The Magical Marvels of MongoDB

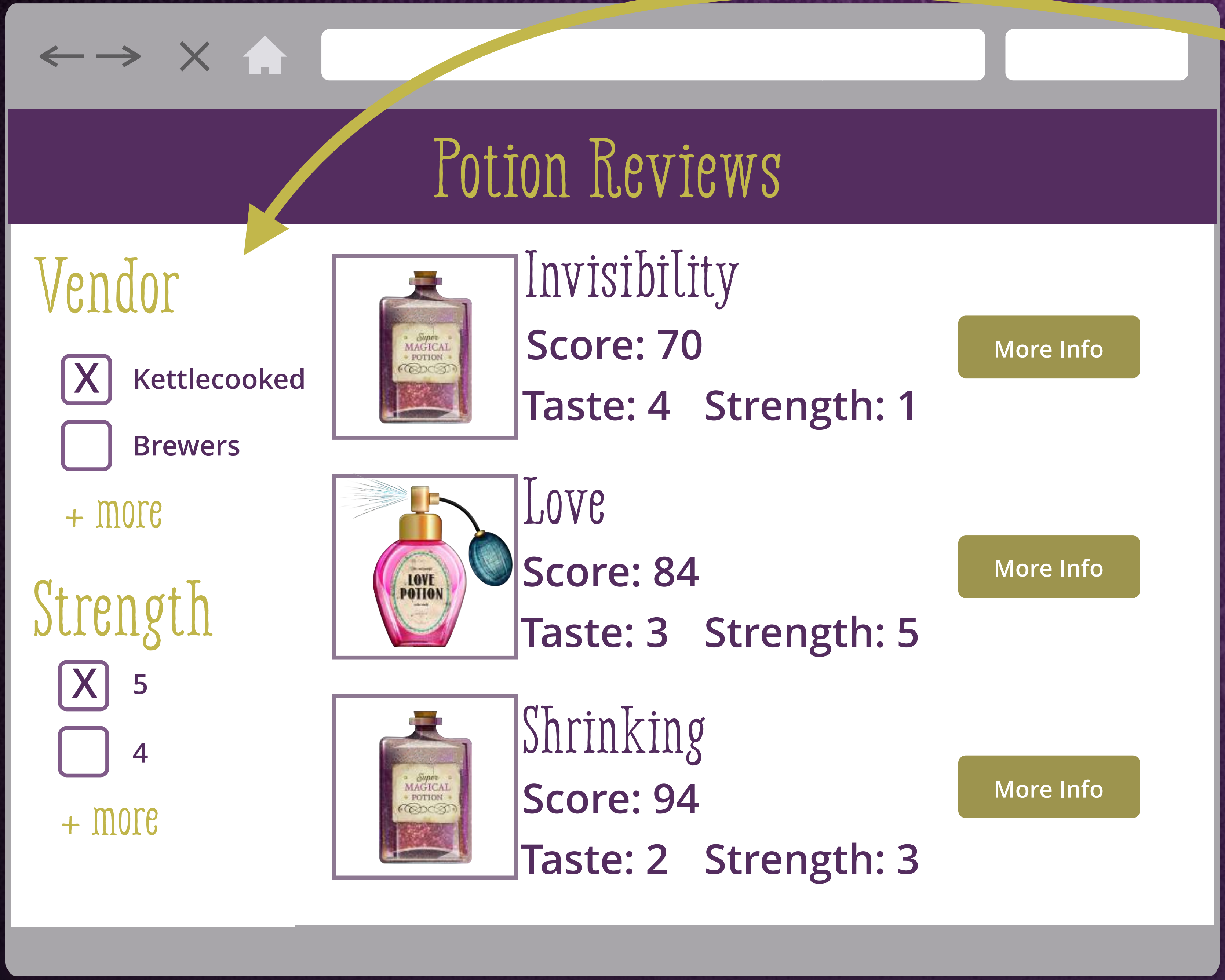
Materializing Potions

Level 3 – Section 1

Query Operators

Adding a Filter for Potions

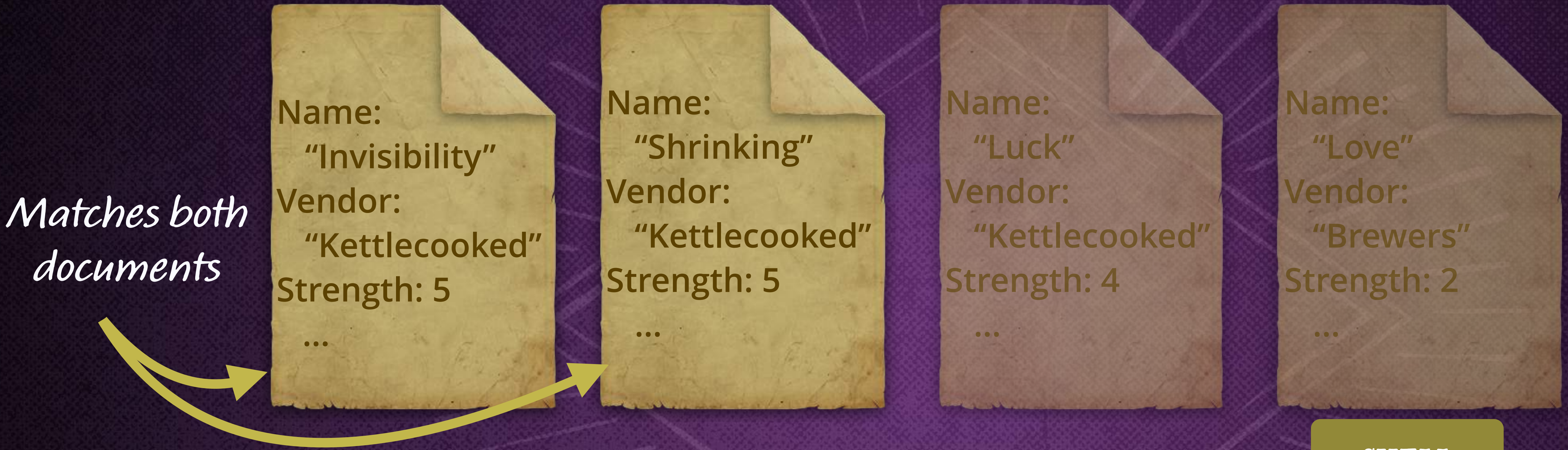
We've received a new feature request to allow users to filter potions based on multiple criteria.



Only show potions made by Kettlecooked that have a strength of 5

Querying With Multiple Criteria

We can query based on multiple criteria by passing in comma-separated queries.


Matches both documents
Name: "Invisibility" Vendor: "Kettlecooked" Strength: 5 ...
Name: "Shrinking" Vendor: "Kettlecooked" Strength: 5 ...
Name: "Luck" Vendor: "Kettlecooked" Strength: 4 ...
Name: "Love" Vendor: "Brewers" Strength: 2 ...

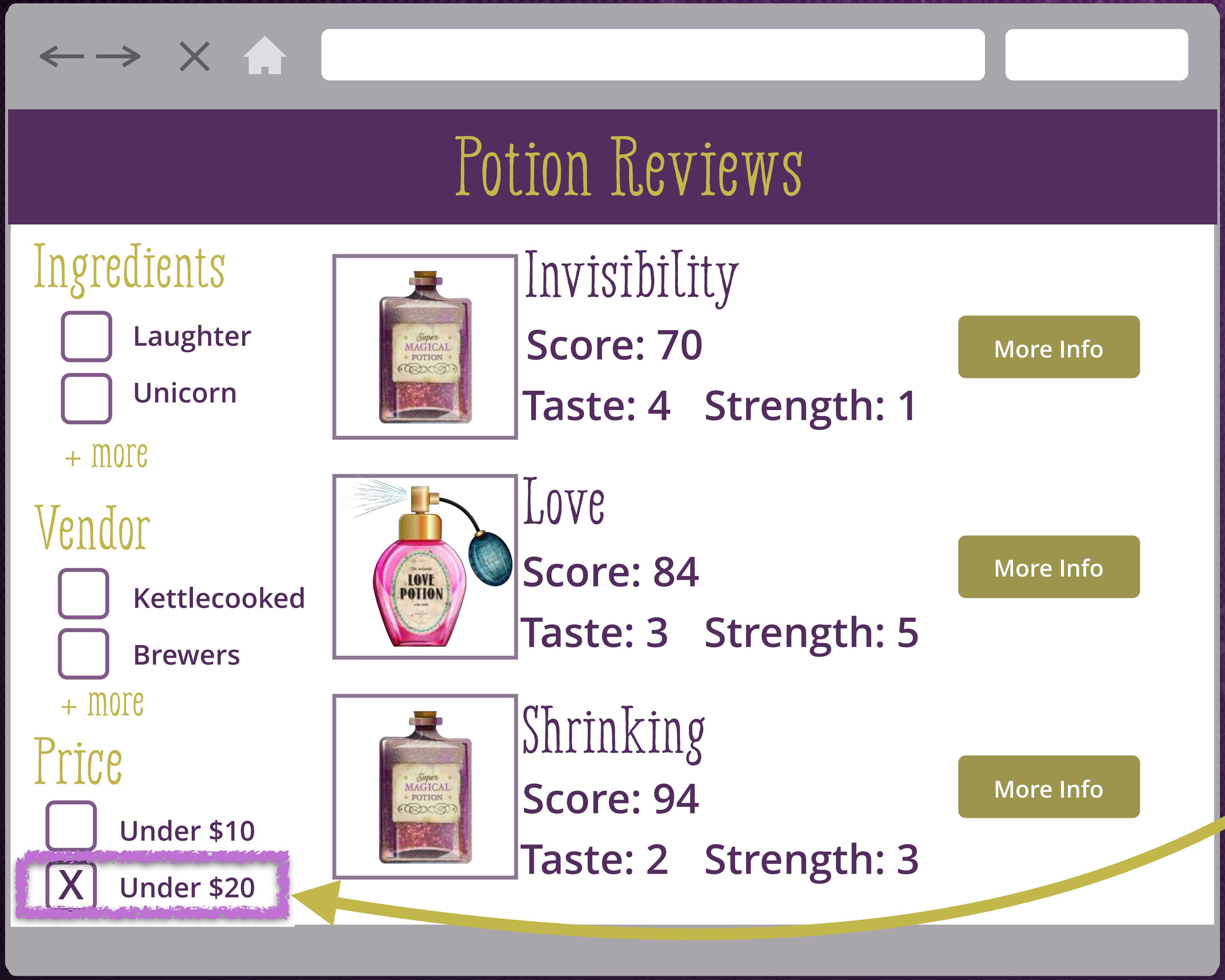
SHELL

```
> db.potions.find(
    {
     "vendor": "Kettlecooked",
     "ratings.strength": 5
    }
  )
```

We can pass in more than 1 query

Finding Potions Based on Conditions

Queries of equality are great, but sometimes we'll need to query based on conditions.



Potion Reviews

Ingredients

- [ ] Laughter
- [ ] Unicorn

+ more

Vendor

- [ ] Kettlecooked
- [ ] Brewers

+ more

Price

- [ ] Under $10
- [X] Under $20

Invisibility	Score: 70	Taste: 4 Strength: 1
Love	Score: 84	Taste: 3 Strength: 5
Shrinking	Score: 94	Taste: 2 Strength: 3

Search for potions with a price less than 20

Comparison Query Operators

We can use comparison query operators to match documents based on the comparison of a specified value.

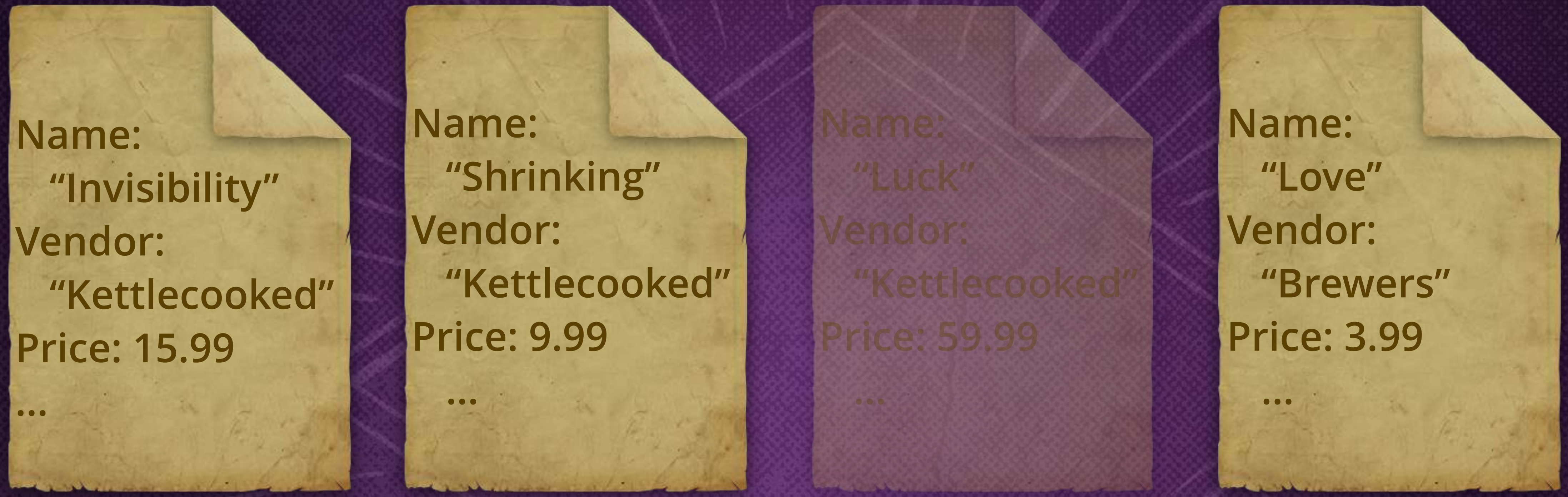
Common Comparisons

`$gt`	*greater than*
`$gte`	*greater than or equal to*
`$ne`	*not equal to*
`$lt`	*less than*
`$lte`	*less than or equal to*

The Magical Marvels of MongoDB

Finding Potions That Are Less Than $20

We can match the appropriate documents by using the ***$lt*** comparison operator.



Name:
  "Invisibility"
Vendor:
  "Kettlecooked"
Price: 15.99
…

Name:
  "Shrinking"
Vendor:
  "Kettlecooked"
Price: 9.99
…

Name:
  "Luck"
Vendor:
  "Kettlecooked"
Price: 59.99
…

Name:
  "Love"
Vendor:
  "Brewers"
Price: 3.99
…

SHELL

```
> db.potions.find({"price": {"$lt": 20}})
```

Price less than 20

THE MAGICAL MARVELS OF MONGODB

Finding Potions Between Prices

We can query with a range by combining comparison operators.

Name:
  "Invisibility"
Vendor:
  "Kettlecooked"
Price: 15.99
...

Name:
  "Shrinking"
Vendor:
  "Kettlecooked"
Price: 9.99
...

Name:
  "Luck"
Vendor:
  "Kettlecooked"
Price: 59.99
...

Name:
  "Love"
Vendor:
  "Brewers"
Price: 3.99
...

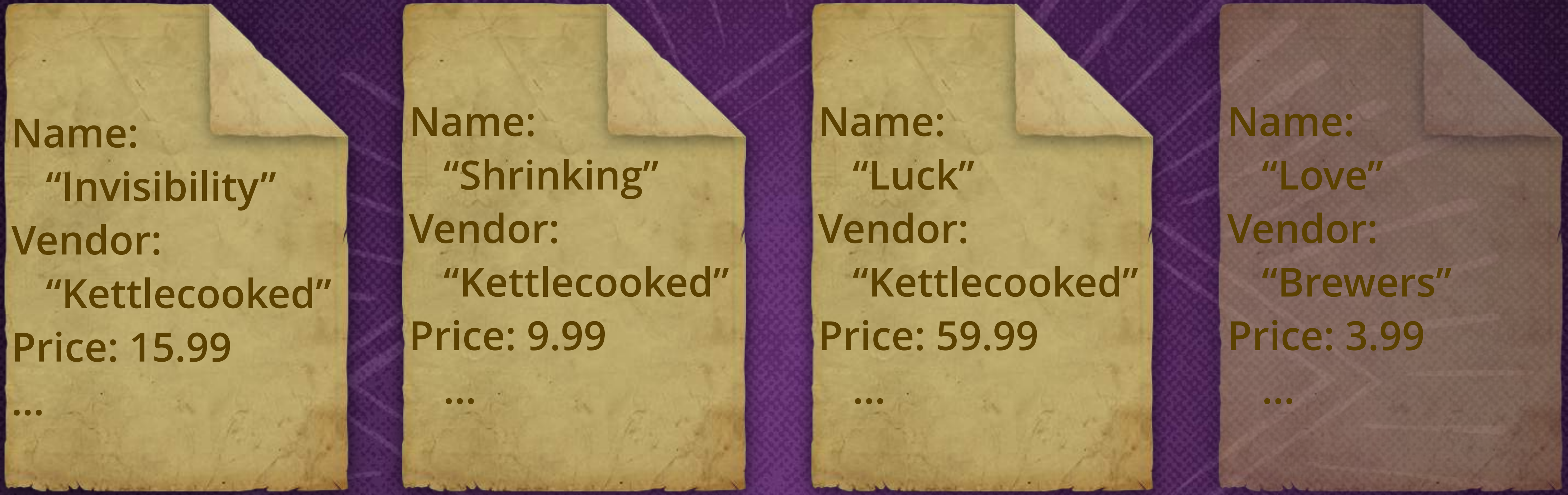
SHELL

```
> db.potions.find({"price": {"$gt":10, "$lt": 20}})
```

Price greater than 10 and less than 20

Queries of Non-equality

We can use the ***$ne*** operator to find potions with fields that don’t equal the specified value.



Name:
“Invisibility”
Vendor:
“Kettlecooked”
Price: 15.99
…

Name:
“Shrinking”
Vendor:
“Kettlecooked”
Price: 9.99
…

Name:
“Luck”
Vendor:
“Kettlecooked”
Price: 59.99
…

Name:
“Love”
Vendor:
“Brewers”
Price: 3.99
…

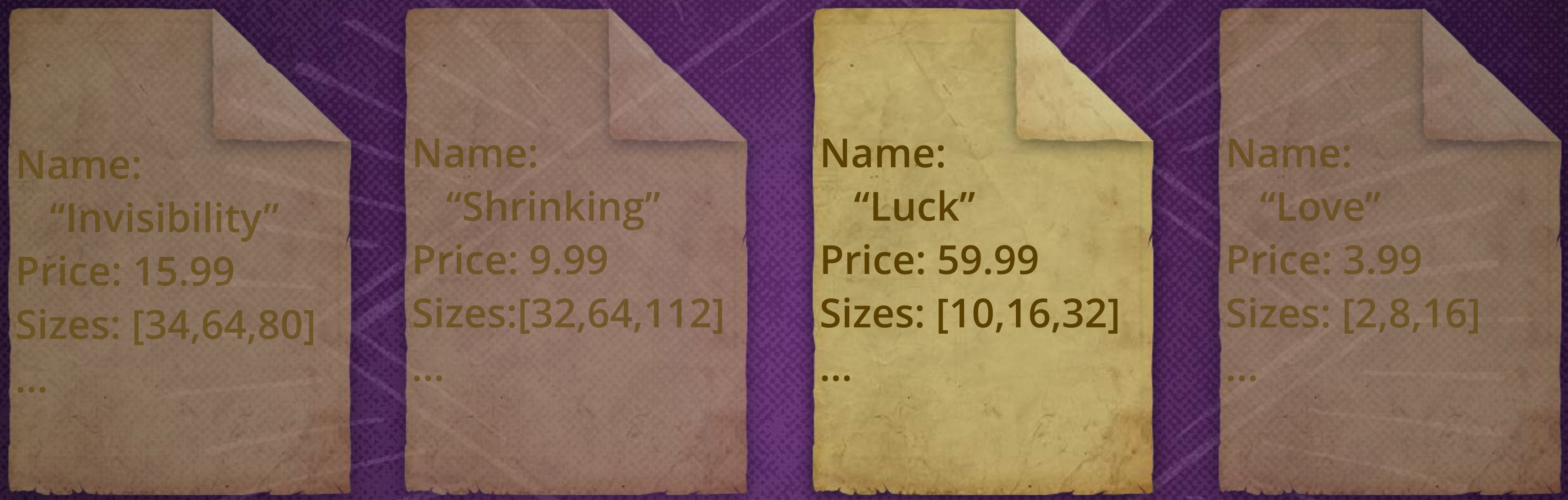
SHELL

```
> db.potions.find({"vendor": {"$ne": "Brewers"}})
```

Vendor not equal to “Brewers”

Range Queries on an Array

Each potion has a size field that contains an array of available sizes. We can use ***$elemMatch*** to make sure at least 1 element matches all criteria.



Name: "Invisibility"
Price: 15.99
Sizes: [34,64,80]
...

Name: "Shrinking"
Price: 9.99
Sizes:[32,64,112]
...

Name: "Luck"
Price: 59.99
Sizes: [10,16,32]
...

Name: "Love"
Price: 3.99
Sizes: [2,8,16]
...

Potion sizes

At least 1 value in an array MUST be greater than 8 and less than 16

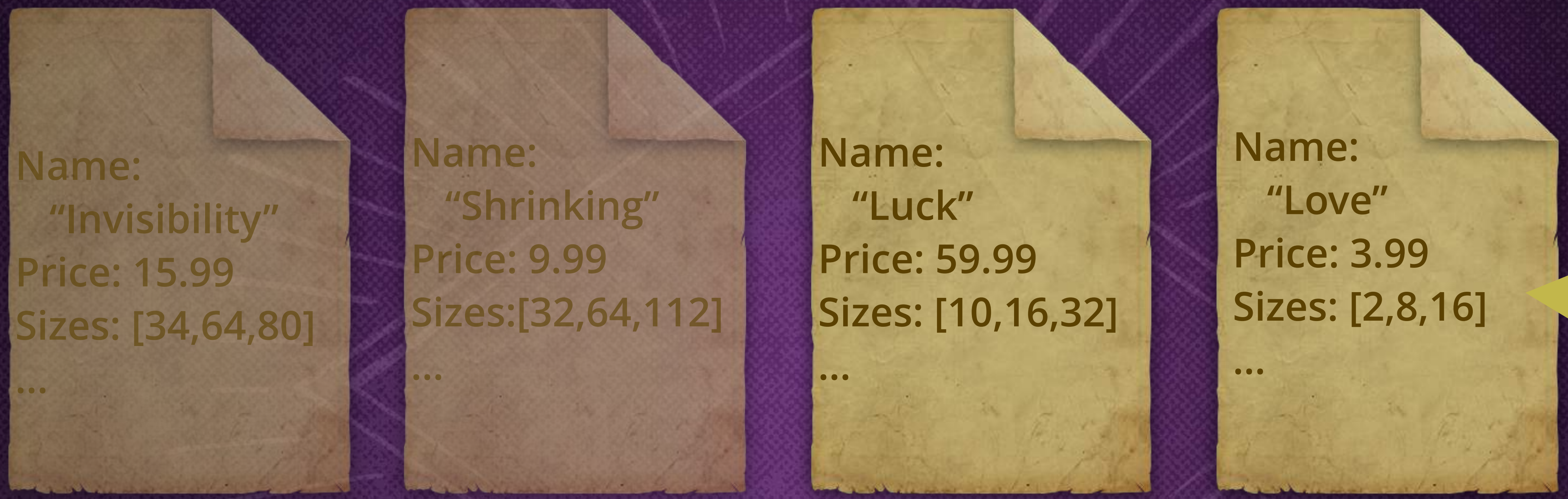
The value 10 matches!

SHELL

```
> db.potions.find(
  {"sizes" : {"$elemMatch": {"$gt": 8, "$lt": 16}}}
 )
```

Be Careful When Querying Arrays With Ranges

What happens when we try to perform a normal range query on an array?



Name: "Invisibility"
Price: 15.99
Sizes: [34,64,80]
…

Name: "Shrinking"
Price: 9.99
Sizes:[32,64,112]
…

Name: "Luck"
Price: 59.99
Sizes: [10,16,32]
…

Name: "Love"
Price: 3.99
Sizes: [2,8,16]
…

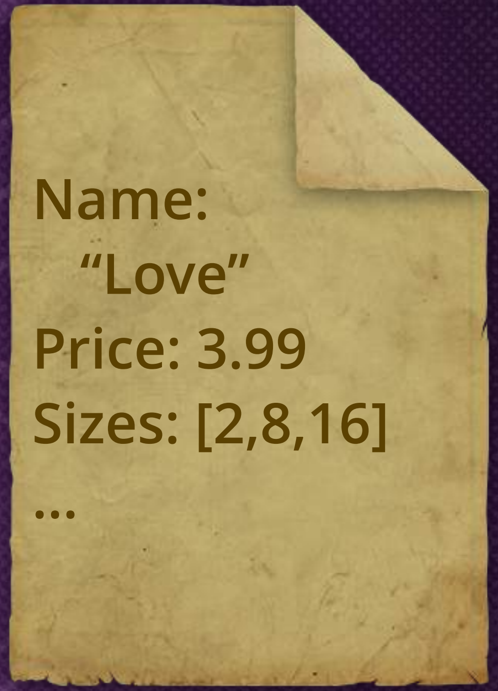
SHELL

```
> db.potions.find(
  {"sizes" : {"$gt": 8, "$lt": 16}}
 )
```

Doesn't contain any matching sizes, so why did it match?

Be Careful When Querying Arrays With Ranges

What happens when we try to perform a normal range query on an array?

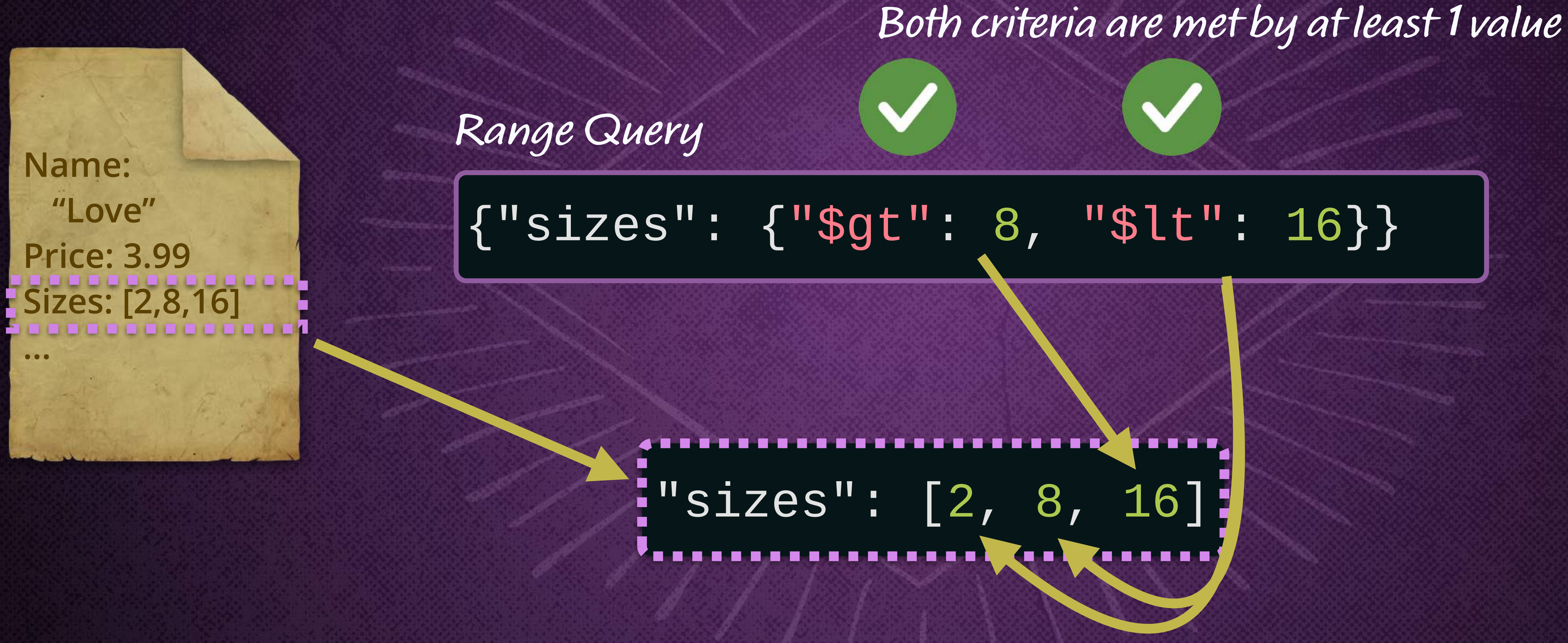

Name:
  “Love”
Price: 3.99
Sizes: [2,8,16]
...

SHELL

```
> db.potions.find(
  {"sizes" : {"$gt": 8, "$lt": 16}}
 )
```

Why Did the Document Match?

Each value in the array is checked individually. If at least 1 array value is true for each criteria, the entire document matches.



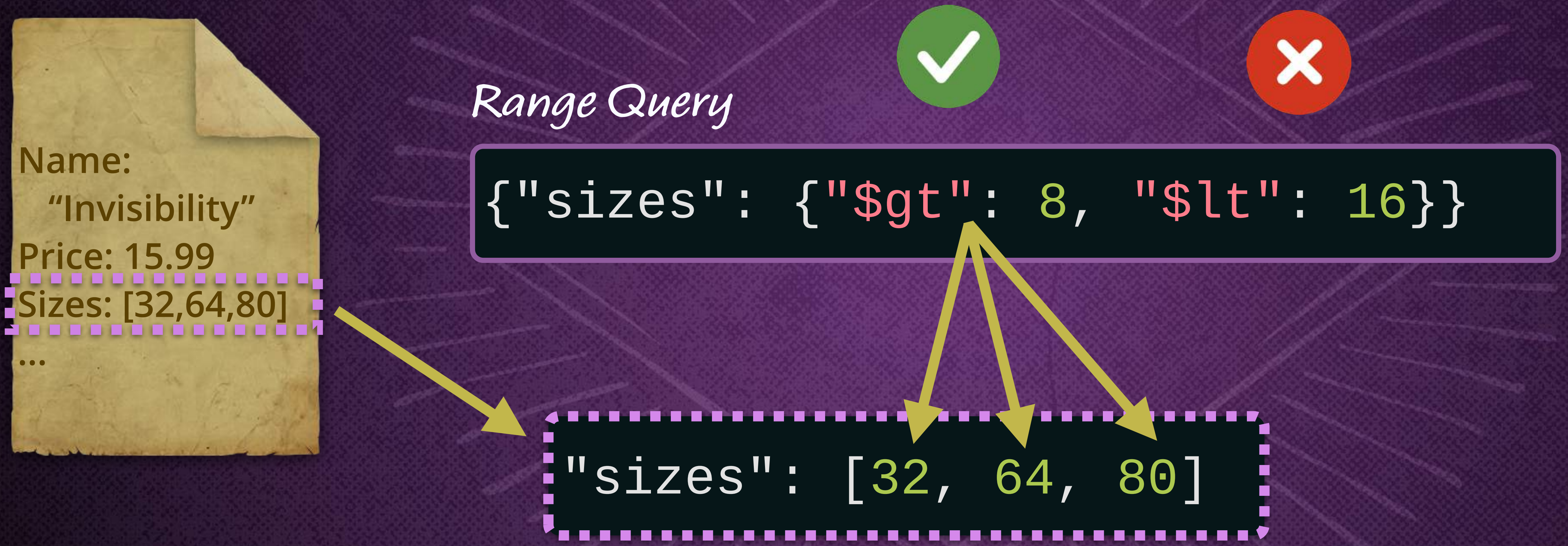
```
{"sizes": {"$gt": 8, "$lt": 16}}
```

```
"sizes": [2, 8, 16]
```

Not Matching a Document

Conversely, the document will not match if only 1 criteria is met.

Only 1 criteria is met, so the document doesn't match



```
{"sizes": {"$gt": 8, "$lt": 16}}
```

```
"sizes": [32, 64, 80]
```

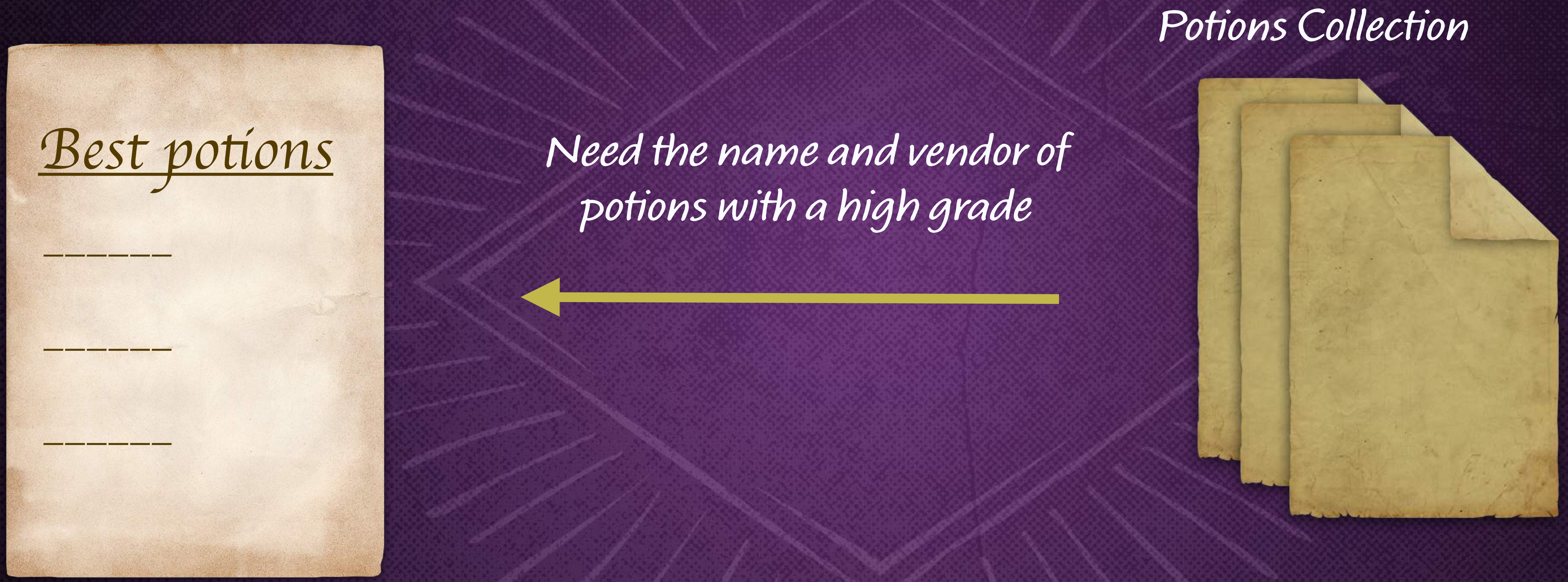
Materializing Potions

Level 3 – Section 2

Customizing Queries

Listing Our Best Potions

We're putting together a list of the best potions we've used. Let's find potions with a grade equal to or greater than 80.



Introducing Projections

find() takes a second parameter called a “projection” that we can use to specify the exact fields we want back by setting their value to true.

SHELL

```
> db.potions.find(
  {"grade": {"$gte": 80}},
  {"vendor": true, "name": true}
)
{
  "_id": ObjectId(...),
  "vendor": "Kettlecooked",
  "name": "Shrinking"
}
...
```

When selecting fields, all other fields but the _id are automatically set to false

Only retrieve what’s needed

THE MAGICAL MARVELS OF MONGODB

Excluding Fields

Sometimes you want all the fields except for a few. In that case, we can exclude specific fields.

SHELL

```
> db.potions.find(
  {"grade": {"$gte": 80}},
  {"vendor": false, "price": false}
)
{
  "_id": ObjectId(...),
  "name": "Shrinking",
  "grade": 94,
  "ingredients": [...],
  ...
}
```

When excluding fields, all fields but those set to false are defaulted to true

Great for removing sensitive data

THE MAGICAL MARVELS OF MONGODB

Excluding the _id

The ***_id*** field is always returned whenever selecting or excluding fields. It's the only field that can be set to false when selecting other fields.

SHELL

```
> db.potions.find(
  {"grade": {"$gte": 80}},
  {"vendor": true, "price": true, "_id": false}
)
{
  "vendor": "Homebrewed",
  "price": 9.99
}
```

The only time we can mix an exclusion with selections

Removing the id is common when preparing data reports for non-developers.

THE MAGICAL MARVELS OF MONGODB

Either Select or Exclude Fields

Whenever projecting, we either select or exclude the fields we want — we don't do both.

SHELL

```
> db.potions.find(
  {"grade": {"$gte": 80}},
  {"name": true, "vendor": false}
)
```

Causes an error to be raised

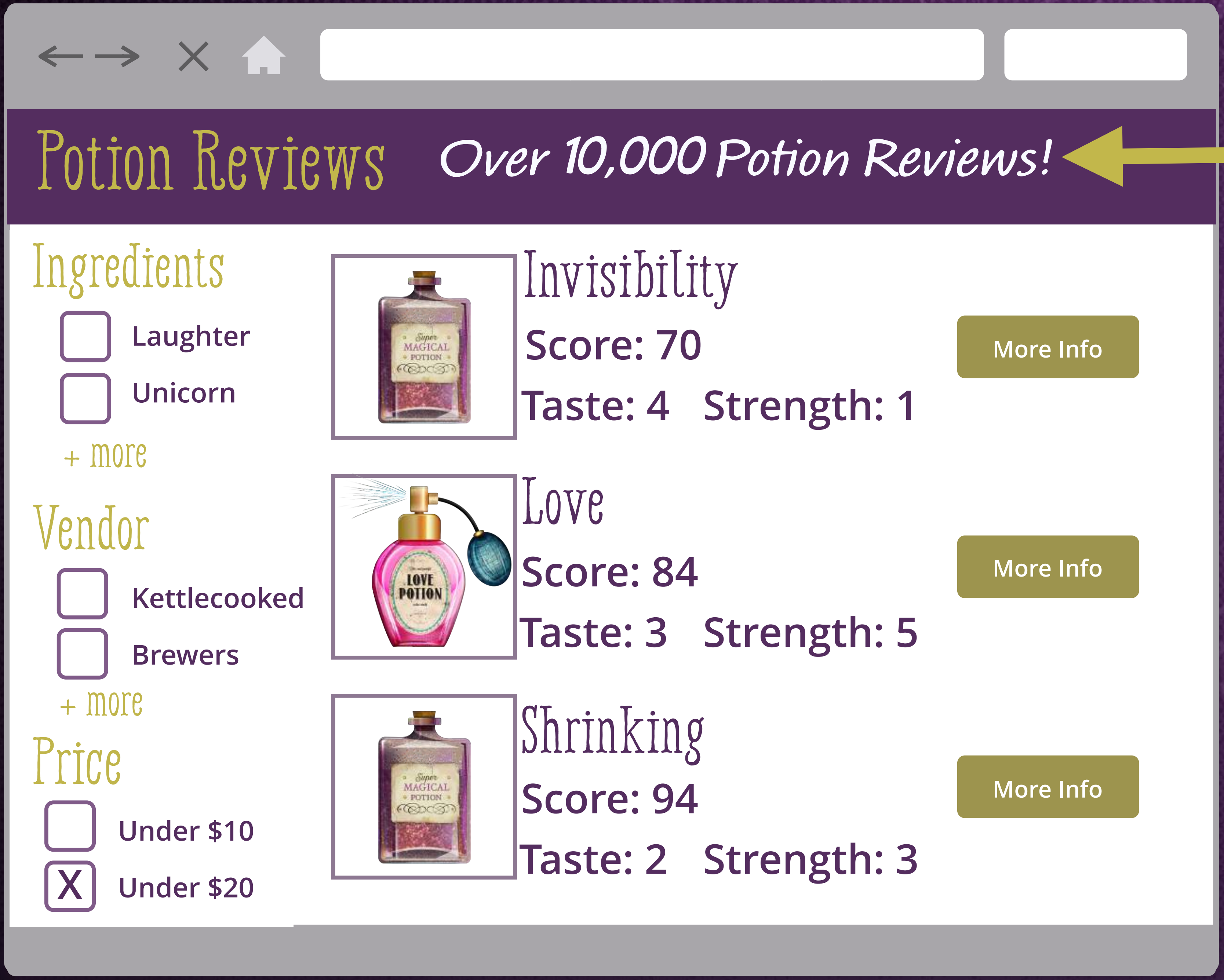
ERROR

```
"$err": "Can't canonicalize query: BadValue
Projection cannot have a mix of inclusion
and exclusion."
```

THE MAGICAL MARVELS OF MONGODB

Counting Our Potions

Time to advertise our expertise and list the total number of potions we've reviewed.



Potion Reviews *Over 10,000 Potion Reviews!*

Ingredients

☐ Laughter

☐ Unicorn

+ more

Vendor

☐ Kettlecooked

☐ Brewers

+ more

Price

☐ Under $10

☒ Under $20

Invisibility
Score: 70
Taste: 4 Strength: 1
More Info

Love
Score: 84
Taste: 3 Strength: 5
More Info

Shrinking
Score: 94
Taste: 2 Strength: 3
More Info

Need to count the total number of potions in the potions collection

Introducing the Cursor

Whenever we search for documents, an object is returned from the find method called a "cursor object."

SHELL

```
> db.potions.find("vendor": "Kettlecooked"})
{"_id": ObjectId(...), ... }
{"_id": ObjectId(...), ... }
{"_id": ObjectId(...), ... }
...
```

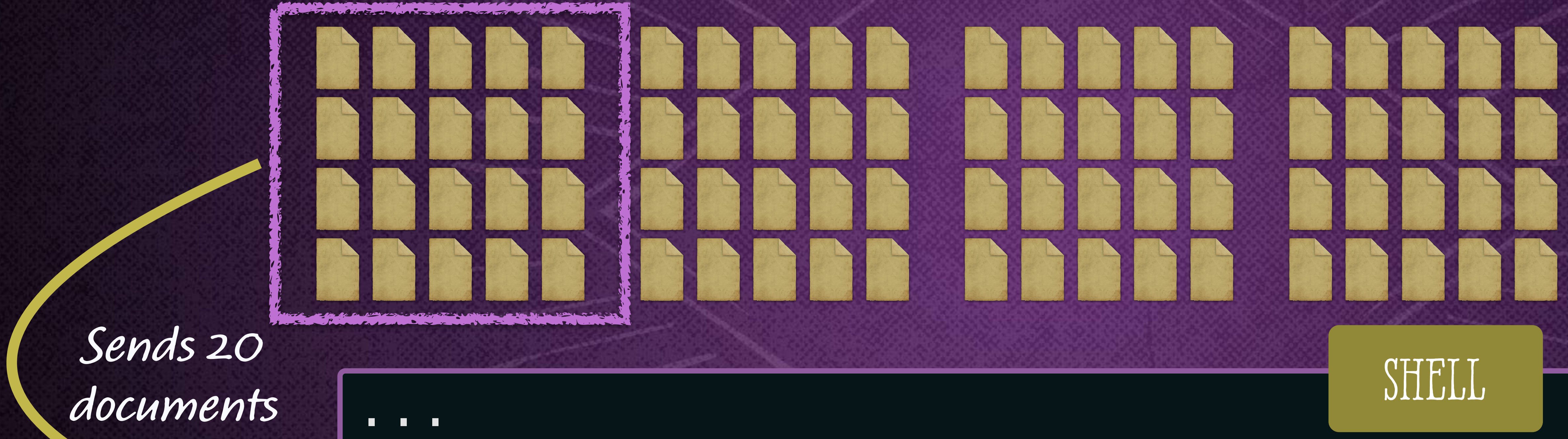
First 20 documents

By default, the first 20 documents are printed out

Iterating Through the Cursor

When there are more than 20 documents, the cursor will iterate through them 20 at a time.

```
db.potions.find()
```



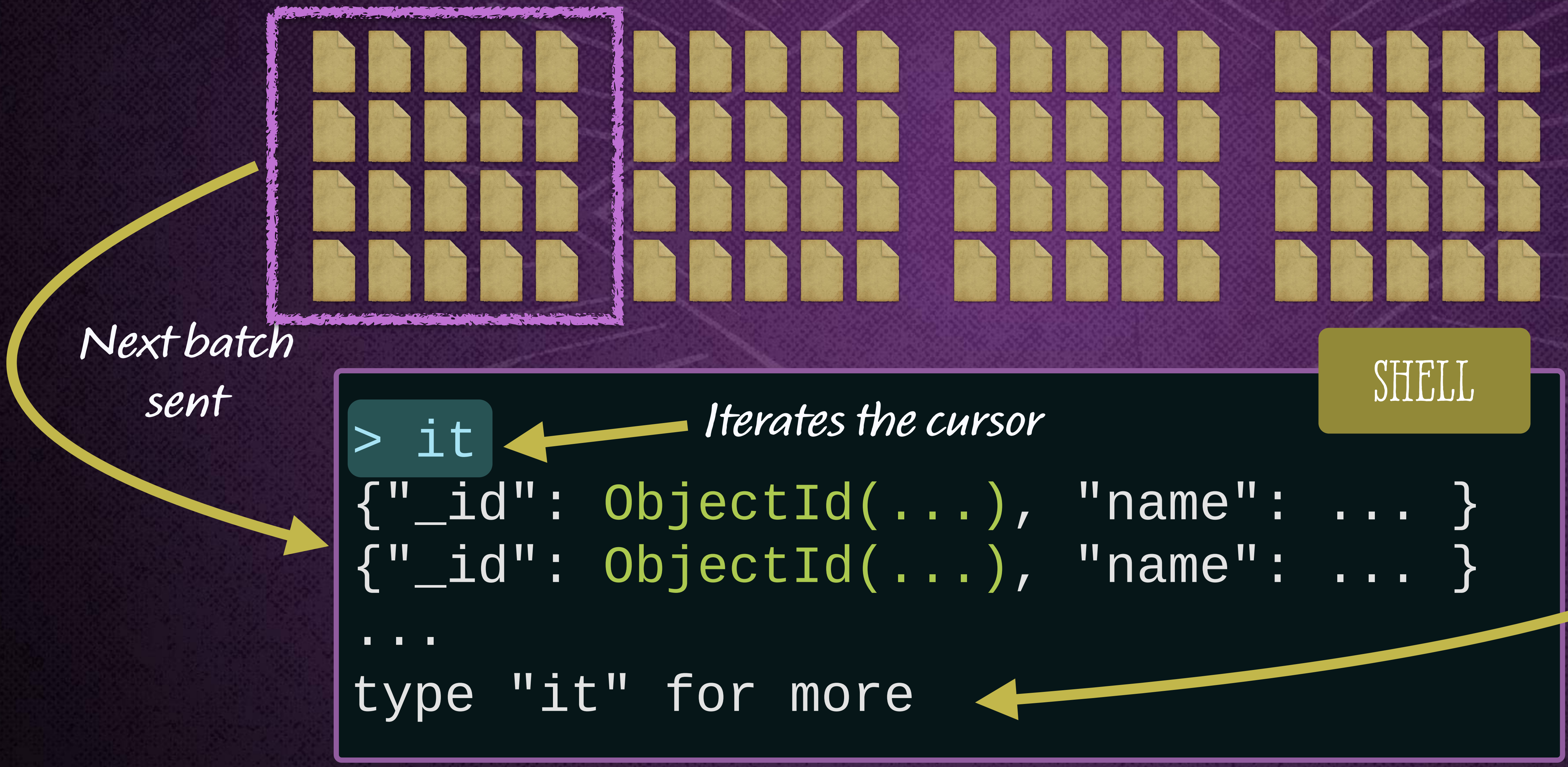
SHELL

```
...
{"_id": ObjectId(...), "name": ... }
{"_id": ObjectId(...), "name": ... }
{"_id": ObjectId(...), "name": ... }
type "it" for more
```

Continuing to Iterate Through the Cursor

Typing "it" will display the next 20 documents in the cursor.

```
db.potions.find()
```



```
> it
{"_id": ObjectId(...), "name": ... }
{"_id": ObjectId(...), "name": ... }
...
type "it" for more
```

Cursor Methods

Since the cursor is actually an object, we can chain methods on it.

Method on cursor that returns the count of matching documents

Returns cursor object

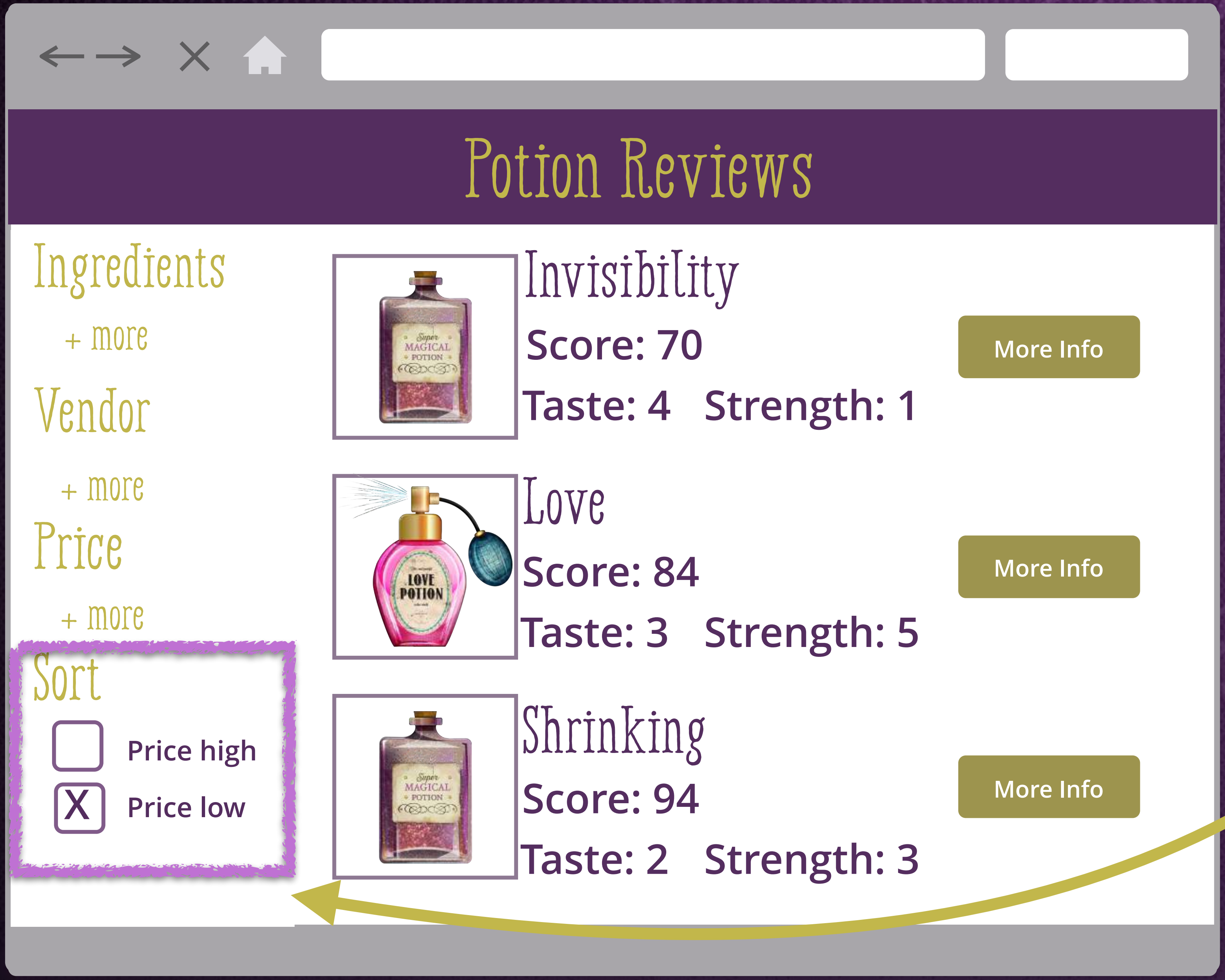
SHELL

```
> db.potions.find().count()
80
```

Cursor methods always come after find() since it returns the cursor object.

Sort by Price

We want to implement a way for users to sort potions by price.



Potion Reviews

[ ] Price high
[✓ Price low]

Invisibility
Score: 70
Taste: 4 Strength: 1

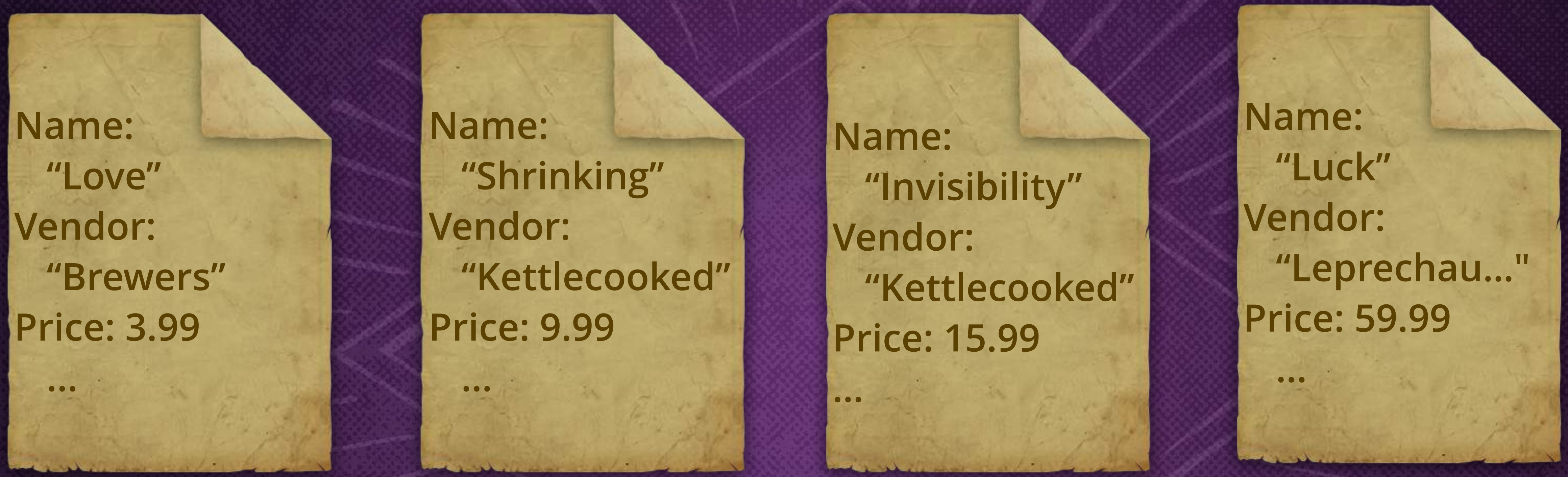
Love
Score: 84
Taste: 3 Strength: 5

Shrinking
Score: 94
Taste: 2 Strength: 3

Sort potions with the lowest price first

Sorting Potions

We can use the ***sort()*** cursor method to sort documents.



Name:
  "Love"
Vendor:
  "Brewers"
Price: 3.99
  …

Name:
  "Shrinking"
Vendor:
  "Kettlecooked"
Price: 9.99
  …

Name:
  "Invisibility"
Vendor:
  "Kettlecooked"
Price: 15.99
…

Name:
  "Luck"
Vendor:
  "Leprechau…"
Price: 59.99
  …

SHELL

```
> db.potions.find().sort({"price": 1})
```

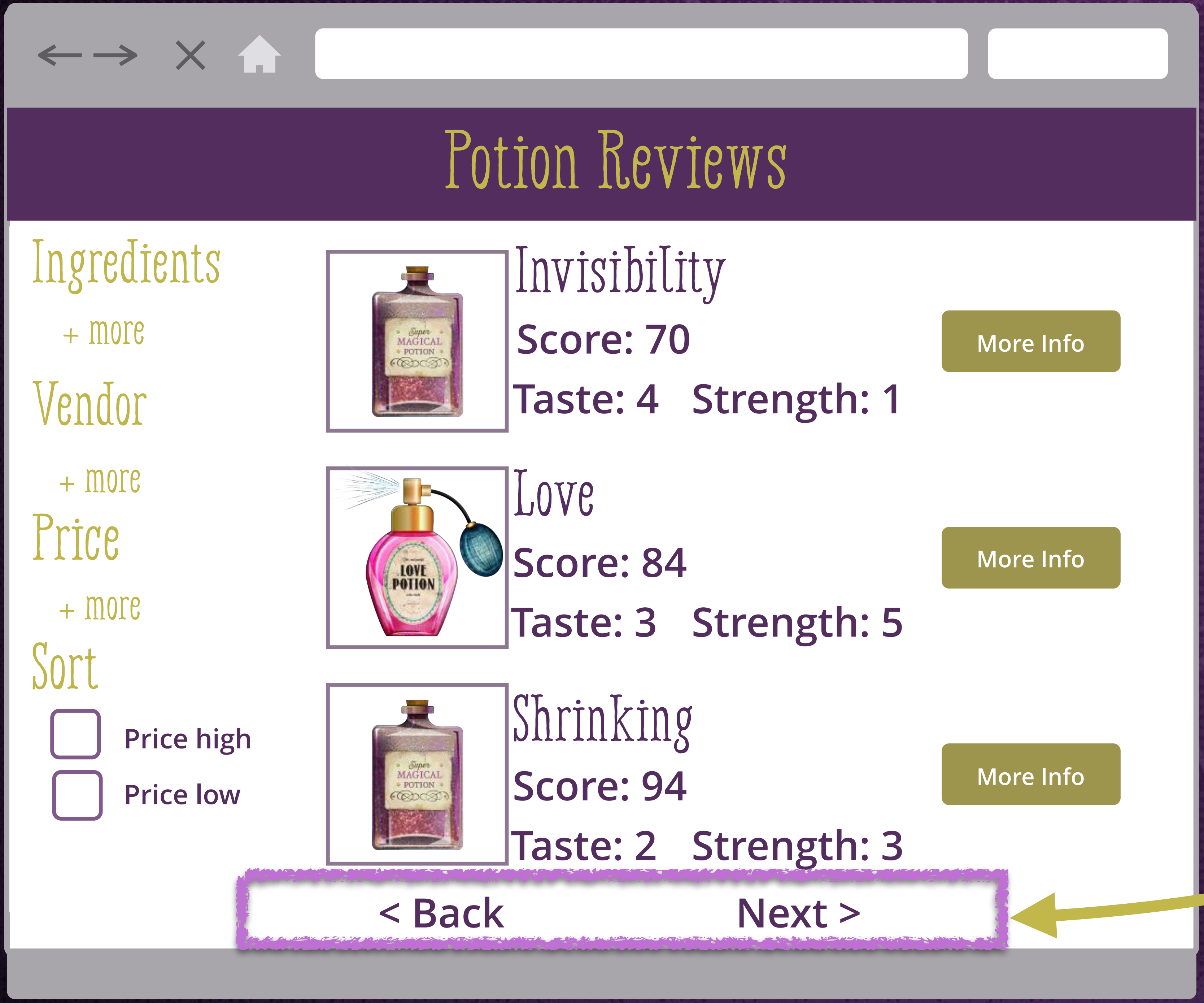
Field to sort

`-1` *to order descending*

`1` *to order ascending*

Paginating the Potions Page

We only want to show 3 potions per page. Time to implement pagination!



Potion Reviews

Ingredients
+ more
Vendor
+ more
Price
+ more
Sort
☐ Price high
☐ Price low

Invisibility
Score: 70
Taste: 4 Strength: 1
More Info

Love
Score: 84
Taste: 3 Strength: 5
More Info

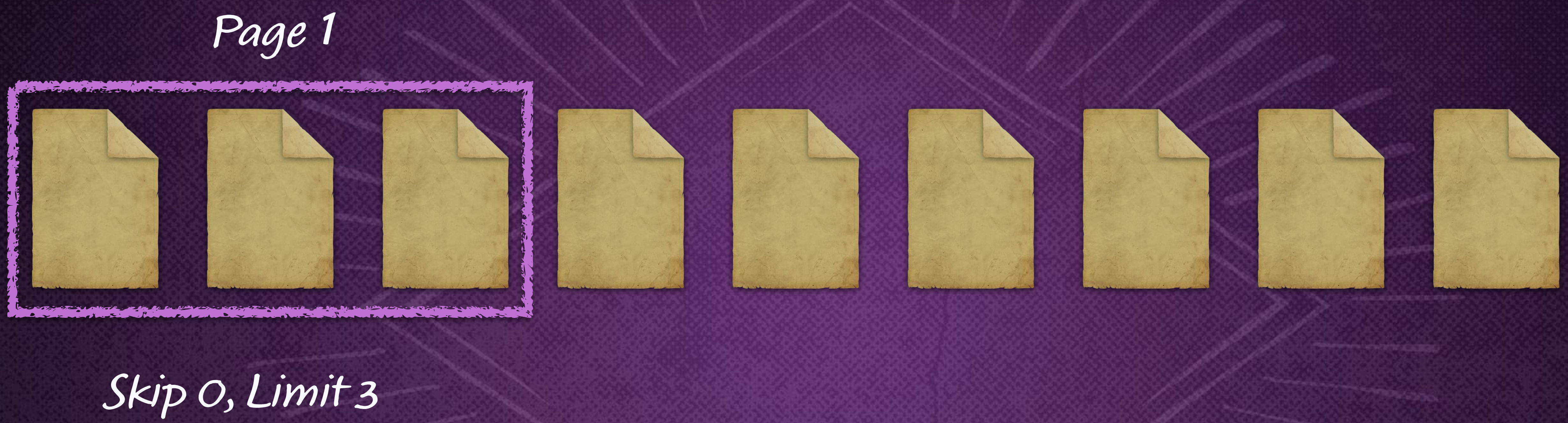
Shrinking
Score: 94
Taste: 2 Strength: 3
More Info

< Back Next >

Paginate results so we only see 3 potions on each page

Basic Pagination

We can implement basic pagination by limiting and skipping over documents. To do this, we'll use the ***skip()*** and ***limit()*** cursor methods.



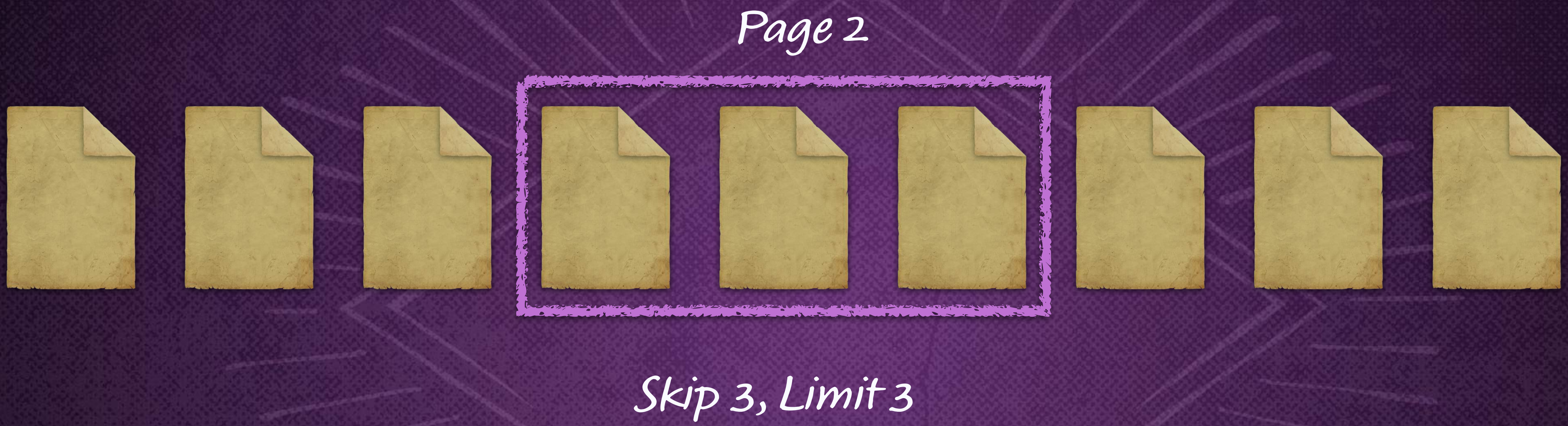
SHELL

```
> db.potions.find().limit(3)
```

Since we're not skipping, we can leave off the skip method and just limit 3

Basic Pagination

We can implement basic pagination by limiting and skipping over documents.

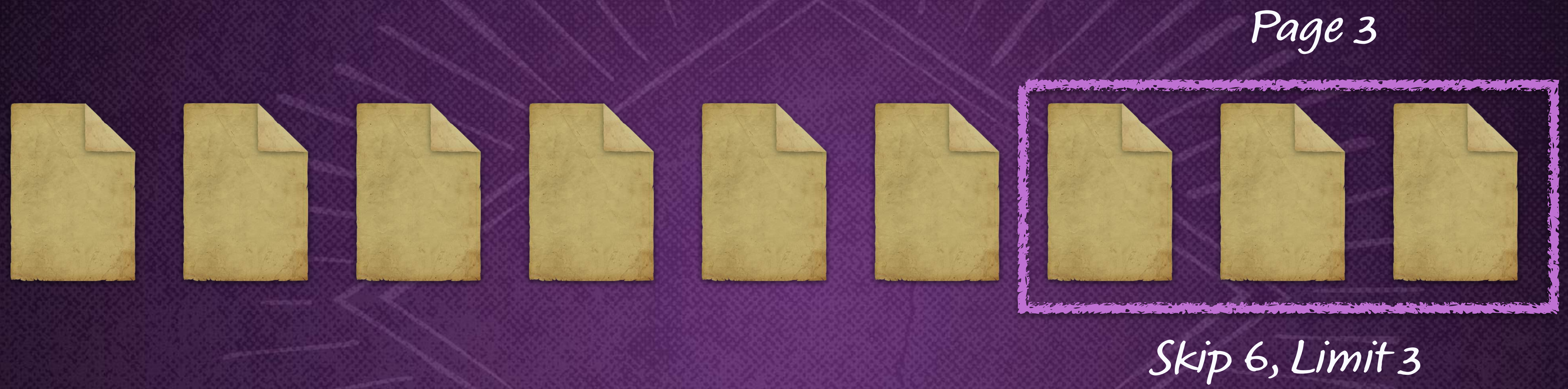


SHELL

```
> db.potions.find().skip(3).limit(3)
```

Basic Pagination

We can implement basic pagination by limiting and skipping over documents.



SHELL

```
> db.potions.find().skip(6).limit(3)
```

❗ *This approach can become really expensive with large collections.*

THE MAGICAL MARVELS OF MONGODB