Cheatsheets / **Learn MongoDB**

Explore MongoDB

MongoDB Atlas Offerings

Atlas is a cloud-based database service offered by MongoDB. Atlas hosts a suite of tools that aid developers in setting up, deploying, and managing databases. In addition to these capabilities, Atlas offers additional integrated features that allow developers to search for, visualize, and analyze their data.

MongoDB Atlas Clusters

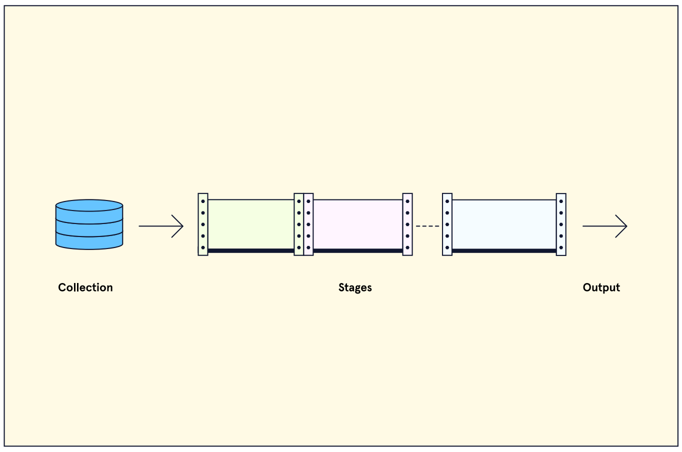
MongoDB Atlas houses data in units of storage called clusters. Depending on their Atlas plan, developers may use a shared cluster, where resources for that cluster, such as hardware and network bandwidth, are shared amongst multiple users. Alternatively, they may have their own dedicated cluster where its resources are allocated solely to them.

Atlas via the MongoDB Shell

In addition to the website interface offered by MongoDB Atlas, we can also connect to our Atlas cluster and interact with our databases locally via the MongoDB Shell. The MongoDB shell allows us to perform database operations using our computer's terminal or command line.

Aggregation Pipeline Stages

A stage in a MongoDB aggregation pipeline performs a specific operation on the data in the form of filtering or modifying the data before passing the result to either another stage or returning the result if there are no more stages. The above visual shows data flow from a collection to the final output via stages.



The `$match` Stage

The `$match` stage in a MongoDB pipeline filters input documents to pass only the documents that match the specified condition(s) to the next pipeline stage. This stage is similar to using the `find()` method because a query argument needs to be used to filter documents based on specific criteria.
The following example uses the `$match` stage to filter documents from a `movies` collection based on having a `rating` field with a value of `"R"`:

```
db.movies.aggregate([
  {
    $match: {rating: "R"}
  }
])
```

```
{ $match: { <query> } }
```

codecademy

The `$addFields` Stage

The $addFields stage in a MongoDB pipeline adds a new field to records. In addition to a new field, this stage also uses an aggregation expression which performs some type of logic such as arithmetic or comparisons.
Check out the example below where the $addFields stage is used as the third stage to create a new field called highest_score .

```
db.students.aggregate([
  {
    // First stage
    $match: {grade_level: 6, a
  },
  {
    // Second Stage
    $sort: { first_name: 1}
  },
  {
    // Third Stage
    $addFields:  {
      highest_score: { $max: '
    }
  }
])
```

```
{ $addFields: { <newField>:
<expression>, …}}
```

codecademy

The `$sort` Stage

The $sort stage in a MongoDB pipeline sorts results in either ascending order, specified with 1 , or descending order, specified with -1 . Check the example below where the $sort stage is used to take the output from the first stage (ie. $match) in order to sort the results in ascending order based on the first_name field.

```
db.students.aggregate([
  {
    // First stage
    $match: {grade_level: 6, a
  },
  {
    // Second Stage
    $sort: { first_name: 1}
  }
])
```

```
{ $sort: { <query> } }
```

The `.aggregate()` Method

Aggregation in MongoDB can be accomplished using an aggregation pipeline via the `.aggregate()` method. The first argument is an array containing the pipeline stages that will be used.
Check out the following example of the `.aggregate()` method used on a `students` collection with the pipeline stages `$match` and `$sort` :

```
db.students.aggregate([
  {
    // First stage
    $match: {grade_level: 6}
  },
  {
    // Second Stage
    $sort: { first_name: 1}
  }
])
```

Aggregation Pipeline Stages

A MongoDB aggregation pipeline can be built using stages such as `$match` or `$sort` . Other common stages include:

- `$group`
- `$addFields`
- `$out`
- `$count`

And many more!

codecademy

Aggregation Expressions

MongoDB aggregation pipeline stages can utilize different types of expressions such as field paths, literals, system variables, expression objects, and expression operators. Expressions can be nested.

Aggregation Pipelines

Aggregation pipelines allow for data to filter incrementally through the use of stages, where each stage filters or modifies the data in a specific way and then passes that data to the next stage or returns the data if there are no more stages.

Referencing Fields Inside Aggregation Expressions

Aggregation expressions use field paths to access fields in the input documents. To specify a field path, prefix the field name or the dotted field name (if the field is in the embedded document) with a dollar sign `$`.
In the below example, the field path allows us to access the `test_scores` field from the document to use with the `$max` expression operator:

```
db.students.aggregate([
  {
    // First stage
    $match: {grade_level: 6}
  },
  {
    // Second Stage
    $sort: { first_name: 1}
  },
  {
    // Third Stage
    $addFields:  {
        highest_score: { $max
  },
])
```

The `$out` Stage

The `$out` stage can output the final result of a MongoDB aggregation pipeline to a new database and or a new collection. When used, it must be the final stage in the pipeline.
The following example uses a aggregation pipeline with the `$out` stage as the final stage:

```
db.students.aggregate([
  {
    // First stage
    $match: {grade_level: 6}
  },
  {
    // Second Stage
    $sort: { first_name: 1}
  },
  {
     // Third Stage
     $addFields:  {
       highest_score: { $max:
     }
  },
  {
    // Fourth Stage
    $out : "candidates"
  }
])
```

codecademy

Aggregation Use Cases

Aggregation is useful when tasks cannot be accomplished with common CRUD methods easily or when performing complex analytics on datasets, such as: grouping values from multiple documents, computations on data, and analyzing data changes over time.

Print Share