**MongoDB Coding Challenge**

**Name: PRIYESHWAR**

**Mail:** [**priyesh2664@gamil.com**](mailto:priyesh2664@gamil.com)

**Relationships in MongoDB**

MongoDB is a NoSQL database that offers flexible ways to model relationships between data. Unlike relational databases, MongoDB does not enforce foreign key constraints, so managing relationships is handled through data modeling strategies.

**Types of Relationships**

MongoDB supports two primary methods for representing relationships between documents:

1. **Embedded Documents (Denormalization)**

This approach stores related data within a single document, ideal for data that is frequently accessed together. It simplifies data retrieval and ensures data locality.

**Use When:**

* Data is tightly coupled and accessed together.
* The relationship is one-to-few.
* Performance is more important than data redundancy.

**Example:**

{

"\_id": 1,

"name": "Alice",

"orders": [

{

"order\_id": 1001,

"product": "Laptop",

"price": 1200

},

{

"order\_id": 1002,

"product": "Mouse",

"price": 25

}

]

}



**Pros:**

* Fewer queries
* Faster read performance

**Cons:**

* Document size can grow quickly
* Harder to update deeply nested data

1. **Referenced Documents (Normalization)**

This method involves storing references to related documents using unique identifiers, suitable for large or independently accessed data. It allows for normalization and maintains data consistency.

**Use When:**

* Data is reused across documents (e.g., users, products).
* The relationship is one-to-many or many-to-many.
* Data integrity and separation are important.

**Example:**

**Users Collection**

{

"\_id": ObjectId("64f0a..."),

"name": "Bob"

}

**Orders Collection**

{

"\_id": ObjectId("64f0b..."),

"user\_id": ObjectId("64f0a..."),

"product": "Keyboard",

"price": 50

}

**Pros:**

* More flexible for complex relationships
* Easier to update data in isolation

**Cons:**

* Requires multiple queries or aggregation joins

**Using $lookup for Joins**

MongoDB supports a form of join using the $lookup aggregation stage. the $lookup stage, enabling complex many-to-many relationships.

**Example:**

db.orders.aggregate([

{

$lookup: {

from: "users",

localField: "user\_id",

foreignField: "\_id",

as: "user\_info"

}

}

])

This returns each order document with a new field user\_info containing the referenced user data.

**Choosing Between Embedding vs Referencing**

| **Feature** | **Embedding** | **Referencing** |
| --- | --- | --- |
| Simplicity | Simple | More complex |
| Query Speed | Faster reads | Slower (needs join) |
| Write Efficiency | Fewer writes | May need multiple writes |
| Data Integrity | Harder | Easier |
| Document Size | Can grow large | Smaller units |

**One-to-One Relationship**

A one-to-one relationship is when one document is associated with exactly one other document.

**Use Cases:**

* Storing user profiles separately from user credentials
* Separating large optional fields (e.g., user settings, metadata)

**Embedded Document**

{

"\_id": ObjectId("user\_id\_1"),

"username": "john\_doe",

"profile": {

"full\_name": "John Doe",

"age": 30,

"bio": "Developer at Acme Corp"

}

}



**Referenced Document**

**Users Collection**

{

"\_id": ObjectId("user\_id\_1"),

"username": "john\_doe",

"profile\_id": ObjectId("profile\_id\_1")

}

**Profiles Collection**

{

"\_id": ObjectId("profile\_id\_1"),

"full\_name": "John Doe",

"age": 30,

"bio": "Developer at Acme Corp"

}



**One-to-Many Relationship with Embedded Documents**

A one-to-many relationship is when one document contains many related sub-documents. Embedding works best when the related items are limited in number and used together.

**Use Cases:**

* A blog post with comments
* A product with specifications
* A user with a list of saved items

**Example: Blog Post with Embedded Comments**

{

"\_id": ObjectId("post\_id\_1"),

"title": "MongoDB Relationships",

"content": "Understanding embedded and referenced documents.",

"comments": [

{

"author": "Alice",

"text": "Great explanation!",

"date": "2025-07-20"

},

{

"author": "Bob",

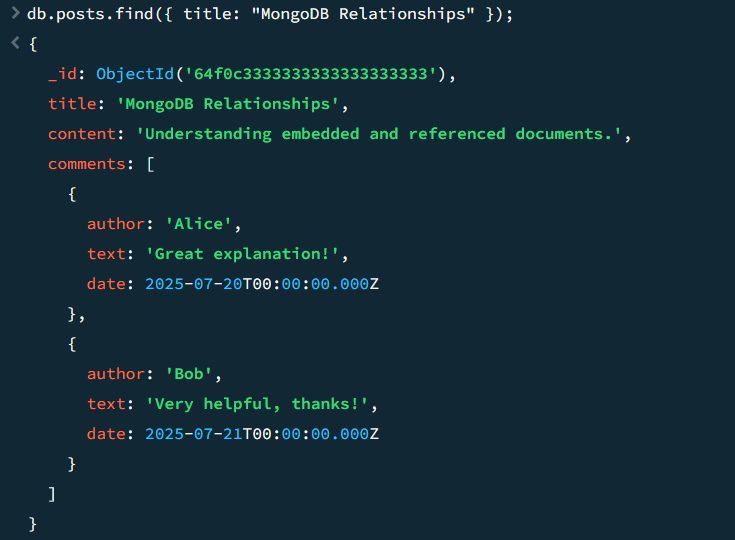
"text": "Very helpful, thanks!",

"date": "2025-07-21"

}

]

}

****

**One-to-Many Relationship with Document References**

Referencing is ideal when the number of related documents is large or when sub-documents are accessed independently.

**Use Cases:**

* Users and their orders
* Authors and their books
* Categories and products

**Example: User and Orders (Referenced)**

**Users Collection**

{

"\_id": ObjectId("user\_id\_1"),

"name": "Alice"

}

**Orders Collection**

{

"\_id": ObjectId("order\_id\_1"),

"user\_id": ObjectId("user\_id\_1"),

"item": "Monitor",

"price": 199.99

}

**Fetching Related Data with $lookup**

db.users.aggregate([

{

$lookup: {

from: "orders",

localField: "\_id",

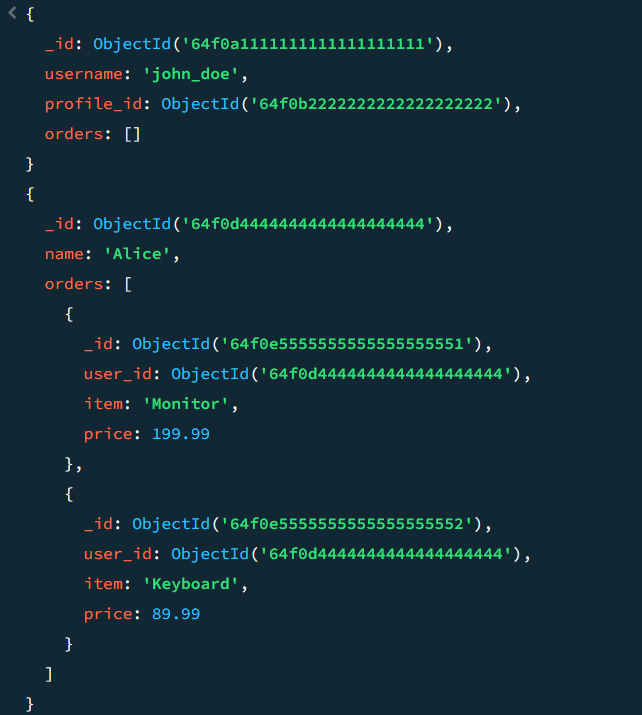
foreignField: "user\_id",

as: "orders"

}

}

])



| **Relationship Type** | **Embedding** | **Referencing** |
| --- | --- | --- |
| One-to-One | Small, always together | Large, separate lifecycle |
| One-to-Many | Few items, read together | Many items, queried separately |