Kayla McDanel  
09/07/2022  
Aggregation Operations

Aggregation operations can be used to group values, perform operations on grouped data, and analyze data changes over time (Aggregation Operations, n.d.). The aggregation operations return the computed result and are typically performed through pipelines or single purpose aggregation methods. The pipeline method is where you’ll find use of $project, $match, and $sort. A pipeline consists of a series of stages that process documents within MongoDB. After data is input into the database, aggregation operations can be used to filter and sort it in various ways.

The $project aggregation operation adds requested fields into a document, then passes it along down the pipeline. These documents can specify the inclusion of fields, suppression of the \_id field, addition of new fields, updating of existing fields, and even exclusion of fields ($project (aggregation), n.d.). One important thing to note is that you cannot employ other $project specifications if you specify the exclusion of a field other than \_id. An example of the $project aggregation operation would consider the following data:

{

"\_id" : 1,  
title: "abc123",  
isbn: "0001122223334",  
author: { last: "zzz", first: "aaa" },  
copies: 5

}

This $project stage includes the \_id, title, and author in the output document:

db.books.aggregate( [ { $project : { title : 1 , author : 1 } } ] )

It would return the following operation results:

{ "\_id" : 1, "title" : "abc123", "author" : { "last" : "zzz", "first" : "aaa" } }

The $match operation is used to filter documents by fields, such as date or a date range, then passes the remaining documents to the $group stage. An example of the $match operation would be to filter pizzas by size:

db.orders.aggregate( [

{

$match: { size: "medium" }

},

The $sort aggregation operation is used to sort the input documents and return a sorted order. This order can be from ascending or descending, or by textScore metadata in descending order. Multiple fields are sorted left to right with a maximum of 32 keys ($sort aggregation, n.d.). Consider the following data for restaurants:

db.restaurants.insertMany( [

{ "\_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan"},

{ "\_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens"},

{ "\_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn"},

{ "\_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan"},

{ "\_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn"},

] );

This aggregation operation will sort the “borough” field:

db.restaurants.aggregate(

[

{ $sort : { borough : 1 } }

])

Which will return the following results. It is important to note that the return “borough” fields are in alphabetical order, but duplicate boroughs are not organized further. To do so, more specifications would need to be added.

{ "\_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn" }

{ "\_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn" }

{ "\_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan" }

{ "\_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan" }

{ "\_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens" }

# References

*$project (aggregation)*. (n.d.). Retrieved from MongoDB: https://www.mongodb.com/docs/manual/reference/operator/aggregation/project/

*$sort aggregation*. (n.d.). Retrieved from MongoDB: https://www.mongodb.com/docs/manual/reference/operator/aggregation/sort/#mongodb-pipeline-pipe.-sort

*Aggregation Operations*. (n.d.). Retrieved from MongoDB: https://www.mongodb.com/docs/manual/aggregation/