

Data model

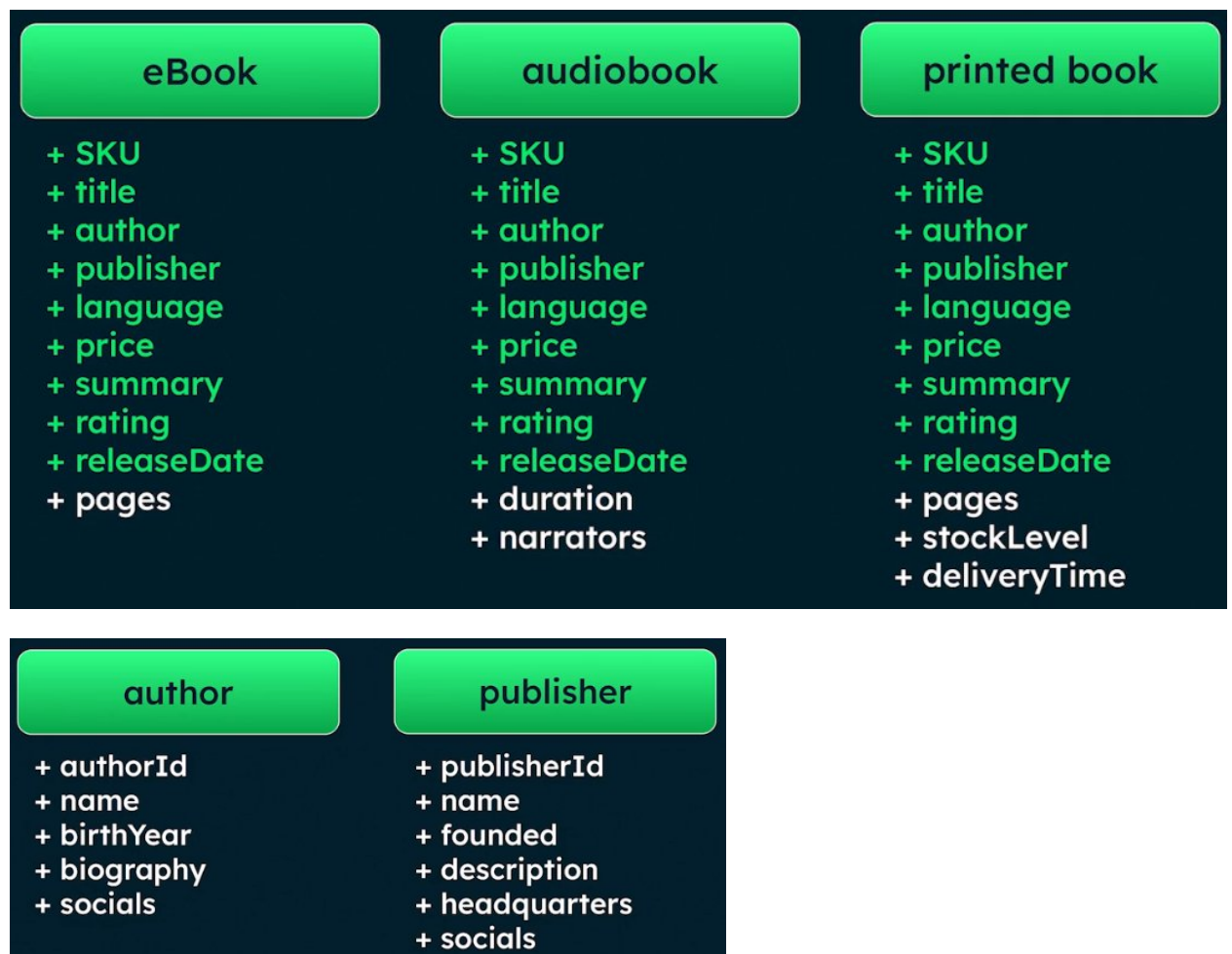
MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called BSON (Binary JSON).

Creating a data model in MongoDB involves understanding your application's data requirements, designing the structure of your collections and documents, and deciding on a normalized or denormalized approach. By following these steps, you can create a flexible and efficient data model tailored to your application's needs.

Entities

Identify

A MongoDB entity refers to a single document within a collection in a MongoDB database. It is the fundamental unit of data storage in MongoDB, similar to a row in a relational database, but with a flexible, schema-less structure that allows for a diverse set of field types and nested data.



users	reviews
+ userId	+ productSKU
+ name	+ userId
+ email	+ date
+ phone	+ stars
+ address	+ reviewTitle
+ memberSince	+ reviewBody

Number of documents

The number of documents in a MongoDB collection is a critical factor that influences the database's performance, resource usage, scalability, and management. Proper data modeling, indexing, sharding, and maintenance strategies are essential to handle large collections effectively and ensure the database performs well under various workloads.

Entity	Quantity
eBooks	450,000
Audiobooks	200,000
Printed books	50,000
Authors	20,000
Publishers	500
Users	25,000,000
Reviews	1,000,000,000

Type of access

The access type in MongoDB is important because it dictates how data is read from and written to the database, impacting performance, security, and data integrity. Different applications have varying requirements for read and write operations, and understanding the access patterns helps in optimizing the database design and configuration.

Entities	Operations	Information Needed	Type
Books*	Fetch book details	Book details + rating	Read
Reviews	Fetch 10 reviews for a book	Reviews + reviewer rating	Read
Users	Fetch user details	User details	Read
Authors, Books*	Fetch an author and their books	Book titles + author details	Read
Books*	Add/update book	Book details + stock level	Write
Printed Books	Sell copy of printed book	Stock level	Write
Users	Add/update user	User details	Write
Reviews	Add review	Review + book rating	Write
Printed Books	Fetch printed book titles where the stock level has fallen below 50	Book details + stock level	Read

Access rate

The access type in MongoDB is critical as it influences the database's performance, security, scalability, and resource management. Optimizing for specific access patterns—whether read-heavy, write-heavy, or balanced—ensures that the database performs efficiently and meets the application's requirements. Proper indexing, sharding, write and read concern levels, and access control are all tailored based on the access type to achieve the best results.

Entities	Operation	Information Needed	Type	Rate
Books*	Fetch book details	Book details + rating	Read	1000/sec
Authors, Books*	Fetch an author and their books	Book titles + author details	Read	50/sec
Print Books	Fetch printed book titles where the stock level has fallen below 50	Book details + stock level	Read	2/day
Books*	Add/update book	Book details + stock level	Write	10/hour
Print Books	Sell copy of printed book	Stock level	Write	5/sec
Reviews	Fetch 10 reviews for a book	Reviews + reviewer rating	Read	200/sec
Reviews	Add review	Review + book rating	Write	50/sec
Users	Fetch user details	User details	Read	5/min
Users	Add/update user	User details	Write	1/sec

Document

Data that is accessed together should be stored together

The principle "data that is accessed together should be stored together" is a fundamental guideline in database design, particularly relevant to document-oriented databases like MongoDB. This principle helps optimize performance, simplify data retrieval, and enhance the overall efficiency of your application.

Embedding

Document embedding in MongoDB is a technique where related data is stored within a single document by nesting documents or arrays of documents inside a parent document. This approach leverages MongoDB's flexible schema design and is particularly useful for representing one-to-one and one-to-many relationships.

Key Concepts of Document Embedding

Single Document Storage

Related data is stored together in one document, reducing the need for joins or multiple queries.

Nested Structures

Documents can contain nested documents and arrays, allowing for complex, hierarchical data models.

Improved Read Performance

Embedding can improve read performance by allowing the retrieval of all related data in a single query.

```
{
  "_id": "book0003",
  "title": "MongoDB Data Modeling and Schema Design",
  "authors": [
    {
      "author_id": "author0029",
      "name": "Daniel Coupal"
    },
    {
      "author_id": "author0073",
      "name": "Pascal Desmarets"
    },
    {
      "author_id": "author0045",
      "name": "Steve Hoberman"
    }
  ]
}
```

Referencing

Document referencing in MongoDB is a technique where related data is stored in separate documents, and relationships between these documents are established using references. Instead of embedding related data within a single document, you store related data in different collections and use references (usually by storing the `_id` of one document in another) to link them. This approach is similar to foreign keys in relational databases.

Reasons

Data Reuse

When the same piece of data is used in multiple places, referencing avoids data duplication.

Example: Products in an e-commerce application, where the same product can be part of multiple orders.

Large or Growing Data

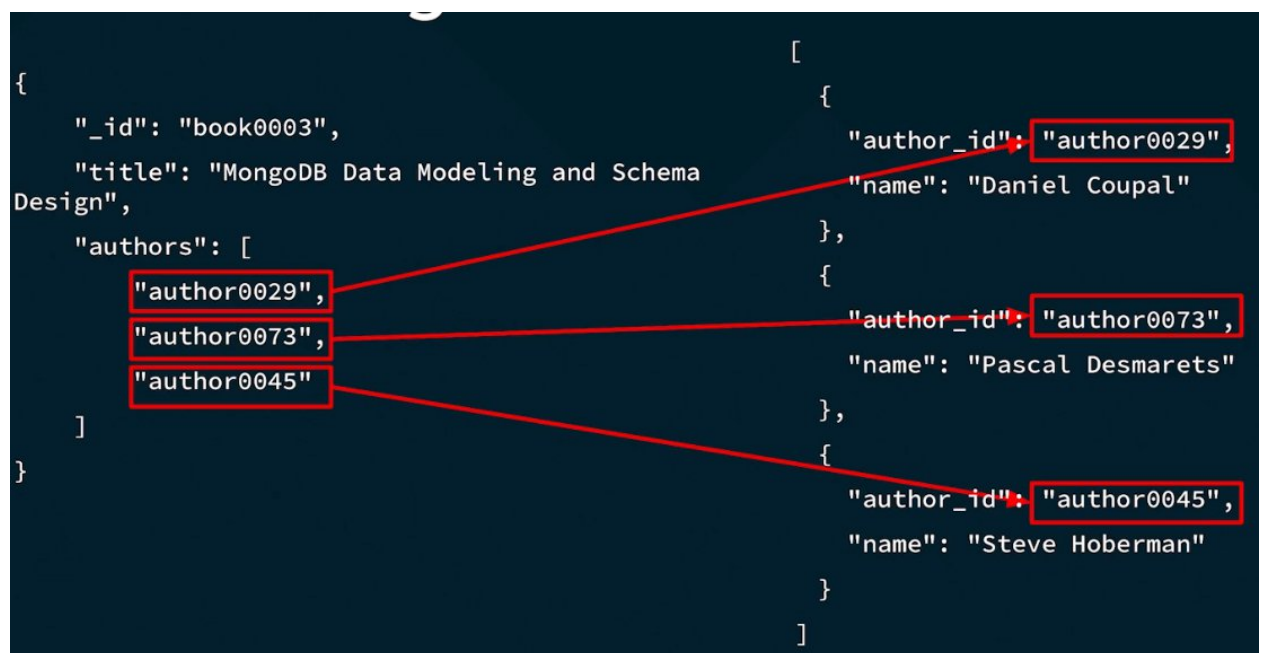
When related data can grow indefinitely or is too large to fit comfortably within a single document.

Example: Comments on a blog post that can accumulate over time.

Complex Relationships

When dealing with many-to-many relationships or when related data is frequently updated independently.

Example: Students and courses in an educational application, where students can enroll in multiple courses and courses can have multiple students.



Advantages of Document Referencing

Avoids Data Duplication

By storing a reference instead of duplicating data, you save storage space and ensure consistency across documents.

Scalability

Documents remain manageable in size, making it easier to work with large datasets and adhere to MongoDB's 16MB document size limit.

Independent Updates

Related data can be updated independently. For example, updating a product's price does not require updating every order that includes that product.

Disadvantages of Document Referencing

Increased Read Complexity

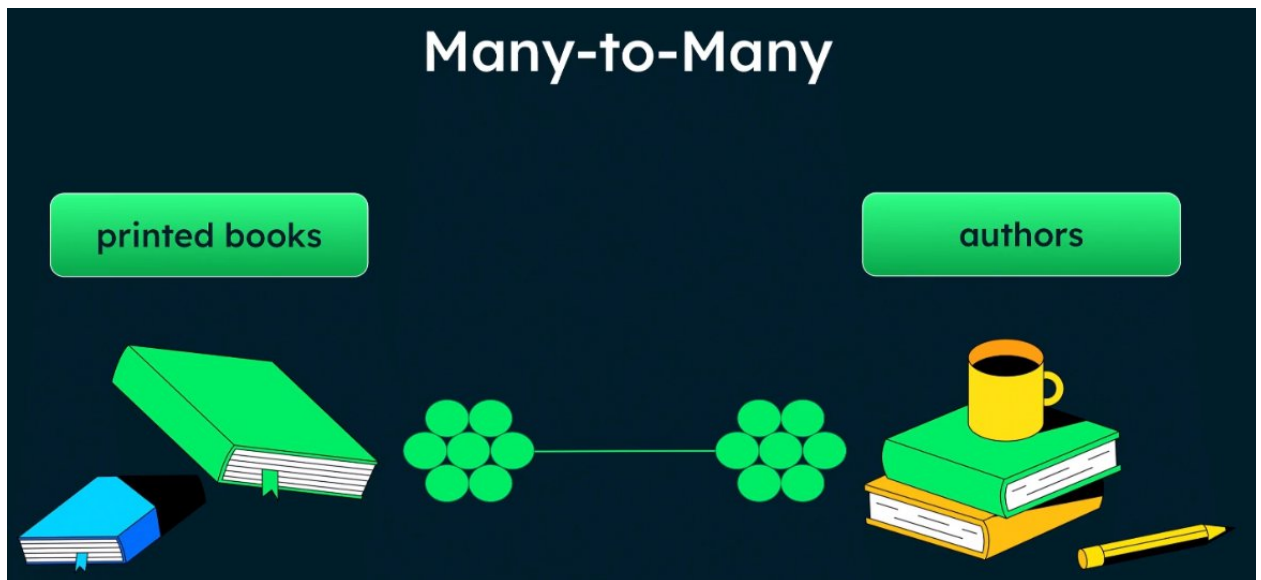
Fetching data requires multiple queries or joins, which can complicate queries and potentially slow down read operations.

Joins in Application Code

Unlike SQL databases, MongoDB does not support traditional joins. The application code needs to handle the logic to fetch and assemble related documents, which can add complexity.

Sample

Assume we have the following many-to-many relation:



Simplicity

Simplicity

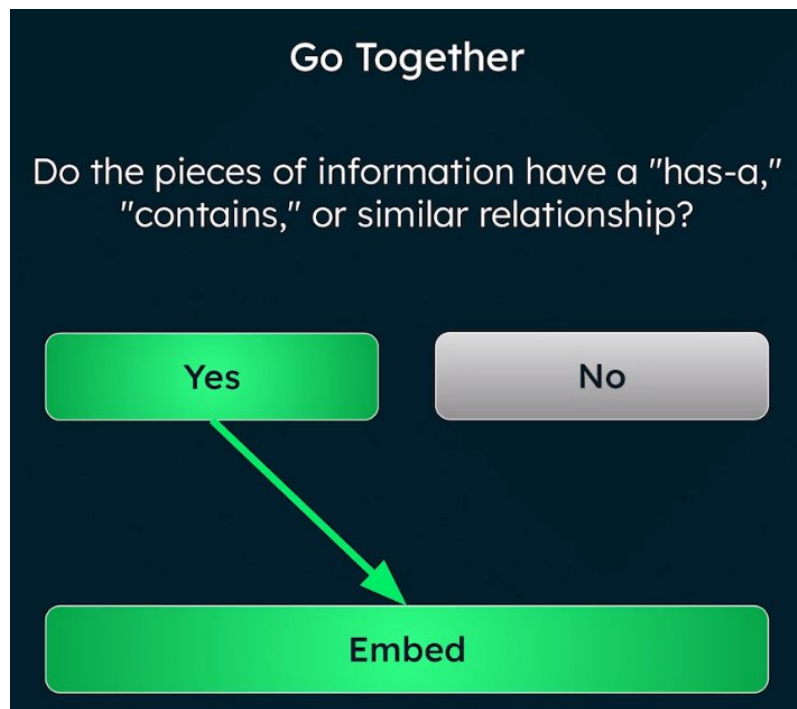
Would keeping the pieces of information together lead to a simpler data model and code?

Yes **No**

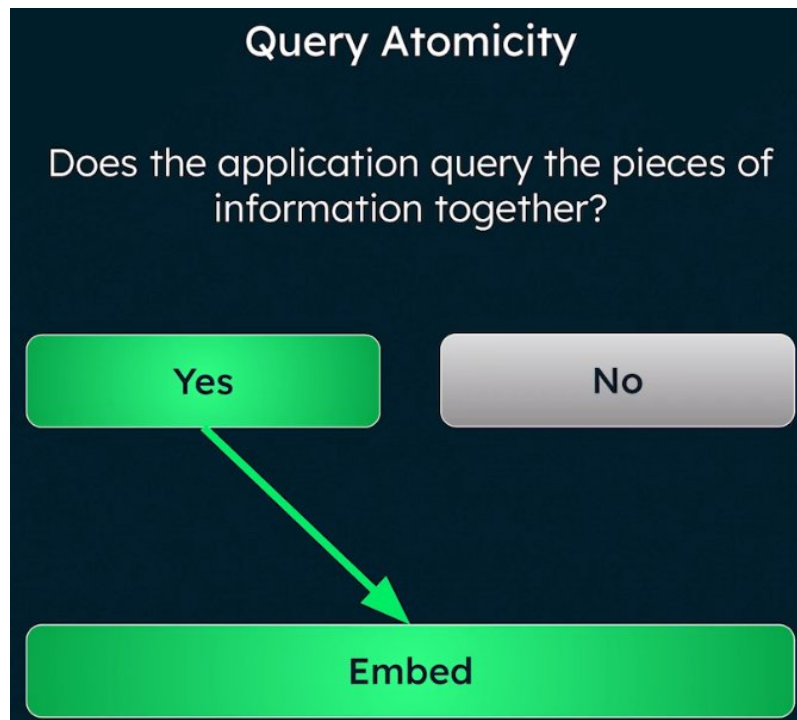
Embed

```
graph TD; Q[Would keeping the pieces of information together lead to a simpler data model and code?]; Q -- Yes --> E[Embed]; Q -- No --> Exit[ ];
```

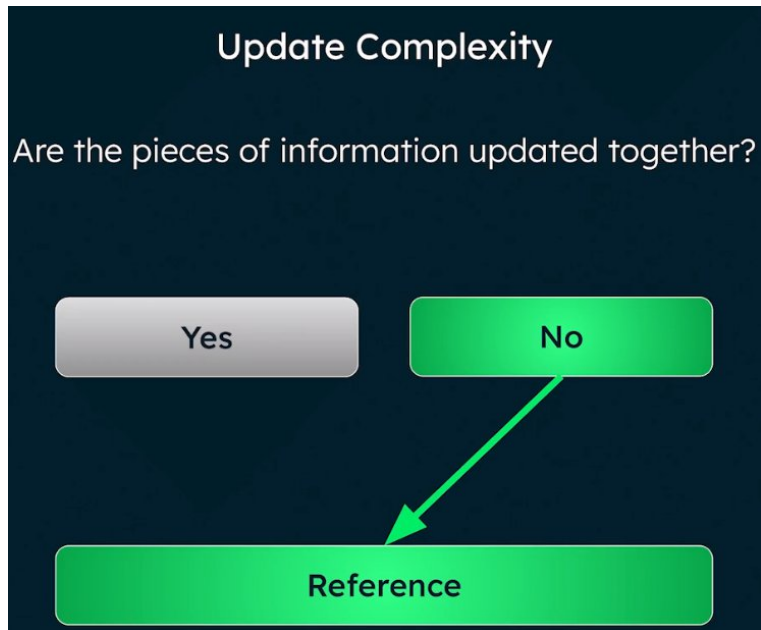

Go Together



Query Atomicity

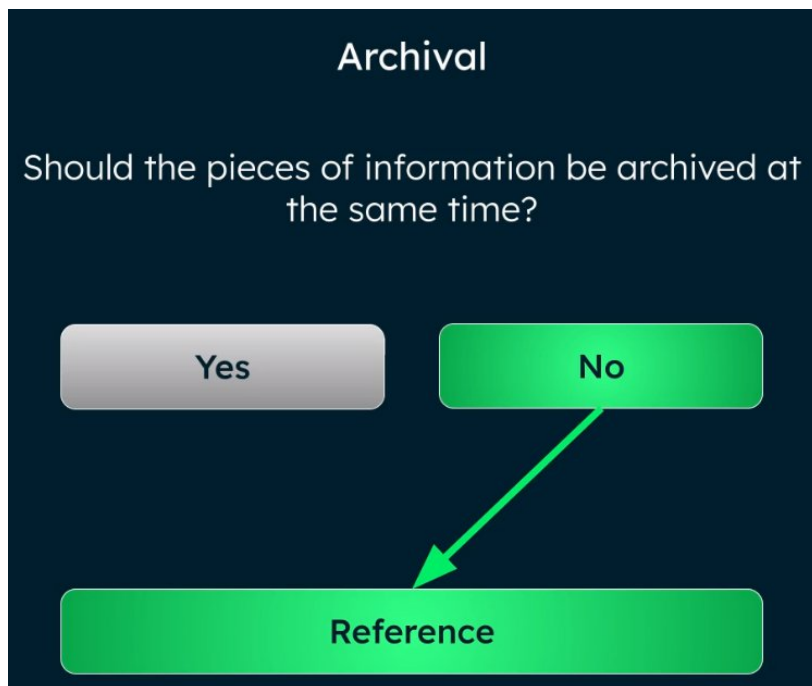


Update Complexity

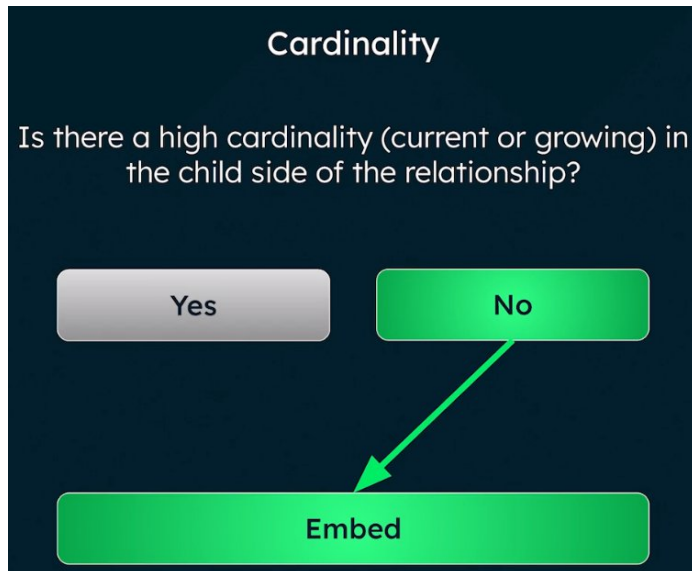


Archival

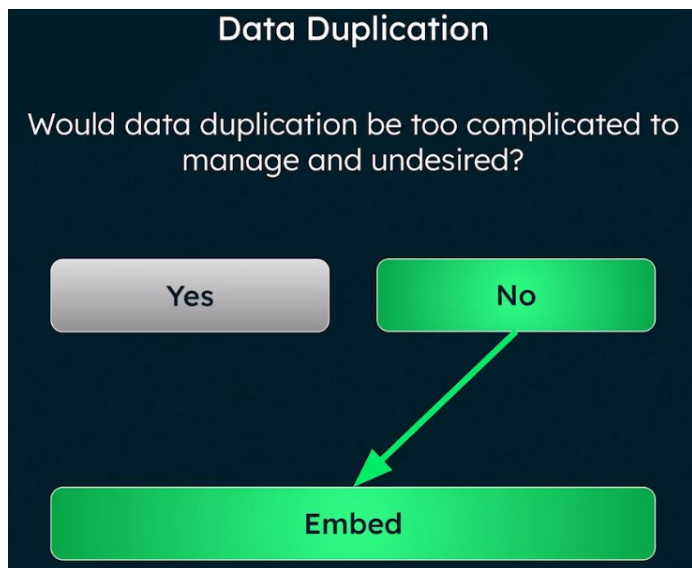
DELETE both



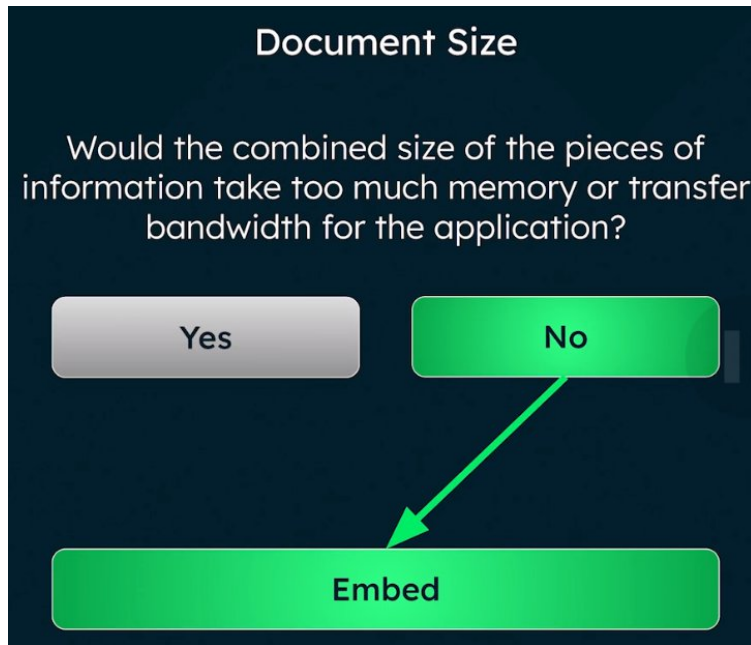
Cardinality



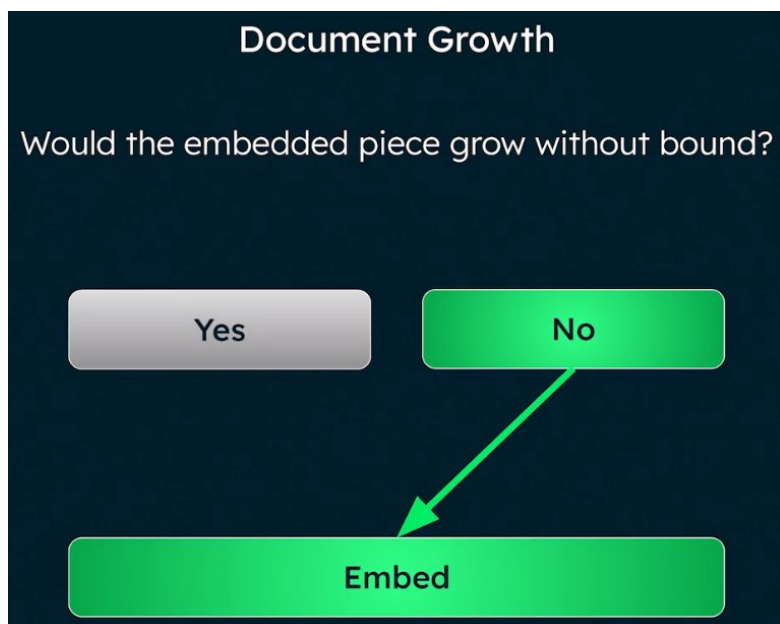
Data Duplication



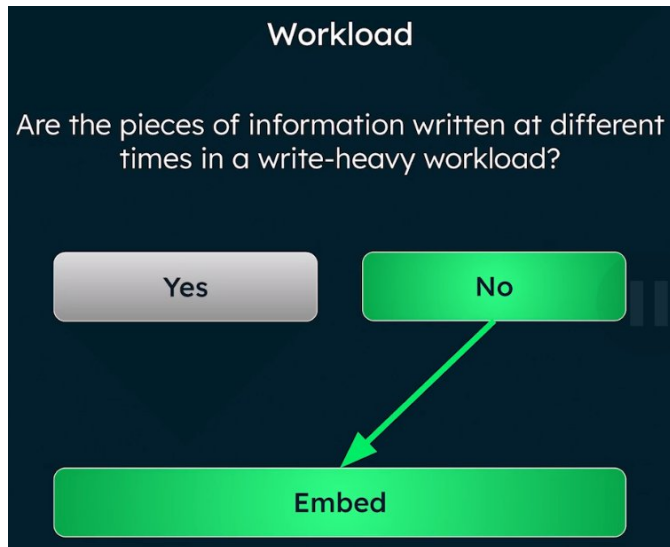
Document Size (16 MByte)



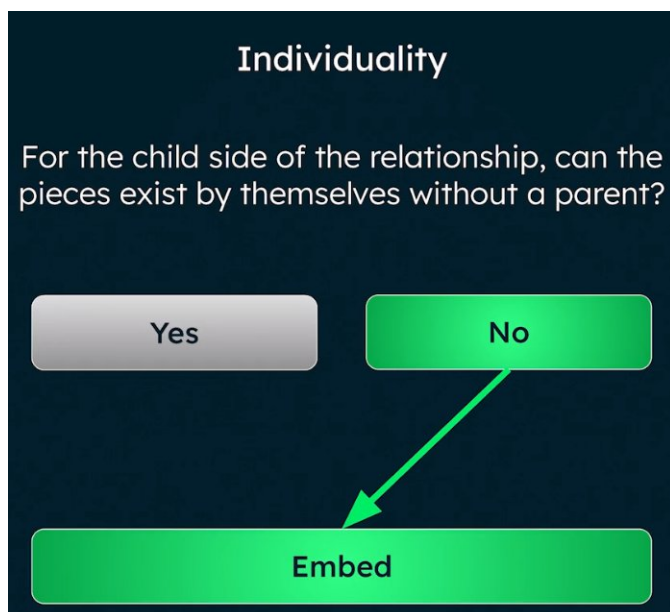
Document Growth (16 MByte)



Workload



Individuality



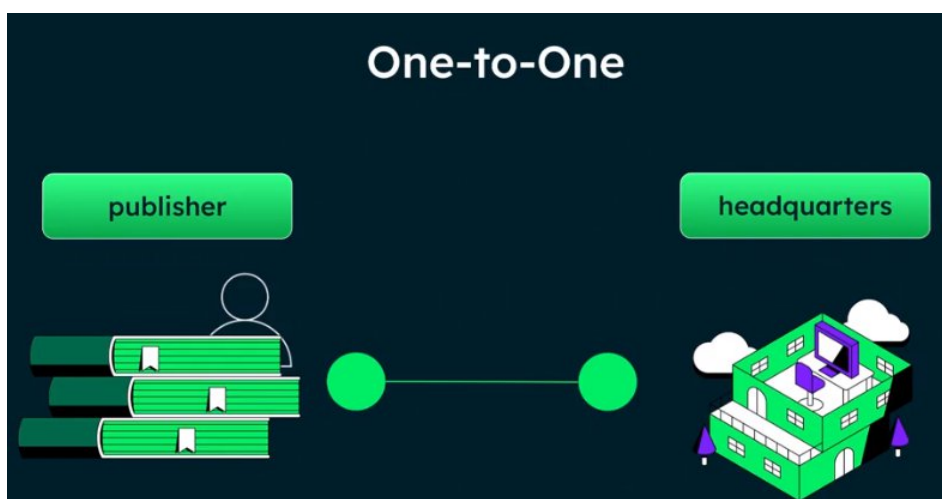
Result

Guidelines	Results
Simplicity	Embed
Go Together	Embed
Query Atomicity	Embed
Update Complexity	Reference
Archival	Reference
Cardinality	Embed
Data Duplication	Embed
Document Size	Embed
Document Growth	Embed
Workload	Embed
Individuality	Embed

Relationships

One-to-One

In MongoDB, a one-to-one relationship refers to a relationship between two collections where one document in each collection is related to exactly one document in the other collection. This relationship is established by embedding one document within another or by referencing documents between collections.





To decide if embedding or referencing is the better option, check the 11 guidelines.

Embedding Colocate

```
{
  "_id": "publisher00023",
  "name": "MongoDB Press",
  "street": "1234 Ave",
  "city": "New York",
  "state": "New York",
  "country": "US",
  "zip": "00000"
}
```

Embedding Subdocument

```
{
  "_id": "publisher00023",
  "name": "MongoDB Press",
  "headquarters": {
    "street": "1234 Ave",
    "city": "New York",
    "state": "New York",
    "country": "US",
    "zip": "00000"
  }
}
```

Reference

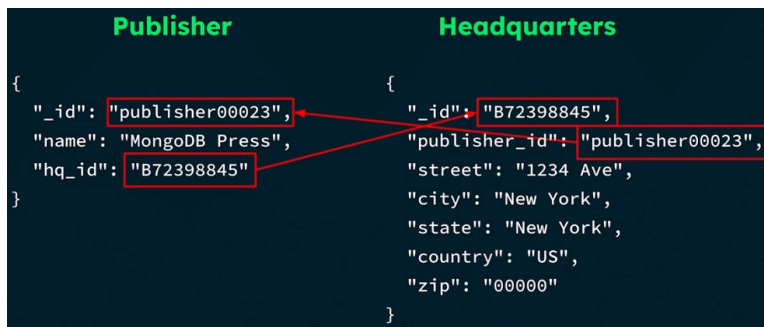
Primary read Publisher



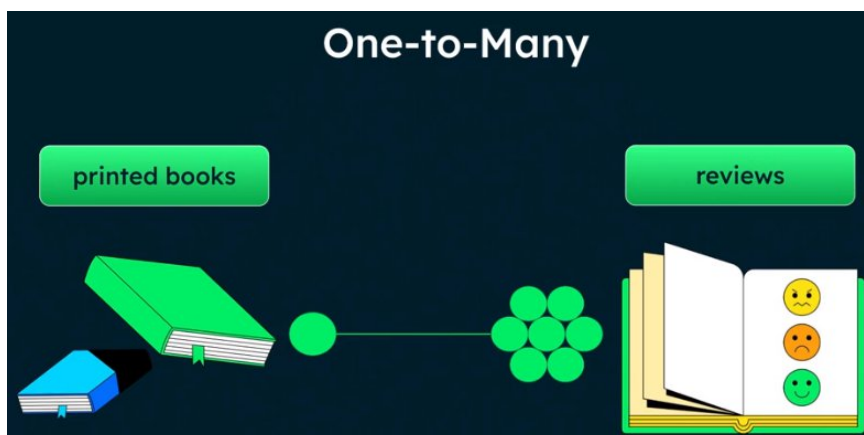
Primary read Headquarters



Bidirectional



One-to-Many



Array of subdocuments

Book

```
{
  "_id": "book0456",
  "title": "Practical MongoDB Aggregations",
  "author": "Paul Done",
  "reviews": [
    {
      "user_id": "user0956",
      "title": "Lorem Ipsum",
      "review_text": "dolor sit amet...",
      "rating": 3
    },
    {
      "user_id": "user0345",
      "title": "Ut enim",
      "review_text": "ad minim veniam...",
      "rating": 4
    }
  ]
}
```

Subdocument with subdocuments

Book

```
{
  "_id": "book0456",
  "title": "Practical MongoDB Aggregations",
  "author": "Paul Done",
  "reviews": {
    "user0956": {
      "title": "Lorem Ipsum",
      "review_text": "dolor sit amet...",
      "rating": 3
    },
    "user0345": {
      "title": "Ut enim",
      "review_text": "ad minim veniam...",
      "rating": 4
    }
  }
}
```

Reference

Book

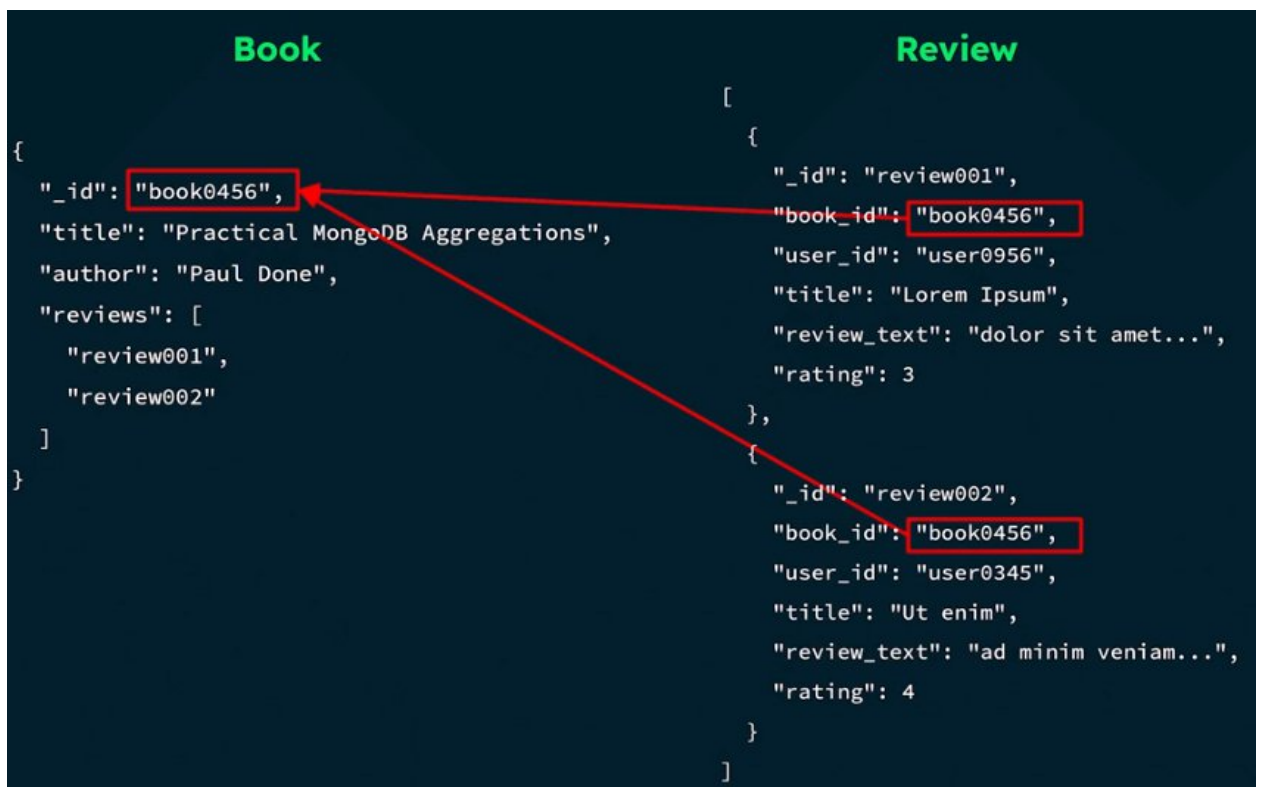
```
{
  "_id": "book0456",
  "title": "Practical MongoDB Aggregations",
  "author": "Paul Done",
  "reviews": [
    "review001",
    "review002"
  ]
}
```

Review

```
[
  {
    "_id": "review001",
    "user_id": "user0956",
    "title": "Lorem Ipsum",
    "review_text": "dolor sit amet...",
    "rating": 3
  },
  {
    "_id": "review002",
    "user_id": "user0345",
    "title": "Ut enim",
    "review_text": "ad minim veniam...",
    "rating": 4
  }
]
```



Both sides



Do a reference

Cardinality

Is there a high cardinality (current or growing) in the child side of the relationship?

Best solution

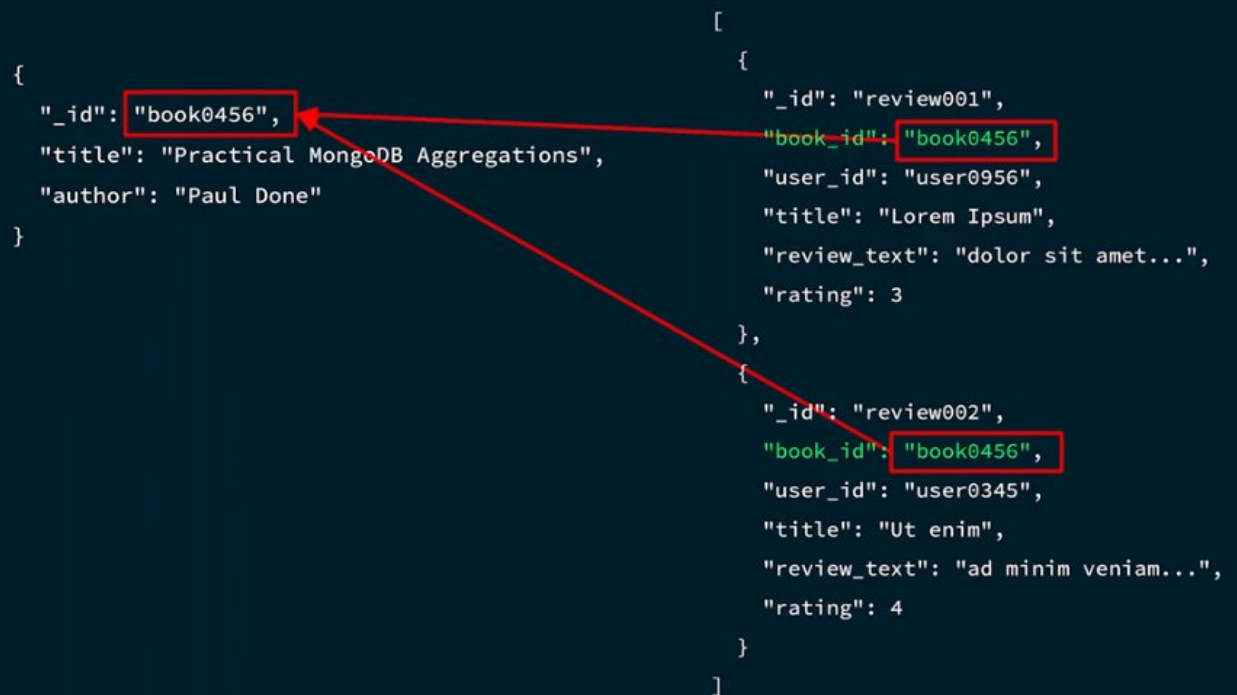
Array of subdocuments preferred

Avoid unbounded arrays

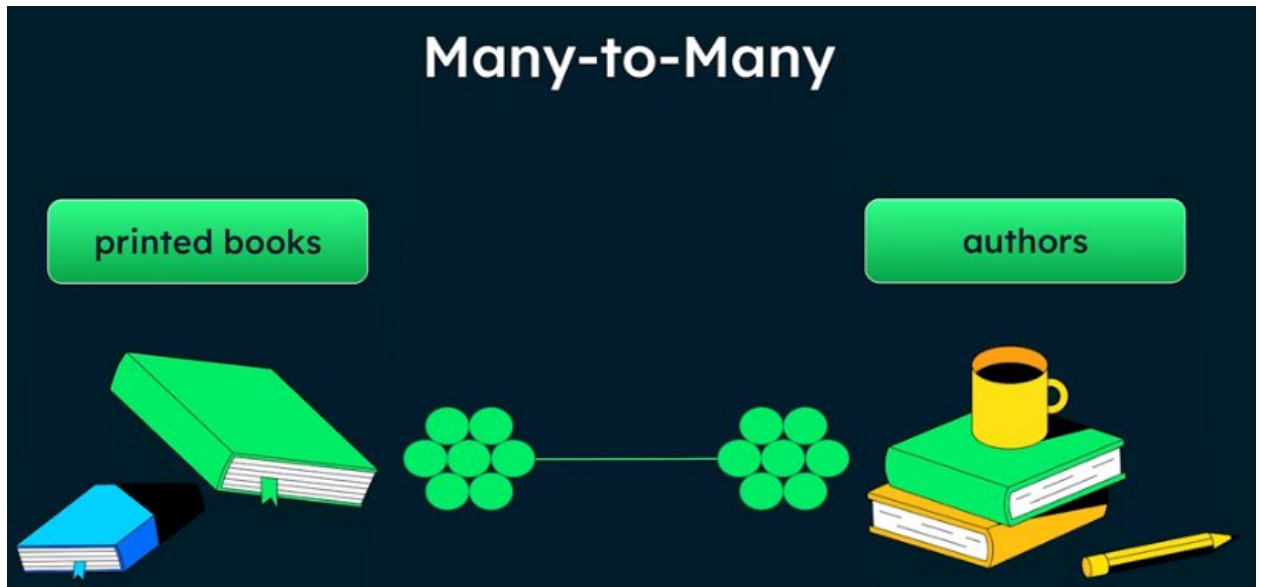
Referenced one-to-many relationship

```
{
  "_id": "book0456",
  "title": "Practical MongoDB Aggregations",
  "author": "Paul Done"
}

[
  {
    "_id": "review001",
    "book_id": "book0456",
    "user_id": "user0956",
    "title": "Lorem Ipsum",
    "review_text": "dolor sit amet...",
    "rating": 3
  },
  {
    "_id": "review002",
    "book_id": "book0456",
    "user_id": "user0345",
    "title": "Ut enim",
    "review_text": "ad minim veniam...",
    "rating": 4
  }
]
```



Many-to-Many

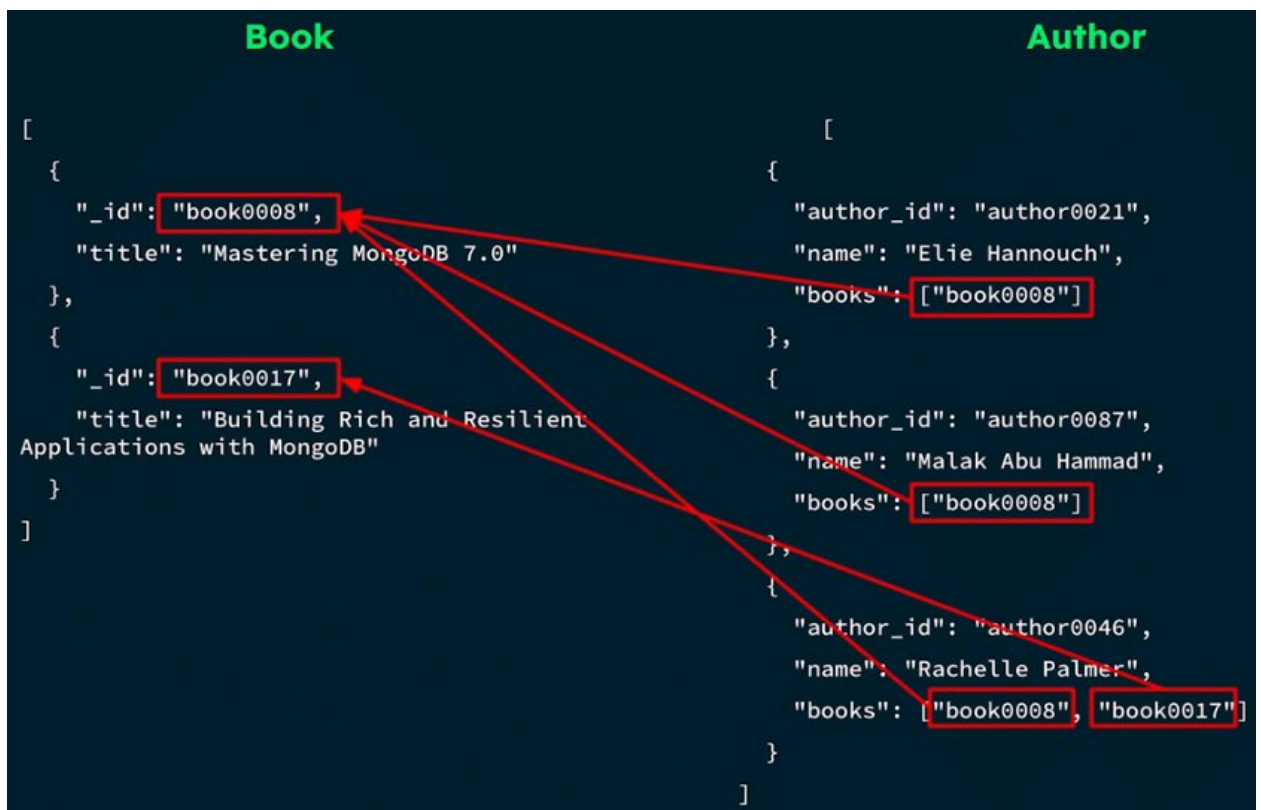
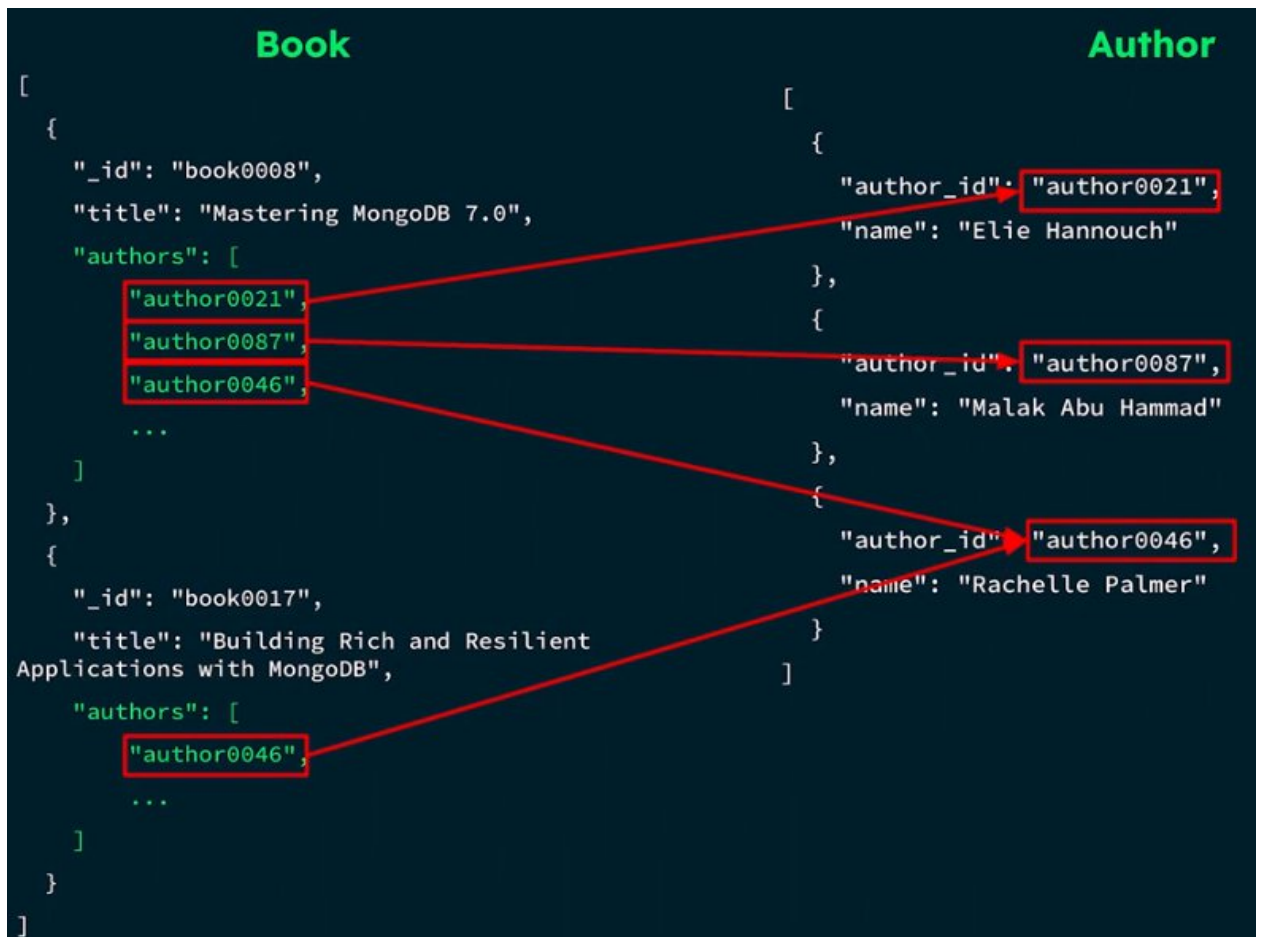


Subdocument

Not all data duplication is bad

```
Book                                     Book
{                                       {
  "_id": "book0008",
  "title": "Mastering MongoDB 7.0",
  "authors": [
    {
      "author_id": "author0021",
      "name": "Elie Hannouch"
    },
    {
      "author_id": "author0087",
      "name": "Malak Abu Hammad"
    },
    {
      "author_id": "author0046",
      "name": "Rachelle Palmer"
    },
    ...
  ]
}
```

```
    {
      "_id": "book0017",
      "title": "Building Rich and Resilient
Applications with MongoDB",
      "authors": [
        {
          "author_id": "author0046",
          "name": "Rachelle Palmer"
        },
        ...
      ]
    }
```



NO BIDIRECTIONAL REFERENCES

Embed Total: 9

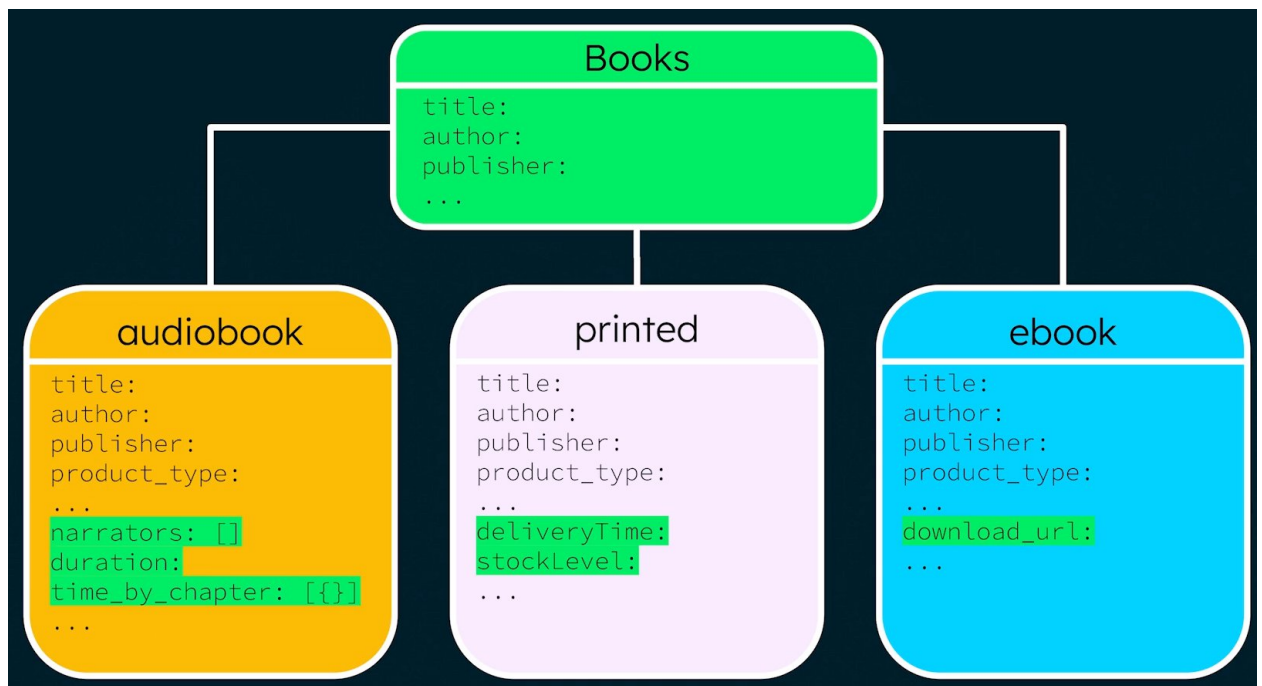
Reference Total: 2

Patterns

Inheritance

Data that is accessed together should be stored together.

Searching for books - no matter which type.



ebook

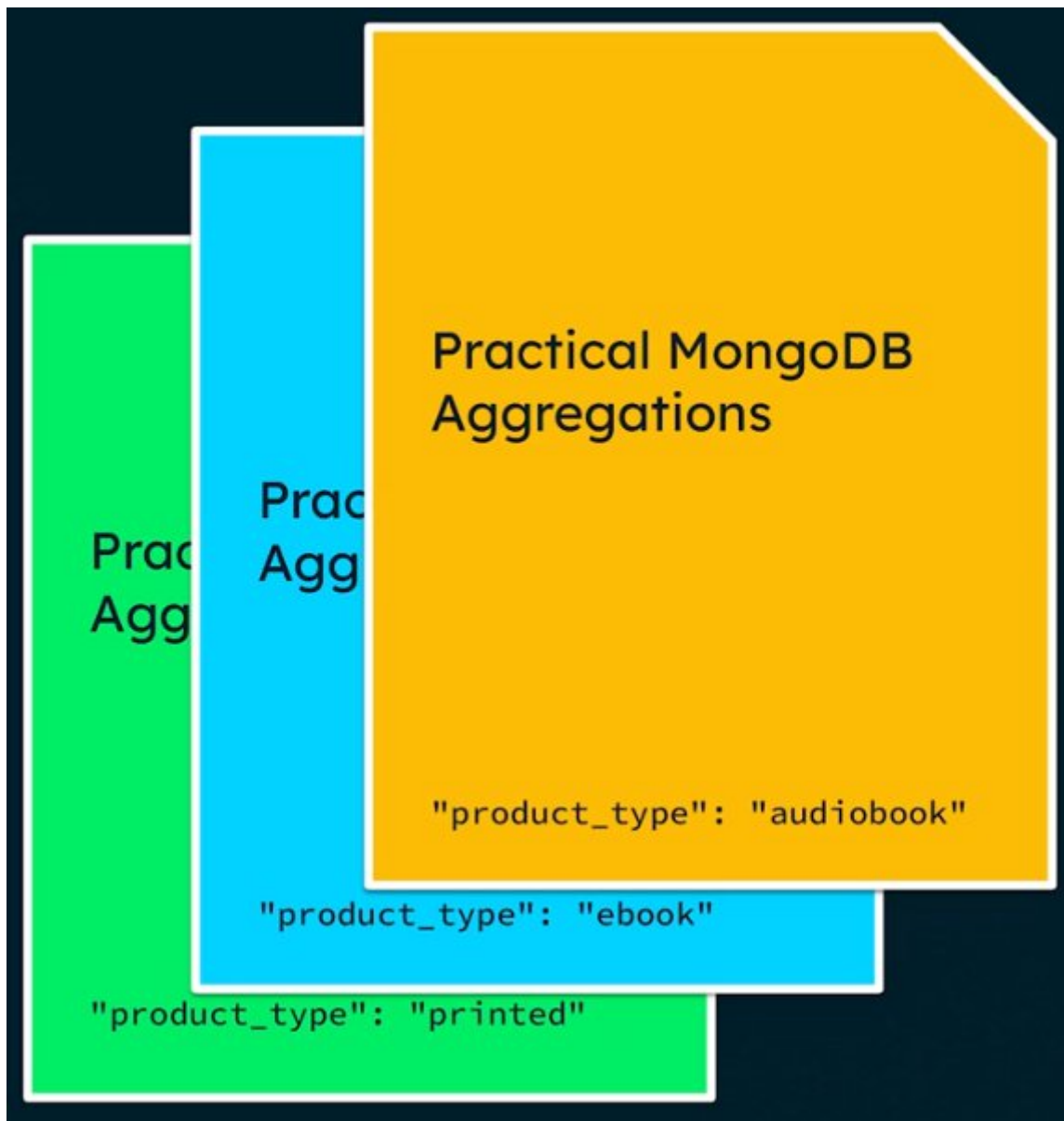
```
{  
  ...  
  "title": "Practical MongoDB Aggregations",  
  "author": "Paul Done",  
  "authorId": "author0034",  
  "rating": 4.8,  
  "genres": ["Programming"],  
  "pages": 338,  
  "price": 0.00,  
  "publisher": "MongoDB Press",  
  ...  
  "product_type": "ebook",  
  "download_url": "https://mdb.com/dune.ebpub"  
}
```

audiobook

```
{
  "title": "Practical MongoDB Aggregations",
  "author": "Paul Done",
  ...
  "genres": ["Programming"],
  "price": 0.00,
  "publisher": "MongoDB Press",
  ...
  "product_type": "audiobook",
  "download_url": "https://dl.mdb.com/dune.m4a",
  "narrators": ["Paul Done"],
  "duration": "21 hours 8 minutes",
  "time_by_chapter": [{
    "chapter": 1, "start": "00:00:00",
    "end": "01:00:00"
  },
  ...
]
}
```

printed book

```
{  
  ...  
  "title": "Practical MongoDB Aggregations",  
  "author": "Paul Done",  
  "authorId": "author0034",  
  "rating": 4.8,  
  "genres": ["Programming"],  
  "pages": 338,  
  "price": 0.00,  
  "publisher": "MongoDB Press",  
  ...  
  "product_type": "printed_book",  
  "stockLevel": 132,  
  "deliveryTime": "2",  
}
```

```
home_adr: {  
  street: "Mullergasse",  
  house_number: "13A",  
  zip: "2700",  
  city: "Wiener Neustadt"  
}, home_adr: {  
  street: "Mullergasse",  
  house_number: "13A",  
  zip: "2700",  
  city: "Wiener Neustadt"  
}, home_adr: {
```

```
    street: "Mullergasse",  
      house_number: "13A",  
      zip: "2700",  
      city: "Wiener Neustadt"  
  }, home_adr: {  
    street: "Mullergasse",  
      house_number: "13A",  
      zip: "2700",  
      city: "Wiener Neustadt"  
  },
```