# The Magical Marvels of MongoDB

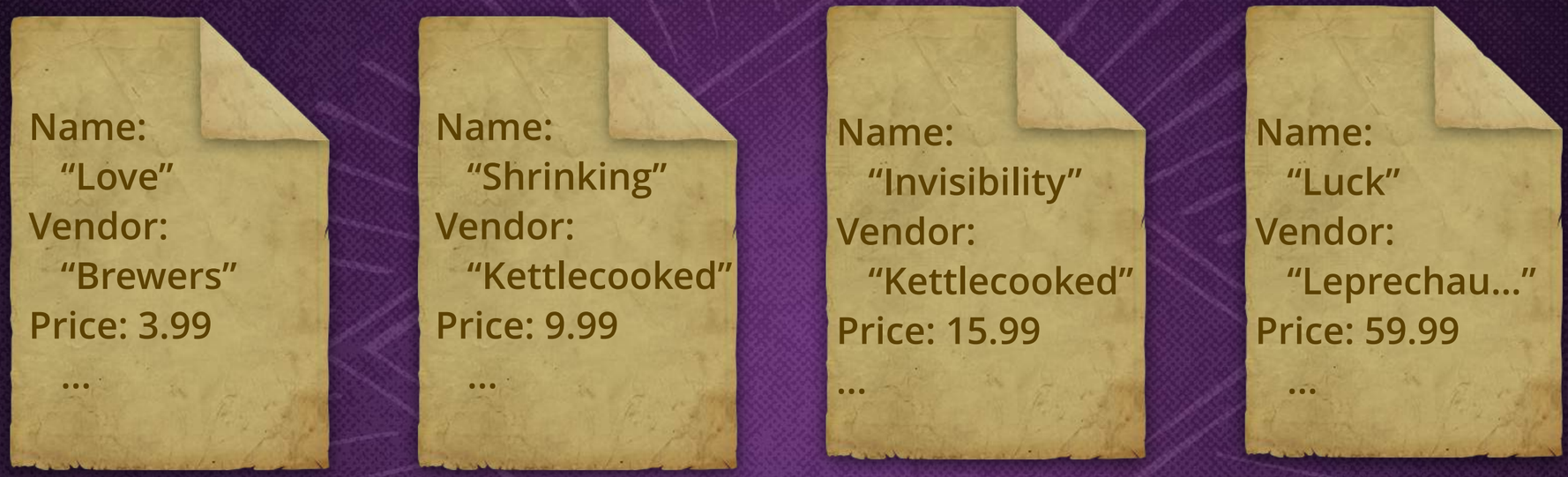
# Aggregation Apparitions

Level 5 – Section 1

Common Aggregations

# Finding the Number of Potions Per Vendor

Time for an audit! We need to know how many potions we have per vendor.



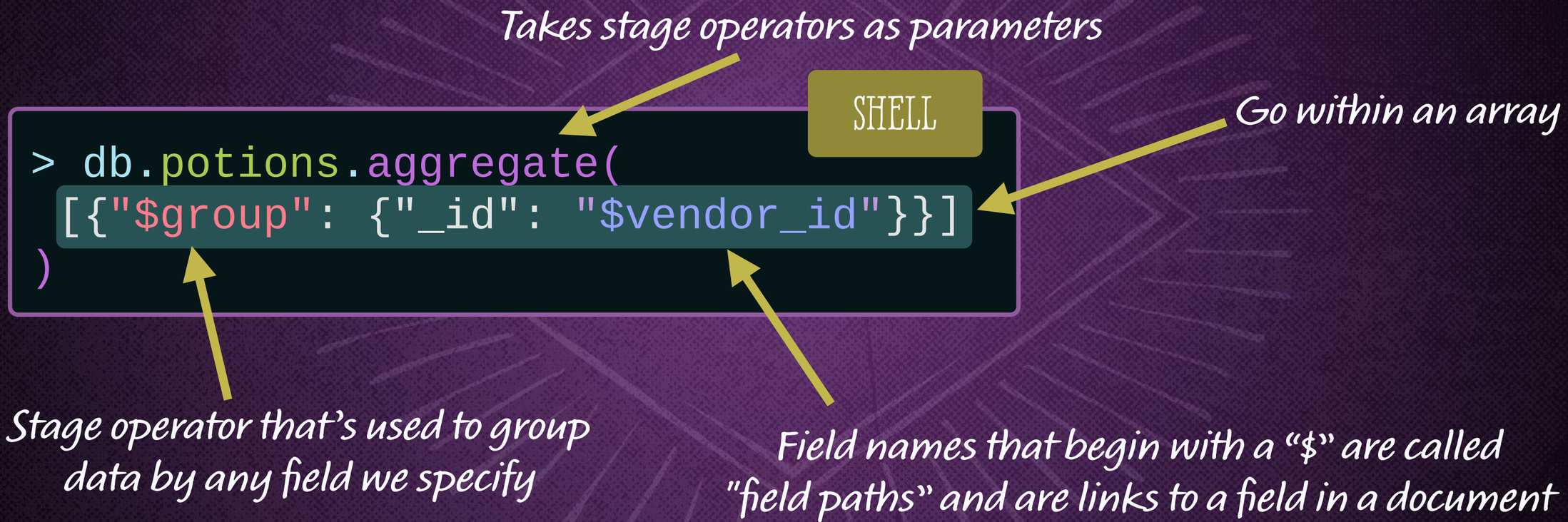

```
> db.potions.find({}, {"name": true, "vendor": true})
```

*We could manually pull all the data and count everything, but it's better to have MongoDB handle that for us!*

THE MAGICAL MARVELS OF MONGODB

# Introducing the Aggregation Framework

The aggregation framework allows for advanced computations.




"Aggregate" is a fancy word for combining data.

# Using the Aggregation Framework to Group Data

SHELL

```
> db.potions.aggregate(
  [{"$group": {"_id": "$vendor_id"}}]
)
```

*This is known as the "group key" and is required*

*Returns result object containing the unique vendors in the inventory collection*

*Potions Collection*

*$group*

*Results*

```
{"_id": "Kettlecooked"},
{"_id": "Brewers"},
{"_id": "Leprechaun Inc"}
```

# Using Accumulators

Anything specified after the group key is considered an accumulator. Accumulators take a single expression and compute the expression for grouped documents.

SHELL

```
> db.potions.aggregate([
 {"$group": {"_id": "$vendor_id", "total": {"$sum": 1}}}
])
```

*Group key*

*Will add 1 for each matching document*

*Accumulator*

*Inventory*

*$group*

*Results*

*Total number of documents per vendor!*

```
{"_id": "Kettlecooked", "total": 2},
{"_id": "Brewers", "total": 1,},
{"_id": "Leprechaun Inc”, "total": 1}
```

# Field Paths Vs. Operators

*When values begin with a "$", they represent field paths that point to the value*

SHELL

```
> db.potions.aggregate([
 {"$group": {"_id": "$vendor_id", "total": {"$sum": 1}}}
])
```

*When fields begin with a "$", they are operators that perform a task*

# Summing the Grade Per Vendor

SHELL

```
> db.potions.aggregate([
   { "$group": {
                    "_id": "$vendor_id",
                    "total": {"$sum": 1},
                    "grade_total": {"$sum": "$grade"}
                }
   }
])
```

*Field path*

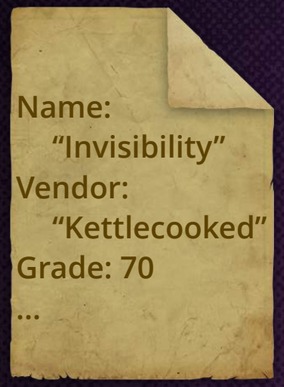
*Sums the grade values for potions in their group*

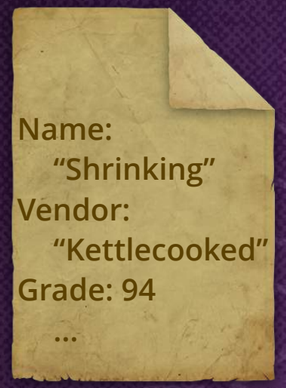
*Second accumulator*

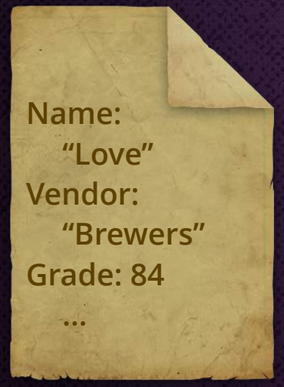
*Results*

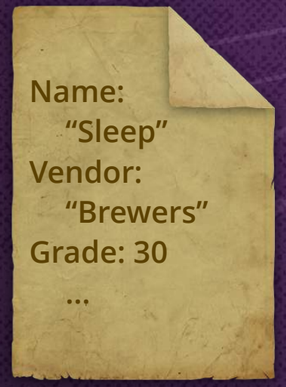
```
{"_id": "Kettlecooked", "total": 2, "grade_total": 400},
{"_id": "Brewers","total: 1, "grade_total": 340},
{"_id": "Leprechaun Inc”, "total": 1, "grade_total": 92}
```

# Averaging Potion Grade Per Vendor



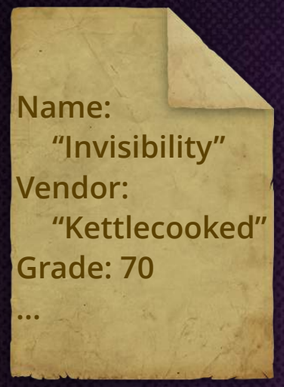

SHELL

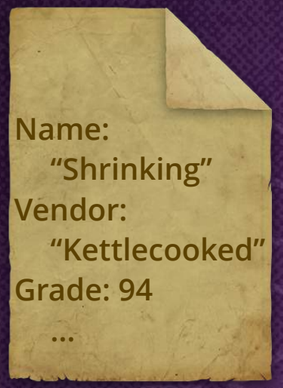
```
> db.potions.aggregate([
 { "$group": {
     "_id": "$vendor_id",
     "avg_grade": {"$avg": "$grade"}
   }
 }
])
```

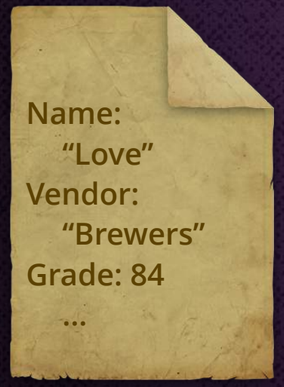
*Results*

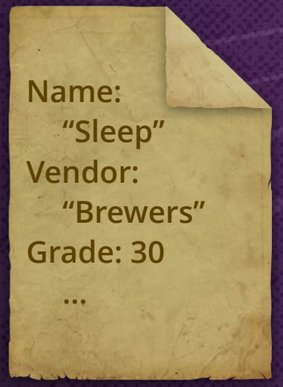
```
{"_id": "Kettlecooked", "avg_grade": 82 },
{"_id": "Brewers","avg_grade": 57 }
```

# Returning the Max Grade Per Vendor



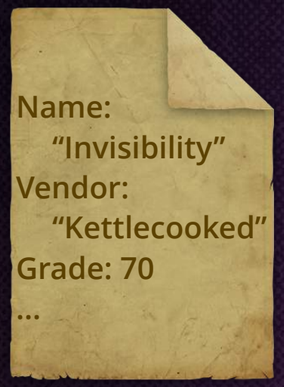

SHELL

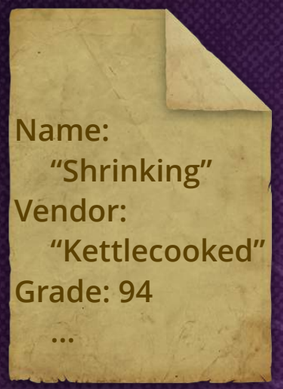
```
> db.potions.aggregate([
 { "$group": {
     "_id": "$vendor_id",
     "max_grade": {"$max": "$grade"}
   }
 }
])
```

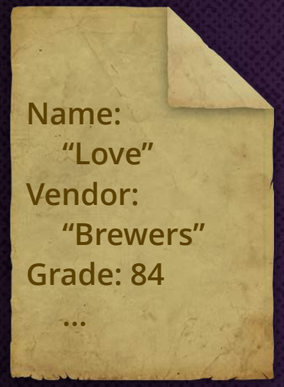
*Results*

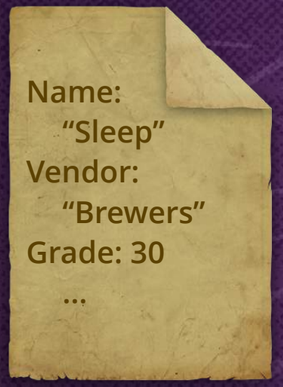
```
{"_id": "Kettlecooked", "max_grade": 94},
{"_id": "Brewers","max_grade": 84}
```

# Using $max and $min Together




SHELL

```
> db.potions.aggregate([
  { "$group": {
      "_id": "$vendor_id",
      "max_grade": {"$max": "$grade"},
      "min_grade": {"$min": "$grade"}
    }
  }
])
```

*We can use the same field in multiple accumulators*

*Results*

```
{"_id": "Kettlecooked", "max_grade": 94, "min_grade": 70 },
{"_id": "Brewers", "max_grade": 84, "min_grade": 30 }
```

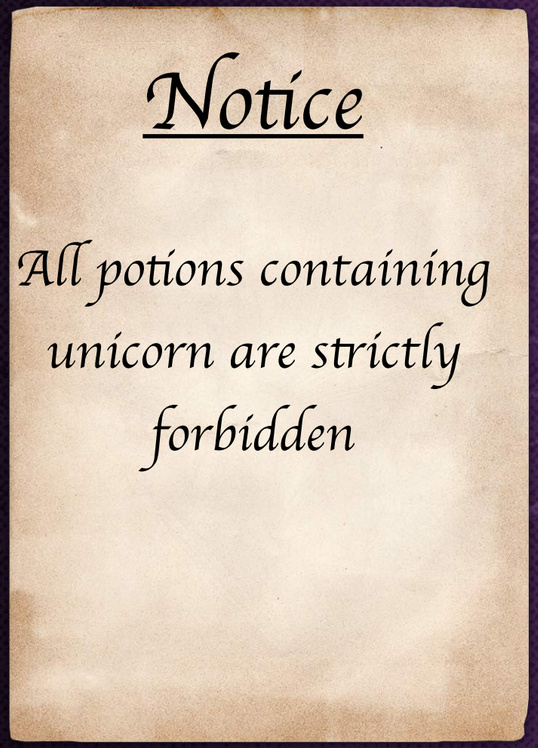
# Aggregation Apparitions

Level 5 – Section 2

The Aggregation Pipeline

# Pulling Conditional Vendor Information

It turns out that potions made with unicorn aren't permitted, so we need to count the number of potions per vendor that contain it.




**Steps to find potions with unicorns:**

1) Query potions

2) Group by vendor

3) Sum the number of potions per vendor

THE MAGICAL MARVELS OF MONGODB

# Introducing the Aggregation Pipeline

The aggregate method acts like a pipeline, where we can pass data through many stages in order to change it along the way.

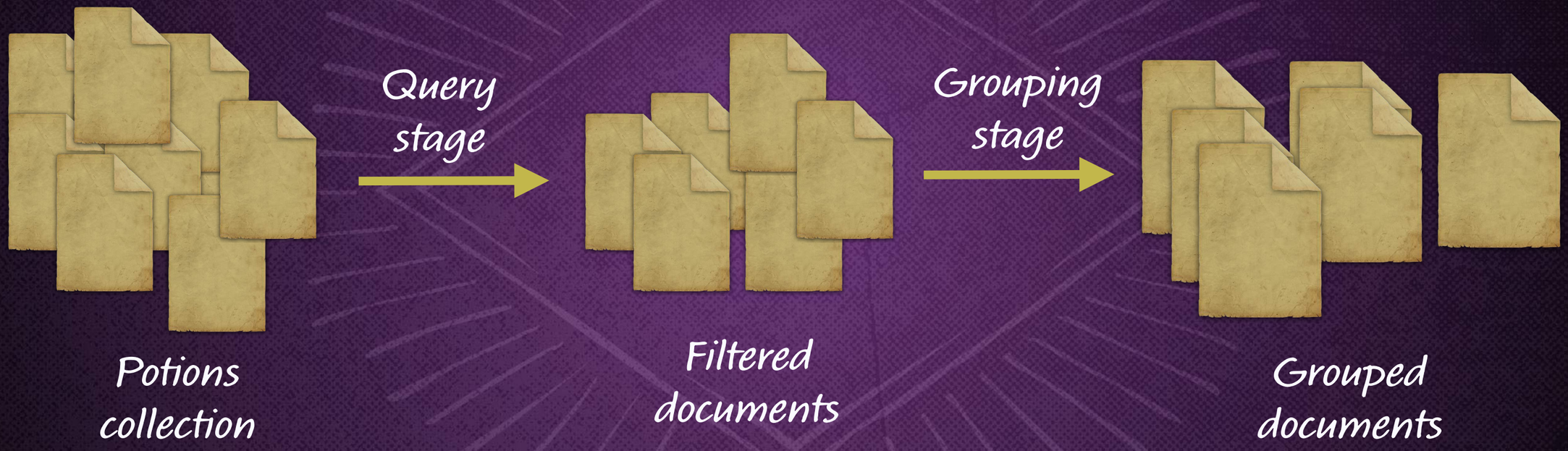
```
db.potions.aggregate([stage, stage, stage])
```

*We've already seen what a stage looks like in the last section*

```
db.potions.aggregate([
 {"$group": {"_id": "$vendor_id", "total": {"$sum": 1}}}
])
```

# How the Pipeline Works

Each stage modifies the working data set and then passes the altered documents to the next stage until we get our desired result.



# Using the $match Stage Operator

***$match*** is just like a normal query and will only pass documents to the next stage if they meet the specified condition(s).

SHELL

```
db.potions.aggregate([
 {"$match": {"ingredients": "unicorn"}}
])
```

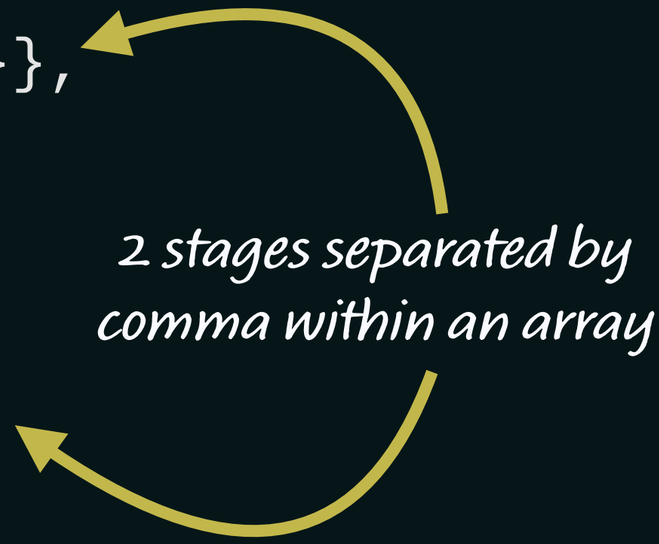
*We can use the same query we would use with find()*

*Use match early to reduce the number of documents for better performance.*

# Grouping Potions With Data

SHELL

```
db.potions.aggregate([
 {"$match": {"ingredients": "unicorn"}},
 {"$group":
   {
     "_id": "$vendor_id",
     "potion_count": {"$sum": 1}
   }
 }
])
```




Result

```
{"_id": "Poof", "potion_count": 20},
{"_id": "Kettlecooked","potion_count": 1}
```

# Top 3 Vendors With Potions Under $15


1) Query for potions with a price less than 15

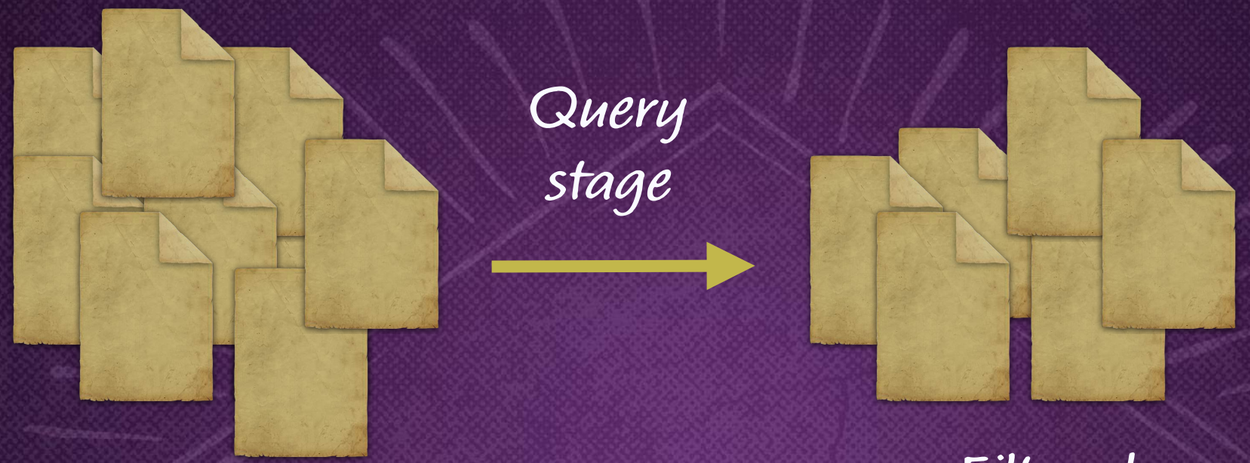
2) Group potions by vendor and average their grades

3) Sort the results by grade average

4) Limit results to only 3 vendors

THE MAGICAL MARVELS OF MONGODB

# Matching Potions Under $15




SHELL

```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}}
])
```

*It's good practice to limit the number of results early on*

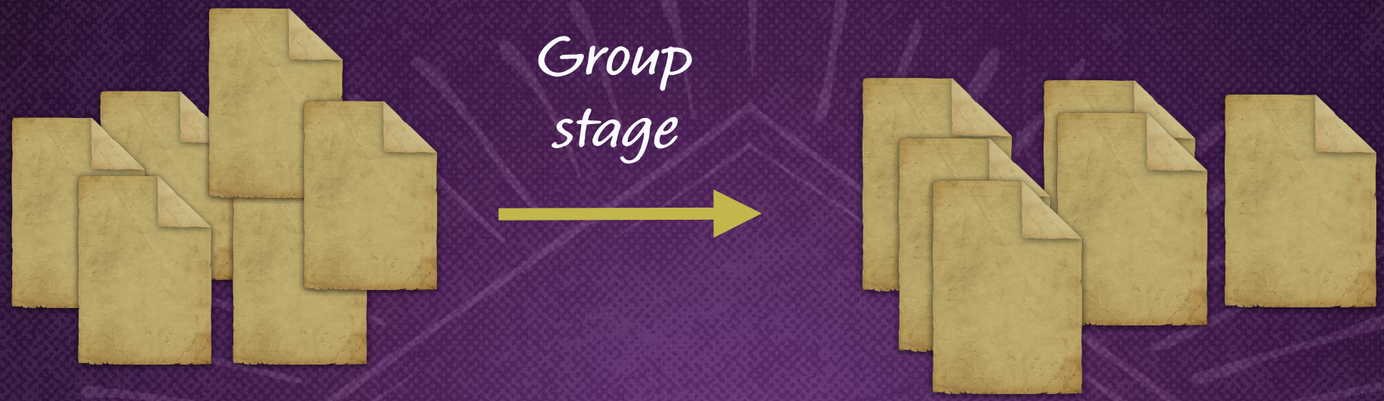
# Matching Potions Under $15

*Filtered documents*

SHELL

```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}}
])
```

# Grouping Potions by Vendor




SHELL

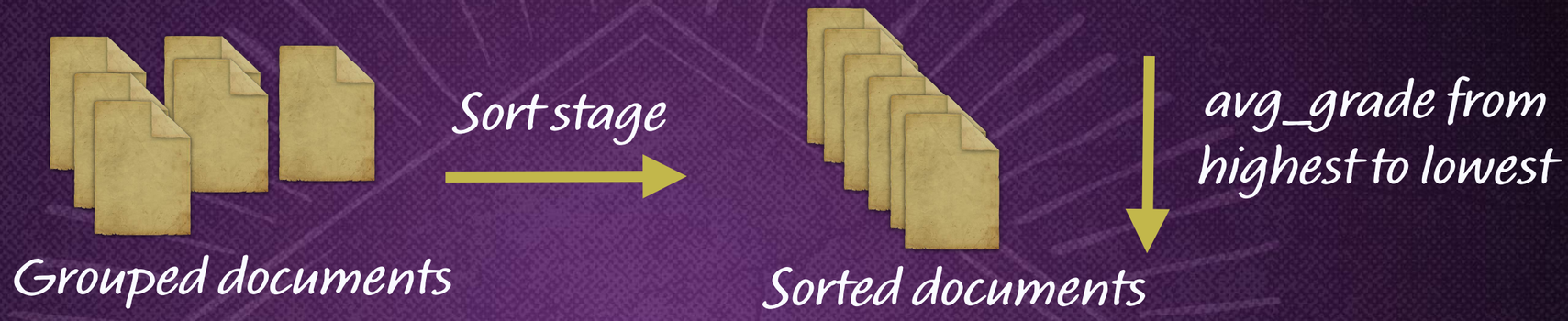
```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}},
 {"$group": {"_id": "$vendor_id","avg_grade": {"$avg": "$grade"}}}
])
```

Vendor name as group key

Average of potion grades

# Sorting Vendors by Average Grade

We can sort potions by using the ***$sort*** stage operator.




SHELL

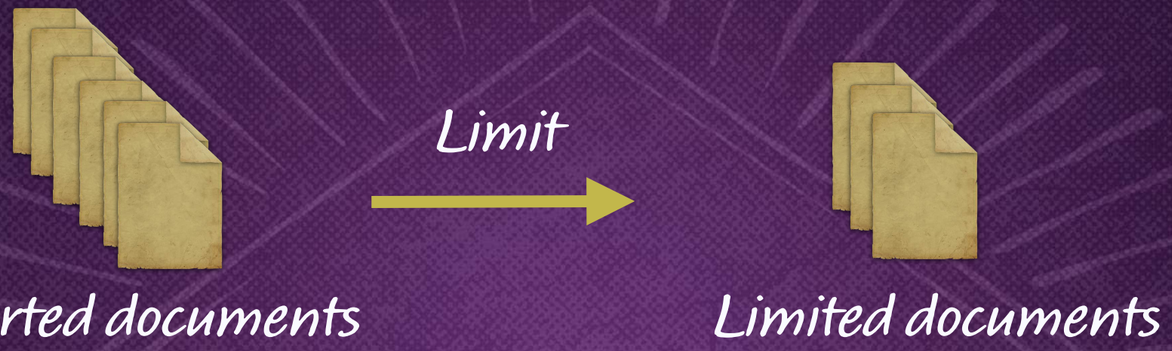
```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}},
 {"$group": {"_id": "$vendor_id","avg_grade": {"$avg": "$grade"}}},
 {"$sort": {"avg_grade": -1}}
])
```

We can sort the field we created during the group stage

Sort by the field in descending order

# Limiting the Number of Documents

We can use the ***$limit*** stage operator to limit the number of documents that get passed on.




SHELL

```
db.potions.aggregate([
  {"$match": {"price": {"$lt": 15}}},
  {"$group": {"_id": "$vendor_id","avg_grade": {"$avg": "$grade"}}},
  {"$sort": {"avg_grade": -1}},
  {"$limit": 3}
])
```

*Specify the number of documents to limit*

# Optimizing the Pipeline

It’s best practice to only send the needed data from stage to stage. Let’s see what data we really need.

SHELL

```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}},
 {"$group": {"_id": "$vendor_id","avg_grade": {"$avg": "$grade"}}},
 {"$sort": {"avg_grade": -1}},
 {"$limit": 3}
])
```

*Only need vendor and grade for each potion after the match stage*

# Projections While Aggregating

We can limit the fields we send over by using ***$project***, which functions the same way as projections when we're querying with ***find()***.

SHELL

```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}},
 {"$project":{"_id": false, "vendor_id": true, "grade": true}},
 {"$group": {"_id": "$vendor_id","avg_grade": {"$avg": "$grade"}}},
 {"$sort": {"avg_grade": -1}},
 {"$limit": 3}
])
```

*We want to use $project as soon as possible*

*Vendor and grade for each potion after the match stage*

*It's common to see $match and $project used together early on and throughout the pipeline.*

# Aggregation Results

SHELL

```
db.potions.aggregate([
 {"$match": {"price": {"$lt": 15}}},
 {"$project":{"_id": false, "vendor_id": true, "grade": true}},
 {"$group": {"_id": "$vendor_id","avg_grade": {"$avg": "$grade"}}},
 {"$sort": {"avg_grade": -1}},
 {"$limit": 3}
])
```

```
 {"_id": "Kettlecooked", "avg_grade": 99 },
 {"_id": "Leprechaun Inc", "avg_grade": 95 },
 {"_id": "Brewers", "avg_grade": 90 }
```

Group key

Average potion grade