MONGODB AGGREGATION PIPELINES

An aggr. pipeline consist of one (or) more stages that process documents.

- Each stage performs an operation on i/p documents. Ex: a stage can filter documents & calculate values.
- The documents that are output from a stage are passed to next stage.
- An aggregation pipeline can return results for group of documents. Ex: return total, avg, max & min. values

- **References:** → MongoDB for VSCode → Atlas → Database → Browse Collections

- **$match:** → filters the docs to pass only the docs that match the specified condition(s) to the next pipeline stage
  → {$match: {<query>}}

- **$count:** → passes a document to the next stage that contains a count of the no. of documents i/p to the stage.
  → {$count: <string>}

Ex:
```
[{
  $match: { isActive : true }
},
{
  $count: "activeUsers",
}]
```
(key, value in DB; just a name)

o/p: activeUsers 516

- **$group:** → This seperates documents into groups according to a "group key". The o/p is 1 document for each unique group key

Ex: find avg. age of all users →
```
[{
  $group : {
    _id : null,
    averageAge : {
      $avg: "$age"
    }
  }
}]
```
(_id: null → basically groups whole data into 1 group; averageAge: just a name; "$age" → key in DB)

o/p:
```
_id: null
averageAge: 29.83
```

- **$sum:** → calculates & returns the collective sum of numeric values.
  → {$sum: <expression>}
  (is an accumulator)

- **$limit:** → limits the no. of documents passed to the next stage in pipeline.
  → {$limit: <positive integer>}

- **$sort** : → It orders the elements of an array during a $push operation.

ex: find top 5 favourite fruit of people

```
[{ $group: {
     _id: "$favoriteFruit",        ← key in DB
     count: {
       $sum: 1
     }
   }
},
{ $sort : { count : -1 } },        ← sorts in descending
{ $limit : 5 }                     ← +1 : sorts in ascending
]
```

o/p:
```
{ _id: "apple", count: 337 }
{ _id:
```
(only 5)

- **$avg** : → returns the avg. value of the numeric values
  → { $avg: <expression> }

ex: which country has highest no. of registered users?

```
[{ $group: {                       ← 1st stage
     _id: "$company.location.country",   ← if key present inside obj., we can access like this
     userCount: { $sum: 1 }
   }
},
{ $sort: { userCount: -1 } },      ← 2nd stage
{ $limit: 1 }                      ← 3rd stage
]
```

o/p:
```
{ _id: "Germany", userCount: 213 }
```

→ must be same, as we are applying 'sort' based on that

→ it's not a key in DB, but since it's pipeline here, we can use it in 'sort'

- **$unwind** : → deconstructs an array field from the input documents to output a document for each element. Each o/p doc. is the i/p doc. with the value of the array field replaced by that element.

  → { $unwind: <field path> }

- **$addFields**: → adds new fields to documents. It o/ps docs that contain all existing fields from the i/p docs. & newly added fields.

  → { $addFields: { <newfield> : <expression>, ... } }

ex: What is avg. no. of tags per user?

```
[{ $addFields: {
     numOfTags: {
       $size: { $ifNull: ["$tags", []] }      ← keyword in DB is an array
     }
   }
},
{ $group: {
     _id: null,
     avgNumOfTags: { $avg: "$numOfTags" }
   }
}]
```

o/p:
```
{ _id: null, avgNumOfTags: 3.556 }
```

- **$push**: → appends a specified value to an array.
- **$project**: → passes along the docs. with the requested fields to the next stage of pipeline. The specified fields can be existing fields from i/p documents or newly computed fields.

ex: How many users have a ph. no. starting with '+1 (940)'?

```
[{
  $match: {
    "company.phone": /^\+1 \(940\)/
  }},
  { $count: 'userPhNo.' }
]
```

regex exp. → find from GPT

- **$all**: → This operator selects the documents where the value of a field is an array that contains all the specified elements.
  → { <field> : { $all : [<value1>, <value2>, ...] } }

- **$lookup**: → Performs a left outer join of the stream of messages from your $source to an Atlas collection

ex: Find users who have both 'enim' & 'id' as their tags?

```
[{
  $match: {
    tags: {
      $all: ["enim", "id"]
    }
  }
}]
```

ex: How many users have 'ad' as the 2nd tag in their list of tags?

```
[{
  $match: {
    "tags.1": "ad"
  }
},
{ $count: 'secondTagAd' }]
```

→ in 'tags' → 1st index

op: secondTagAd : 12

ex: Categorize users by their favourite fruit?

```
[{
  $group: {
    _id: "$favoriteFruit",
    users: { $push: "$name" }
  }
}]
```

→ Key in DB

op: _id: "apple"
users: Array (338)

ex: What are the names & age of users who are inactive & have 'velit' as a tag?

```
[{
  $match: {
    isActive: false,
    tags: "velit"
  }
},
{
  $project: {
    name: 1,
    age: 1
  }
}]
```

'1' = show

op: name :
age :
_id :

Or:

```
[{$lookup:{
    from: "authors",
    localField: "author_id",
    foreignField: "_id",
    as: "author_details"
  }
},
{
  $addFields:{
    author_details:{
      $arrayElemAt: ["$author_details", 0]
    }
  }
}]
```

→ not mandatory to add, it just provides a better view.