MongoDB : Regular Expressions

Pattern Matching

mongoDB®

MongoDB Regular Expression

- Regular expressions are used for pattern matching, for findings strings within documents.

- The regex operator in MongoDB is used to search for specific strings in the collection. An example is :

```
db.trainee.find({first_name:{$regex:"Ja" }})
```

Case Insensitive Regular Expression

- We can also provide additional options by using the $options keyword

- Option ‘i’ is used to specify case insensitivity. An example is :

```
db.trainee.find({first_name:{$regex:"Ja",
$options:"i" }})
```

Regular Expression : Exact Pattern

- To do an exact pattern matching, we will use the ^ and $ character.

- The ^ character is added in the beginning of the string and $ at the end of the string.

```
db.trainee.find({first_name:{$regex: "^James$" }})
```

Pattern match without Regular Expression

- The “//” options can be used to specify your search criteria within these delimiters.

```
db.trainee.find({first_name:/Ja/})
```

MongoDB : Cursors

The pointer to find method output

mongoDB®

What are Cursors in MongoDB?

- The find() method used to find the documents in the given collection, returns a pointer which will points to the documents of the collection
- This pointer can be called as a ‘Cursor’
- Using this pointer we can access the document data.
- By default, mongo shell automatically iterates the cursor up to 20 documents and will display the result.
- We can override this and iterate the cursor manually.

Saving the Cursor

- To iterate a cursor manually, assign the cursor return by the find() method to a JavaScript variable using the var keyword Or.

```
var trainees = db.trainee.find()
trainees
```

- If a cursor is not used for 10 min then MongoDB server will automatically close/flush that cursor.

Manually Iterating the Cursor

- Using next() method:

```
var trainees = db.trainee.find({fee:{$gt:2000}})

while(trainees.hasNext()){
    print(trainees.next());
}
```

Manually Iterating the Cursor

- Using forEach() method:

```
var trainees = db.trainee.find({fee:{$gt:2000}})

trainees.forEach(printjson);

The printjson() helper method will print the
document in json format
```

Manually Iterating the Cursor

- Using Iterator Index from Array

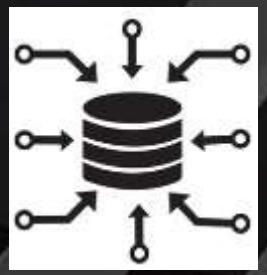
```
var trainees = db.trainee.find({fee:{$gt:2000}})
var traineesArray = trainees.toArray();

var firstTrainee = traineesArray[0];
firstTrainee
```

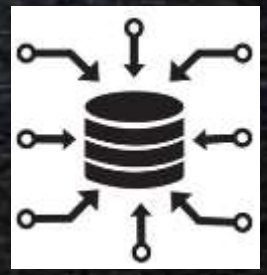
MongoDB : Aggregation

Process Documents and Output Computed Results

mongoDB®

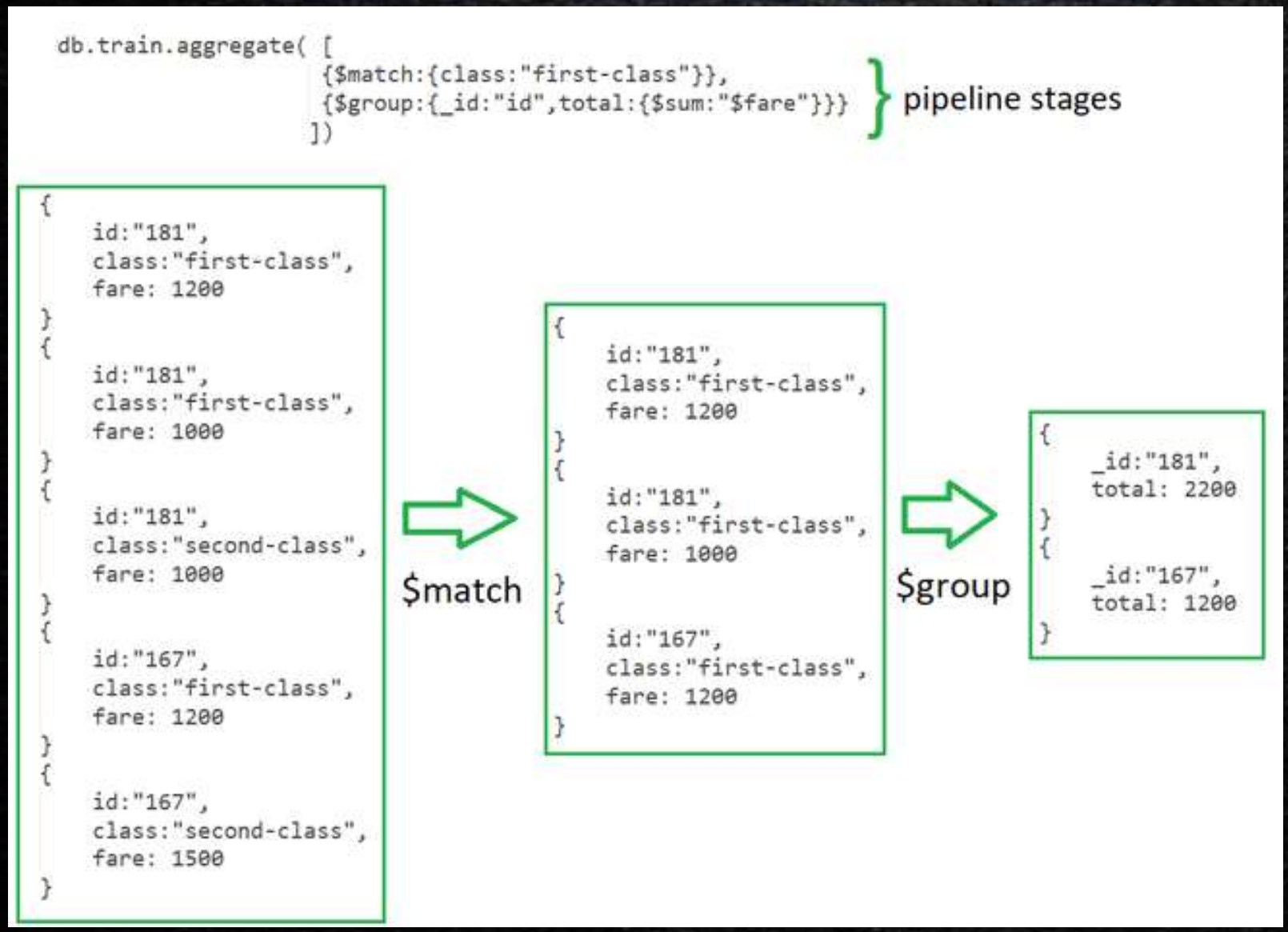


What is Aggregation



- Aggregation operations process the data records/documents and return computed results.
- It collects values from various documents and groups them together
- Different types of operations are done on that grouped data like sum, average, min-max, etc to return a computed result.
- It is similar to the aggregate functions of SQL like the sum(), min(), max(), avg() etc.

A Sample Aggregation



Source: https://www.geeksforgeeks.org/

Types of MongoDB Aggregation

- In MongoDB there are three ways to perform aggregation. Using :

- Single-purpose aggregation
  - With simple aggregation methods, count(), distinct(), and estimatedDocumentCount()

- Aggregation pipeline
  - Consist of different stages in which the documents taken as input and produce the output to the next stage, this process is going on till the last stage.

- Map-reduce function (Deprecated As of MongoDB 5.0)
  - **map** function for grouping and **reduce** to performs operation on the grouped data.

Type 1 : Single Purpose Aggregation

Using distinct():

```
db.trainee.distinct("first_name")
```

```
training> db.trainee.distinct("first_name")
[
  'Captain', 'Harry',
  'Jack',    'James',
  'John',    'Optimus',
  'Rose',    'Spider'
]
```

Type 1 : Single Purpose Aggregation

Using count():

```
db.trainee.count()
```

```
training> db.trainee.count()
8
```

Type 1 : Single Purpose Aggregation

Using estimatedDocumentCount(<options>):

- The optional parameter is maxTimeMS, which is an integer
- It is the maximum amount of time to allow the count to run.

```
db.trainee.estimatedDocumentCount(100)
```

```
training> db.trainee.estimatedDocumentCount(100)
8
```

Why Aggregation Pipelines are needed ?

- The find() command for querying data and it will probably be sufficient for normal data selection and retrieval
- But as soon as we start doing process more advanced than data retrieval, the MongoDB aggregation pipeline will be very helpful.
- MongoDB’s aggregation pipeline is a framework to perform a series of data transformations on a dataset.
- The first stage takes the entire collection of documents as input, and from then on each subsequent stage takes the previous transformation’s result set as input and produces some transformed output.

The aggregation function syntax

- The aggregate() function is used to perform aggregation
-
- It can have three operators stages, expression and accumulator.

```
db.train.aggregate([{$group : { _id :"$id", total : { $sum : "$fare" }}}])
```

Stage Expression Accumulator

MongoDB aggregation pipeline Stages

- we can break down a complex query into easier stages, in each of which we complete a different operation on the data.
- The most important stages of the aggregation pipeline are :

- The input data into the pipeline can be from one or more collections.
- $match() stage – filters those documents we need
- $group() stage – does the aggregation task
- $sort() stage – sorts the resulting documents
- These successive transformations on the data gives us the output

Popular Stage Operators

$match: It is used for filtering the documents can reduce the amount of documents that are given as input to the next stage.

$project: It is used to select some specific fields from a collection.

$group: It is used to group documents based on some value.

$sort: It is used to sort the document that is rearranging them

$skip: It is used to skip n number of documents and passes the remaining documents

Popular Stage Operators

$limit: It is used to pass first n number of documents thus limiting them.

$unwind: It is used to unwind documents that are using arrays i.e. it deconstructs an array field in the documents to return documents for each element.

$out: It is used to write resulting documents to a new collection

Expressions

It refers to the name of the field in input documents

for e.g. { $group : { _id : “**$id**“, total:{$sum:”**$fare**“}}}

Here **$id** and **$fare** are expressions.

Popular Accumulators

- **Accumulators:** These are basically used in the group stage

- **sum:** It sums numeric values for the documents in each group
- **count:** It counts total numbers of documents
- **avg:** It calculates the average of all given values from all documents

- **min:** It gets the minimum value from all the documents
- **max:** It gets the maximum value from all the documents

- **first:** It gets the first document from the grouping
- **last:** It gets the last document from the grouping

The aggregation function syntax

```
pipeline = [
   { $match : { … },
   { $group : { … },
   { $sort : { … },
   ...
]
```

```
db.collectionName.aggregate(pipeline, options)
```

- 100 MB of RAM is the max limit for each stage
- To overcome this, we can use disk page file option

- `db.collectionName.aggregate(pipeline, { allowDiskUse : true })`

Example:

```
db.trainee.aggregate([{$match:{city:"New York"}},
{$count:"city:New York"}])
```

For taking a count of the number of trainees in city new york we first filter the documents using the **$match operator,**

And then we use the **$count** accumulator to count the total number of documents that are passed after filtering from the $match.

Example:

```
db.trainee.aggregate([{$group: { _id:"$city",
total_strength: {$sum:1}, max_age:{$max:"$age"} }
}])
```

- **Displaying the total number of trainees in both the cities and maximum age from both cities**

we use **$group** to group, so that we can count for every other city in the documents, here **$sum** sums up the document in each group and **$max** accumulator is applied on age expression which will find the maximum age in each document.

Recommended Reference:

https://www.practical-mongodb-aggregations.com/