The Magical Marvels of MongoDB

Morphing Models

Level 4 – Section 1

Data Modeling

Introducing User Profiles

We've added a user profile page to the site and want to allow users to enter up to 3 of their favorite potions.

User Document

```
{
    "_id": ObjectId(...),
    "username": "Azrius",
    "email": "azrius@example.com",
    "favorites": "Luck"
}
```

Need a way to store up to 3 potions

THE MAGICAL MARVELS OF MONGODB

Storing Favorites Within a User

Users and their favorites are strongly related and will be used together often.

User Document

```
{
    "_id": ObjectId(...),
    "username": "Azrius",
    "email": "azrius@example.com",
    "favorites": [
      "Newt Tonic",
      "Sleeping",
      "Love"
    ]
}
```

Use an array to hold multiple values

THE MAGICAL MARVELS OF MONGODB

Adding Vendor Information

We'd like to add more vendor information to our potions so users can be more informed about where they get their potions.

Vendor

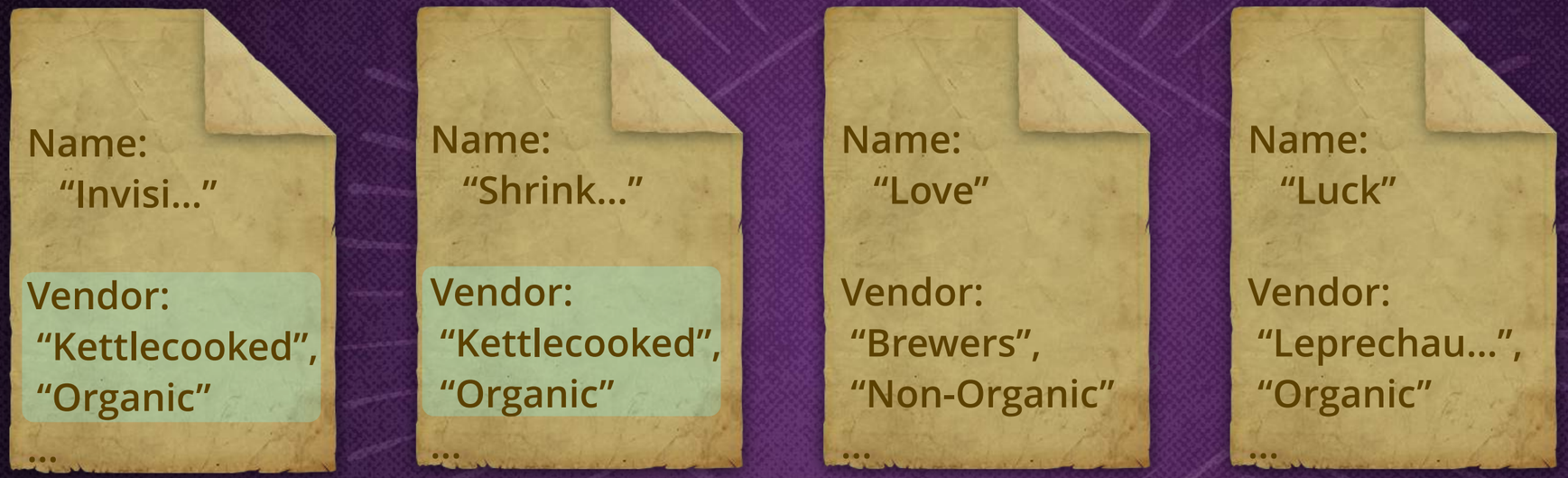
Inserting the data as an embedded document

Potion Document

```
{
  "_id": ObjectId(...),
  "name": "Invisibility",
  ...
  "vendor": {
    "name": "Kettlecooked",
    "phone": 5555555555,
    "organic": true
  }
}
```

Storing Vendor Information

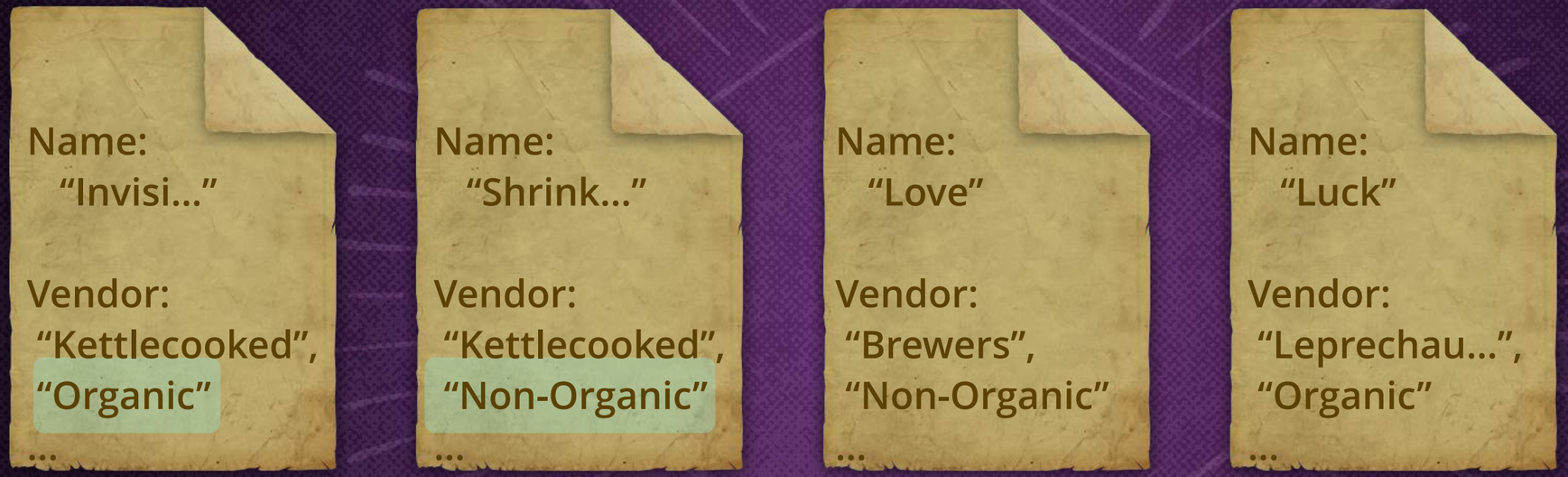
Each potion now contains its vendor information, which means we have vendor information repeated throughout our documents.



We're going to have duplicate vendor information for each potion.

Dangers of Duplication

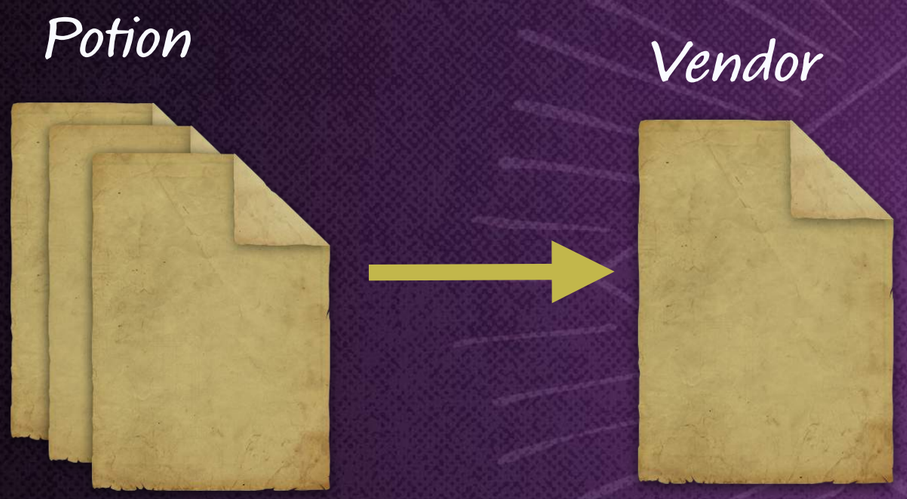
Duplicate data can be hard to keep consistent throughout the database.



If 1 potion gets updated with new information and the rest don't, our data is no longer correct

Referencing Vendor Information

Instead of embedding the vendor information, we can create a vendors collection and reference the vendor document in each potion document.



Vendor names are unique and don't change

Potion

```
{
  "_id": ObjectId(...),
  "name": "Invisibility",
  "vendor_id": "Kettlecooked",
  ...
}
```

Vendor

```
{
  "_id": "Kettlecooked",
  "phone": 5555555555,
  "organic": true
}
```

Inserting Referenced Documents

SHELL

```
> db.vendors.insert({
    "_id": "Kettlecooked",
    "phone": 5555555555,
    "organic": true
})
```

We can specify the unique _id of document

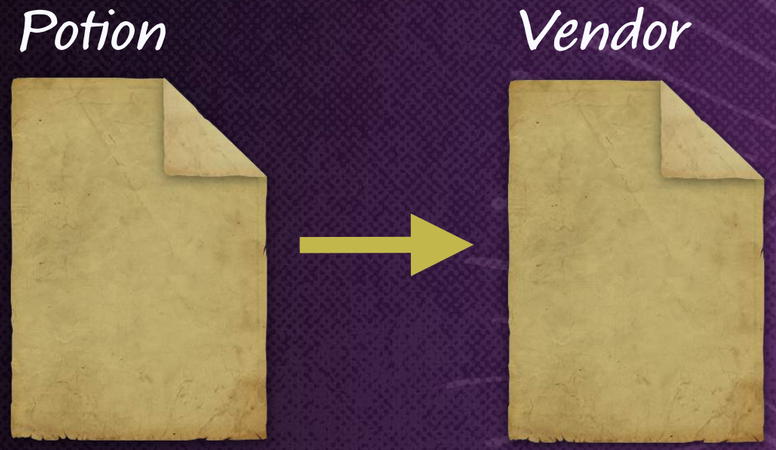
SHELL

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

Referenced document

Querying a Referenced Document

In order to pull a potion document and the vendor information, we must first query for the potion to get the vendor_id and then query once more to get their vendor information.




```
{
  "_id": ObjectId(...),
  "name": "Invisibility",
  "vendor_id": "Kettlecooked",
  ...
}
```

First, query to retrieve potion information

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

Second, query to retrieve vendor information

```
db.vendors.find({"_id": "Kettlecooked"})
```

We get the vendor _id from the first query

Some Features of Embedded Documents

With a single query, we can grab a user's email and their addresses

Data readily available

SHELL

```
db.users.find({},{"email": true, "favorites": true})
```

```
{
    "_id": ObjectId(...),
    "email": "azrius@example.com"
    "favorites": [
      "Newt Tonic",
      "Sleeping",
      "Love"
     ]
}
```

More Features of Embedded Documents

Data readily available

SHELL

```
db.users.find({},{"email": true, "favorites": true})
```

Atomic write operations

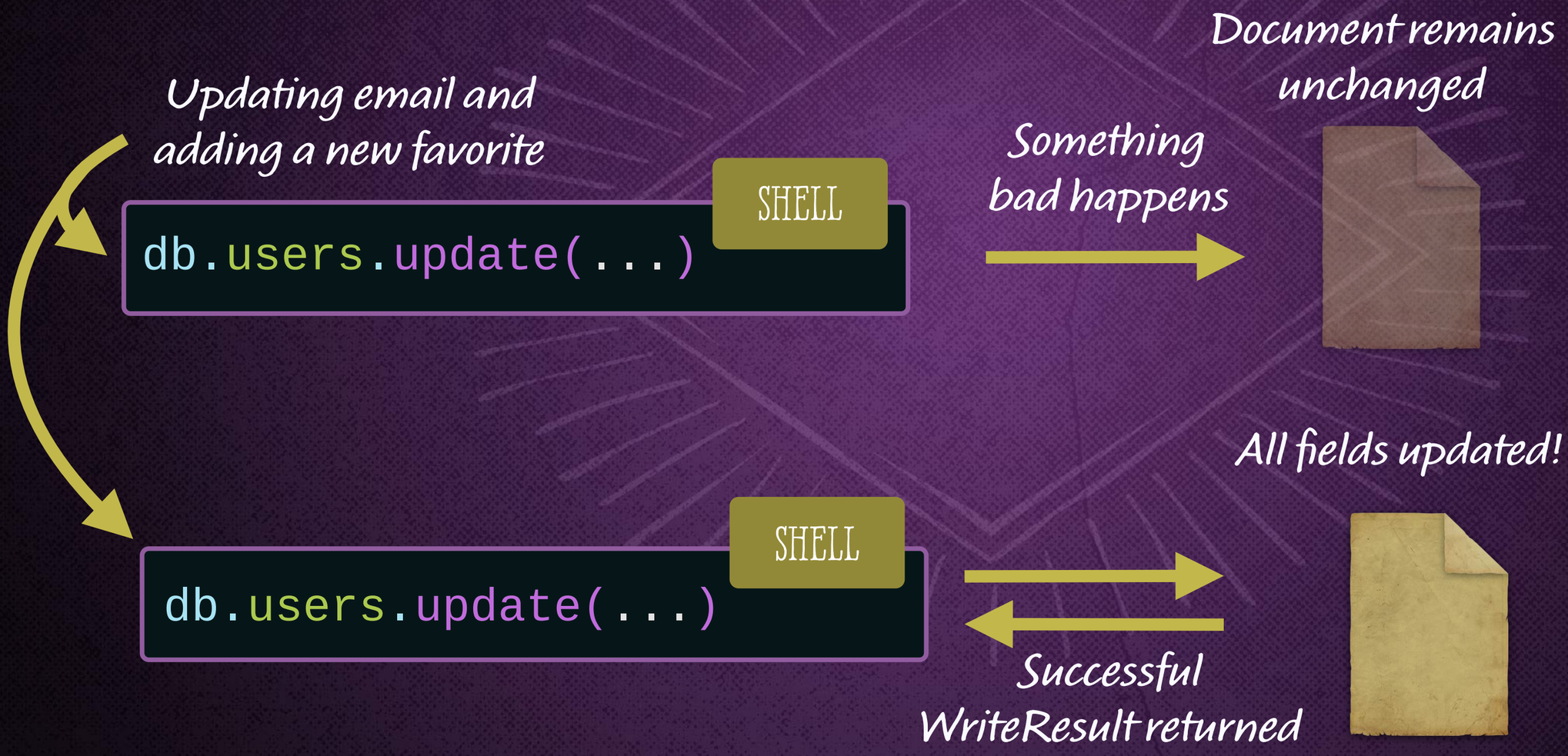
SHELL

```
db.users.insert({
  "username": "Azrius",
  "email": "azrius@example.com",
  "favorites": ["Newt...", "Sleeping", "Love"]
})
```

Guarantee that the document write completely happens or doesn't at all

Why Does Atomicity Matter?

If we update a user's email and add a favorite potion, but an error occurs in the favorites portion of the update, then none of the update will occur.



Referenced Documents Exist Independently

Since we've referenced vendor documents, they now exist entirely on their own outside of potion documents.

One place to edit vendor information

```
db.vendors.update({"_id": ObjectId(...)},{...})
```

Multi-document writes not supported

New Potion

SHELL

```
db.potions.insert({...})
```

New Potion's Vendor

SHELL

```
db.vendors.insert({...})
```

No guarantee that both write operations will occur

Multi-document Write Operations

Currently, MongoDB doesn't support multi-document writes. We aren't able to guarantee write operations on an atomic level.

Adding new potion

```
db.potions.insert(...)
```

SHELL

Successful write

New potion created!

Adding the new vendor

```
db.vendors.insert(...)
```

SHELL

Error occurs

No vendor document created

Dangers of Not Having Transactions

MongoDB doesn't recognize document relationships, so we now have a potion with a nonexistent vendor.

New potion created!

No vendor document created

Vendor never got created!

Potion document

```
{
  "_id": ObjectId(...),
  "name": "Time Travel",
  "vendor_id": "Magical Inc.",
  ...
}
```

The references still exist!

Morphing Models

Level 4 – Section 2

Data Modeling Decisions

Choosing Which Route to Take

When deciding how to model a one-to-many relationship, we must carefully consider how our data will be used in order to decide between embedding and referencing.

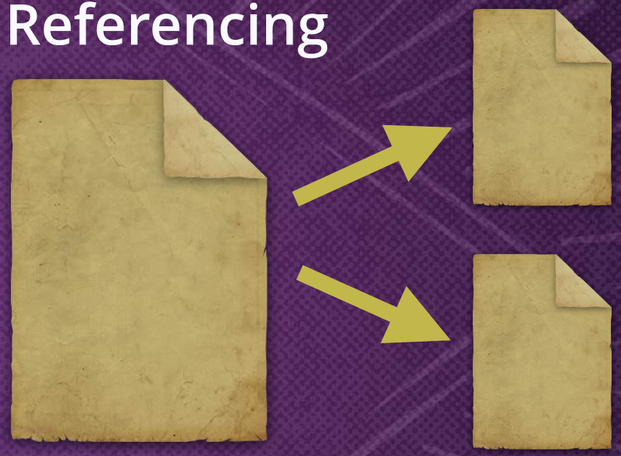
Embedding

Single query

Documents accessed through parent

Atomic writes

Referencing



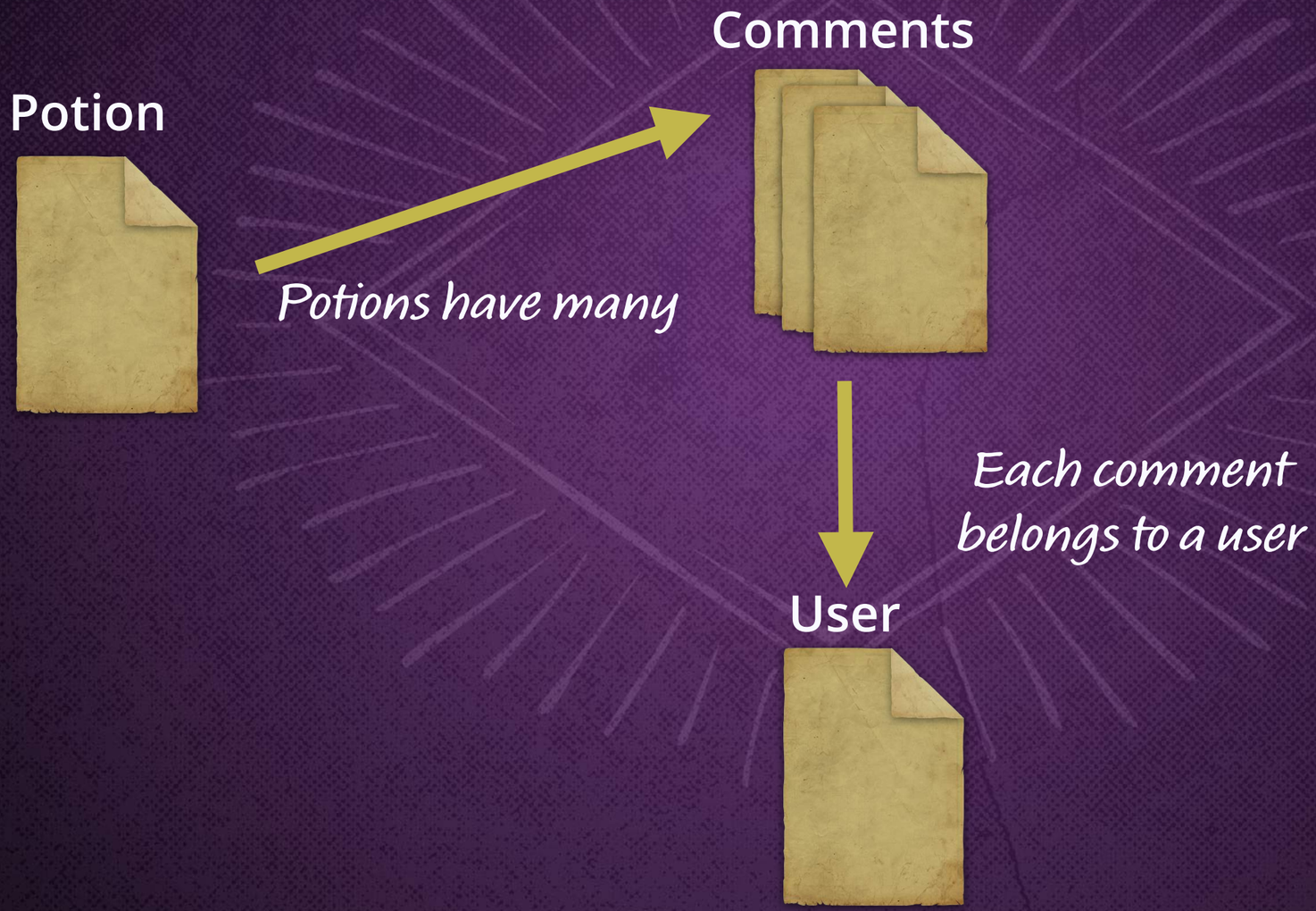
Requires 2 queries

Documents exist independently

Doesn't support multi-document writes

Adding Comments to Potions

We'd like to allow users to comment on the potions and need to determine the best route.



First Question: How Will the Data Be Used?

Data that is frequently used together will benefit from being embedded while data that's rarely used can afford the cost of referencing.

How often is the data used together?	Always	Sometimes	Rarely
Embed	✓	✓	✓
Reference		✓	✓

Embedding will work most of the time

Either option can work effectively

Data Usage: Potion Comments

Embedding

Referencing

We would be forced to perform multiple queries

Whenever we display potions, we'll always want to display comments

Username displayed for each comment

THE MAGICAL MARVELS OF MONGODB

Second Question: What's the Size of the Data?

The size of the data in a relationship has a significant impact on data modeling. It's crucial to think ahead!

Expected Size	Less than 100	More than a few hundred	Thousands
Embed	✓	✓	
Reference		✓	✓

Might start to see a decline in read performance when embedded

Data Size: Potion Reviews

Embedding

Referencing

When the data size is over 100, we can consider referencing

Most potions won't get more than 50 comments, and each comment has a single author

THE MAGICAL MARVELS OF MONGODB

Third Question: Will the Data Change Often?

Sometimes embedding data can lead to data duplication. Depending on whether or not the duplicate data changes a lot can factor into our decision making.

Frequency of Change	Never/Rarely	Occasionally	Constantly
Embed	✓	✓	
Reference		✓	✓

Prevents inconsistencies from duplication

Data duplication is okay if we don't expect change

Depends on whether or not we want the overhead of managing duplication

Data Change: Potion Comments

Embedding

Referencing

Requires updating all of a user's comments if their information changes

Embedding

Referencing

Embedding Comments in Potions

We can confidently embed comments within potions. We know we'll have less than 100 comments per potion, they're used together often, and the data doesn't change often.

```
{
   "name": "Invisibility",
   ...
   "comments":[
     {
      "title": "The best potion!",
      "body": "Lorem ipsum abra cadrbra"
     },
     ...
   ]
}
```

Referencing Users in Comments

We only need the username from each user, and embedding user documents could lead to duplication issues. Let's reference a user by using his or her unique username.

```
{
    "name": "Invisibility",
    ...
    "comments":[
      {
       "title": "The best potion!",
       "body": "Lorem ipsum abra cadrbra",
       "user_id": "Azrius"
      },
      ...
    ]
}
```

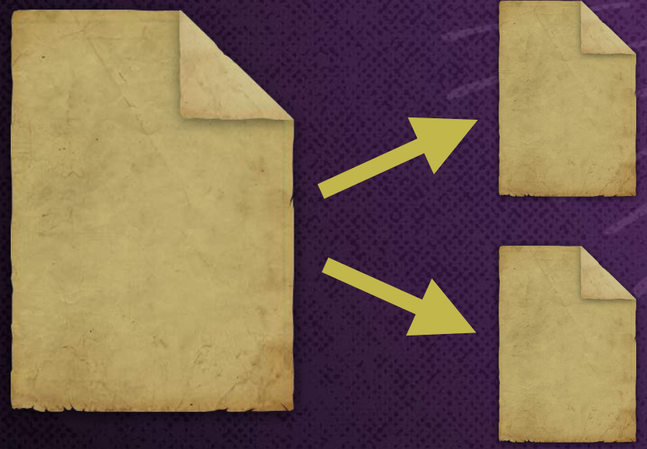
Usernames can't be changed and are unique

Data Modeling Guidelines and Takeaways

Generally, embedding is the best starting point

Focus on how your data will be used

If you find you need complex references, consider a relational database



Reference data when you need to access document independently

Consider referencing when you have large data sizes

THE MAGICAL MARVELS OF MONGODB