**1. What is the difference between embedded documents and referencing in MongoDB? When would you use each approach?**

In an embedded document, related data is stored within the same document. For a bookstore, this could mean storing a book's reviews inside the same document as the book itself.

**Pros of Embedded Documents:**

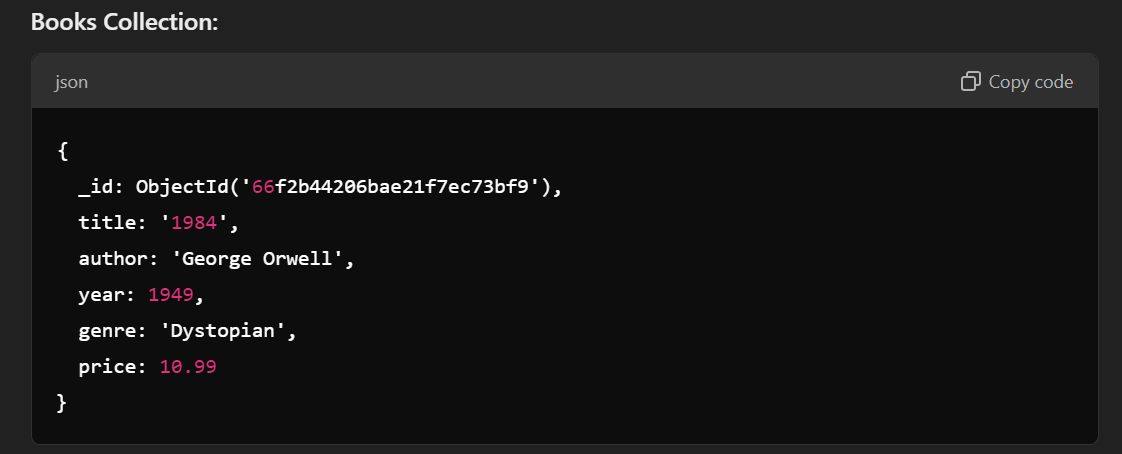
1. **Faster Read Operations**: All related data is stored together in one document, making reads quicker. For example, when you retrieve a book, you also get its reviews.
2. **Atomicity**: Updates to the document are atomic, meaning changes to the document and its subdocuments happen together. This is useful if you need to update book information and reviews simultaneously.
3. **Simpler Queries**: Queries don’t need to join different collections to retrieve related data. You can retrieve everything you need in one call.

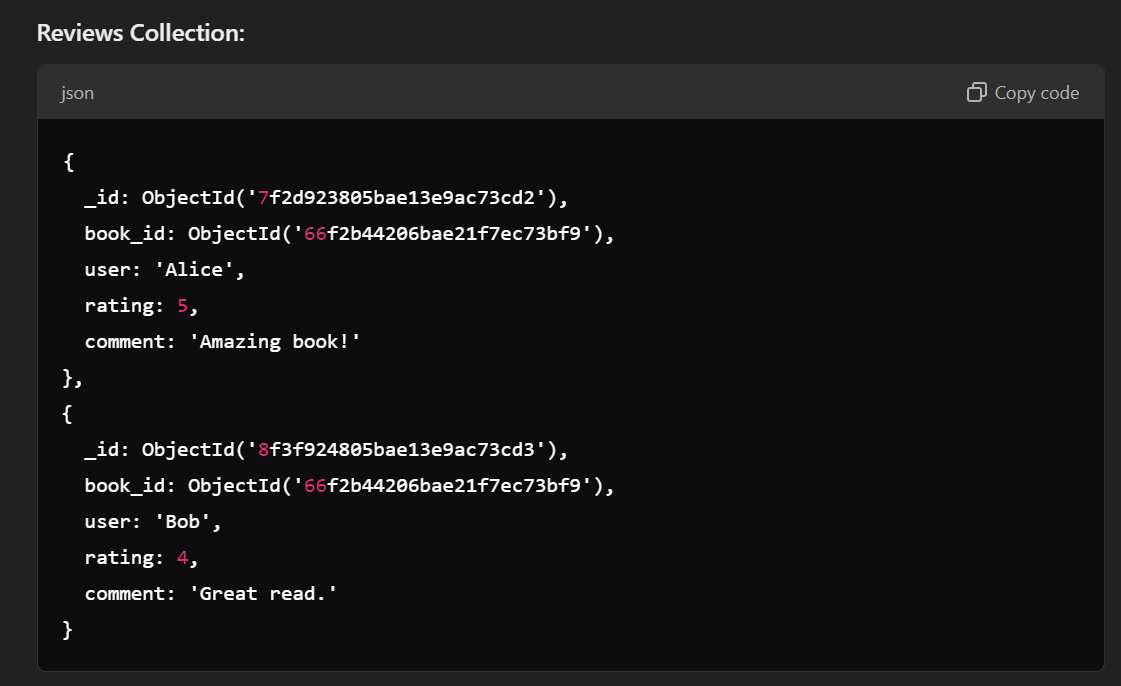
**Cons of Embedded Documents:**

1. **Document Size Limitation**: MongoDB has a 16 MB document size limit. If the embedded data (e.g., reviews) grows too large, this can become problematic.
2. **Duplication and Updates**: If reviews are frequently updated independently of the book, embedding may not be ideal. Every time you update a review, you need to update the whole book document.
3. **Data Redundancy**: If the same data is embedded in multiple documents, redundancy and inconsistencies can occur when updating.

**Referenced Document Example**

In a referenced document structure, related data is stored in separate collections, and references (e.g., IDs) link them together. For a bookstore, this could mean having a books collection and a reviews collection, where reviews are linked to books by a book\_id.





**Pros of Referenced Documents:**

1. **Scalability**: Related data (like reviews) can grow independently from the main document (book). This avoids the 16 MB document size limit.
2. **Reduced Redundancy**: Common data (like reviews) is stored once and referenced. This makes updates more efficient and reduces redundancy.
3. **Flexibility**: Data can be updated separately. For instance, you can update reviews independently of the book details.

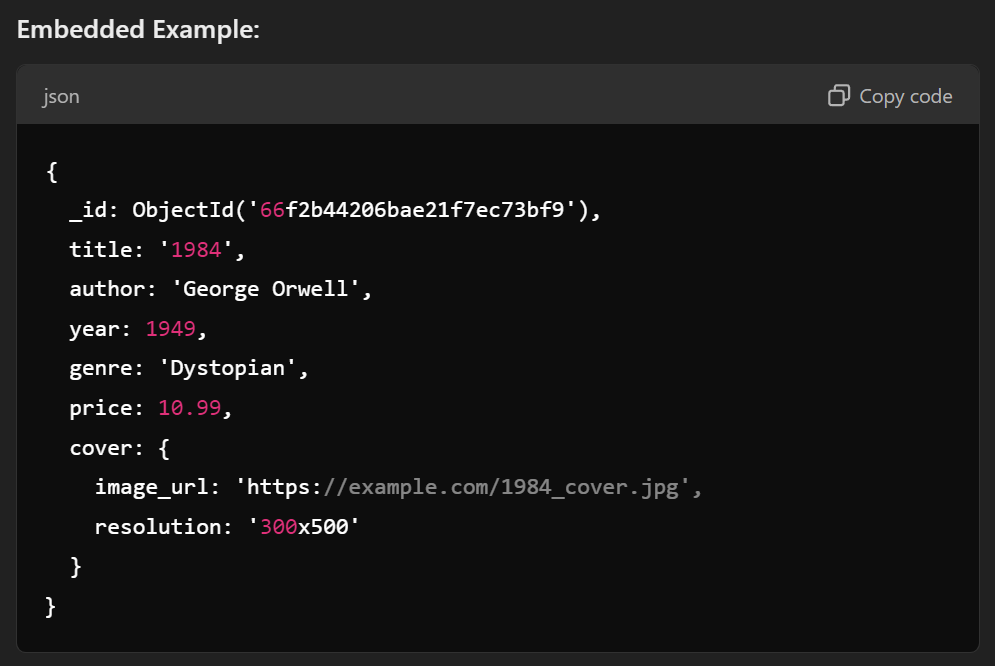
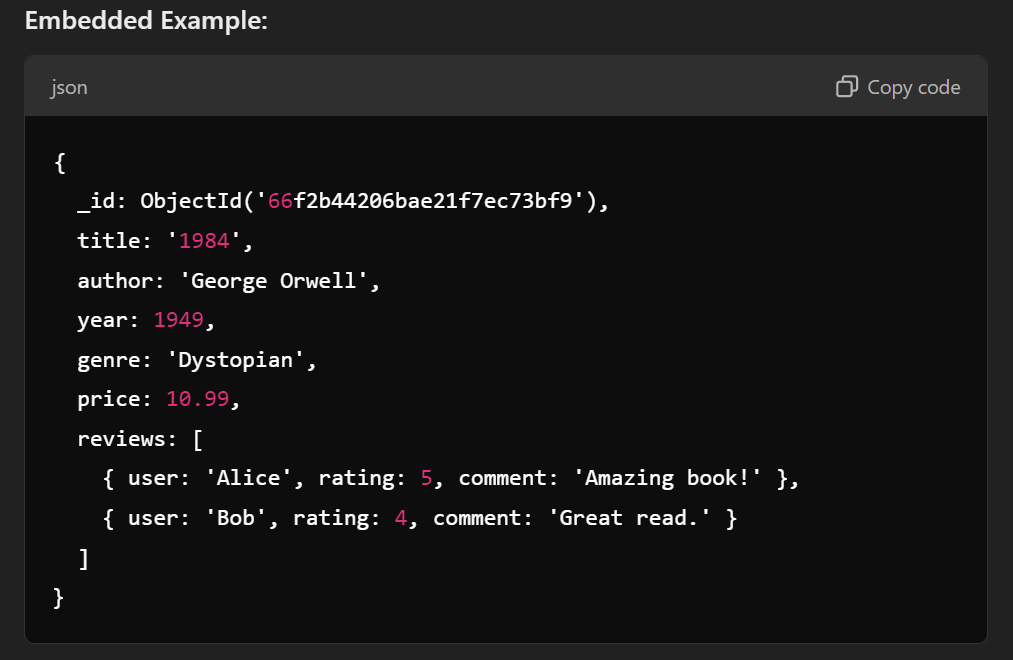
**Cons of Referenced Documents:**

1. **Slower Read Operations**: You need to make multiple queries or use joins (through $lookup in MongoDB) to retrieve related data from different collections. For example, retrieving a book and its reviews requires querying both the books and reviews collections.
2. **More Complex Queries**: You have to write queries that join data across collections, which can increase complexity.
3. **Lack of Atomicity**: Updates to related documents (e.g., book and its reviews) are not atomic. If a transaction fails, the data may become inconsistent.

**When to Use Each Approach:**

* Use Embedded Documents when:
  + The related data is tightly coupled and will be accessed frequently together.
  + The related data is not expected to grow significantly (e.g., a limited number of reviews per book).
  + Simplicity and atomicity are a priority for your use case.
* Use Referenced Documents when:
  + The related data is large and grows independently (e.g., thousands of reviews).
  + The related data needs to be accessed or updated separately from the parent document.
  + You want to reduce redundancy and avoid document size limits.

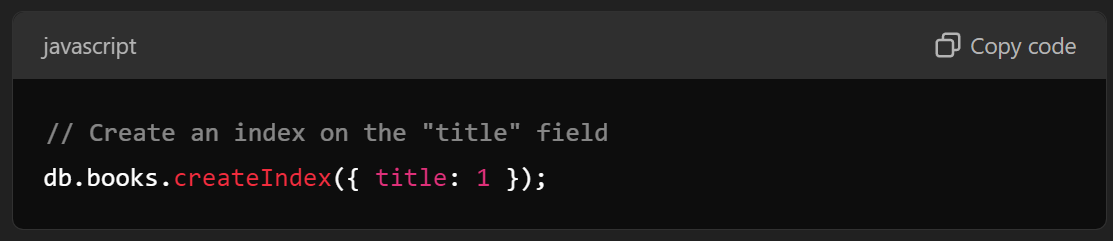
**Relationships** **in** **MongoDB**:

1. **One-to-One:** Each document in a collection is **related to one and only one** document in another collection. In MongoDB, this can be implemented by embedding one document within another or by referencing documents using unique identifiers.  
     
   
2. **One-to-Many:** A single document in one collection is related to multiple documents in another collection. For example, one book can have multiple reviews.  
   ****
3. **Many-to-Many Relationship:** A Many-to-Many relationship occurs when multiple documents in one collection are related to multiple documents in another collection. For example, multiple books can be written by multiple authors, and one author can write multiple books.

**Indexing** **in** **MongoDB**:

* In MongoDB, an **index** is a special data structure that improves the speed and efficiency of query operations.
* It works by maintaining a sorted order of the values of the indexed fields, similar to how an index in a book allows you to quickly find information without scanning through every page.

**Types of Indexes in MongoDB:**

1. Single Field Index:  
   
2. Compound Field Index:  
   