Introduction to the MongoDB aggregation operations

MongoDB aggregation operations allow you to process multiple documents and return the calculated results.

Typically, you use aggregation operations to group documents by specific field values and perform aggregations on the grouped documents to return computed results.

For example, you can use aggregation operations to take a list of sales orders and calculate the total sales amounts grouped by the products.

To perform aggregation operations, you use aggregation pipelines. An aggregation pipeline contains one or more stages that process the input documents:

Each stage in the aggregation pipeline performs an operation on the input documents and returns the output documents. The output documents are then passed to the next stage. The final stage returns the calculated result.

The operations on each stage can be one of the following:

- $project – select fields for the output documents.
- $match – select documents to be processed.
- $limit – limit the number of documents to be passed to the next stage.
- $skip – skip a specified number of documents.
- $sort – sort documents.
- $group – group documents by a specified key.
- …

The following shows the syntax for defining an aggregation pipeline:

```
db.collection.aggregate([{ $match:...},{$group:...},{$sort:...}]);
```

Code language: CSS (css)

In this syntax:

- First, call the aggregate() method on the collection.
- Second, pass an array of documents, where each document describes a stage in the pipeline.

MongoDB 4.2 or later allows you to use an aggregation pipeline to update documents.

MongoDB aggregation example

First, switch to the coffeeshop database that stores the coffee sales:

```
use coffeeshop
```

Code language: PHP (php)

Second, insert documents into the sales collection:

```
db.sales.insertMany([
	{ "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },
	{ "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },
	{ "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },
	{ "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },
	{ "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },
	{ "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },
	{ "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },
	{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },
	{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },
	{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}
]);
```

Code language: JavaScript (javascript)

Third, use an aggregation pipeline to filter the sales by the Americanos, calculate the sum of quantity grouped by sizes, and sort the result document by the total quantity in descending order.

```
db.sales.aggregate([
	{
		$match: { item: "Americanos" }
	},
	{
		$group: {
			_id: "$size",
			totalQty: {$sum: "$quantity"}
		}
	},
	{
		$sort: { totalQty : -1}
	}
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Grande', totalQty: 33 },
  { _id: 'Short', totalQty: 22 },
  { _id: 'Tall', totalQty: 15 }
]
```

Code language: JavaScript (javascript)

This aggregation pipeline contains three stages:

- Stage 1: the $match stage filters the orders by Americanos coffee and passes the filtered documents to the $group stage.
- Stage 2: the $group stage groups the filtered documents by coffee size and uses the $sum to calculate the total quantity. The $group stage creates a new collection of documents where each document contains two fields _id and totalQty, and passed these documents to the $sort stage.

- Stage 3: the $sort stage sorts the documents by the totalQty field in the descending order and returns the result documents.

SQL equivalent to MongoDB aggregation

If you’re familiar with SQL, the above aggregation pipeline is equivalent to the following SQL statement:

```sql
select
  name as _id,
  sum(quantity) as totalQty
from
  sales
where name = 'Americanos'
group by name
order by totalQty desc;
```

Code language: SQL (Structured Query Language) (sql)

The following table shows the comparison between SQL and MongoDB aggregation:

SQL clause	**MongoDB Aggregation**
select	$project $group functions: $avg, $count, $sum, $max, $min
from	db.collection.aggregate(…)
join	$unwind
where	$match
group by	$group
having	$match

Introduction to the MongoDB $avg

The MongoDB $avg returns the average value of numeric values. The syntax of the $avg is as follows:

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

Code language: HTML, XML (xml)

The $avg ignores the non-numeric and missing values. If all values are non-numeric, the $avg returns null.

MongoDB $avg examples

Let's take some examples of using the $avg. We'll use the following sales collection:

```
db.sales.insertMany([
	{ "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },
	{ "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },
	{ "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },
	{ "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },
	{ "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },
	{ "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },
	{ "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },
	{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },
	{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },
	{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}
]);
```

Code language: JavaScript (javascript)

1) Using the MongoDB $avg to calculate the average quantity for each group

The following example groups the documents by the item field and uses the $avg to calculate the average quantity for each group:

```
db.sales.aggregate([
 {
   $group: {
     _id: '$item',
     averageQty: { $avg: '$quantity' },
   },
 },
]);
```

Code language: PHP (php)

Output:

```
[
 { _id: 'Americanos', averageQty: 17.5 },
 { _id: 'Lattes', averageQty: 27.5 },
 { _id: 'Mochas', averageQty: 11 },
 { _id: 'Cappuccino', averageQty: 16.333333333333332 }
]
```

Code language: JavaScript (javascript)

2) Using the MongoDB $avg to calcualte the average amount

The following example groups the documents by the item field and use the $avg to calculate the average amount for each group:

```
db.sales.aggregate([
 {
   $group: {
     _id: '$item',
     averageAmount: { $avg: { $multiply: ['$quantity', '$price'] } },
   },
 },
 { $sort: { averageAmount: 1 } },
])
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Cappuccino', averageAmount: 127.33333333333333 },
  { _id: 'Americanos', averageAmount: 140 },
  { _id: 'Mochas', averageAmount: 275 },
  { _id: 'Lattes', averageAmount: 562.5 }
]
```

Code language: JavaScript (javascript)

3) Using the MongoDB $avg with $match example

The following example uses the $avg to calculate the average amount per group and returns the group with the average amount greater than 150:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      averageAmount: { $avg: { $multiply: ['$quantity', '$price'] } },
    },
  },
  { $match: { averageAmount: { $gt: 150 } } },
  { $sort: { averageAmount: 1 } },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Mochas', averageAmount: 275 },
  { _id: 'Lattes', averageAmount: 562.5 }
]
```

Code language: JavaScript (javascript)

Summary

- Use MongoDB $avg to return the average value of the numeric values.

Introduction to the MongoDB $count

MongoDB $count returns the number of documents in a group. Here's the syntax of the $count:

{ $count: {} }

Code language: PHP (php)

Note that the $count does not accept any parameters.

The $count is functionally equivalent to using the following [$sum] in the $group stage:

{ $sum: 1 }

Code language: PHP (php)

MongoDB $count examples

We'll use the following sales collection to demonstrate the $count:

db.sales.insertMany([

    { "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },

    { "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },

    { "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },

    { "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },

    { "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },

    { "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },

    { "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },

```
	{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },
	{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },
	{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}
]);
```

Code language: JavaScript (javascript)

1) Using MongoDB $count to count the number of documents per group example

The following example uses the MongoDB $count to return the number of items in the sales collection:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      itemCount: { $count: {} },
    },
  },
])
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Mochas', itemCount: 1 },
  { _id: 'Americanos', itemCount: 4 },
  { _id: 'Lattes', itemCount: 2 },
  { _id: 'Cappuccino', itemCount: 3 }
]
```

Code language: JavaScript (javascript)

In this example:

- _id: "$item" groups the documents by the item field value. It returns four groups for Mochas, Americanos, Lattes, and Cappuchino.
- $count: {} assigns the number of documents that have the same item field value to the itemCount field.

2) Using MongoDB $count with the $match example

The following example uses the $count to calculate the number of documents per item and returns the item with a count greater than two:

```
db.sales.aggregate([
 {
   $group: {
     _id: '$item',
     itemCount: { $count: {} },
   },
 },
 {
   $match: { itemCount: { $gt: 2 } },
 },
]);
```

Code language: PHP (php)

Output:

```
[
 { _id: 'Americanos', itemCount: 4 },
 { _id: 'Cappuccino', itemCount: 3 }
]
```

Code language: JavaScript (javascript)

Summary

- Use MongoDB $count to return the number of documents in a group.

Introduction to the MongoDB $sum

MongoDB $sum returns the sum of numeric values. Here's the syntax of the $sum:

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

Code language: HTML, XML (xml)

Typically, you apply the $sum to the numeric values. However, if a field contains a non-numeric value, the $sum ignores that value. Also, if the field doesn't exist in any document, the $sum returns 0 for that field.

MongoDB $sum examples

We'll use the following sales collection to demonstrate the $sum:

```
db.sales.insertMany([
	{ "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },
	{ "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },
	{ "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },
	{ "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },
	{ "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },
	{ "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },
	{ "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },
	{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },
	{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },
	{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}
]);
```

Code language: JavaScript (javascript)

1) A simple MongoDB $sum example

The following example calculates the total quantity of coffee sales in the sales collection:

```
db.sales.aggregate([
 {
   $group: {
     _id: null,
     totalQty: { $sum: '$quantity' },
   },
 },
]);
```

Code language: PHP (php)

Output:

```
[ { _id: null, totalQty: 185 } ]
```

Code language: JavaScript (javascript)

To remove the _id field from the output document, you can use the $project like this:

```
db.sales.aggregate([
 {
   $group: {
     _id: null,
     totalQty: { $sum: '$quantity' },
   },
 },
 { $project: { _id: 0 } },
]);
```

Code language: PHP (php)

Output:

```
[ { totalQty: 185 } ]
```

Code language: CSS (css)

2) Using mongoDB $sum to calculcate the sum of groups

The following example uses the $sum to calculate the sum of quantity grouped by items:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      totalQty: { $sum: '$quantity' },
    },
  },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Cappuccino', totalQty: 49 },
  { _id: 'Lattes', totalQty: 55 },
  { _id: 'Americanos', totalQty: 70 },
  { _id: 'Mochas', totalQty: 11 }
]
```

Code language: JavaScript (javascript)

In this example, the $group groups the documents by items and the $sum returns the total quantity for each group.

3) Using the MongoDB $sum with $sort example

The following example uses the $sum to returns the total quantity of each item and sorts the result documents by the totalQty in descending order:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      totalQty: { $sum: '$quantity' },
    },
  },
  { $sort: { totalQty: -1 } },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Americanos', totalQty: 70 },
  { _id: 'Lattes', totalQty: 55 },
  { _id: 'Cappuccino', totalQty: 49 },
  { _id: 'Mochas', totalQty: 11 }
]
```

Code language: JavaScript (javascript)

4) Using the MongoDB $sum with $match example

The following example uses the $sum to return the total quantity of each item and $match to include only items with a total quantity greater than 50:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      totalQty: { $sum: '$quantity' },
    },
  },
  { $match: { totalQty: { $gt: 50 } } },
  { $sort: { totalQty: -1 } },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Americanos', totalQty: 70 },
  { _id: 'Lattes', totalQty: 55 }
]
```

Code language: JavaScript (javascript)

5) Using the MongoDB $sum with an expression

The following example uses the $sum to calculate the total amount by multiplying price with quantity for coffee sales group by sizes:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$size',
      totalAmount: { $sum: { $multiply: ['$price', '$quantity'] } },
    },
  },
  { $sort: { totalAmount: -1 } },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Tall', totalAmount: 1285 },
  { _id: 'Grande', totalAmount: 875 },
  { _id: 'Short', totalAmount: 182 }
]
```

Code language: JavaScript (javascript)

Summary

- Use MongoDB $sum to return the collective sum of numeric values.
- The $sum ignores numeric values and return 0 for non-existing fields.

Introduction to the MongoDB $max

The MongoDB $max returns the maximum value. The $max operator uses both value and type for comparing according to the BSON comparison order for values of different types.

The $max operator has the following syntax:

{ $max: <expression> }

Code language: HTML, XML (xml)

If you apply the $max to the field that has a null or missing value in all documents, the $max returns null.

However, if you apply the $max to the field that has a null or missing value in some documents, but not all, the $max only considers non-null and non-missing values for that field.

MongoDB $max examples

We'll use the following sales collection to demonstrate the $max:

db.sales.insertMany([

{ "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },

{ "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },

{ "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },

{ "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },

{ "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },

{ "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },

{ "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },

{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },

{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },

```
	{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}
]);
```

Code language: JavaScript (javascript)

1) Using MongoDB $max to find the maximum value example

The following example uses the $max to find the maximum quantity from all the documents:

```
db.sales.aggregate([
 {
   $group: {
     _id: null,
     maxQty: { $max: '$quantity' },
   },
 },
 {
   $project: {
     _id: 0,
   },
 },
]);
```

Code language: PHP (php)

Output:

```
[ { maxQty: 30 } ]
```

Code language: CSS (css)

2) Using MongoDB $max operator to find the maximum value for each group

The following example uses the $max operator to group documents in the item field and returns the maximum quantity per group of documents:

```
db.sales.aggregate([
 {
   $group: {
      _id: '$item',
      maxQty: { $max: '$quantity' },
    },
  },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Mochas', maxQty: 11 },
  { _id: 'Americanos', maxQty: 22 },
  { _id: 'Lattes', maxQty: 30 },
  { _id: 'Cappuccino', maxQty: 20 }
]
```

Code language: JavaScript (javascript)

3) Using MongoDB $max operator example

The following groups the documents by the item field and returns the maximum amount for each group of sales:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      maxQty: { $max: { $multiply: ['$quantity', '$price'] } },
    },
  },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Mochas', maxQty: 275 },
  { _id: 'Cappuccino', maxQty: 170 },
  { _id: 'Americanos', maxQty: 210 },
  { _id: 'Lattes', maxQty: 750 }
]
```

Code language: JavaScript (javascript)

Summary

- Use the MongoDB $max operator to find the maximum value.

Here's the syntax of the $min operator:

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

Code language: HTML, XML (xml)

If you apply the $min to the field that has a null or missing value in all documents, the $min returns null.

However, if you apply the $min to the field that has a null or missing value in some documents, but not all, the $min only considers non-null and non-missing values for that field.

MongoDB $min examples

We'll use the following sales collection to demonstrate the $min:

```
db.sales.insertMany([
	{ "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },
	{ "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },
	{ "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },
	{ "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },
	{ "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },
	{ "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },

	{ "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },

	{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },

	{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },

	{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}

]);
```

Code language: JavaScript (javascript)

1) Using MongoDB $min to find the minimum value example

The following example uses the $min to find the minimum quantity from all the documents:

```
db.sales.aggregate([
 {
   $group: {
     _id: null,
     maxQty: { $min: '$quantity' },
   },
 },
 {
   $project: {
     _id: 0,
   },
 },
]);
```

Code language: PHP (php)

Output:

```
[ { minQty: 11 } ]
```

Code language: CSS (css)

2) Using MongoDB $min operator to find the minimum value for each group

The following example uses the $min operator to group documents in the item field and returns the minimum quantity per group of documents:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      minQty: { $min: '$quantity' },
    },
  },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Mochas', minQty: 11 },
  { _id: 'Americanos', minQty: 12 },
  { _id: 'Lattes', minQty: 25 },
  { _id: 'Cappuccino', minQty: 12 }
]
```

Code language: JavaScript (javascript)

3) Using MongoDB $min operator example

The following groups the documents by the item field and returns the minimum amount for each group of sales:

```
db.sales.aggregate([
  {
    $group: {
      _id: '$item',
      maxQty: { $min: { $multiply: ['$quantity', '$price'] } },
    },
  },
]);
```

Code language: PHP (php)

Output:

```
[
  { _id: 'Cappuccino', minQty: 72 },
  { _id: 'Americanos', minQty: 110 },
  { _id: 'Lattes', minQty: 375 },
  { _id: 'Mochas', minQty: 275 }
]
```

Code language: JavaScript (javascript)

Summary

- Use the MongoDB $min operator to find the minimum value.