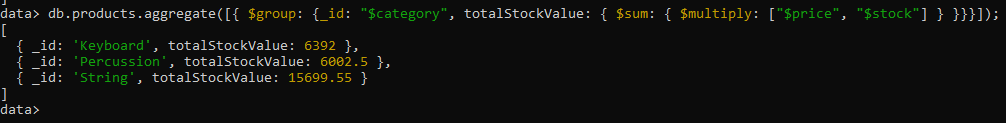
# Write queries for the given questions using the above data.

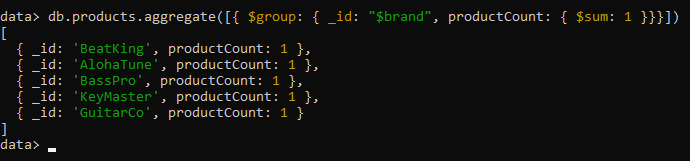
## Calculate Total Stock Value by Category.

db.products.aggregate([{ $group: {\_id: "$category", totalStockValue: { $sum: { $multiply: ["$price", "$stock"] } }}}]);



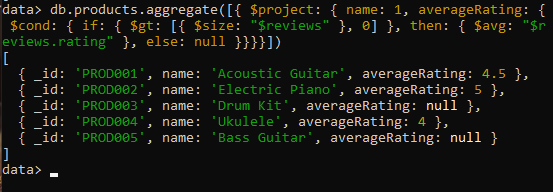
## Count Products per Brand.

db.products.aggregate([{ $group: { \_id: "$brand", productCount: { $sum: 1 }}}])



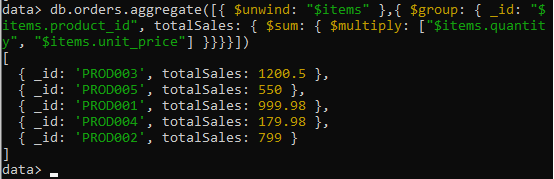
## Find the Average Rating for Each Product.

db.products.aggregate([{ $project: { name: 1, averageRating: { $cond: { if: { $gt: [{ $size: "$reviews" }, 0] }, then: { $avg: "$reviews.rating" }, else: null }}}}])



## Calculate Total Sales for Each Product.

db.orders.aggregate([{ $unwind: "$items" },{ $group: { \_id: "$items.product\_id", totalSales: { $sum: { $multiply: ["$items.quantity", "$items.unit\_price"] }}}}])



## Recalculate each order's total amount using $reduce.

db.orders.aggregate([

{

$project: {

\_id: 1, customer\_id: 1, order\_date: 1, items: 1, status: 1,

recalculated\_total: {

$reduce: {

input: "$items",

initialValue: 0,

in: {

$add: [

"$$value",

{ $multiply: ["$$this.quantity", "$$this.unit\_price"] }

]

}

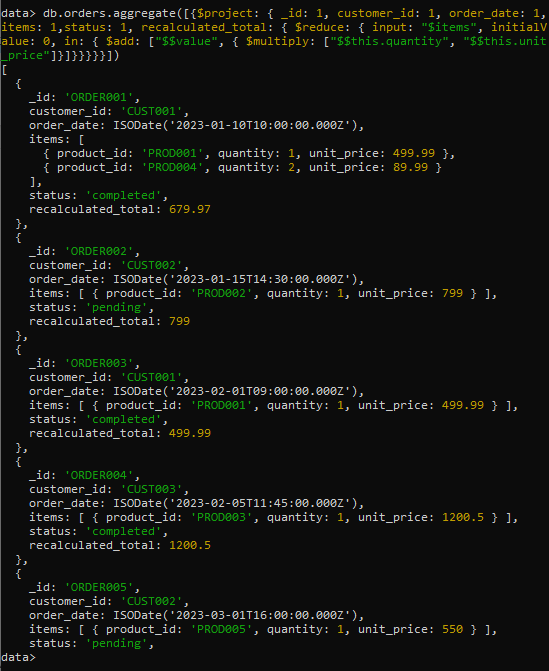
}

}

}

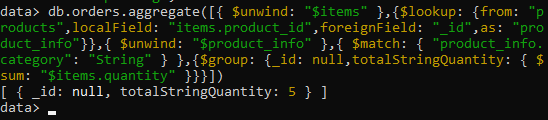
}

])



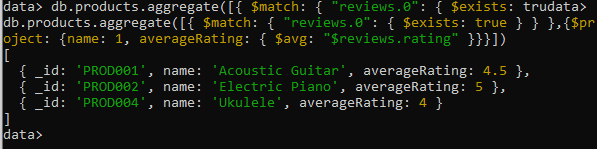
## Find the total quantity of 'String' products sold.

db.orders.aggregate([{ $unwind: "$items" },{$lookup: {from: "products",localField: "items.product\_id",foreignField: "\_id",as: "product\_info"}},{ $unwind: "$product\_info" },{ $match: { "product\_info.category": "String" } },{$group: {\_id: null,totalStringQuantity: { $sum: "$items.quantity" }}}])



## Find the average rating for each product that has received at least one review.

db.products.aggregate([{ $match: { "reviews.0": { $exists: true } } },{$project: {name: 1, averageRating: { $avg: "$reviews.rating" }}}])



## For each product, create a summary that includes its name and price, a simple list of the usernames who reviewed it, and a list of the order IDs in which it was sold.

db.products.aggregate([

{

$lookup: {

from: "orders",localField: "\_id",foreignField: "items.product\_id", as: "order\_details"

}

},{

$project: {

name: 1, price: 1,

reviewers: {

$map: {

input: "$reviews", as: "review", in: "$$review.user"

}

},

order\_ids: {

$map: {

input: "$order\_details", as: "order", in: "$$order.\_id"

}

}

}

}

])



## Retrieve a list of all orders, but instead of just showing product IDs in the items array, replace them with a more detailed object containing the product's name, brand, and category.

db.orders.aggregate([

{ $unwind: "$items" },

{

$lookup: {

from: "products", localField: "items.product\_id", foreignField: "\_id",

as: "product\_details"

}

},

{ $unwind: "$product\_details" },

{

$group: {

\_id: "$\_id",

customer\_id: { $first: "$customer\_id" },

order\_date: { $first: "$order\_date" },

status: { $first: "$status" },

total\_amount: { $first: "$total\_amount" },

items: {

$push: {

product\_id: "$items.product\_id",

quantity: "$items.quantity",

unit\_price: "$items.unit\_price",

product\_name: "$product\_details.name",

brand: "$product\_details.brand",

category: "$product\_details.category"

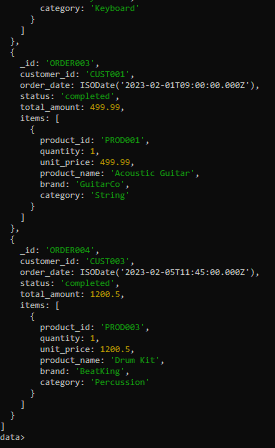
}

}

}

}

])

## Create a profile for each customer showing a list of products they have purchased. The list should not contain duplicates and should include the product name and category.

db.orders.aggregate([

{ $unwind: "$items" },

{

$lookup: {

from: "products", localField: "items.product\_id", foreignField: "\_id",

as: "product\_details"

}

},

{ $unwind: "$product\_details" },

{

$group: {

\_id: "$customer\_id",

purchased\_products: {

$addToSet: {

name: "$product\_details.name",

category: "$product\_details.category"

}

}

}

}

])



# Create documentation on Map-Reduce and the Input-Output model in MongoDB.

**Map-Reduce in MongoDB**

**Overview:**  
Map-Reduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. In MongoDB, it's used for complex aggregation operations that are difficult to express using the aggregation framework.

**Map Function:**

* Processes each input document
* Emits key-value pairs
* Can be customized for specific data processing needs

**Reduce Function:**

* Processes all values associated with a particular key
* Returns a single aggregated result for each key
* Must be idempotent (can be called multiple times with same input)

**Basic Syntax:**

db.collection.mapReduce(

mapFunction,

reduceFunction,

{

out: "output\_collection",

query: { /\* optional filter \*/ }

}

)

**Input-Output Model:**

**Input:**

* Documents from a MongoDB collection
* Can be filtered using the query parameter
* Each document is processed by the map function

**Processing:**

1. **Map Phase:** Each document → emits key-value pairs
2. **Shuffle Phase:** Groups values by key
3. **Reduce Phase:** Processes grouped values → emits results

**Output Options:**

* {out: "collection"} - Output to a new collection
* {out: {inline: 1}} - Return results directly (limited to 16MB)
* {out: {merge: "collection"}} - Merge with existing collection
* {out: {reduce: "collection"}} - Reduce with existing collection
* {out: {replace: "collection"}} - Replace existing collection

**Example:**

// Count products by category

var mapFunction = function() {

emit(this.category, 1);

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.products.mapReduce(

mapFunction,

reduceFunction,

{ out: "product\_counts\_by\_category" }

);

**When to Use Map-Reduce:**

* Complex data processing not supported by aggregation framework
* Custom JavaScript logic required
* Batch processing of large datasets
* Historical data analysis

**Limitations:**

* Slower than aggregation framework for most operations
* JavaScript execution can be resource-intensive
* Not suitable for real-time queries
* Limited to single MongoDB instance for inline results