MongoDB Aggregation Pipelines

Cynthia Amoaba

Aggregation Pipelines

- Aggregation operations are used in MongoDB to allow us group, sort, perform calculations, analyze data, and much more. Using this framework, MongoDB passes the documents of a single collection through a pipeline.

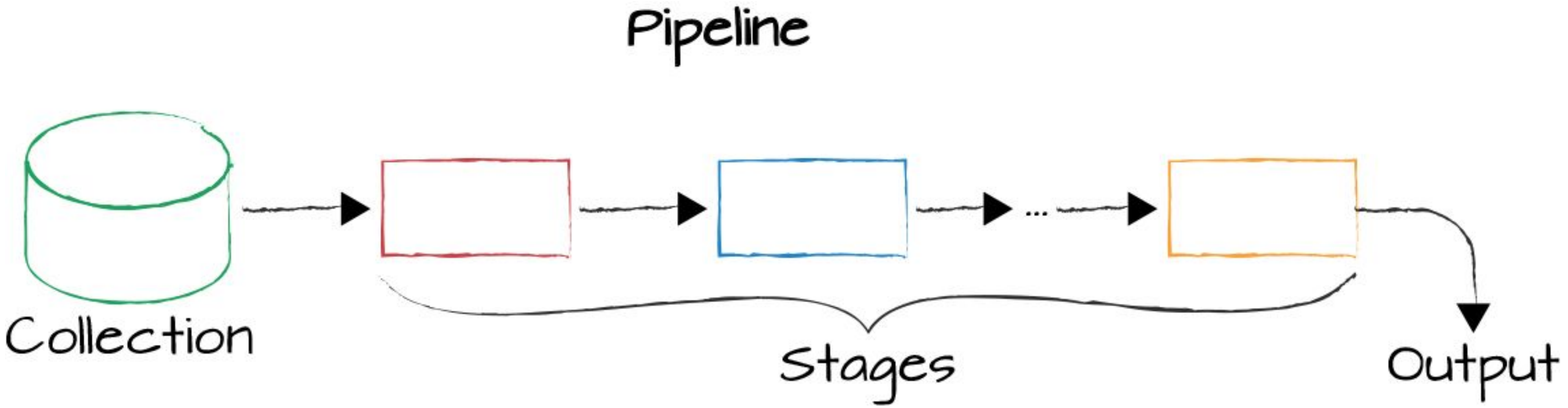
- Aggregation pipelines can have one or more stages. The order of these stages are important.

- Aggregation **knobs** or **tunables** typically take the form of **operators** that we can supply, that will modify fields, perform arithmetic operations, reshape documents, or do some sort of accumulation task or a variety of other things.

- Each stage acts upon the results of the previous stage.

The Aggregation Pipeline

- An individual stage of an aggregation pipeline is a data processing unit. It **takes** in a stream of input documents **one at a time**, **processes** each document **one at a time**, and **produces** an **output** stream of documents **one at a time**.



Example

```
db.posts.aggregate([
  // Stage 1: Only find documents that have more than 1 like
  {
    $match: { likes: { $gt: 1 } }
  },
  // Stage 2: Group documents by category and sum each categories likes
  {
    $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
  }
    ])
```

Aggregation $group

- This aggregation stage groups documents by the unique `_id` expression provided.

- Don't confuse this `_id` expression with the `_id` **ObjectId** provided to each document.

-
  ```
  db.listingsAndReviews.aggregate(
          [ { $group : { _id : "$property_type" } } ]
  )
  ```

- This will return the distinct values from the `property_type` field.

Aggregation $limit

- This aggregation stage limits the number of documents passed to the next stage.

- `db.movies.aggregate([ { $limit: 1 } ])`

- This will return the 1 movie from the collection.
-

(data from -"sample_mflix" database loaded from our sample data in the Intro to Aggregations section - W3Schools)

Aggregation $project

- This aggregation stage passes only the specified fields along to the next aggregation stage.
- Same projection that is used in the find method.
- This will return the documents but only include the specified fields.
- _id field is also included unless specifically excluded.
- We can use 1 to include a field and 0 to exclude a field

Aggregation $project

```
db.restaurants.aggregate([
  {    $project: {
       "name": 1,
       "cuisine": 1,
       "address": 1
    }
  },
  {    $limit: 5
  }
  ])
```

Aggregation $sort

- This aggregation stage groups sorts all documents in the specified sort order.
- Remember that each stage will only work on the documents that the previous stage provides.
- This will return the documents sorted in descending order by the **accommodates** field.
- Sort order can be chosen using -1 or 1.

Aggregation $sort

```
db.listingsAndReviews.aggregate([
 {    $sort: { "accommodates": -1 }
 },
 {   $project: {
    "name": 1,
    "accommodates": 1
  } },
 {
  $limit: 5
 }
   ])
```

Aggregation $match

- This aggregation stage behaves like a find. It will filter documents that match the query provided.
- It is best to use $match early in the pipeline to help improve performance since it limits the number of documents the next stage must process.
- This will only return documents that have the **property_type** of "House".

```
db.listingsAndReviews.aggregate([
  { $match : { property_type : "House" } },
  { $limit: 2 },
  { $project: {
    "name": 1,
    "bedrooms": 1,
    "price": 1
  }}

])
```

Aggregation $addFields

- This aggregation stage adds new fields to documents.
- This will return the documents along with a new field, **avgGrade**, which will contain the average of each restaurants **grades.score.**

```
db.restaurants.aggregate([

 {   $addFields: {

    avgGrade: { $avg: "$grades.score" }

   }

 },

    {
```

Aggregation $addFields

```
$project: {
     "name": 1,
     "avgGrade": 1
   }
 },
 {
   $limit: 5
 }
   ])
```

Aggregation $count

- This aggregation stage counts the total amount of documents passed from the previous stage.
- This will return the number of documents at the **$count** stage as a field called **"totalChinese".**

```
db.restaurants.aggregate([
 {
  $match: { "cuisine": "Chinese" }
 },
 {
  $count: "totalChinese"
 }

])
```

Data From- In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

Aggregation $lookup

- This aggregation stage performs a left outer join to a collection in the same database.

- There are four required fields:

  from: The collection to use for lookup in the same database

  localField: The field in the primary collection that can be used as a unique identifier in the **from** collection.

  foreignField: The field in the **from** collection that can be used as a unique identifier in the primary collection.

  as: The name of the new field that will contain the matching documents from the **from** collection.

Aggregation $lookup

```
db.comments.aggregate([ {   $lookup: {
      from: "movies",
      localField: "movie_id",
      foreignField: "_id",
      as: "movie_details",
   }, },
  {   $limit: 1
  }])
```

This will return the movie data along with each comment.

Aggregation $out

- This aggregation stage writes the returned documents from the aggregation pipeline to a collection.

- This must be the last stage in the pipeline

- In the example below, The first stage will group properties by the **property_type** and include the **name**, **accommodates**, and **price** fields for each. The **$out** stage will create a new collection called **properties_by_type** in the current database and write the resulting documents into that collection.

Aggregation $out

```
db.listingsAndReviews.aggregate([

 {   $group: {

    _id: "$property_type",

    properties: {

      $push: {

        name: "$name",

        accommodates: "$accommodates",

        price: "$price",

      },     },

  },  },  { $out: "properties_by_type" },

  ])
```

Thank you.

:)