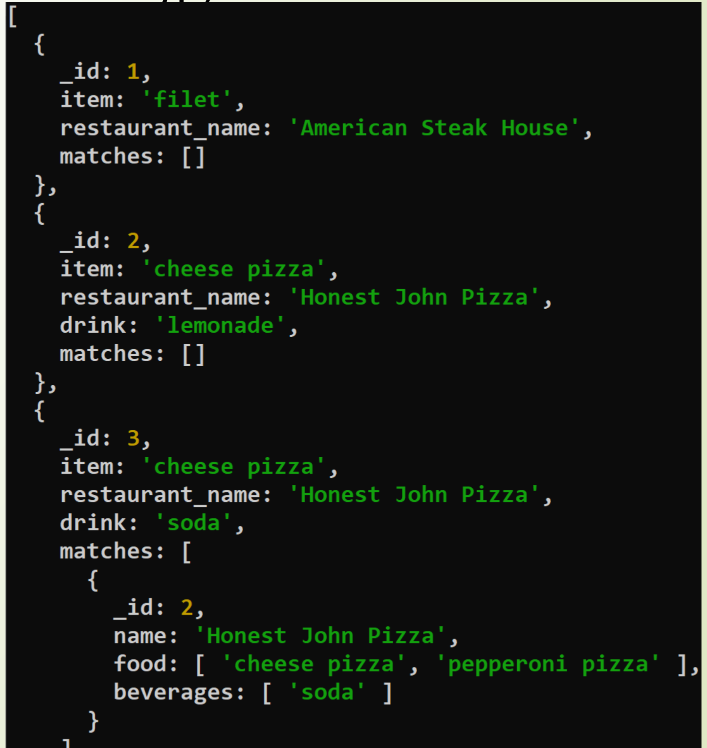
```
{  $lookup :
        {      from : <collection to join>,
               localField : <field from local collection's documents>,
               foreignField : <field from foreign collection's documents>,
               let : { <var_1>: <expression>, …, <var_n>: <expression> },
               pipeline : [ <pipeline to run> ],
               as : <output array field>
        }
}
```

  - *let*: Optional. Specifies the variables to use in the pipeline stages.
  - *pipeline*: determines the resulting documents from the joined collection. To return all documents, specify an empty pipeline []. The pipeline cannot directly access the document fields. Instead, define variables for the document fields using the let option and then reference the variables in the pipeline stages.

2. Aggregation Structure and Syntax $lookup stage: Correlated subquery example

```
db.restaurant.insertMany( [
    { _id: 1, name: "American Steak House", food: [ "filet", "sirloin" ], beverages: [ "beer", "wine" ] },
    { _id: 2, name: "Honest John Pizza", food: [ "cheese pizza", "pepperoni pizza" ], beverages: [ "soda" ] }] )

db.order.insertMany( [
    { _id: 1, item: "filet", restaurant_name: "American Steak House" },
    { _id: 2, item: "cheese pizza", restaurant_name: "Honest John Pizza", drink: "lemonade" },
    { _id: 3, item: "cheese pizza", restaurant_name: "Honest John Pizza", drink: "soda" }] )

db.order.aggregate( [
    { $lookup : {
            from : 'restaurant',
            localField : 'restaurant_name',
            foreignField : 'name',
            let : { orders_drink : '$drink' },
            pipeline : [ {
                $match : { $expr : { $in : [ '$$orders_drink', '$beverages' ] } }
            } ],
            as : 'matches'
    }
} ] )
```

Find the restaurant that match with drink order



```
[
  {
    _id: 1,
    item: 'filet',
    restaurant_name: 'American Steak House',
    matches: []
  },
  {
    _id: 2,
    item: 'cheese pizza',
    restaurant_name: 'Honest John Pizza',
    drink: 'lemonade',
    matches: []
  },
  {
    _id: 3,
    item: 'cheese pizza',
    restaurant_name: 'Honest John Pizza',
    drink: 'soda',
    matches: [
      {
        _id: 2,
        name: 'Honest John Pizza',
        food: [ 'cheese pizza', 'pepperoni pizza' ],
        beverages: [ 'soda' ]
      }
    ]
```

2. Aggregation Structure and Syntax
$lookup stage: Multiple joins and a correlated subquery

- Syntax:

```
{  $lookup :
        {       from : <foreign collection>,
                let : { <var_1>: <expression>, …, <var_n>: <expression> },
                pipeline : [ <pipeline to run> ],
                as : <output array field>
        }
}
```

- Example:

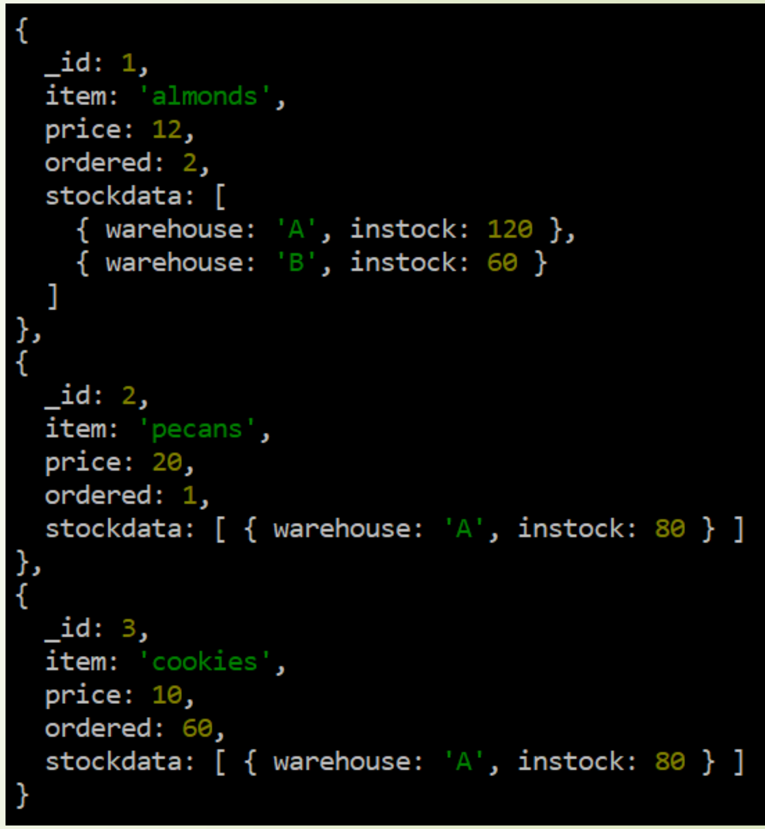
```
db.warehouse.insertMany( [
{ _id : 1, stock_item : "almonds", warehouse: "A", instock : 120 },
{ _id : 2, stock_item : "pecans", warehouse: "A", instock : 80 },
{ _id : 3, stock_item : "almonds", warehouse: "B", instock : 60 },
{ _id : 4, stock_item : "cookies", warehouse: "B", instock : 40 },
{ _id : 5, stock_item : "cookies", warehouse: "A", instock : 80 }] )
```

```
db.order.insertMany( [
{ _id : 1, item : "almonds", price : 12, ordered : 2 },
{ _id : 2, item : "pecans", price : 20, ordered : 1 },
{ _id : 3, item : "cookies", price : 10, ordered : 60 }] )
```

2. Aggregation Structure and Syntax
$lookup stage: Multiple joins and a correlated subquery

Ensures the quantity of the item in stock can fulfill the ordered quantity:

```
db.order.aggregate( [
    { $lookup:
            {   from: "warehouse",
                let: { order_item: "$item", order_qty: "$ordered" },
                pipeline: [
                    { $match:
                        { $expr:
                            { $and: [  { $eq: [ "$stock_item", "$$order_item" ] },
                                          { $gte: [ "$instock", "$$order_qty" ] }
                                       ]
                            }
                        }
                    },
                    { $project: { stock_item: 0, _id: 0 } }
                ],
                as: "stockdata"
    }
}] )
```



```
{
  _id: 1,
  item: 'almonds',
  price: 12,
  ordered: 2,
  stockdata: [
    { warehouse: 'A', instock: 120 },
    { warehouse: 'B', instock: 60 }
  ]
},
{
  _id: 2,
  item: 'pecans',
  price: 20,
  ordered: 1,
  stockdata: [ { warehouse: 'A', instock: 80 } ]
},
{
  _id: 3,
  item: 'cookies',
  price: 10,
  ordered: 60,
  stockdata: [ { warehouse: 'A', instock: 80 } ]
}
```

2. Aggregation Structure and Syntax
$out stage

- Takes the documents returned by the aggregation pipeline and writes them to a specified collection.
- The $out stage must be the last stage in the pipeline.
- Syntax:

```
{ $out: { db: '<output-db>', coll: '<output-collection>' } }
```

  - The $out operation creates a new collection if one does not already exist.
  - If the collection specified by the $out operation already exists, the $out stage atomically replaces the existing collection with the new results collection
- Example:

```
db.movies.aggregate( [
        { $group: { _id: '$year', 'count': { $sum: 1 } } },
        { $sort: { count: -1} },
        { $out: { db: 'reporting', coll: 'movies' } } ] )
```

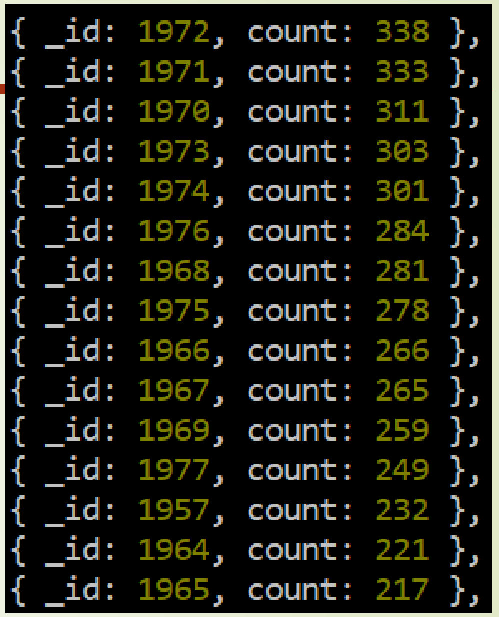
2. Aggregation Structure and Syntax
$merge stage

- Writes the results of the aggregation pipeline to a specified collection.
- The $merge stage must be the last stage in the pipeline.
  - Can output to a collection in the same or different database.
  - Creates a new collection if the output collection does not already exist
  - Can incorporate results (insert new documents, merge documents, replace documents, keep existing documents, process documents with a custom update pipeline) into an existing collection.
- Syntax:

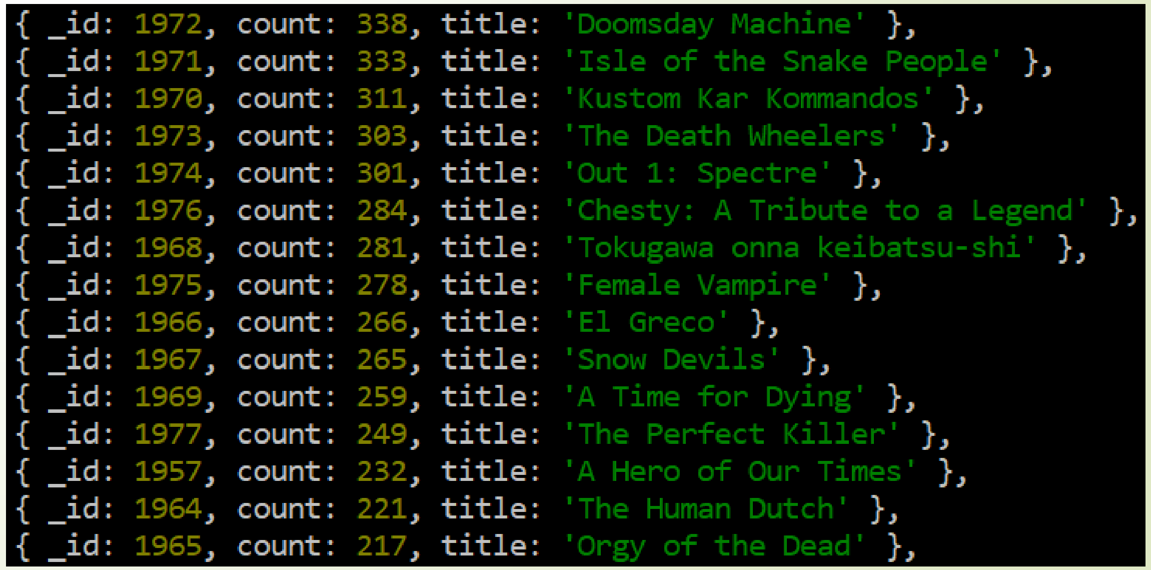
```
{ $merge: {
        into: <collection> -or- {db: <db>, coll: <collection> },
        on: <identifier field> -or- [ <identifier field1>, ...],
        let: <variables>,
        whenMatched:
        <replace|keepExisting|merge|fail|pipeline>,
        whenNotMatched: <insert|discard|fail>
} }
```

2. Aggregation Structure and Syntax
$merge stage: Example

```
db.movies.aggregate( [
      { $group: { _id: '$year', 'count': { $sum: 1 } } },
      { $sort: { count: -1} },
      { $out : { db: 'reporting', coll: 'movies' } }
] )
db.movies.aggregate( [
    { $group: {_id : '$year', 'count': { $sum: 1 }, 'title': {'$first': '$title' }}},
    { $sort: {count:-1} },
    { $merge:
            {  into: { db: 'reporting', coll: 'movies' },
                on:_id,
                whenMatched: 'merge',
                whenNotMatched: 'insert'
            }
    }
] )
```



```
{ _id: 1972, count: 338 },
{ _id: 1971, count: 333 },
{ _id: 1970, count: 311 },
{ _id: 1973, count: 303 },
{ _id: 1974, count: 301 },
{ _id: 1976, count: 284 },
{ _id: 1968, count: 281 },
{ _id: 1975, count: 278 },
{ _id: 1966, count: 266 },
{ _id: 1967, count: 265 },
{ _id: 1969, count: 259 },
{ _id: 1977, count: 249 },
{ _id: 1957, count: 232 },
{ _id: 1964, count: 221 },
{ _id: 1965, count: 217 },
```



```
{ _id: 1972, count: 338, title: 'Doomsday Machine' },
{ _id: 1971, count: 333, title: 'Isle of the Snake People' },
{ _id: 1970, count: 311, title: 'Kustom Kar Kommandos' },
{ _id: 1973, count: 303, title: 'The Death Wheelers' },
{ _id: 1974, count: 301, title: 'Out 1: Spectre' },
{ _id: 1976, count: 284, title: 'Chesty: A Tribute to a Legend' },
{ _id: 1968, count: 281, title: 'Tokugawa onna keibatsu-shi' },
{ _id: 1975, count: 278, title: 'Female Vampire' },
{ _id: 1966, count: 266, title: 'El Greco' },
{ _id: 1967, count: 265, title: 'Snow Devils' },
{ _id: 1969, count: 259, title: 'A Time for Dying' },
{ _id: 1977, count: 249, title: 'The Perfect Killer' },
{ _id: 1957, count: 232, title: 'A Hero of Our Times' },
{ _id: 1964, count: 221, title: 'The Human Dutch' },
{ _id: 1965, count: 217, title: 'Orgy of the Dead' },
```

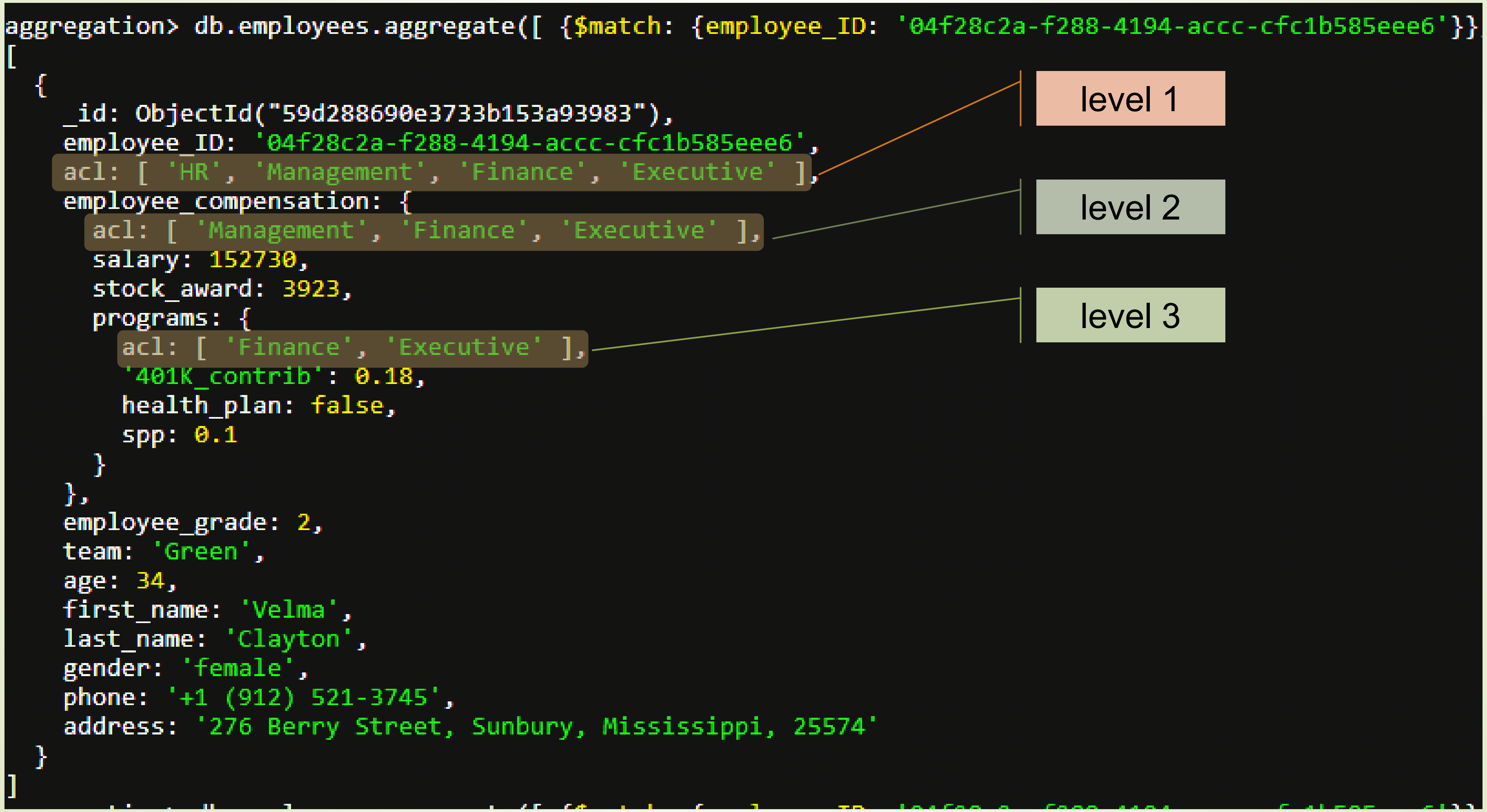
2. Aggregation Structure and Syntax
$redact stage

- Restricts the contents of the documents based on information stored in the documents themselves
- Syntax: { $redact : <expression> }
- The argument can be any valid expression as long as it resolves to the $$DESCEND, $$PRUNE, or $$KEEP system variables

System variable	Description
$$DESCEND	$redact returns the fields at the current document level, excluding embedded documents
$$PRUNE	$redact excludes all fields at this current document/embedded document level, without further inspection of any of the excluded fields
$$KEEP	$redact returns or keeps all fields at this current document/embedded document level, without further inspection of the fields at this level

2. Aggregation Structure and Syntax
$redact stage: Example

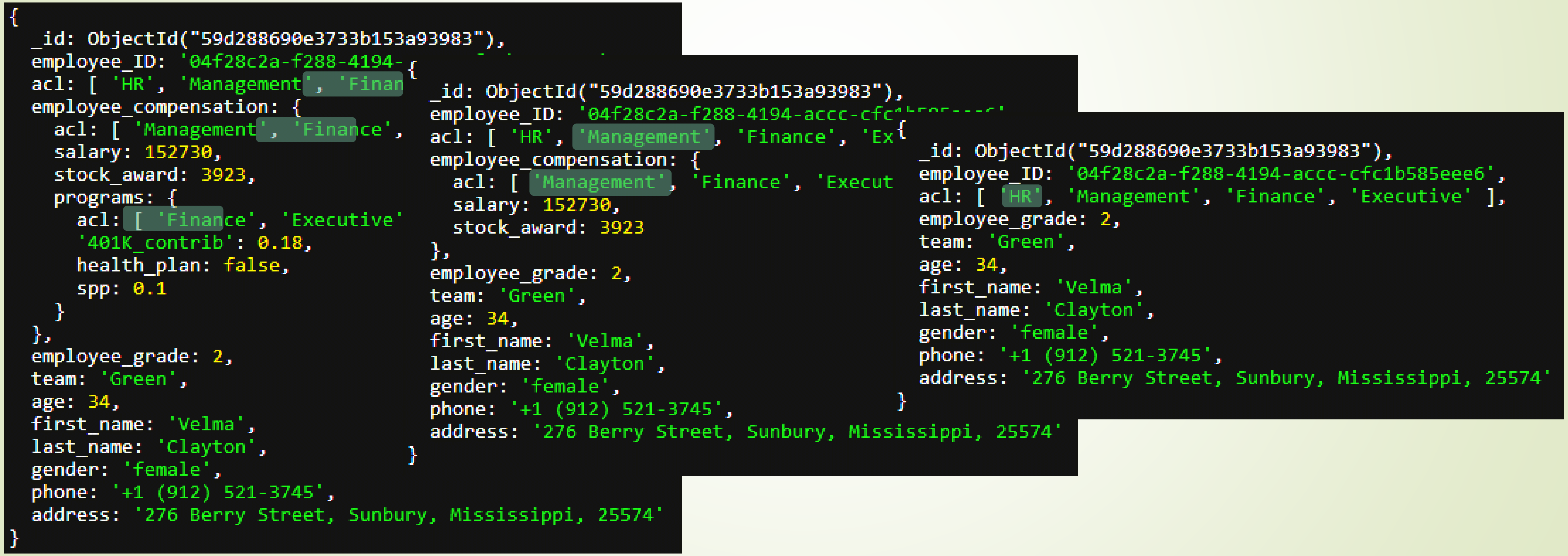
db.employees.aggregate([{ $match: { employee_ID : '04f28c2a-f288-4194-accc-cfc1b585eee6' } }])



```
aggregation> db.employees.aggregate([ {$match: {employee_ID: '04f28c2a-f288-4194-accc-cfc1b585eee6'}}
[
  {
    _id: ObjectId("59d288690e3733b153a93983"),
    employee_ID: '04f28c2a-f288-4194-accc-cfc1b585eee6',
    acl: [ 'HR', 'Management', 'Finance', 'Executive' ],
    employee_compensation: {
      acl: [ 'Management', 'Finance', 'Executive' ],
      salary: 152730,
      stock_award: 3923,
      programs: {
        acl: [ 'Finance', 'Executive' ],
        '401K_contrib': 0.18,
        health_plan: false,
        spp: 0.1
      }
    },
    employee_grade: 2,
    team: 'Green',
    age: 34,
    first_name: 'Velma',
    last_name: 'Clayton',
    gender: 'female',
    phone: '+1 (912) 521-3745',
    address: '276 Berry Street, Sunbury, Mississippi, 25574'
  }
]
```

2. Aggregation Structure and Syntax
$redact stage: Example

```
db.employees.aggregate( [ { $redact : { $cond : [ { $in : [ 'Finance', '$acl' ] }, '$$DESCEND', '$$PRUNE'] } } ] )
db.employees.aggregate( [ { $redact : { $cond : [ { $in : ['Management', '$acl'] }, '$$DESCEND', '$$PRUNE'] } } ] )
db.employees.aggregate( [ { $redact : { $cond : [ { $in : ['HR', '$acl'] }, '$$DESCEND', '$$PRUNE'] } } ] )
```



```
{
  _id: ObjectId("59d288690e3733b153a93983"),
  employee_ID: '04f28c2a-f288-4194-
  acl: [ 'HR', 'Management', 'Finan
  employee_compensation: {
    acl: [ 'Management', 'Finance',
    salary: 152730,
    stock_award: 3923,
    programs: {
      acl: [ 'Finance', 'Executive'
      '401K_contrib': 0.18,
      health_plan: false,
      spp: 0.1
    }
  },
  employee_grade: 2,
  team: 'Green',
  age: 34,
  first_name: 'Velma',
  last_name: 'Clayton',
  gender: 'female',
  phone: '+1 (912) 521-3745',
  address: '276 Berry Street, Sunbury, Mississippi, 25574'
}
```

```
{
  _id: ObjectId("59d288690e3733b153a93983"),
  employee_ID: '04f28c2a-f288-4194-accc-cfc
  acl: [ 'HR', 'Management', 'Finance', 'Ex
  employee_compensation: {
    acl: [ 'Management', 'Finance', 'Execut
    salary: 152730,
    stock_award: 3923
  },
  employee_grade: 2,
  team: 'Green',
  age: 34,
  first_name: 'Velma',
  last_name: 'Clayton',
  gender: 'female',
  phone: '+1 (912) 521-3745',
  address: '276 Berry Street, Sunbury, Mississippi, 25574'
}
```

```
{
  _id: ObjectId("59d288690e3733b153a93983"),
  employee_ID: '04f28c2a-f288-4194-accc-cfc1b585eee6',
  acl: [ 'HR', 'Management', 'Finance', 'Executive' ],
  employee_grade: 2,
  team: 'Green',
  age: 34,
  first_name: 'Velma',
  last_name: 'Clayton',
  gender: 'female',
  phone: '+1 (912) 521-3745',
  address: '276 Berry Street, Sunbury, Mississippi, 25574'
}
```

Contents

3. Create and using View
View

- A MongoDB view is a queryable object whose contents are defined by an aggregation pipeline on other collections or views.
- MongoDB does not persist the view contents to disk. A view's content is computed on-demand when a client queries the view.
- You can:
    - Create a view on a collection of employee data to exclude any private or personal information (PII). Applications can query the view for employee data that does not contain any PII.
    - Create a view on a collection of collected sensor data to add computed fields and metrics. Applications can use simple find operations to query the data
    - …

3. Create and using View
Create View

- Syntax:

```
db.createView(<viewName>, <source>, [<pipeline>], <options>)
```

Parameter	Type	Description
viewName	String	The name of the view to create.
source	String	The name of the source collection or view from which to create the view. You must create views in the same database as the source collection.
pipeline	Array	An array that consists of the aggregation pipeline stage(s). The view definition pipeline cannot include the $out or the $merge stage.
options	Document	Optional. Additional options for the method.

3. Create and using View
Create View: Example

- Create a view in aggregations DB

```
db.createView( 'maleEmployees',
               'employees',
               [   { $match : {'gender' : 'male' } },
                   { $project : {'acl' : 0, 'employee_compensation' : 0} }
               ]
)
```

- Query the view:

```
db.maleEmployees.find()
```

Question?