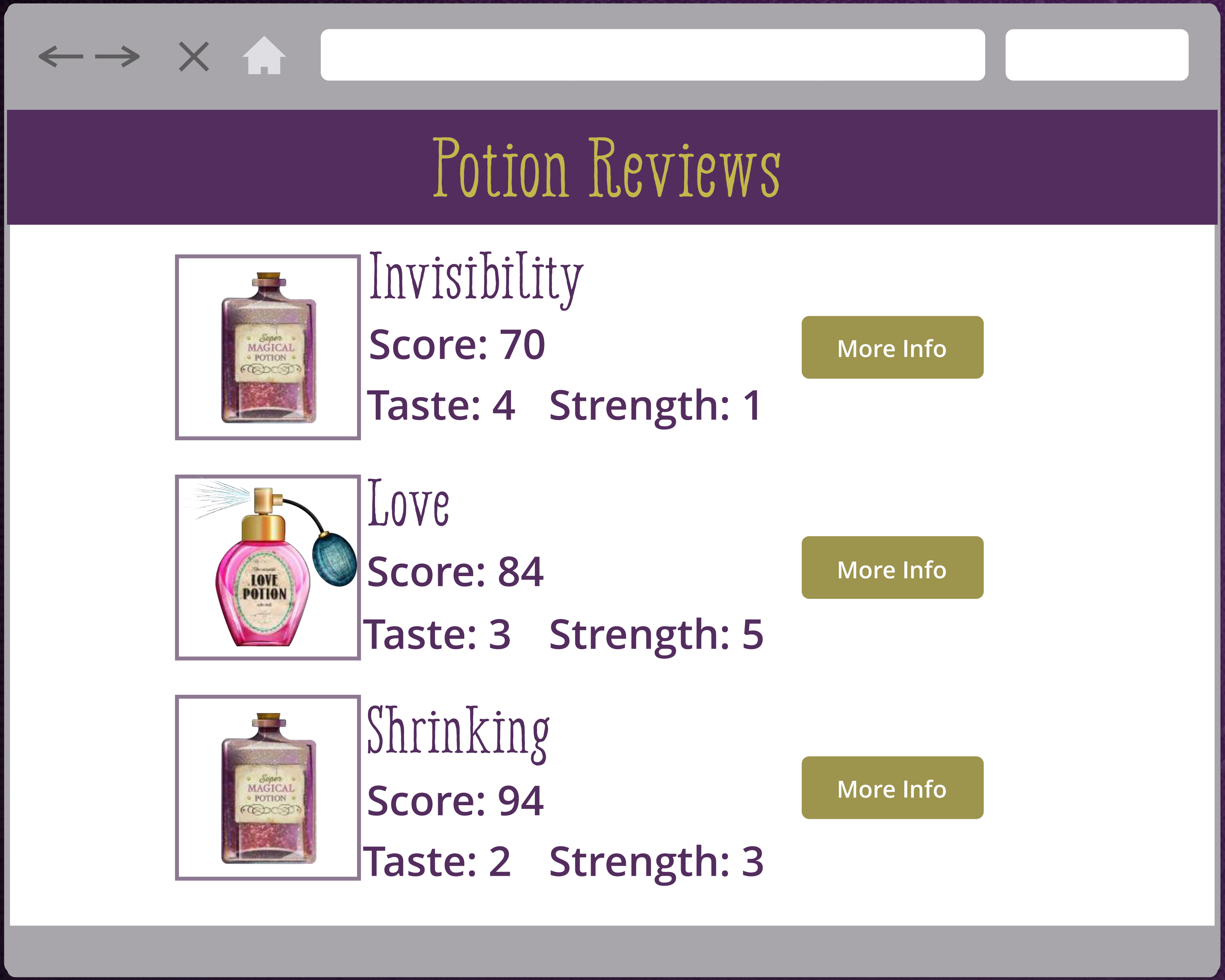
The Magical Marvels of MongoDB

Example Application



Potion Reviews

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

Course Outline

Conjuring MongoDB

Level 1 – Section 1

Introducing MongoDB

What Is MongoDB?

- Open-source NoSQL database
- Document-oriented
- Great for unstructured data, especially when you have a lot of it

Catch-all term for databases that generally aren't relational and don't have a query language like SQL

Began developing MongoDB as part of their stack

Open-sourced

Name comes from the word "humongous"

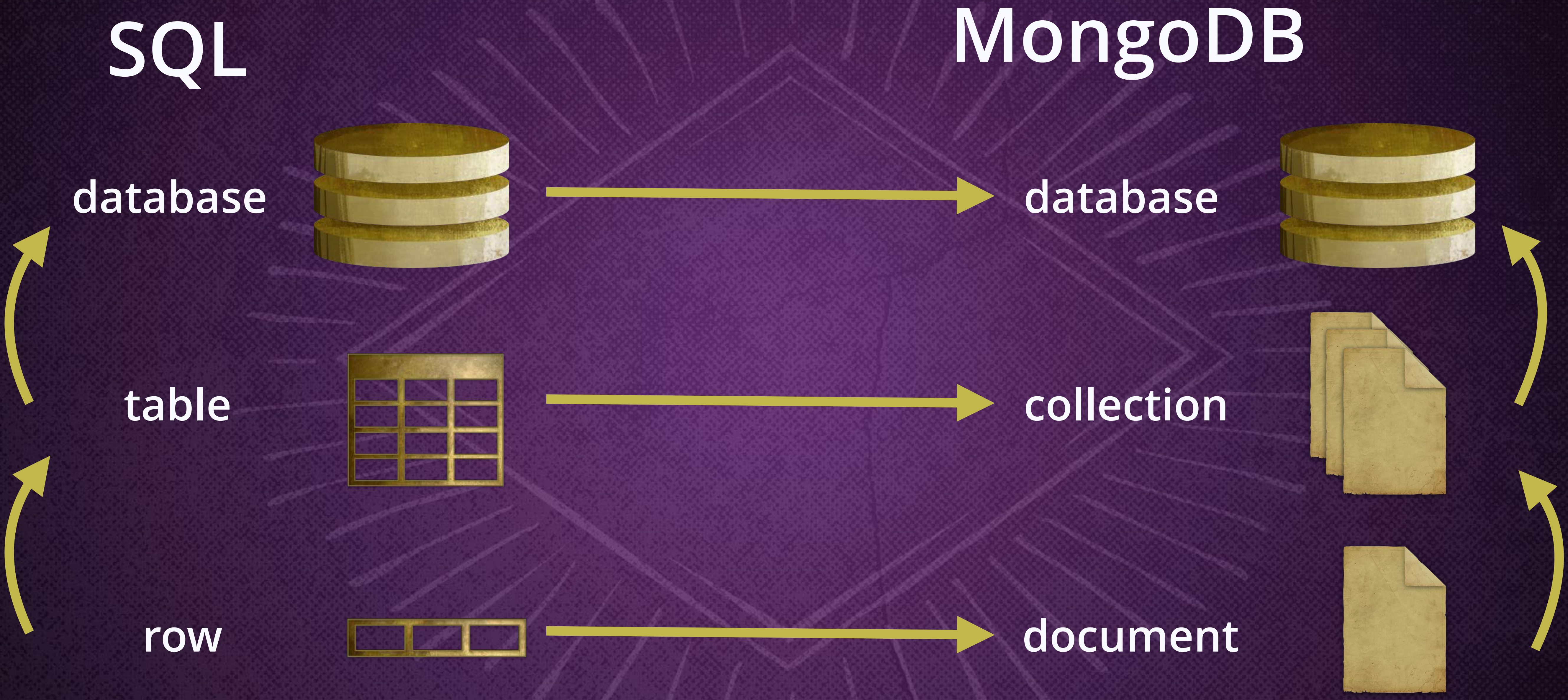
Renamed to MongoDB

2007

2009

2013

MongoDB Comparison to SQL



The main difference? SQL is *relational* and MongoDB is *document-oriented.*

Relational vs. Document-oriented

Relational database management systems save data in rows within tables. MongoDB saves data as documents within collections.

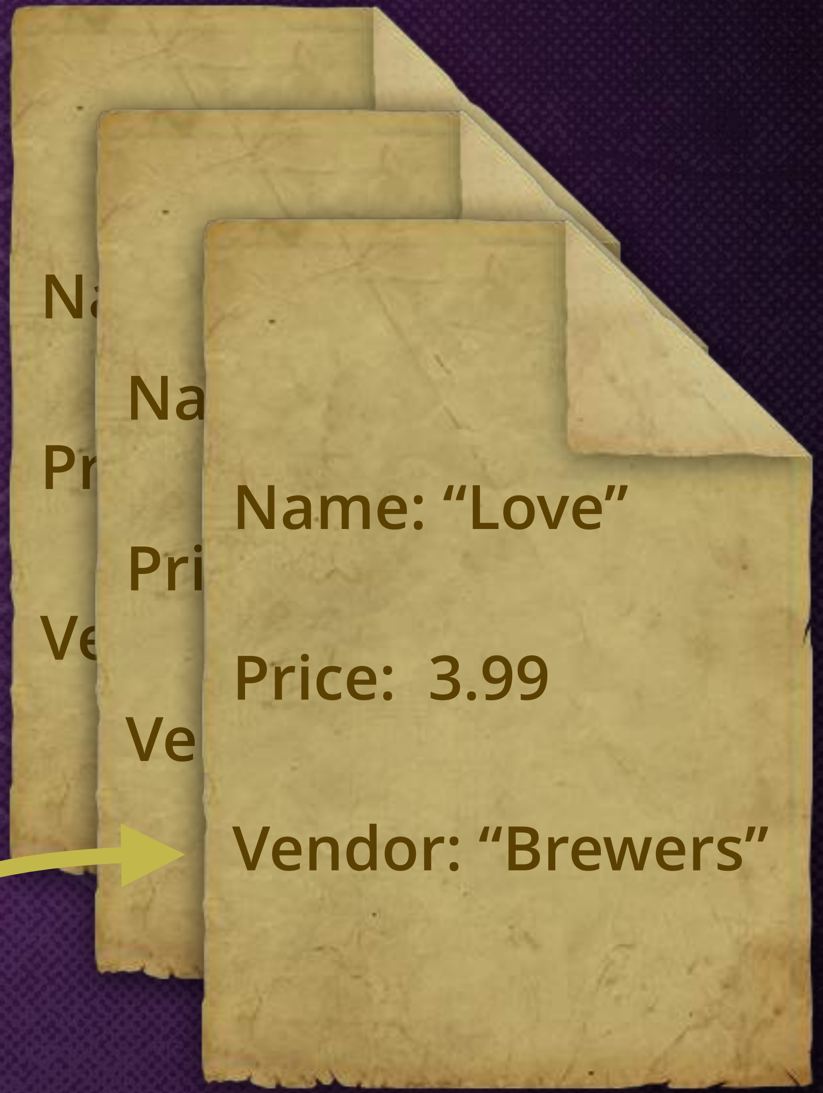
Potions Table

potion_id	name	price	vendor_id
1	"Love"	3.99	2
2	"Invisibility"	15.99	1
3	"Shrinking"	9.99	1

Vendors Table

vendor_id	name
1	"Kettlecooked"
2	"Brewers"

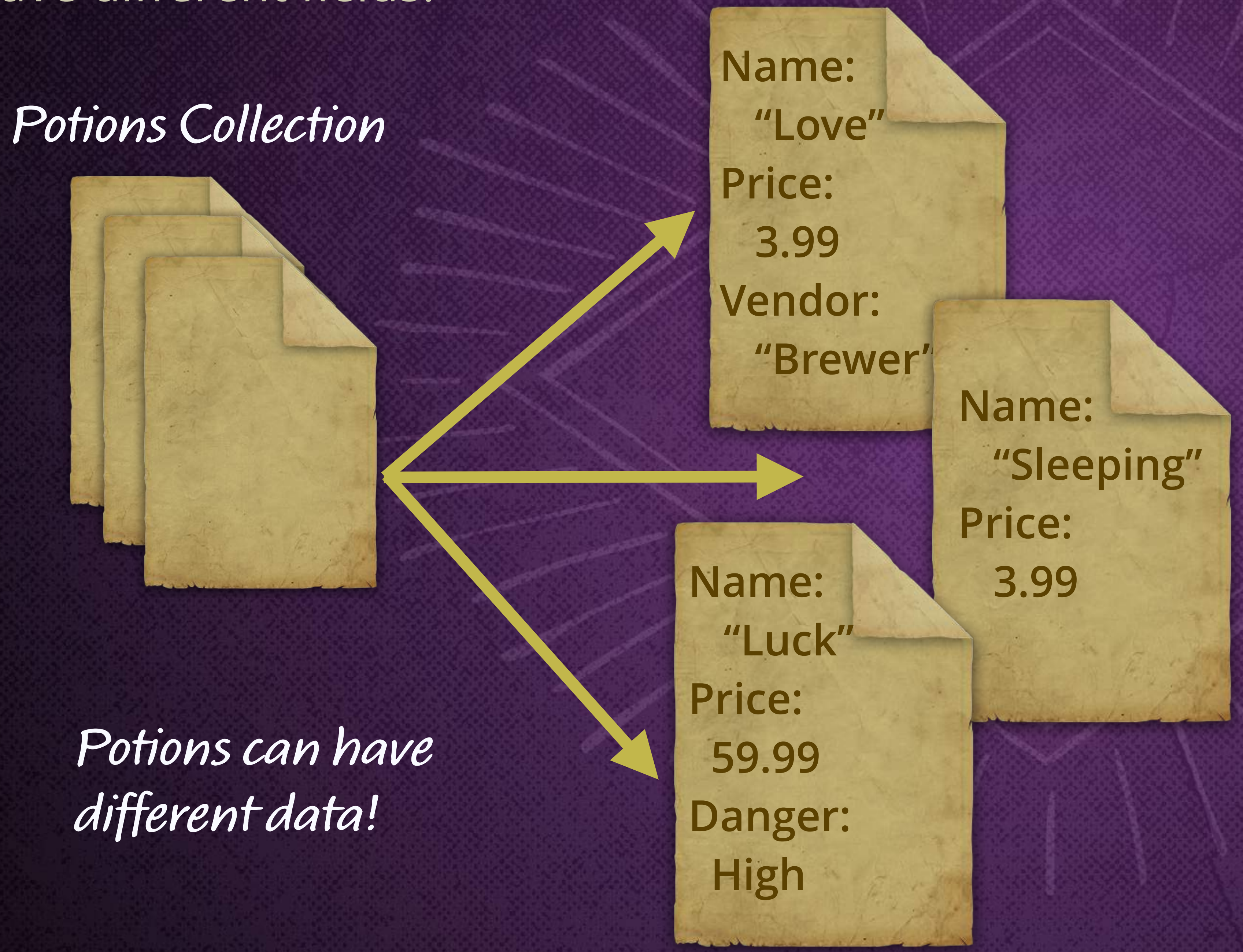
Potions Collection



All data is grouped within documents

Collections Group Documents

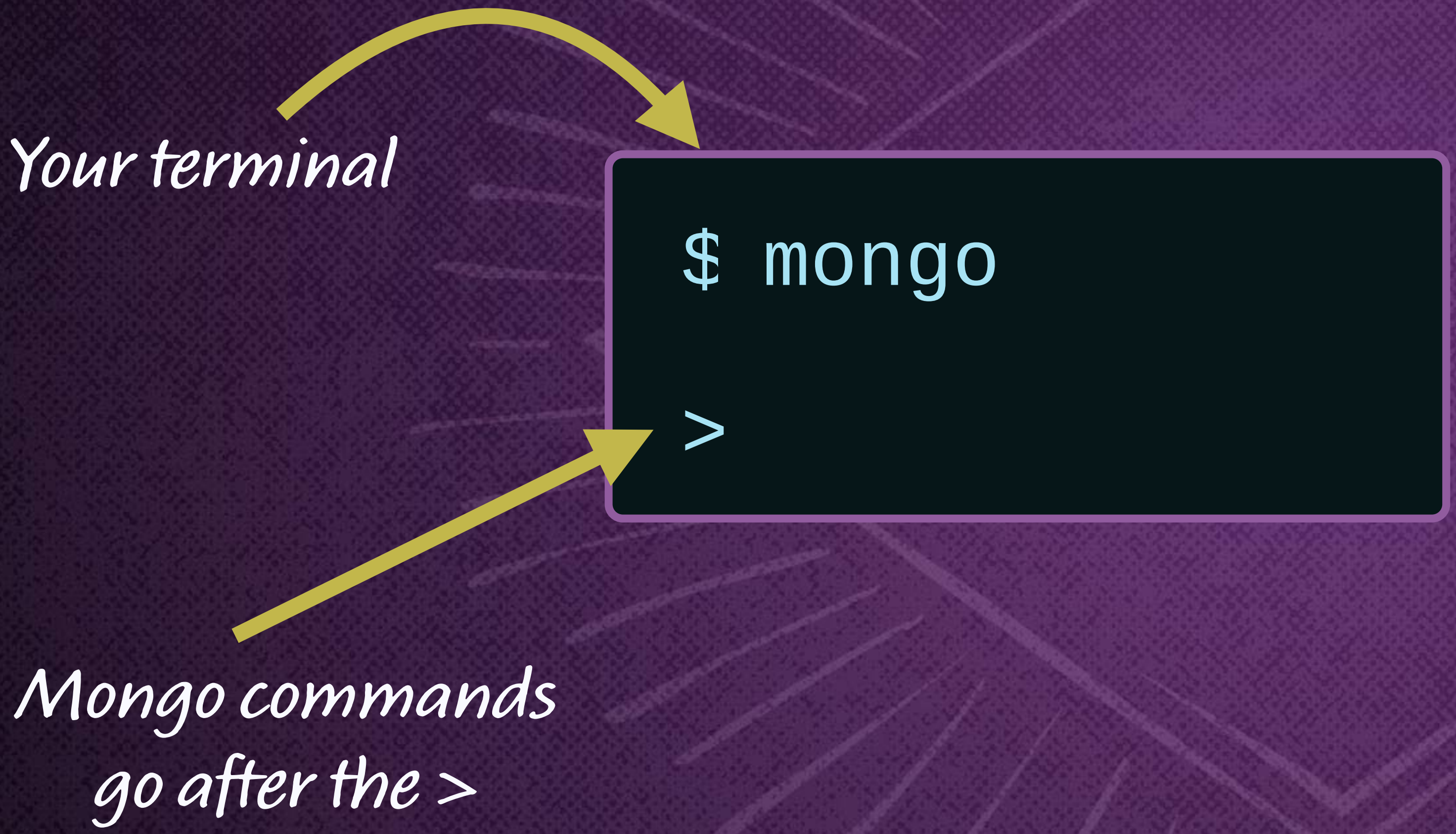
Collections are simply groups of documents. Since documents exist independently, they can have different fields.



This is referred to as a "dynamic schema."

Starting the Shell

We can access MongoDB through the terminal application. If you want try this out locally, follow the link below for MongoDB installation instructions.



```
$ mongo

>
```

Download MongoDB here:
http://go.codeschool.com/download-mongodb

How Do We Interact With MongoDB?

All instances of MongoDB come with a command line program we can use to interact with our database using JavaScript.

SHELL

Regular JavaScript variable assignment

```
> var potion = {
    "name": "Invisibility",
    "vendor": "Kettlecooked"
  }
```

Access the variable to see the contents

```
> potion
```

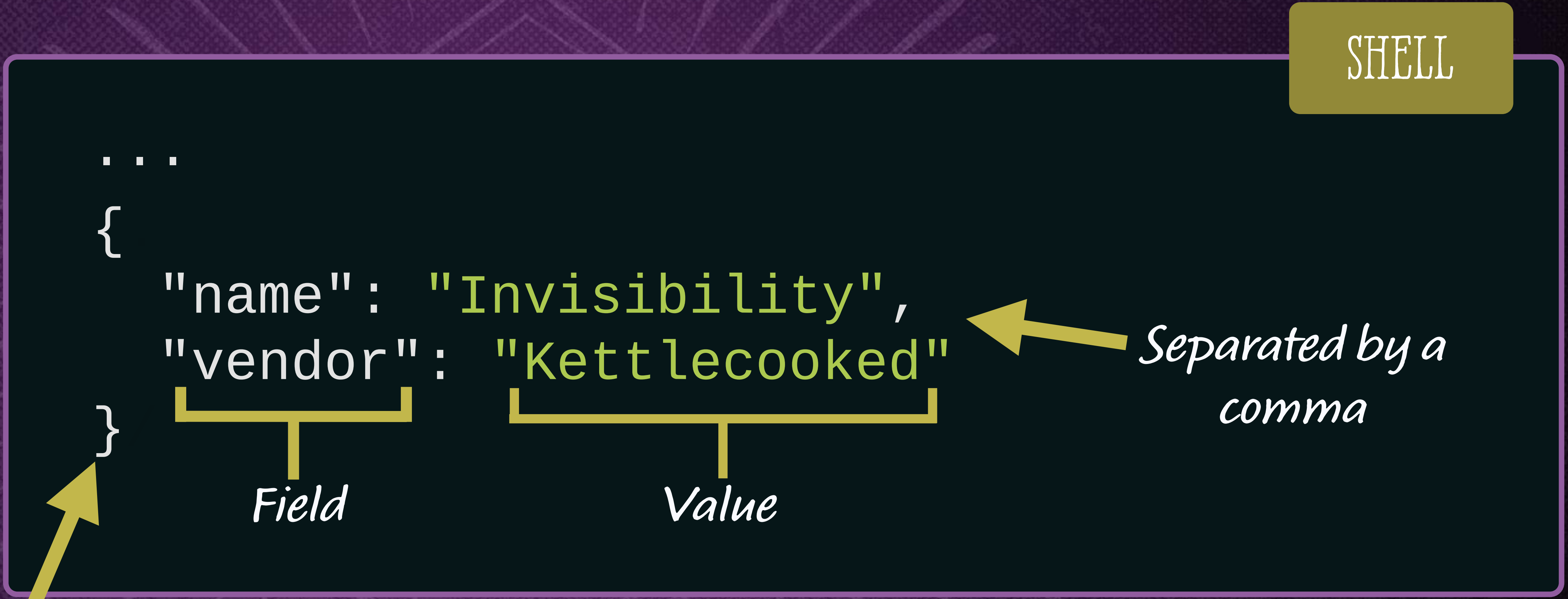
Get a response back

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked"
}
```

This is all just normal JavaScript!

Documents Are Just JSON-like Objects

Here's what a simple document looks like.



```
...
{
  "name": "Invisibility",
  "vendor": "Kettlecooked"
}
```

Using the Shell

MongoDB comes with helper methods to make it easy to interact with the database.

Switches to use the database and creates it if doesn't exist when we write to it

SHELL

```
> use reviews
switched to db reviews
> db
reviews
```

Returns the current database name

Show list of commands

SHELL

```
> help
db.help()            ...
...
show dbs             ...
> show dbs
test            0.078GB
reviews         0.078GB
```

Show list of databases

Name and size of databases

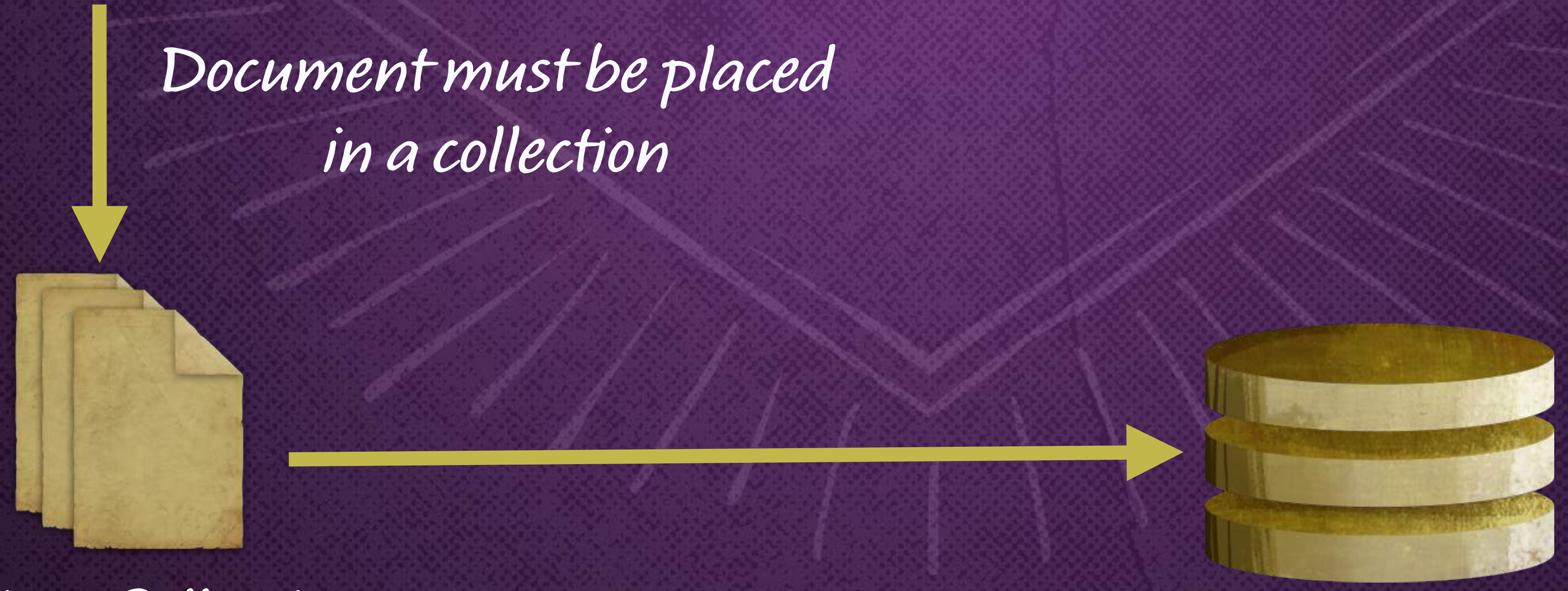
THE MAGICAL MARVELS OF MONGODB

Documents Need to Be Stored in Collections

Documents are always stored in collections within a database.

Potion Document

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked"
}
```



Potions Collection

Inserting a Document Into a Collection

We can use the ***insert()*** collection method to save a potion document to the potions collection.

This collection doesn't exist yet, so it will automatically be created

To write to the database, we specify the collection and the operation to perform

SHELL

```
> db.potions.insert(
  {
    "name": "Invisibility",
    "vendor": "Kettlecooked"
  }
)

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

Potion document as a parameter of the insert method

What's a WriteResult?

Whenever we write to the database, we'll always be returned a WriteResult object that tells us if the operation was successful or not.

SHELL

```
> db.potions.insert(
  {
    "name": "Invisibility",
    "vendor": "Kettlecooked"
  }
)
WriteResult({ "nInserted": 1 })
```

1 document successfully inserted

Finding All Potions

We can use the ***find()*** collection method to retrieve the potion from the inventory collection.

All collection methods must end with parentheses

SHELL

```
> db.potions.find()
 {
   "_id": ObjectId("559f07d741894edebdd8aa6d"),
   "name": "Invisibility",
   "vendor": "Kettlecooked"
 }
```

Unique id that gets automatically generated

Using Find to Return All Documents in a Collection

SHELL

```
> db.potions.insert(...)
  WriteResult({ "nInserted": 1 })
> db.potions.insert(...)
  WriteResult({ "nInserted": 1 })

> db.potions.find()
 { "name": "Invisibility" ... }
 { "name": "Love" ... }
 { "name": "Shrinking" ... }
```

Let's add 2 more potions

Now find returns a total of 3 potions

Conjuring MongoDB

Level 1 – Section 2

Queries and Data Types

ObjectIds Make Documents Unique

Every document is required to have a unique ***_id*** field. If we don't specify one when inserting a document, MongoDB will generate one using the ObjectId data type.

SHELL

```
> db.potions.find()
{
  "_id": ObjectId("559f07d741894edebdd8aa6d"),
  "name": "Invisibility",
  "vendor": "Kettlecooked"
}
```

It's common to let MongoDB handle _id generation.

THE MAGICAL MARVELS OF MONGODB

Finding a Specific Potion With a Query

We can perform a query of equality by specifying a field to query and the value we'd like.

Queries are field/value pairs

SHELL

```
> db.potions.find({"name": "Invisibility"})
{
  "_id": ObjectId("559f07d741894edebdd8aa6d"),
  "name": "Invisibility",
  "vendor": "Kettlecooked"
}
```

Queries will return all the fields of matching documents

The Magical Marvels of MongoDB

Queries That Return Multiple Values

More than 1 document matches the query

SHELL

```
> db.potions.find({"vendor": "Kettlecooked"})
{
  "_id": ObjectId("55d232a5819aa726..."),
  "name": "Invisibility",
  "vendor": "Kettlecooked"
}
{
  "_id": ObjectId("55c3b9501aad0cb0..."),
  "name": "Shrinking",
  "vendor": "Kettlecooked"
}
```

Two separate documents are returned

Queries are case sensitive.

THE MAGICAL MARVELS OF MONGODB

What Else Can We Store?

Documents are persisted in a format called BSON.

BSON is like JSON, so you can store:

Strings	`"Invisibility"`
Numbers	`1400` `3.14`
Booleans	`true` `false`
Arrays	`["newt toes", "pickles"]`
Objects	`{"type" : "potion"}`
Null	`null`

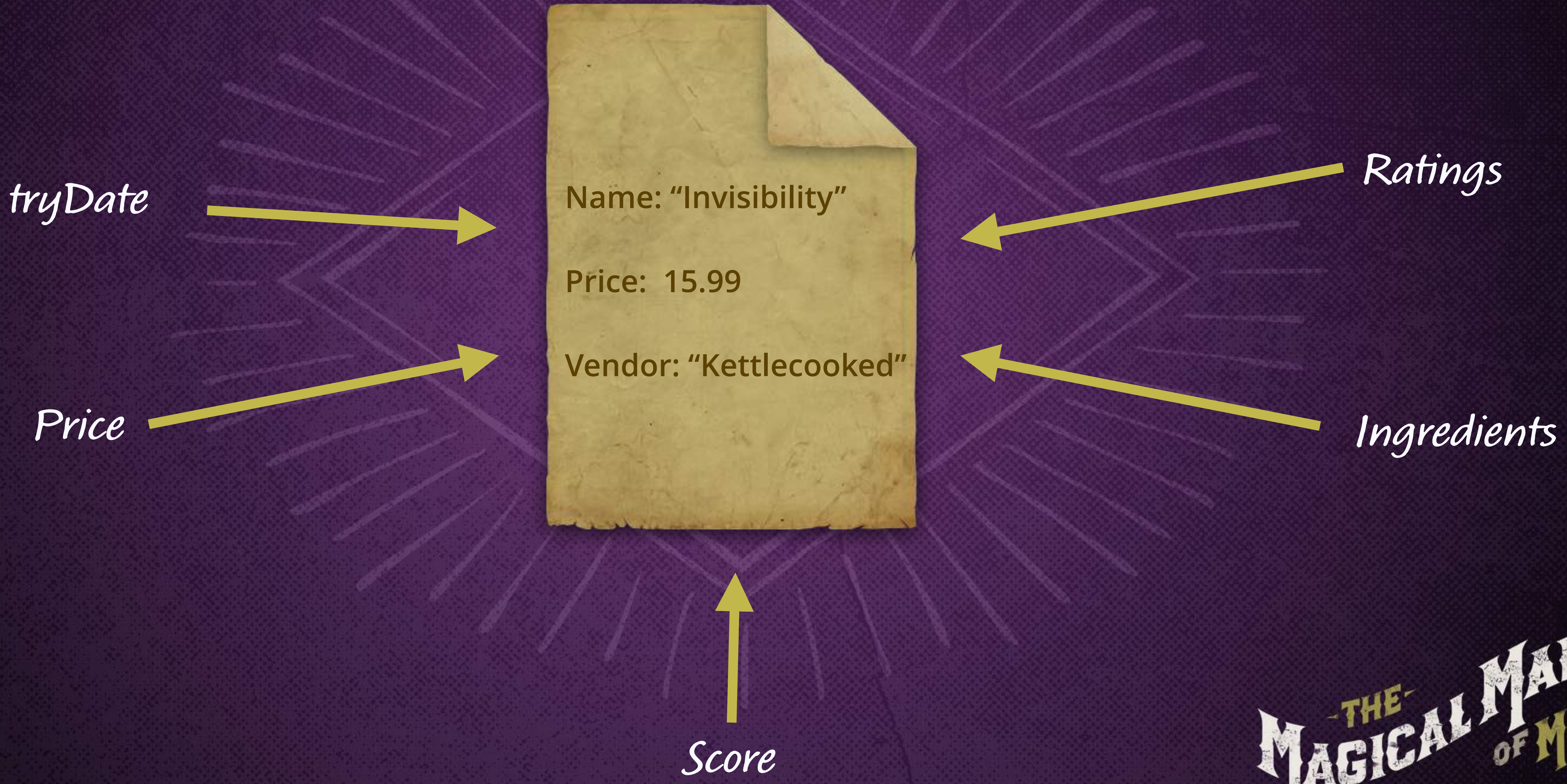
BSON comes with some extras.

ObjectID	`ObjectId(...)`
Date	`ISODate(...)`

Learn more at:
http://go.codeschool.com/bson-spec

Building Out Our Potions

Now that we have a better grasp on documents, let's build out our potion document with all the necessary details.



Adding Price and Score

We can store both integers and floats in a document.

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99,
  "score": 59
}
```

MongoDB will preserve the precision of both floats and integers

Adding a tryDate

Dates can be added using the JavaScript Date object and get saved in the database as an ISODate object.

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99,
  "score": 59,
  "tryDate": new Date(2012, 8, 13)
}
```

Reads as September 13, 2012, since JavaScript months begin at 0

Dates get converted to an ISO format when saved to the database

```
"expiration": ISODate("2012-09-13T04:00:00Z")
```

Adding a List of Ingredients

Arrays are a great option for storing lists of data.

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99,
  "score": 59,
  "tryDate": new Date(2012, 8, 13),
  "ingredients": ["newt toes", 42, "laughter"]
}
```

We can store any data type within an array

THE MAGICAL MARVELS OF MONGODB

Adding a Potion's Ratings

Each potion has 2 different ratings, which are scores based on a scale of 1 to 5.

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99,
  "score": 59,
  ...
}
```

Each rating will have 2 fields

```
{
  "strength": 2,
  "flavor": 5
}
```

MongoDB supports embedded documents so we can simply add this to our potion document

Embedded Documents

We embed documents simply by adding the document as a value for a given field.

```
{
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99,
  "score": 59,
  "tryDate": new Date(2012, 8, 13),
  "ingredients": ["newt toes", 42, "laughter"],
  "ratings": {"strength": 2, "flavor": 5}
}
```

An embedded document doesn't require an id since it's a child of the main document

Inserting Our New Potion

We've cleared out the inventory collection — now let's add our newly constructed potion!

SHELL

```
> db.potions.insert(
 {
   "name": "Invisibility",
   "vendor": "Kettlecooked",
   "price": 10.99,
   "score": 59,
   "tryDate": new Date(2012, 8, 13),
   "ingredients": ["newt toes", 42, "laughter"],
   "ratings": {"strength": 2, "flavor": 5}
 }
)
WriteResult({ "nInserted": 1 })
```

Document successfully inserted!

Finding Potions by Ingredients

Array values are treated individually, which means we can query them by specifying the field of the array and the value we'd like to find.

Same format as basic query for equality

SHELL

```
> db.potions.find({"ingredients": "laughter"})
 {
   "_id": "ObjectId(...)",
   "name": "Invisibility",
   ...
   "ingredients": ["newt toes","secret", "laughter"]
 }
```

Potion contains the right ingredient

Finding a Potion Based on the Flavor

We can search for potions by their ratings using dot notation to specify the embedded field we'd like to search.

```
{
  "_id": "ObjectId(...)",
  "name": "Invisibility",
  ...
  "ratings": {"strength": 2, "flavor": 5}
}
```

"ratings.strength"

"ratings.flavor"

We can easily query embedded documents

```
db.potions.find({"ratings.flavor": 5})
```

What About Insert Validations?

If we were to insert a new potion but accidentally set the price value to a string, the potion would still get saved despite all other potions having integer values.

SHELL

```
> db.potions.insert({
    "name": "Invisibility",
    "vendor": "Kettlecooked",
    "price": "Ten dollars",
    "score": 59
})
WriteResult({ "nInserted": 1 })
```

Data we might consider to be invalid but MongoDB will think is fine

! *The document still got saved to the database!*

Validations Supported by MongoDB

MongoDB will only enforce a few rules, which means we'll need to make sure data is valid client-side before saving it.

```
{
  "_id": ObjectId("55c3b9561..."),
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99
}
{
  "_id": ObjectId("55d232a51..."),
  "name": "Shrinking",
  "vendor": "Kettlecooked",
  "price": 9.99
}
```

- ✓ *No other document shares same _id*
- ✓ *No syntax errors*
- ✓ *Document is less than 16mb*

THE MAGICAL MARVELS OF MONGODB

Validations Supported by MongoDB

MongoDB will only enforce a few rules, which means we'll need to make sure data is valid client-side before saving it.

```
{
  "_id": ObjectId("55c3b9561..."),
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99
,
{
  "_id": ObjectId("55d232a51..."),
  "name": "Shrinking",
  "vendor": "Kettlecooked",
  "price": 9.99
}
```

Missing end bracket

- ✓ *No other document shares same _id*
- ✓ *No syntax errors*
- ✓ *Document is less than 16mb*

Validations Supported by MongoDB

MongoDB will only enforce a few rules, which means we'll need to make sure data is valid client-side before saving it.

```
{
  "_id": 1,
  "name": "Invisibility",
  "vendor": "Kettlecooked",
  "price": 10.99
},
{
  "_id": 1,
  "name": "Shrinking",
  "vendor": "Kettlecooked",
  "price": 9.99
}
```

Duplicate _id

✓ *No other document shares same _id*

✓ *No syntax errors*

✓ *Document is less than 16mb*