



THE MAGICAL MARVELS OF MONGODB

Mystical Modifications


Level 2 – Section 1

Removing and Modifying Documents

Potion Catastrophe

Uh-oh — we sneezed while performing a spell and ruined some potions in our database!


Potion Reviews



Taste: 4

Strength: 1

More Info




Taste: 3

Score: 84

Shrinking Score: 94

More Info



Taste: 2

Strength: 3

More Info

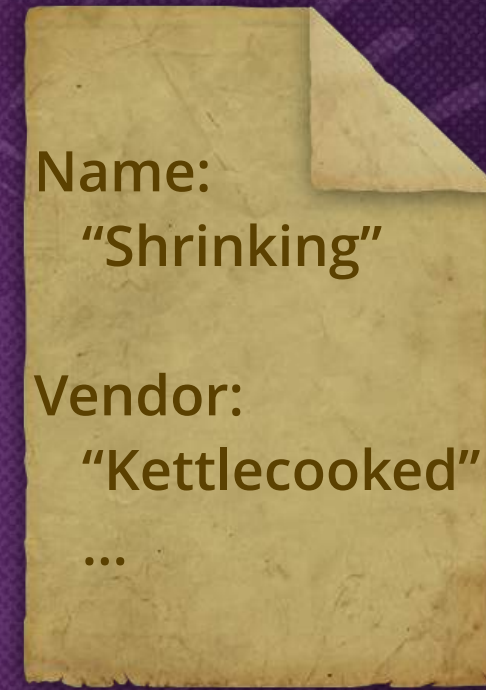
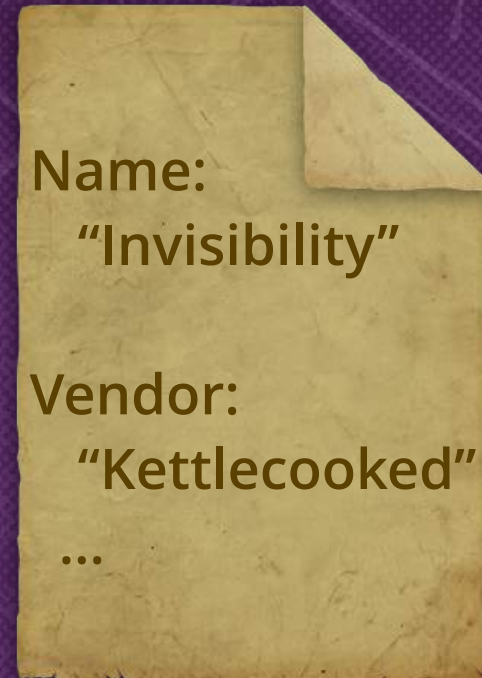
Need a way to remove the affected potions from our collection



Delete a Single Document

The ***remove()*** collection method will delete documents that match the provided query.

Ruined
Potions



SHELL

```
> db.potions.remove(  
  { "name": "Love" }  
)
```

```
WriteResult( { "nRemoved": 1 } )
```

*1 document
successfully removed*

*Query matches
single document*

**THE
MAGICAL MARVELS
OF MONGODB**

Delete a Single Document

The ***remove()*** collection method will delete documents that match the provided query.

Ruined
Potions

Name:
"Invisibility"

Vendor:
"Kettlecooked"

...

Name:
"Shrinking"

Vendor:
"Kettlecooked"

...

SHELL

```
> db.potions.remove(  
  { "name": "Love" }  
)  
WriteResult( { "nRemoved": 1 } )
```

*1 document
successfully removed*

*Query matches
single document*

THE
MAGICAL MARVELS
OF MONGODB

Delete Multiple Documents

If our query matches multiple documents, then ***remove()*** will delete all of them.

Name:
"Invisibility"

Vendor:
"Kettlecooked"
...

Name:
"Shrinking"

Vendor:
"Kettlecooked"
...



Passing {} as the query would delete all documents in the collection.

SHELL

```
> db.potions.remove(  
  {"vendor": "Kettlecooked"}  
)
```

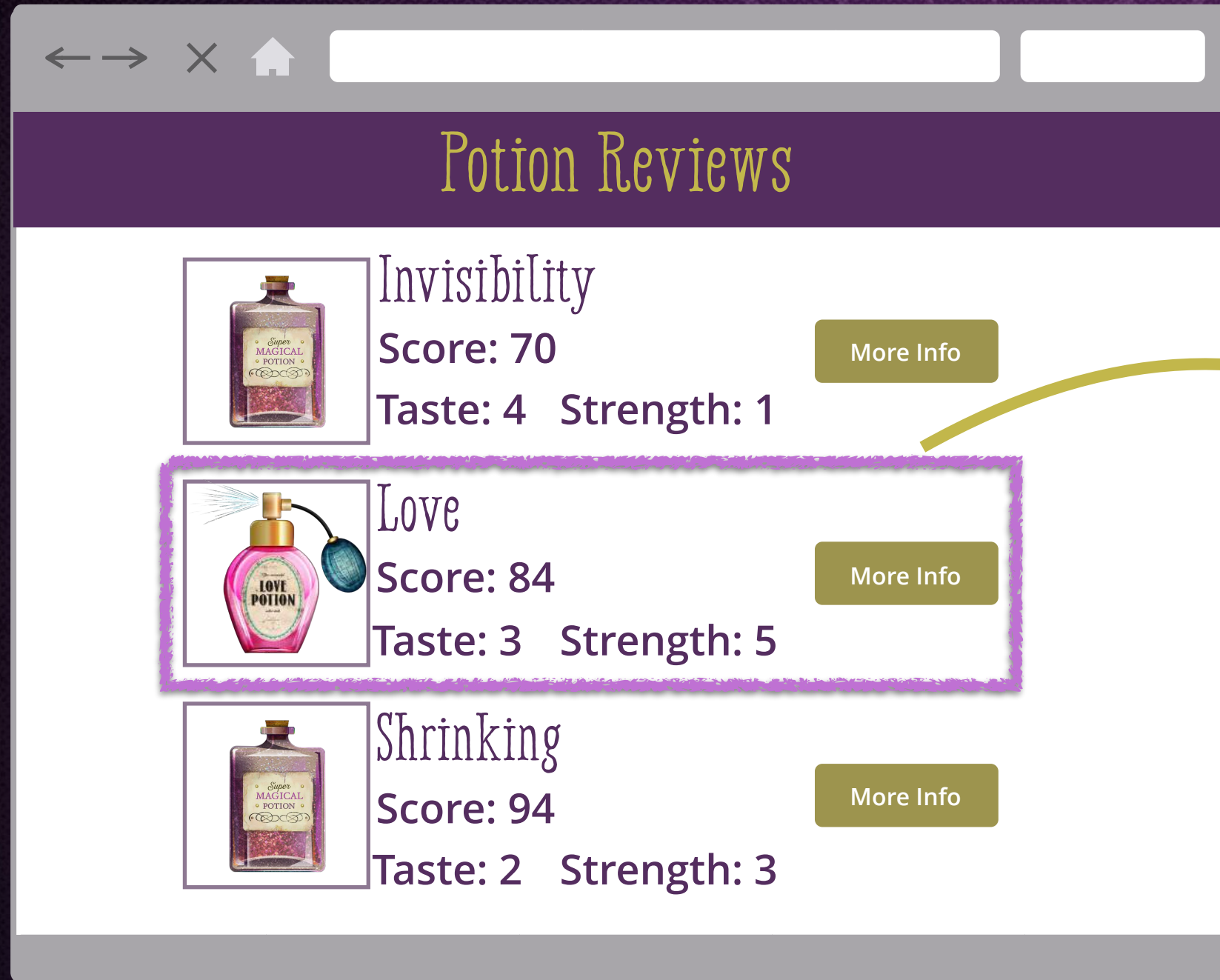
```
WriteResult( { "nRemoved": 2 } )
```

Query matches both documents




Removed 2 documents

Update the Price of a Single Potion

We made a typo while inserting our love potion, so let's update the price.



Potion Reviews

Potion Image	Potion Name	Score	Taste	Strength	More Info
	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

Name:
"Love"

Vendor:
"Brewers"

Price: 40.99

...

*Needs to be updated with
the correct price*

THE
MAGICAL MARVELS
OF MONGODB

Updating a Document

We can use the **update()** collection method to modify existing documents.

Name:
"Love"

Vendor:
"Brewers"

Price: 40.99
...

Update

Name:
"Love"

Vendor:
"Brewers"

Price: 3.99
...

Price is updated!

Update only applied to
first matching document

SHELL

```
> db.potions.update(  
  { "name" : "Love" },  
  { "$set" : { "price" : 3.99 } }  
)
```

Query parameter

Update parameter

Update operators
always begin with a \$

THE
MAGICAL MARVELS
OF MONGODB

Understanding the Update WriteResult

The WriteResult gives a summary of what the **update()** method did.

SHELL

```
> db.potions.update(  
  { "name": "Love" },  
  { "$set": { "price": 3.99 } }  
)  
WriteResult( {  
  "nMatched": 1,  
  "nUpserted": 0,  
  "nModified": 1  
} )
```

Number of documents matched

*Number of documents that
were created*

Number of documents modified



Update Without an Operator

If the update parameter consists of only field/value pairs, then everything but the **_id** is replaced in the matching document.

Name:
"Love"

Vendor:
"Brewers"

Price: 40.99
...

Update

Price: 3.99

Document replaced with the
update parameter



No operator, just field/value pair

SHELL

```
> db.potions.update(  
  { "name": "Love" },  
  { "price": 3.99 }  
)
```

Useful for
importing data

THE
MAGICAL MARVELS
OF MONGODB

Updating Multiple Documents

The update method can take a third parameter for options.

SHELL

Notice

WE ARE NOT
CALLED KC

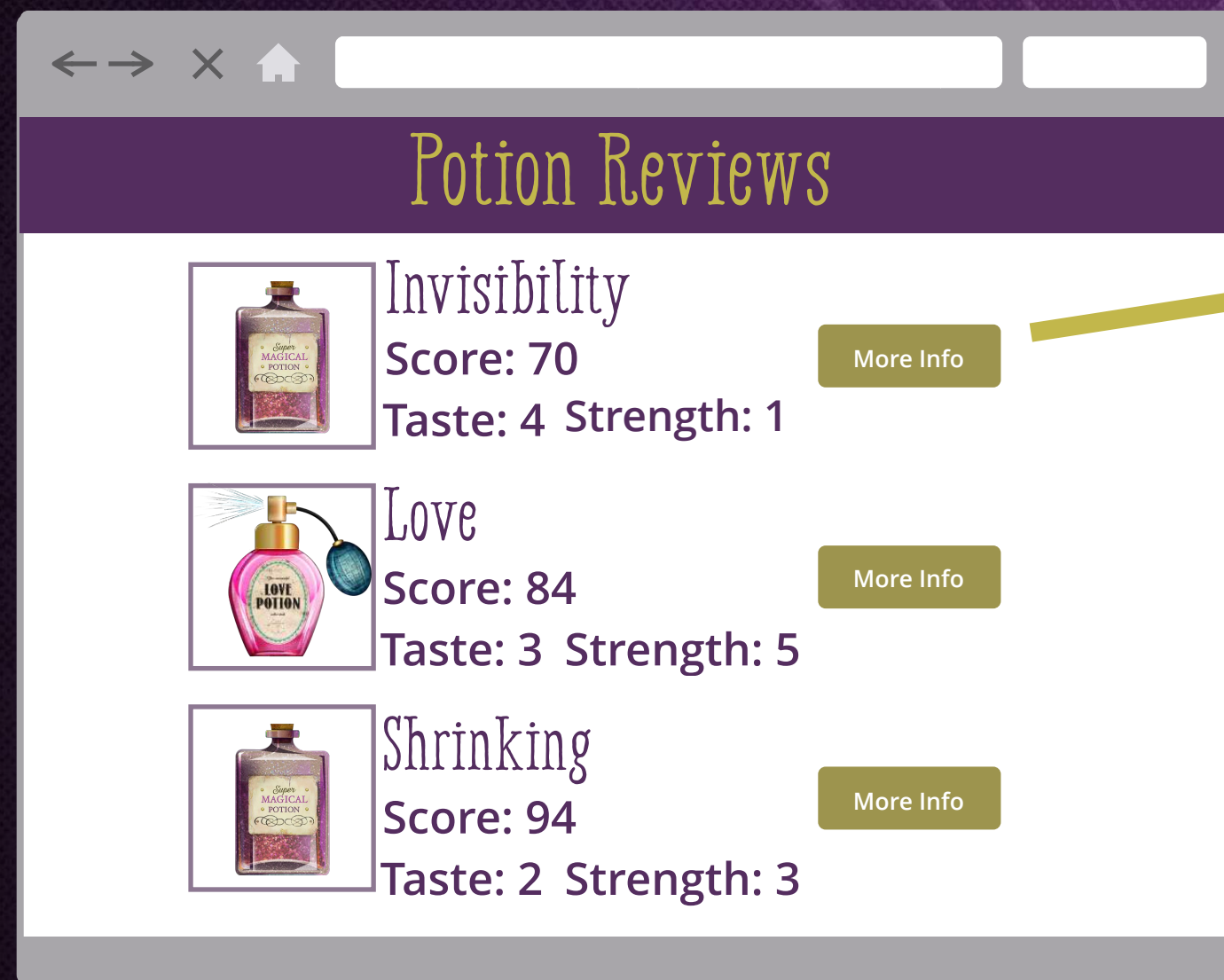
```
> db.potions.update(  
  { "vendor": "KC" },  
  { "$set": { "vendor": "Kettlecooked" } },  
  { "multi": true }  
)  
  
WriteResult({  
  "nMatched": 4,  
  "nUpserted": 0,  
  "nModified": 4  
})
```

← When multi is true, the update modifies all matching documents

4 documents matched
and modified

Recording Potion Views

Time to start analyzing which potions are viewed the most. To do this, we need to record each potion page view.



Create or update
existing log document

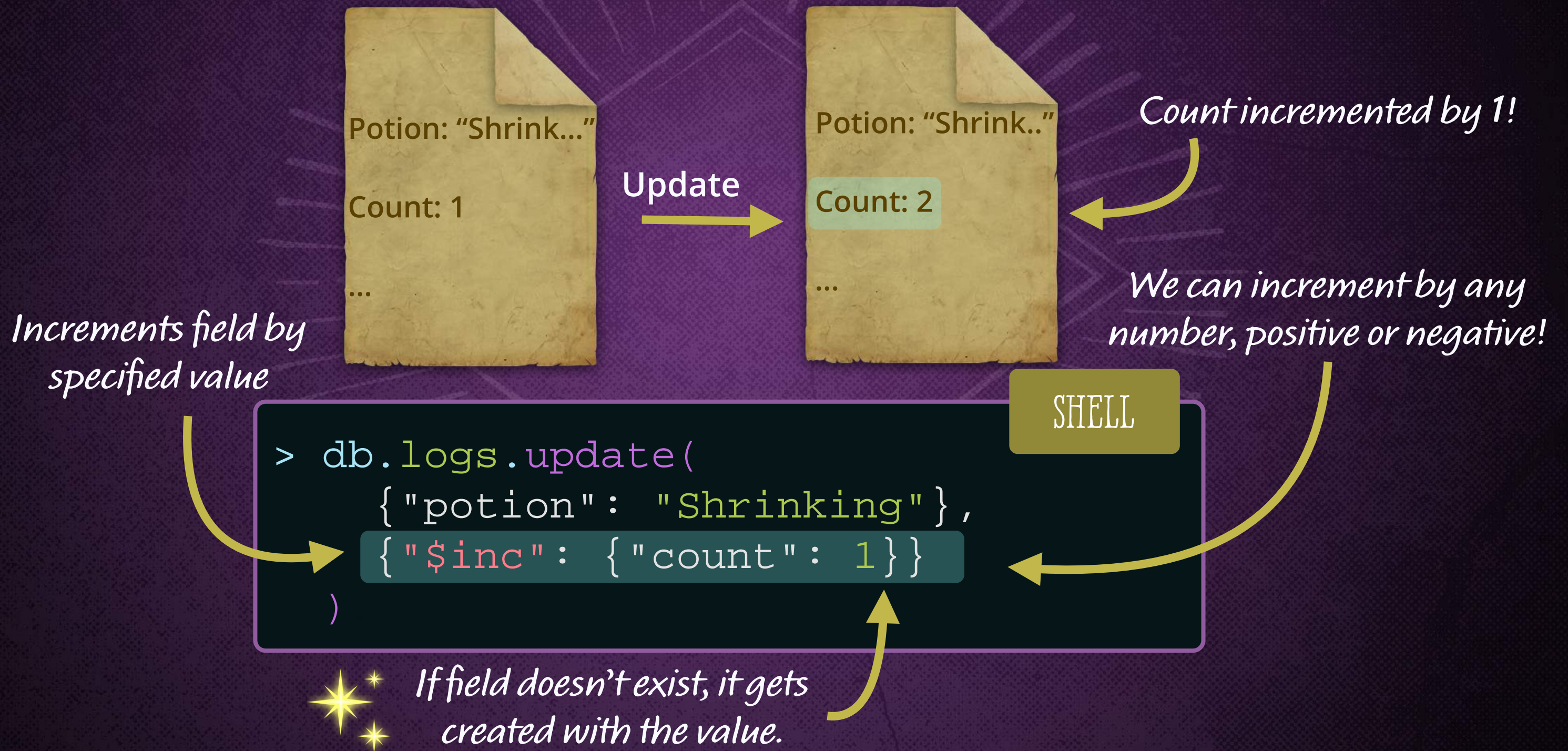
Logs Collection

```
{  
  "_id": ObjectId(...),  
  "potion": "Frog Tonic",  
  "count": 1  
}
```

We'll update count
with each click

Update a Document's Count

We can use the **\$inc** operator to increment the count of an existing log document.



Update a Non-existing Potion

If we run the update on a potion that doesn't exist, then nothing will happen.

SHELL

```
> db.logs.update(  
  {"potion": "Love"},  
  {"$inc": {"count": 1}},  
)  
WriteResult({  
  "nMatched": 0,  
  "nUpserted": 0,  
  "nModified": 0  
})
```

Potion log doesn't exist yet



No potions matched or modified

Find or Create With Upsert

The upsert option either updates an existing document or creates a new one.

If the field doesn't exist, it gets created with the value

SHELL

```
> db.logs.update(  
  {"potion": "Love"},  
  {"$inc": {"count": 1}},  
  {"upsert": true}  
)  
WriteResult({  
  "nMatched": 0,  
  "nUpserted": 1,  
  "nModified": 0  
})
```

Results in new document

Name: "Love"

Count: 1

...

Creates a document using the values from the query and update parameter

1 document created

Updating Once More

If we run the same update again, the update will act normally and **upsert** won't create another document.

SHELL

```
> db.logs.update(  
  {"potion": "Love"},  
  {"$inc": {"count": 1}},  
  {"upsert": true}  
)  
WriteResult({  
  "nMatched": 1,  
  "nUpserted": 0,  
  "nModified": 1  
})
```

Result

Count of 2

Document found and
modified but nothing created

Name: "Love"

Count: 2

...

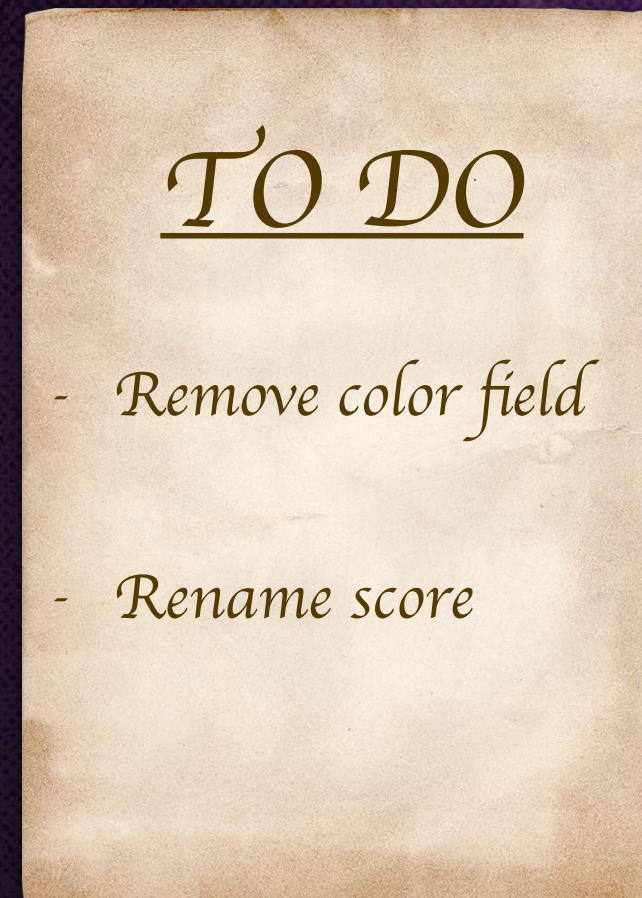
Mystical Modifications

Level 2 – Section 2

Advanced Modification

Improving Potions

We rushed a bit during development and need to fix up our potions. Luckily, we keep a to-do list for what to work on.



We can accomplish these tasks with update operators!



Removing Fields From Documents

We initially thought we'd need a potion's color, but we never use it. The **\$unset** operator can be used to remove specified fields.

Name:
"Love"
~~Color: "red"~~
...

Name:
"Invisibility"
~~Color: "black"~~
...

Name:
"Shrinking"
~~Color: "green"~~
...

*Color field
removed from
documents*

*Query for
all potions*

*Update all
potions*

```
> db.potions.update(  
  {},  
  { "$unset": { "color": "" } },  
  { "multi": true }  
)
```

SHELL

*The value we pass doesn't
impact the operation*

Updating a Field Name With \$rename

We can use **\$rename** to change field names.

```
{
  "_id": ObjectId(...),
  "name": "Love",
  "score": 84,
  ...
}
```



```
{
  "_id": ObjectId(...),
  "name": "Love",
  "grade": 84,
  ...
}
```

Renamed to grade!

*Renames
specified
field*

```
> db.potions.update(
  {},
  { "$rename": { "score": "grade" } },
  { "multi": true }
)
```

SHELL

*New field
name*

*Field to
rename*

**THE
MAGICAL MARVELS
OF MONGODB**

Potion Ingredient Regulation

The Magical Council has passed a new regulation requiring us to list all ingredients in a potion. No more secret ingredients!

Notice

*All secret ingredients
must be listed!*

```
{  
  "_id": ObjectId(...),  
  "name": "Shrinking",  
  ...  
  "ingredients": ["hippo", "secret", "mouse feet"]  
}
```

*Need to update with actual
ingredient: 42*

The Dilemma of Updating an Array

Name: "Shrink..."
Vendor: "Kettle..."
Ingredients...
...

"ingredients": ["hippo", "secret", "mouse feet"]

```
> db.potions.update(  
  {"ingredients": "secret"},  
  {"$set": {"ingredients": "42"}}  
)
```

SHELL



Would overwrite the entire array and set it as 42

"ingredients": 42

Updating Array Values by Location

Since array values are treated individually, we can update a single value by specifying its location in the array using **dot notation**.

```
{
  "_id": ObjectId(...),
  "name": "Shrinking",
  "vendor": "Kettlecooked",
  "score": 94,
  ...
  "ingredients": ["hippo", "secret", "mouse feet"]
}
```

ingredients.0

ingredients.1

ingredients.2

BSON arrays start with an index of 0



Updating Single Array Value

The **\$set** operator will update the value of a specified field.

The secret ingredient!

SHELL

```
> db.potions.update(  
  { "name": "Shrinking" },  
  { "$set": { "ingredients.1" : 42 } }  
)
```

```
WriteResult({ "nMatched": 1, "nUpserted": 0, "nModified": 1 })
```

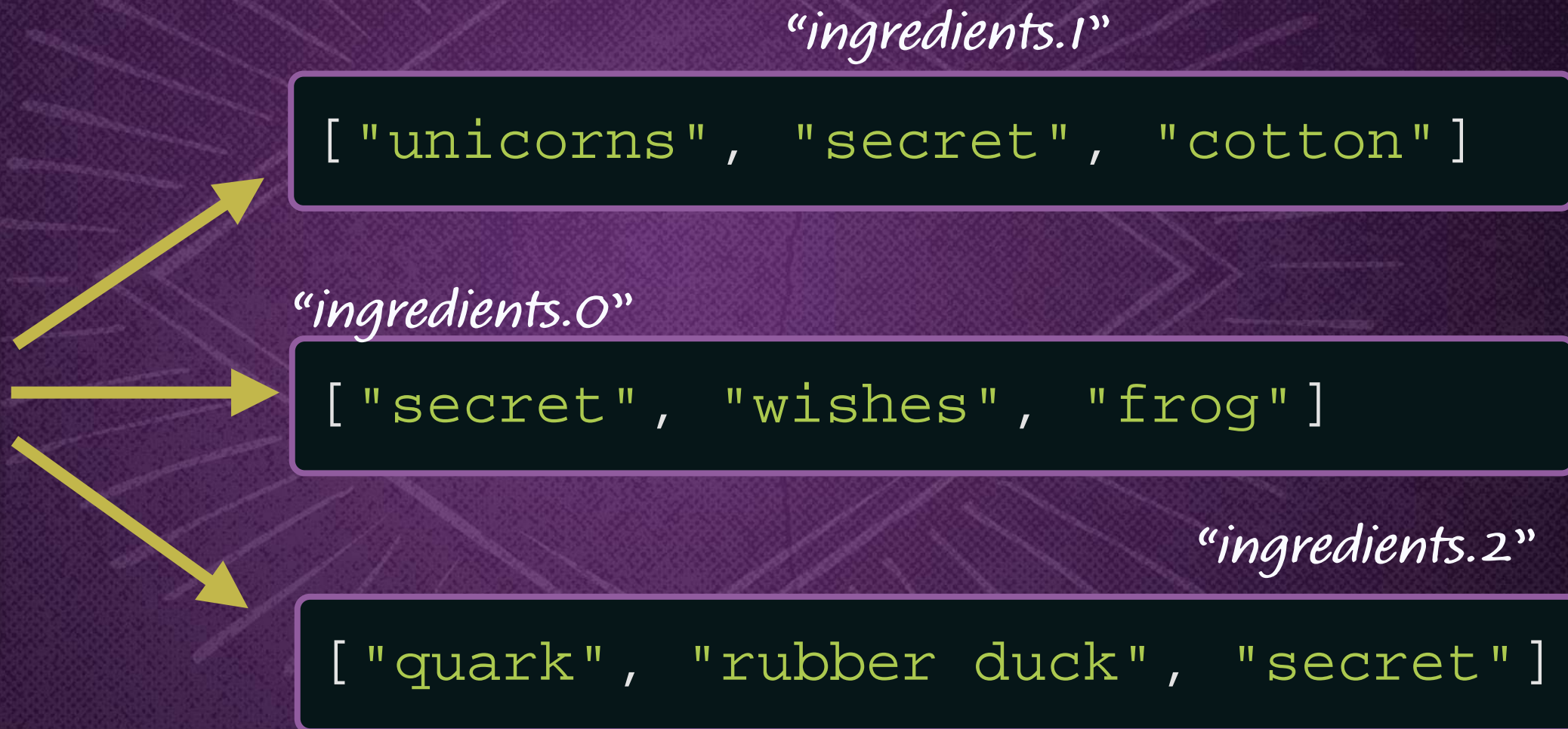
```
{  
  "_id": ObjectId(...),  
  "name": "Shrinking",  
  ...  
  "ingredients": ["hippo", 42, "mouse feet"]  
}
```

*Successful
update*

Updating Multiple Arrays

We need to change "secret" in multiple documents, but the location isn't always the same for every potion.

Potions Collection



Updating Values Without Knowing Position

The positional operator is a placeholder that will set the proper position for the value specified in the query parameter.

```
> db.potions.update(  
  {"ingredients": "secret"},  
  {"$set": {"ingredients.$": 42}},  
  {"multi": true}  
)
```

SHELL

Query for the value we want to change

Multi is true to make the change to all documents

The \$ is a placeholder for the matched value

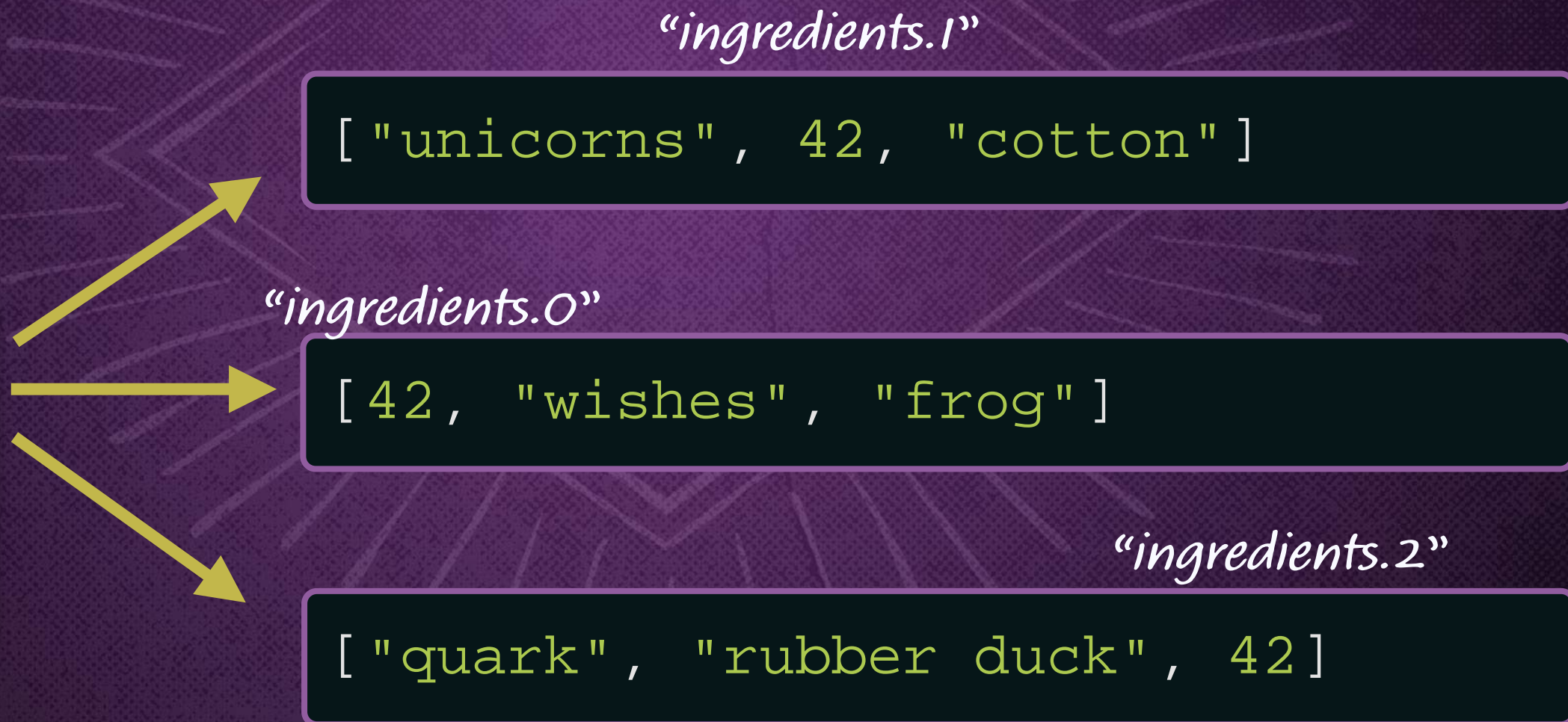


Only updates the first match per document

The Result of Using the Positional Operator to Update

```
...  
  { "ingredients": "secret" },  
  { "$set": { "ingredients.$" : 42 } },  
...
```

Potions Collection



Shrunkened Conundrum

Uh-oh — the shrinking potion hasn't worn off, and we keep on shrinking! We better update that strength rating.



```
{
  "_id": ObjectId(...),
  "name": "Shrinking",
  ...
  "ratings": {
    "strength": 1,
    "flavor": 5
  }
}
```

*Update strength
to 5*

Updating an Embedded Value

We can update using the dot notation to specify the field we want to update.

Name: "Shrink..."
Vendor: "Kettle..."
Ratings...
...

"ratings": {
 "strength": 1,
 "flavor": 5
}

ratings.strength

SHELL

```
> db.potions.update(  
  { "name": "Shrinking" },  
  { "$set": { "ratings.strength" : 5 } }  
)
```

```
WriteResult( { "nMatched": 1, "nUpserted": 0, "nModified": 1 } )
```


Useful Update Operators

MongoDB provides a variety of ways to modify the values of fields.

`$max`

Updates if new value is greater than current or inserts if empty

`$min`

Updates if new value is less than current or inserts if empty

`$mul`

Multiplies current field value by specified value. If empty, it inserts 0.

Operator documentation:
<http://go.codeschool.com/update-operators>

*MongoDB's
documentation is great*

Reference > Operators > Update Operators > Field Update Operators > \$max

\$max

Definition

\$max

The **\$max** operator updates the value of the field to a specified value *if* the specified value is **greater than** the current value of the field. The **\$max** operator can compare values of different types, using the [BSON comparison order](#).

The **\$max** operator expression has the form:

```
{ $max: { <field1>: <value1>, ... } }
```

To specify a **<field>** in an embedded document or in an array, use [dot notation](#).



Modifying Arrays

We've added categories to the potions but need a way to easily manipulate the values.

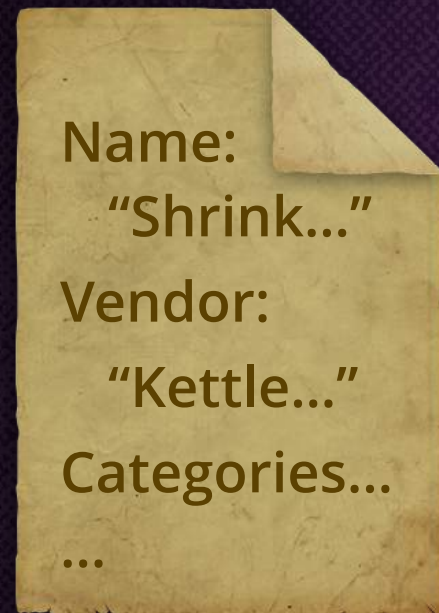
Name:
"Shrink..."
Vendor:
"Kettle..."
Categories...
...

List of categories for the potion

```
"categories": ["tasty", "effective"]
```


Removing the First or Last Value of an Array

The **\$pop** operator will remove either the first or last value of an array.



"categories": ["tasty", "effective"]

```
> db.potions.update(  
  {"name": "Shrinking"},  
  { "$pop": {"categories": 1} })
```

Result

"categories": ["tasty"]



Doesn't return the value — only modifies the array

- 1 Removes the first element
- 1 Removes the last element

Adding Values to the End of an Array

The **\$push** operator will add a value to the end of an array.

Name:
"Shrink..."
Vendor:
"Kettle..."
Categories...
...

"categories": ["tasty"]

```
> db.potions.update(  
  { "name": "Shrinking" },  
  { "$push": { "categories": "budget" } } )
```

Result

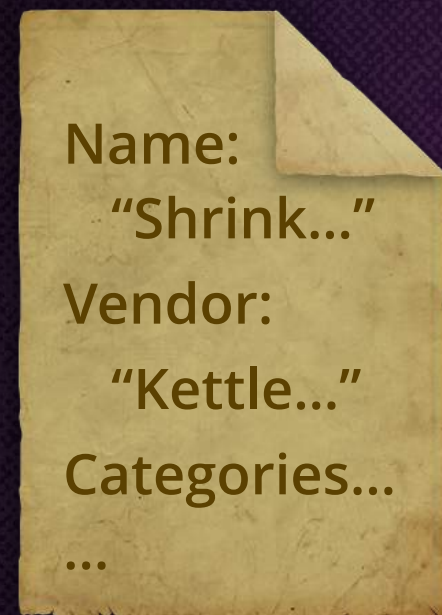
"categories": ["tasty", "budget"]

Added to the end

THE
MAGICAL MARVELS
OF MONGODB

Adding Unique Values to an Array

The ***\$addToSet*** operator will add a value to the end of an array unless it is already present.



"categories": ["tasty", "budget"]

```
> db.potions.update(  
  { "name": "Shrinking" },  
  { "$addToSet": { "categories": "budget" } } )
```

Result

"categories": ["tasty", "budget"]

*Value already exists, so it
doesn't get added again*

Removing Values From an Array

The **\$pull** operator will remove any instance of a value from an array.

Name:
"Shrink..."
Vendor:
"Kettle..."
Categories...
...

"categories": ["tasty", "budget"]

```
> db.potions.update(  
  {"name": "Shrinking"},  
  {"$pull": {"categories": "tasty"}})
```

Result

"categories": ["budget"]



If value isn't unique, then all instances will be removed from the array.