How to use Aggregation Pipeline in MongoDB

If you are using MongoDB's MapReduce, it is best to switch to Aggregation Pipeline for more efficient computation.

Aggregation Pipeline is the recommended way to run complex queries in MongoDB. If you are using MongoDB's MapReduce, it is best to switch to Aggregation Pipeline for more efficient computation.

What is Aggregation Pipeline in MongoDB?

Aggregation Pipeline is a multi-phase process that runs advanced queries in MongoDB. It processes data through different stages called pipelines. You can use the results generated from a level as a working sample

For example, you can pass the results of a match operation to another stage to sort them until you get the desired result.

Each stage of an **Aggregation Pipeline** includes a MongoDB operator and creates one or more transformed documents. Depending on your query, a level may appear multiple times in the process. For example, you may need to use the $count or $sort operator stages multiple times in the aggregation process.

Stages of Aggregation Pipeline

Aggregation Pipeline passes data through multiple stages in a single query. You can find details about some of the document arbitration stages in MongoDB.

Below are some of the most common stages.

$match . stage

This stage helps you define specific filtration conditions before starting the other synthesis stages. You can use it to select the appropriate data that you want to include in the aggregation process.

$group . stage

The grouping phase separates data into different groups based on specific criteria using key-value pairs. Each group represents a key in the output document.

For example, consider the following sales sample data:

Using an aggregation pipeline, you can calculate total sales and peak sales for each product group:

```
{ $group: { _id: $Section, total_sales_count: {$sum : $Sold}, top_sales: {$max: $
```

The _id:$Section pair groups the output document based on sections. By specifying the **top_sale_count** and **top_sale** fields , MongoDB generates new keys based on the activity determined by the aggregator; this can be $sum, $min, $max or $avg.

$skip . stage

You can use the $skip stage to skip a specified number of documents in the output. It usually takes place after the group phase. For example, if you expect two output documents but ignore one, aggregation will only output the second document.

To add a skip stage, insert the **$skip** operator into the aggregation pipeline:

```
., { $skip: 1 },
```

$sort stage

The sorting stage allows you to sort the data in descending or ascending order. For example, sort the data in the previous query example in descending order to decide which section has the highest sales.

Add the $sort operator to the previous query:

```
., { $sort: {top_sales: -1} },
```

$limit . stage

The limit operator reduces the number of output documents you want the Aggregation Pipeline to display. **For example, use the $limit** operator to get the highest revenue portion returned by the previous period:

```
., { $sort: {top_sales: -1} }, {"$limit": 1}
```

The results return only the first document, which has the highest sales volume because it appears at the top of the categorized results.

$project . stage

The $project phase allows you to shape the resulting document as desired. Using the $project operator, you can specify the field to include in the result and customize its key name.

For example, a sample output without the **$project** phase looks like this:

Let's see how it looks when combined with the **$project** stage . To add $project to the pipeline:

```
., { "$project": { "_id": 0, "Section": "$_id", "TotalSold": "$total_sales_count"
```

Since we ungrouped the data based on product parts, the data above includes each product part in the output document. It also ensures that the aggregated sales numbers and top sales features in the output are **TotalSold** and **TopSale** .

The end result is much more compact than the previous version:

How to create Aggregation Pipeline in MongoDB

Although the aggregation process includes several operations, the previously highlighted stages give you an idea of ??how to apply them in the process, including the basic query for each operation.

Using the previous sales data sample, let's synthesize some of the stages discussed above to better understand the Aggregation Pipeline in MongoDB:

```
db.sales.aggregate([ { "$match": { "Sold": { "$gte": 5 } } }, { "$group": { "_id"
```

Result:

Above is **how to use Aggregation Pipeline in MongoDB** . Hope the article is useful to you.

You finished reading the article "**How to use Aggregation Pipeline in MongoDB**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.

© 2019 TipsMake.com