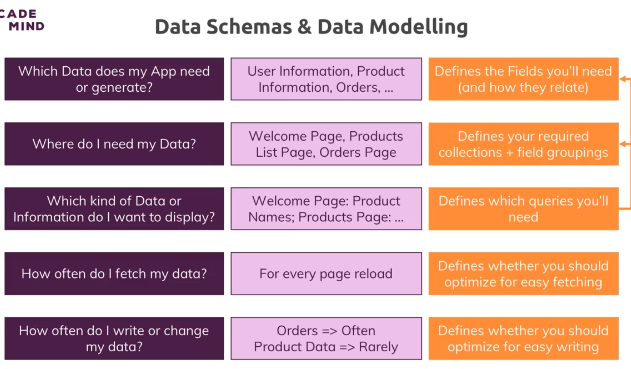
Designing schemas

* Always think about 3 aspects – if u store separately u have to join, we should not have duplicates, and store separately only if u fetch them individually
* Based on the fetching mechanism only decide whether it is embedded document approach or separate collection approach,
* Design collections in such a way that u **should not end up in joining the tables and u should not perform complex joins** to fetch the records if you are facing this problem then prefer embedded document approach
* **Duplicate data issue :-**While designing make sure you should not have duplicate data , if u have duplicate data and if u want to update its painful to update everywhere , consider whether duplicates can hurt you or not
* **For tables having one- one relationship better choose embedded documents instead of storing them separately in separate tables**
* **While de**signing tables think of each field , in future are we going to change to object type from single field, analyse and decide to take it as either single or object type
* **In 1- many ,If u are always fetching child document alone and If u are not interested in parent doc then** store them separately follow separate collection approach, if u store them in embedded way then un necessarily you have to extract the data always
* **In 1-many If you are always fetching together both parent and child all records at same time – then** prefer embedded collection approach , because anyways u want both records then ,this this approach will solve un necessary joins problem
* **1 document max size is 16MB,** don’t embedded/nest too much more than that , mongo can support more than 100 levels
* While designing Think what is your operation –

How often do you fetch your data

How often do u change your data – if u are always writing data make sure you avoid duplicates



2) **Duplicate data issue**

Collection user:-

{

Username : “santhoshi”,

Age: 29,

favBooks:[ {bookname:”java InAction”,AuthorName:”keith serra”},

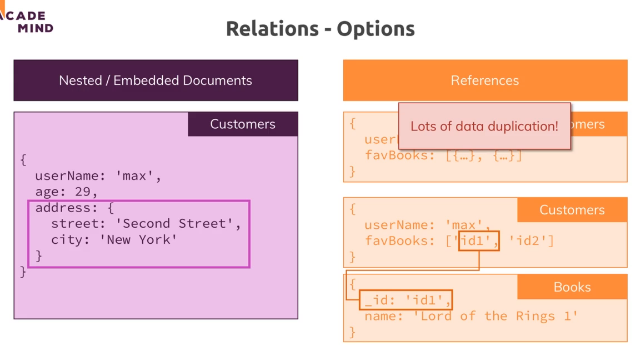
{bookname:”Spring In Action”,AuthorName:”Rod johnson”}

]

}

Think we have hundreds of user almost if all have same favourite book then tomorrow if we have some modifications its practically impossible to change in all places

So in that case use references concept



So as per above create separate collection and store them separately and refer them

Ex:- patient, discharge summary

Every patient will definitely have discharge summary so why to store separately and

join 2 tables while fetching , better store in a single table having embedded document

Embedded document approach

1. Oracle is RDBMS db, since it is relational they store all collections distributed and they used to join while fetching m but here this is not RDBMS, mostly avoid joining collections
2. ***In one to one relationship mostly prefer embedded document approach***
3. *In one to many also sometimes we may need to go for embedded document approach*

*Ex:- each question will have 2-3 answers , as these are tightly coupled* ***mostly we will fetch both questions and answers*** *if we store them separately then we may need to join tables so to avoid it* ***prefer embedded document approach***

1. *Post- comments approach , 1 post will have many comments , we have a post collection but not comments collection*

Embedded doc app ex

In Facebook 1 post will have many comments, generally one - many means if u are mostly interested in fetching child documents alone then go for separate collection approach

**Example1:- 1 to many separate collection approach** 1 disaster order will have many benefits, there we mostly fetch child documents separately so we should prefer separate collection approach

Disaster order separate collection and disasterAssistance separate collection

**Example 2:- 1 to many Embedded document approach**

1 post will have many comments, but generally we are not interested in fetching comments alone

We always fetch with post both are tightly coupled, so there is no chance of fetching comments alone without fetching post, so comments are not stored separately and that to comment is a small document and if we store we should unnecessary join tables better prefer embedded doc approach

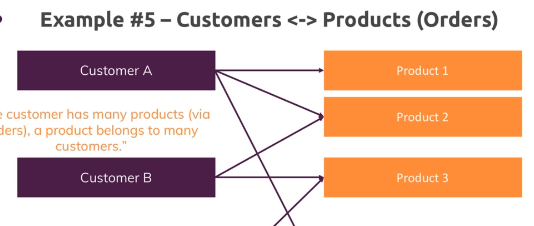
But I am thinking 1 document max size is 16 MB , if 1 post have 1 million of comments which may consume 16 MB, so for size constraint should we consider embedded doc approach?

Separate collection approach

1. 1 document maximum size is 16 MB , 1- many , city- citizens if u follow embedded doc approach 1 document may not be enough to store all citizens information it will easily hit 16MB space constraint so if u have space constraint also prefer separate collection approach
2. One – Many 1 – Mostly separate collection approach, if disaster order id Is having multiple benefits and mostly if we are interested only in **fetching benefits** (FETCHING childs) why to store them in single document and fetch every thing better store them in a separate document – ex:- collections like DisasterOrder, DisasterAssistance
3. In one to one approach 1 person having 1 car and every time if u are interested only in cars then why to fetch entire document since u are only interested in cars make them as a separate collection
4. Embedded document approach not suitable when 1 person having 1 car/ 2 cars , if u have a situation if u want to fetch only cars for all persons, then don’t follow embedded documents methodology
5. One – many 🡪 city –citizens , 1 city have many citizens – mostly store them separately, In case of embedded document approach value is maximum 16 MB only , city –population 1 city have many persons if you are interested only to fetch persons always better store them in separate document approach and that to 1 city have millions of citizens , if u store in embedded document approach all citizens will be stored in single document which will be beyond 16 MB so atleast for this constraint prefer separate collection approach
6. **Many – Many approach**

Simple answer – for this also mostly **prefer separate collection approach- solves duplicate records problem but while fetching u need to join the tables here, if u are fine with duplicates and if u don’t want to join then Still prefer embedded document approach**

1 customer will buy many products and 1 product is being bought by many customers



db.products.find();

|  |
| --- |
| { \_id: 'p1', pname: 'kandipappu', price: 140.123546789 } |

db.customers.find();

|  |
| --- |
| { \_id: **'c1'**, |
| cname: **'manideep'**, |
| age: 29, |
| orders: [ { pid: **'p1'**, quantity: 2 } ] } |

Above is best approach

**Advantage**:- duplicate data gone ,in orders u have just mentioned pid instead of mentioning all product details so in future if u want to update the price u can update in product table alone , since the bill is already generated even if the price hiked to 145 it doesn’t make any issue

**Disadvantage**:- while fetching customer u have to join the tables to see the list of products he bought,

If u don’t want to join the tables and if u are ok with duplicates then prefer embedded doc approach

**Worst embedded document approach –Because this leads to maintain duplicate data**

db.customers.find();

|  |
| --- |
| { \_id: **'c1'**, |
| name: **'N R narayana murthy'**, |
| age: 70, |
| orders: [ { pname: **'kandipappu'**, price: 140.33, quantity: 2 } ] } |

db.products.find();

|  |
| --- |
| { \_id: **'p1'**, pname: **'kandipappu'**, price: 140.123546789 } |

**This is called worst approach because product, price fields are duplicated if tomorrow if update comes we should update all ,if that product is having many more fields having this duplicate data under each customer is memory waste and if any update come difficult to modify like vendor name and all.**

**, but the advantage here is if u want at how much price he bought then this approach is good but while joining the collections since you wont maintain that here always updated price will be there so prefer based on your requirement**

Many to Many separate collection approach

**This** separate collection approach is to avoid duplicates and the problem with duplicates is when there is an update we have to do it everywhere

Books- authors

From both sides it is one to many – hence this is called many to many

1 book is written by many authors and 1 author will write many books

db.books.find();

[

{ \_id: 'b1', bookName: 'kafka in Action',

author: { name: 'thiruppavi', age: 29, address: 'flat 302- kkpalem' }

},

{

\_id: 'b2', bookName: 'micro services in Action',

authors: [

{ name: 'thiruppavi', age: 29, address: 'flat 302- kkpalem' },

{ name: 'rod johnson', age: 40, addresss: ' USA' }

]

}]

Issue with above approach

See the author for every year age changes and after marriage name changes and after some years if the address changes for the author name thiruppavi he wrote 2 books so we have to update in 2 documents this is the problem with the duplicates approach

Best approach-use references

So use references and maintain in separate collection

Authors separate collection and books separate collection and use references

db.authors.find();

|  |
| --- |
| { \_id: **'auth1'**, age: 29, name: **'rod johnson'**, address: **'USA'** } |
| { \_id: **'auth2'**, age: 45, name: **'Gavin king'**, address: **'UK'** }  Ran update queries as below |

db.books.updateOne({\_id:'b1'}, { $set : {authors : ['auth1'] }} );

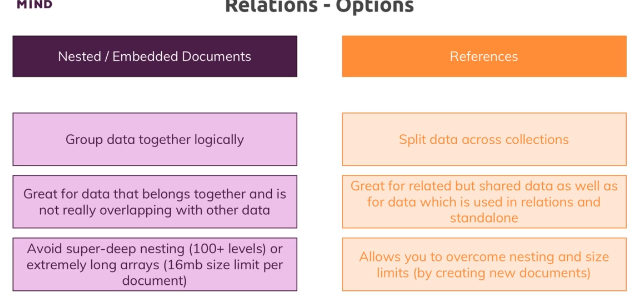
db.books.updateOne({\_id:'b2'}, { $set : {authors : ['auth1','auth2'] }} );

after update records look like

db.books.find();

|  |
| --- |
| { \_id: **'b1'**, bookName: **'kafka in Action'**, authors: [ **'auth1'** ] } |
| { \_id: **'b2'**, |
| bookName: **'micro services in Action'**, |
| authors: [ **'auth1'**, **'auth2'** ] } |

Now for both documents we have authors field instead of having documents we have references

Embedded Doc apprch vs separate coll 

Lookup-Join/Merging the tables when we use relations

When we use separate tables like author, books as we have stored the data in separate tables and by using references we need to merge so use lookup in aggregations

db.books.find();

|  |
| --- |
| { \_id: **'b1'**, bookName: **'kafka in Action'**, authors: [ **'auth1'** ] } |
| { \_id: **'b2'**, |
| bookName: **'micro services in Action'**, |
| authors: [ **'auth1'**, **'auth2'** ] } |

db.authors.find();

|  |
| --- |
| { \_id: **'auth1'**, age: 29, name: **'rod johnson'**, address: **'USA'** } |
| { \_id: **'auth2'**, age: 45, name: **'Gavin king'**, address: **'UK'** } |

Now we can merge using aggregations



Db.books.aggregate( [

{

$lookup: {

From:’ authors’, //indicates we have to merge with authors table

localField:’authors’, // says in current query is querying from table books and use the field called authors

foreignField:’\_id”,// means from authors collection we have to use field called \_id to merging FKey

as: ‘MergedData’ // after merged this is the name of merged field

}

] );

**Plain Query**

db.books.aggregate([{$lookup:{from:'authors',foreignField:'\_id',localField:'authors' , as:'MergedData' } }]);

**QueryResult:-**

|  |
| --- |
| { \_id: **'b1'**, |
| bookName: **'kafka in Action'**, |
| authors: [ **'auth1'** ], |
| **MergedData**: [ { \_id: **'auth1'**, age: 29, name: **'rod johnson'**, address: **'USA'** } ] } |
| { \_id: **'b2'**, |
| bookName: **'micro services in Action'**, |
| authors: [ **'auth1'**, **'auth2'** ], |
| **MergedData**: |
| [ { \_id: **'auth2'**, age: 45, name: **'Gavin king'**, address: **'UK'** }, |
| { \_id: **'auth1'**, age: 29, name: **'rod johnson'**, address: **'USA'** } ] } |

Examples for embedded Doc approach

**1- Many –Embedded approach**- 1 Facebook post will have many comments

Here 1- Many relationship but still Comments is not a separate collection it’s an embedded approach collection. So comments is an array,

Why embedded collection even on 1- many??

1. We are not going to fetch comments alone we always fetch with the post so they are altogether and cannot be separated, so why to separate them if u store them in separate collection again while fetching u have to join them, so prefer embedded collection approach and if post is deleted comments should be automatically deleted automatically and

Each Comments should have an embedded user id, so refer it

Assignment:- post, user ,comments , post comments are in same collection so embedded approach,

**1-Many separate collection approach**

User- posts , 1 user will have many posts- 1-many and mostly we are interested in fetching child documents posts so why to make them embedded document approach, make it separate approach ,

Collection constraints

Generally in mongo there are no constraints every row/document can have diff columns and different datatypes

But still if u want some constraints like for each document if u need to specify some mandatory field you can do that in mongo refer the js file attached

Sample query