MongoDB – Relationships

https://www.navicat.com/en/company/aboutus/blog/1004-create-dbrefs-in-mongodb

Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modeled via **Embedded** and **Referenced** approaches. Such relationships can be either 1:1, 1:N, N:1 or N:N.

Let us consider the case of storing addresses for users. So, one user can have multiple addresses making this a 1:N relationship.

Following is the sample document structure of **user** document −

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"name": "Tom Hanks",

"contact": "987654321",

"dob": "01-01-1991"

}

Following is the sample document structure of **address** document −

{

"\_id":ObjectId("52ffc4a5d85242602e000000"),

"building": "22 A, Indiana Apt",

"pincode": 123456,

"city": "Los Angeles",

"state": "California"

}

Modeling Embedded Relationships

In the embedded approach, we will embed the address document inside the user document.

> db.users.insert({

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin",

"address": [

{

"building": "22 A, Indiana Apt",

"pincode": 123456,

"city": "Los Angeles",

"state": "California"

},

{

"building": "170 A, Acropolis Apt",

"pincode": 456789,

"city": "Chicago",

"state": "Illinois"

}

]

}

})

This approach maintains all the related data in a single document, which makes it easy to retrieve and maintain. The whole document can be retrieved in a single query such as −

>db.users.findOne({"name":"Tom Benzamin"},{"address":1})

Note that in the above query, **db** and **users** are the database and collection respectively.

The drawback is that if the embedded document keeps on growing too much in size, it can impact the read/write performance.

Modeling Referenced Relationships

This is the approach of designing normalized relationship. In this approach, both the user and address documents will be maintained separately but the user document will contain a field that will reference the address document's **id** field.

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin",

"address\_ids": [

ObjectId("52ffc4a5d85242602e000000"),

ObjectId("52ffc4a5d85242602e000001")

]

}

As shown above, the user document contains the array field **address\_ids** which contains ObjectIds of corresponding addresses. Using these ObjectIds, we can query the address documents and get address details from there. With this approach, we will need two queries: first to fetch the **address\_ids** fields from **user** document and second to fetch these addresses from **address** collection.

>var result = db.users.findOne({"name":"Tom Benzamin"},{"address\_ids":1})

>var addresses = db.address.find({"\_id":{"$in":result["address\_ids"]}})

As seen in the last chapter of MongoDB relationships, to implement a normalized database structure in MongoDB, we use the concept of **Referenced Relationships** also referred to as **Manual References** in which we manually store the referenced document's id inside other document. However, in cases where a document contains references from different collections, we can use **MongoDB DBRefs**.

## DBRefs vs Manual References

As an example scenario, where we would use DBRefs instead of manual references, consider a database where we are storing different types of addresses (home, office, mailing, etc.) in different collections (address\_home, address\_office, address\_mailing, etc). Now, when a **user** collection's document references an address, it also needs to specify which collection to look into based on the address type. In such scenarios where a document references documents from many collections, we should use DBRefs.

## Using DBRefs

There are three fields in DBRefs −

* **$ref** − This field specifies the collection of the referenced document
* **$id** − This field specifies the \_id field of the referenced document
* **$db** − This is an optional field and contains the name of the database in which the referenced document lies

Consider a sample user document having DBRef field **address** as shown in the code snippet −

{

"\_id":ObjectId("53402597d852426020000002"),

"address": {

"$ref": "address\_home",

"$id": ObjectId("534009e4d852427820000002"),

"$db": "tutorialspoint"},

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin"

}

The **address** DBRef field here specifies that the referenced address document lies in **address\_home** collection under **tutorialspoint** database and has an id of 534009e4d852427820000002.

The following code dynamically looks in the collection specified by **$ref** parameter (**address\_home** in our case) for a document with id as specified by **$id** parameter in DBRef.

>var user = db.users.findOne({"name":"Tom Benzamin"})

>var dbRef = user.address

>db[dbRef.$ref].findOne({"\_id":(dbRef.$id)})

The above code returns the following address document present in **address\_home** collection −

{

"\_id" : ObjectId("534009e4d852427820000002"),

"building" : "22 A, Indiana Apt",

"pincode" : 123456,

"city" : "Los Angeles",

"state" : "California"

}