DATA RELATIONSHIPS IN MONGODB

Lecturer: Trần Thế Trung

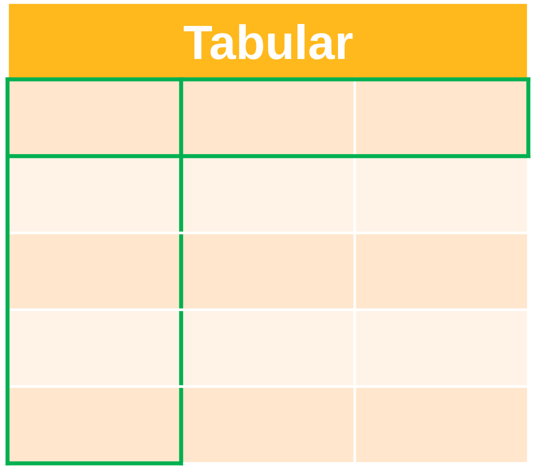
1. Introduction to Relationships

In this chapter, we will cover auto express relationships with MongoDB.

Even if MongoDB is classified as a document database, the pieces of information stored in the database still have relationships between them.

Old style databases are often referred to as relational databases.

However, NoSQL databases are also relational, so a better name for those legacy databases is Tabular, which highlights a notion of columns and tables.



1. Introduction to Relationships

Understanding how to represent relationships, and deciding between embedding and linking relationship information, is crucial.

Having a good model is the single most important thing you can do to ensure you get good performance. The face of identifying and modeling relationships correctly is a step that is not optional in the methodology.

This chapter will give you the knowledge you need to do it right.

Embedding
&
Linking

1. Introduction to Relationships

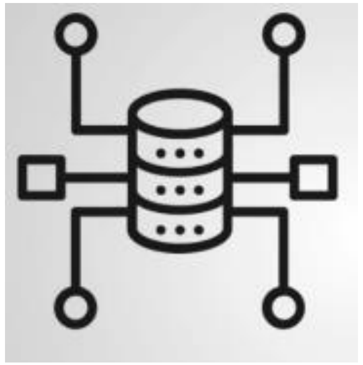
What are relationships in the data model?

If you look at any schema implementation in MongoDB, or any other database, you can observe objects, referred to as entities.

The relationships represent all the entities and the other piece of information are related to each other.

We will start with the different types and cardinalities that describe our relationships, then we will answer one of the essential question when modeling unit of information for MongoDB.

Should the information be embedded or referenced?



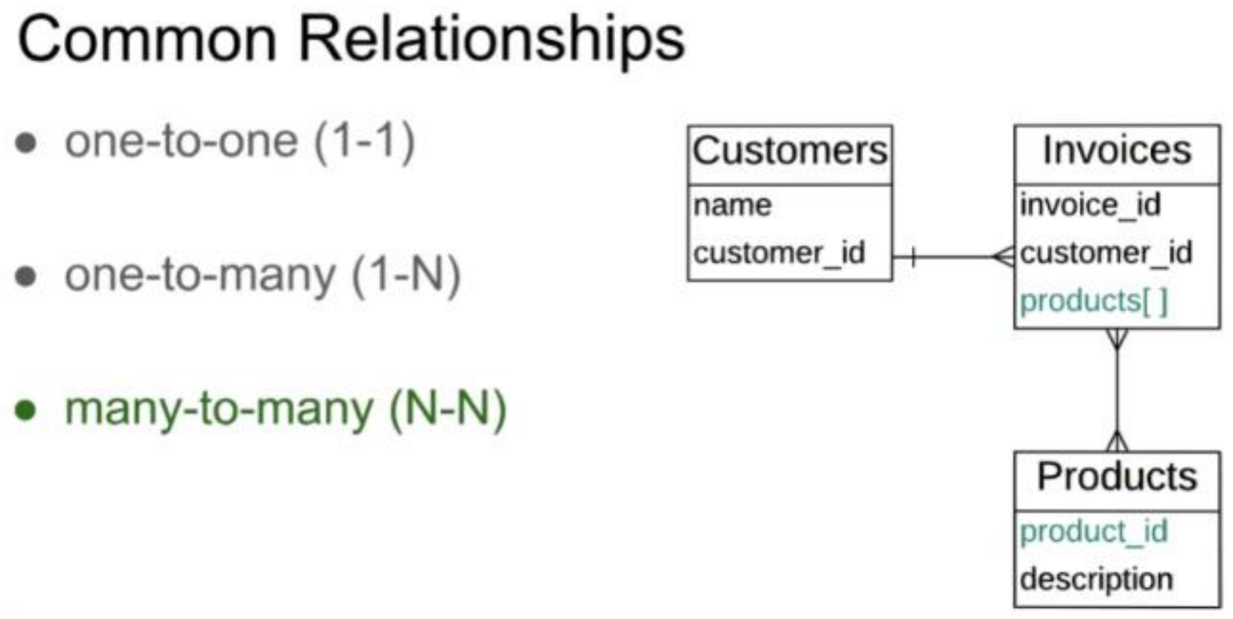
1. Introduction to Relationships

Tabular/RDBMS	Document/MongoDB
Entity	Entity
Relationship	Relationship
Database	Database
Table	Collection
Attribute/Column	Field
Record/Row	Document
Join	Embed or Link

2. Relationship Types and Cardinality

In this lesson, we'll examine the type and cardinality of relationships.

Most of the relationships between units of information can be classified as one-to-one, one-to-many, or as many-to-many.


Common Relationships

- one-to-one (1-1)
- one-to-many (1-N)
- many-to-many (N-N)

However, is this the best and complete way to describe data relationships, especially when dealing with Big Data?

2. Relationship Types and Cardinality

… relationships between

A mother and her children?

- 0, 1, 2, 10 … more?
- Most likely 2

A person and her Xer followers?

- 0, 1, …, 20, may be 100 millions
- Most likely 20

In this case, many-to-many is a very poor way to characterize a relationship.

And this might be true for an increasing number of examples in the world of Big Data.

We could embed the information about the children in the document representing the mother, but it would not make sense to embed 100 million followers into one document.

What we need is a more expressive way to represent the one-to-many relationship so that we know that we are dealing with large numbers and avoid mistakes associated with that distinction.

2. Relationship Types and Cardinality

Looking at earlier examples, we are missing some information.

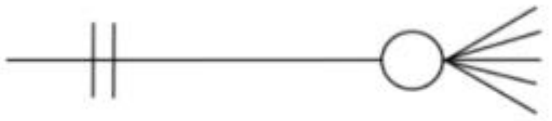
The fact that the relationship can be a large number isn't reflected clearly in the one-to-many description, the <u>value for the maximum of many is not clear</u>. <u>The most likely many value for a given one-to-many relationship is also missing</u>.

Let's introduce this additional symbol for the crow foot notation, and call it zillions. It is based on the many symbols, however, with additional lines.

This relationship would read as from one to zero to zillion. Or in short, one-to-zillions.

Modified Notations

- Crow Foot Notation
  - one-to-zillions



- Numeral Notation
  - minimum
  - most likely, median
  - maximum

[min, likely, max]

2. Relationship Types and Cardinality

This new symbol addresses the identification of large numbers. And if we go to the trouble of identifying the maximum number, why not preserve this information in the model?

For this we use a tuple of one to three values, with the following meaning.

Minimum, usually zero or one, most likely value, or the median, and the maximum.

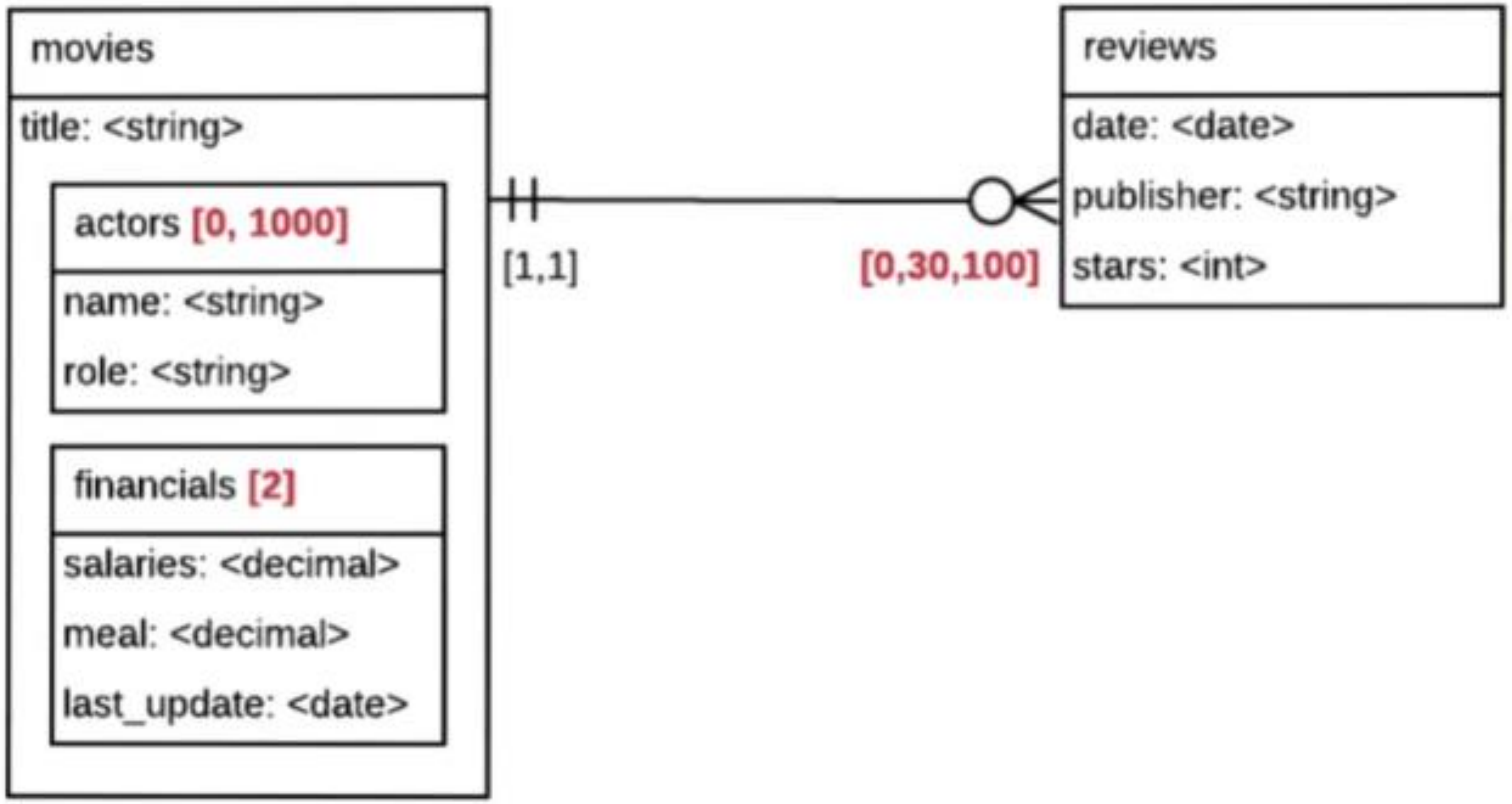
If you have two values they represent the minimum and the maximum.

When a single value is used it means the relationship is fixed to that number.

We will use this notation to describe either arrays or reference between collections in our models.

2. Relationship Types and Cardinality

Example



2. Relationship Types and Cardinality

Recap

- One-to-one, one-to-many, many-to-many are the usual cardinalities.

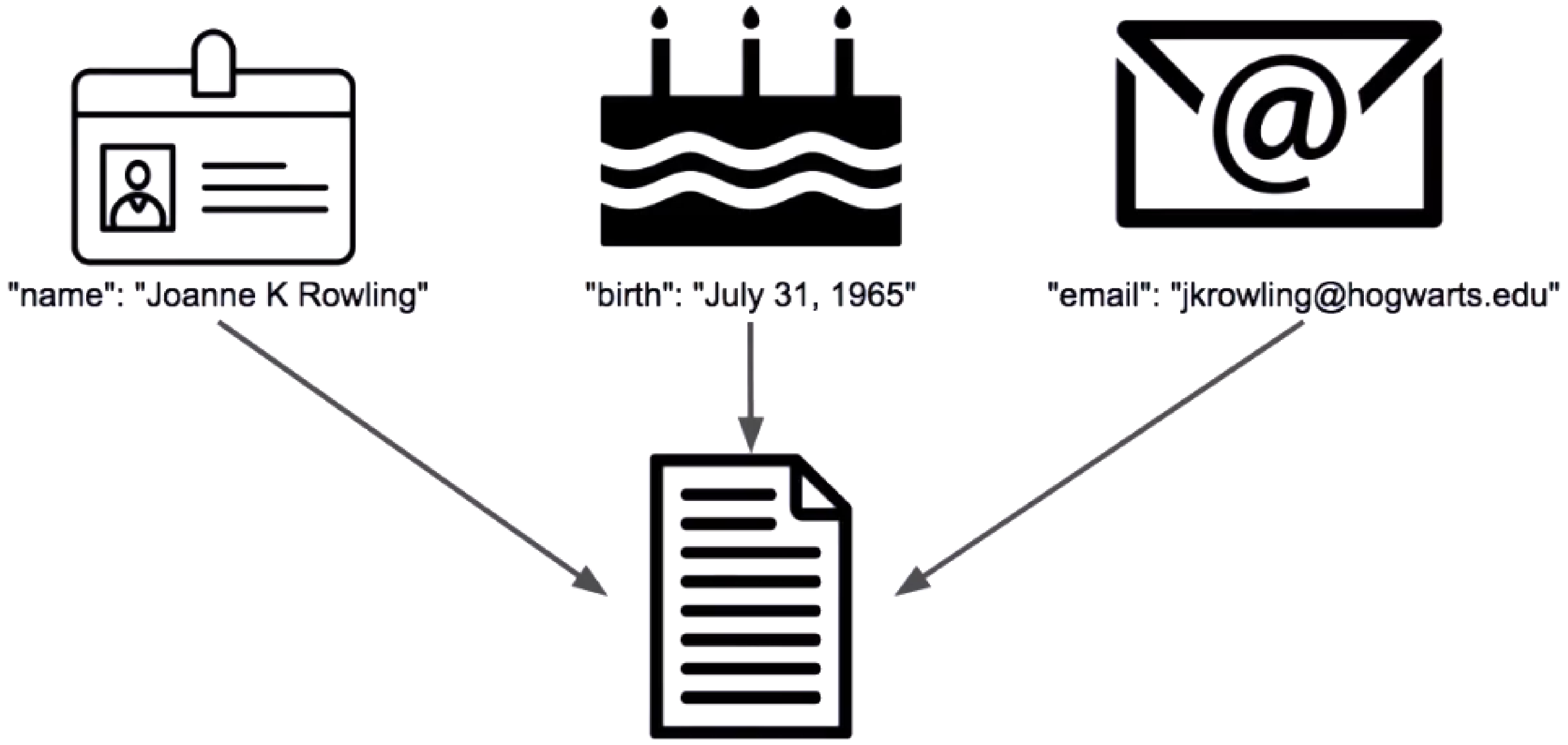
- One-to-zillions is useful in the Big Data World

- Even better, use “maximum” and “most likely” values using a tuple of the form:

  [min, likely, max]

3. One-to-one Relationship

Commonly, a one-to-one relationship is represented by a single table in a tabular database. In general, the same applies to MongoDB.



3. One-to-one Relationship

1. **Embed**
   a. Fields at same level;
   b. Grouping in sub-documents

2. **Reference**
   a. Same identifier in both documents;
   b. In the main “one” side;
   c. In the secondary “one” side.

3. One-to-one Relationship

When we group information together, that is in two different entities, we refer to this action as embedding. This is in contrast to grouping fields together in a given entity. We refer to those fields as attributes of the entities.

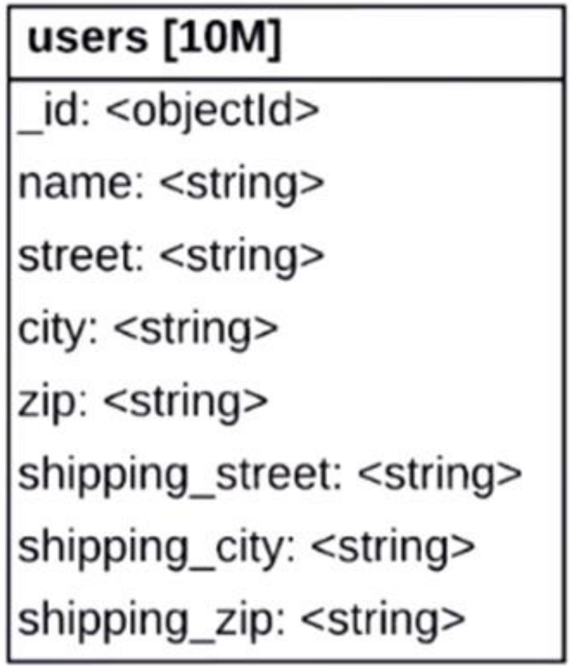
Or we can use the document model ability to add sub-documents to create logical groups of information. This capability also allows us to embed the document or entity inside another one.

Alternatively to embedding the information, we can divide the fields into many documents, usually in separate collections and reference one document from another one.

3. One-to-one Relationship

Example

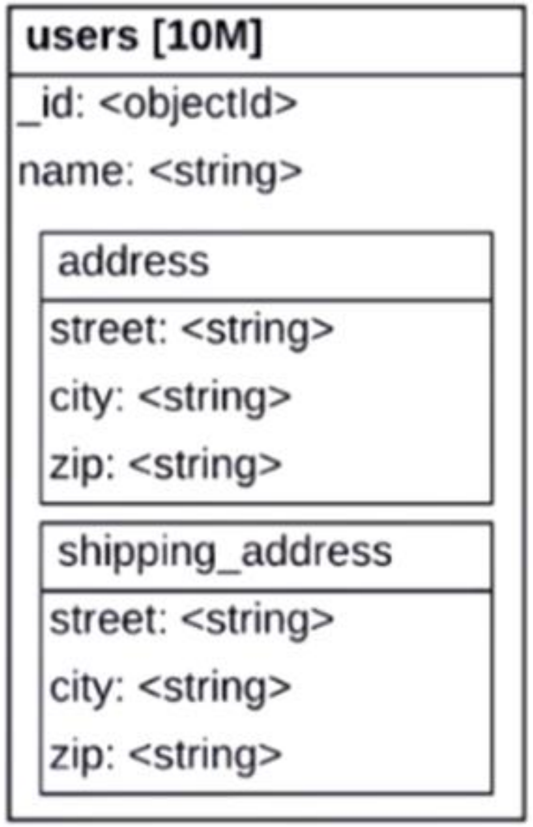
One-to-One: embed, fields at the same level


users [10M]

_id: <objectId>
name: <string>
street: <string>
city: <string>
zip: <string>
shipping_street: <string>
shipping_city: <string>
shipping_zip: <string>

very similar to tabular database

One-to-one: embed, using subdocuments


users [10M]

_id: <objectId>
name: <string>

address
street: <string>
city: <string>
zip: <string>

shipping_address
street: <string>
city: <string>
zip: <string>

preferred representation:

- *preserves simplicity;*
- *documents are clearer.*

3. One-to-one Relationship

Example

Using the user's collection as an example, I would keep track of a given user, the address we use for billing, and the default shipping address if different.

A user as only one street address, city, and zip code for billing and only one street address, city, and zip code used as the default shipping address.

This address information may profit from a little bit more organization. Using the power of the document model, regroup each set of address information into a subgroup.

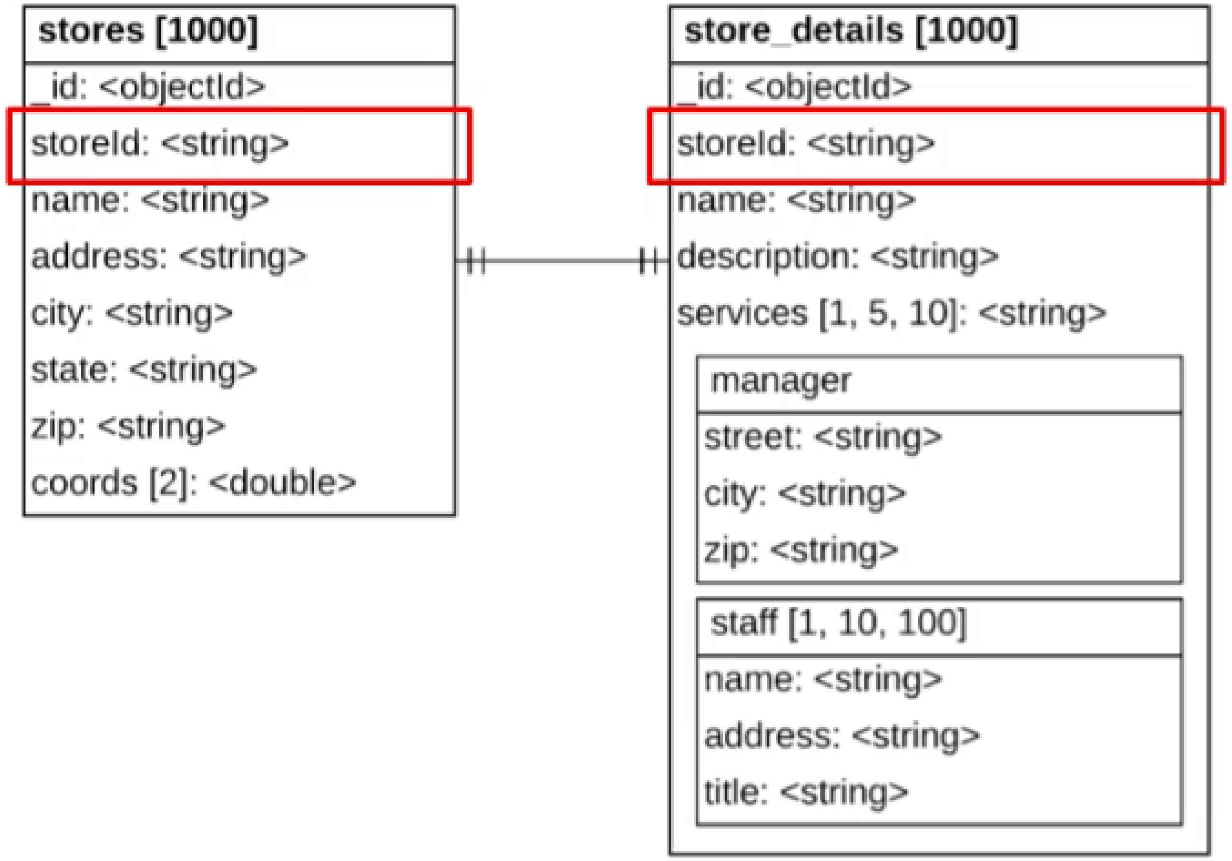
Now, we still have a one-to-one relationship, however, we can't describe it as a user having one and only one billing address, and having one and only one shipping address.

The document is now more clear and easier to understand.
This is the recommended representation of a one-to-one relationship.

3. One-to-one Relationship

Example

One-to-One: reference



- add complexity
- possible performance improvements with:
  - smaller disk access
  - smaller amount of RAM needed

3. One-to-one Relationship

Example

In this example, the list of staff employees is kept within the store document.

If most of the time we create the stores, we don't care about the staff information, we could separate out this set of information, placing it in a second collection.

Consequently this adds some complexity to our model. So we should only do it for schema optimization reasons.

Once we have retrieved a given store, we would find additional information like the manager and staff by querying the store details collection using the link to the corresponding document, as we do for any relation expressed by a reference.

3. One-to-one Relationship

Example

In the case of a one-to-one relationship, it is easy to simply use the same value, here our store ID in both documents.

We also want to prevent this one-to-one relationship from becoming a one-to-many relationship for subdocuments.

In order to do so, we need to ensure that the values in our store ID field are unique for both collections

3. One-to-one Relationship

Example

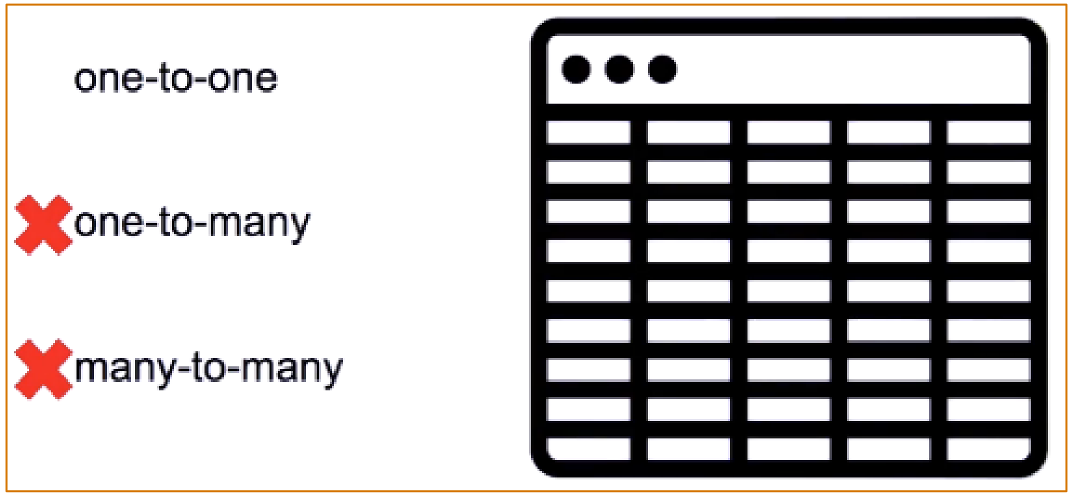
Recap

- Prefer embedding over referencing for simplicity.
- Use subdocuments to organize the fields
- Use a reference for optimization purpose

4. One-to-many Relationship

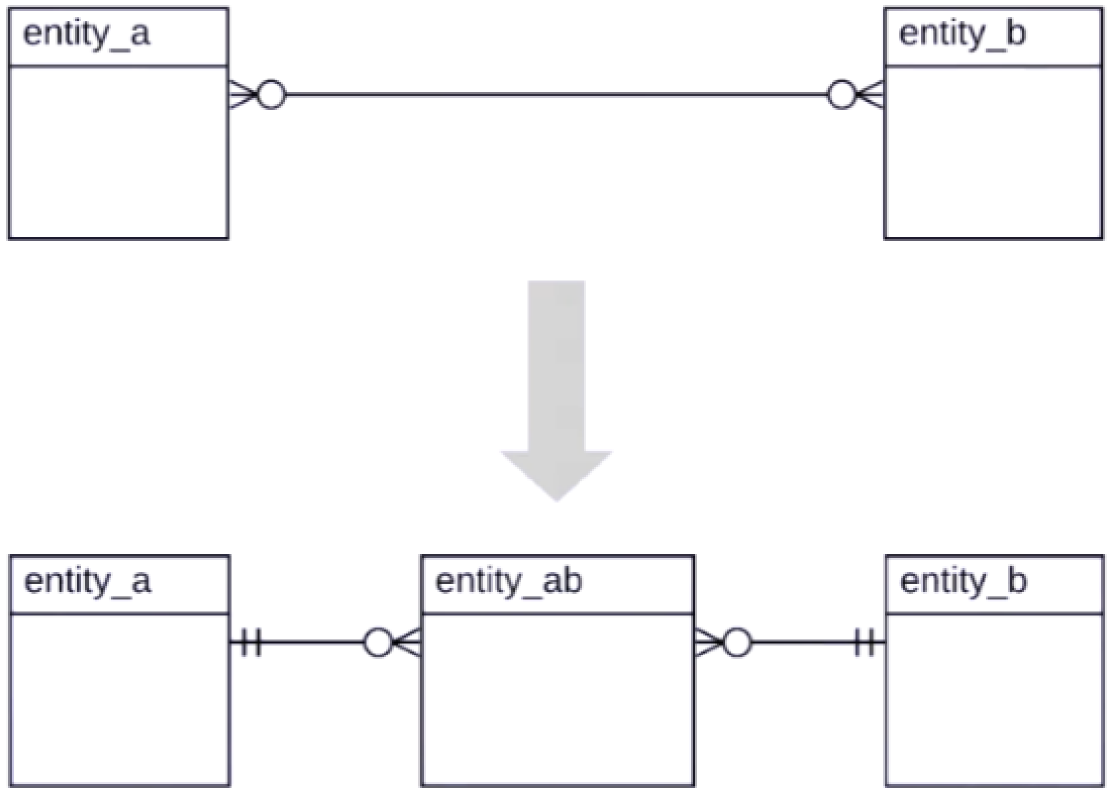
The most interesting type of relationship is the one-to-many relationship.

First, because if all our data is only composed of one-to-one relationships, a spreadsheet application like Excel could do the job, at least for a small data set.



4. One-to-many Relationship

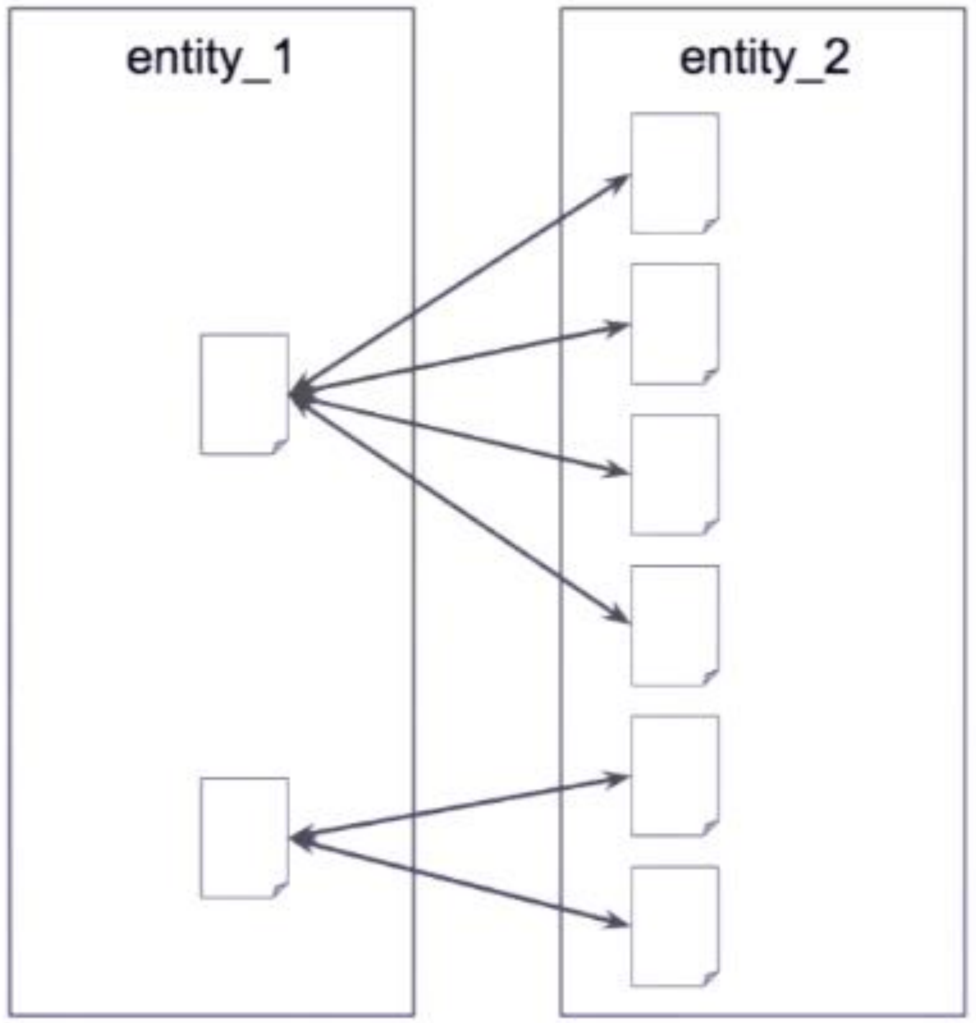
As for the many-to-many relationships, most of them can be expressed as two one-to-many relationships.



We will describe this in more detail in the many-to-many relationship lesson.

4. One-to-many Relationship

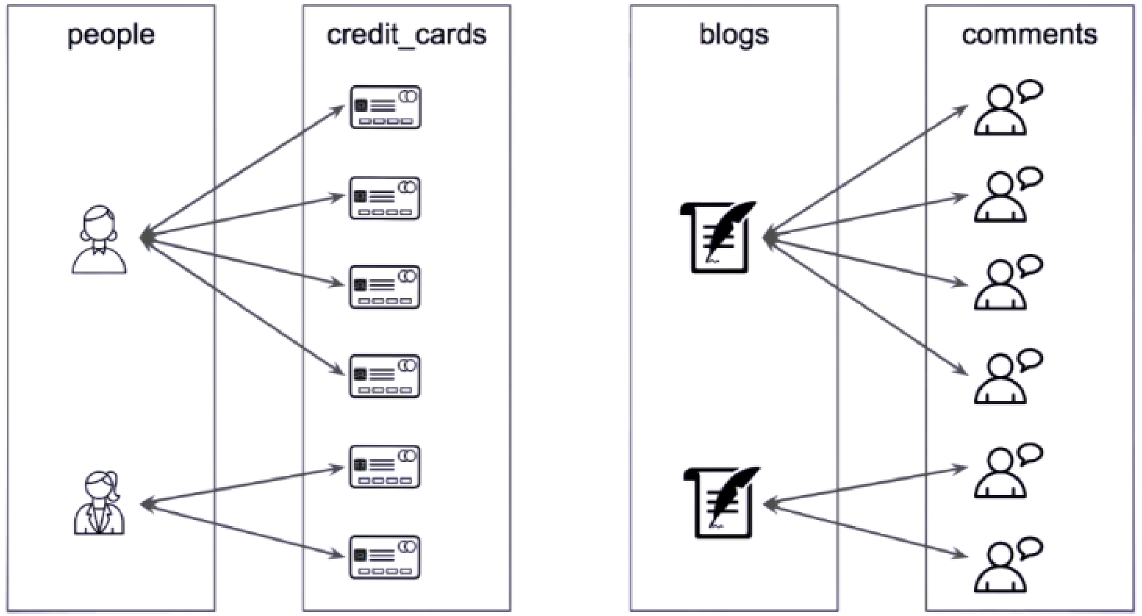
A one-to-many relationship means that an object of a given type is associated with n objects of a second type, while the relationship in the opposite direction means that each of the objects of the second type can only be associated with one object on the one side.



4. One-to-many Relationship

As an example of this relationship, we use a person and their credit cards, or a blog entry and its comments.

A person has n credit cards, but each of these credit cards belongs to one and only one person.



Using MongoDB and its document model, give us a few ways to represent this kind of relationship.

4. One-to-many Relationship

1. **Embed**
   - in the “one” side;
   - In the “many” side;

   Usually, embedding in the entity the most queried

2. **Reference**
   - in the “one” side;
   - in the “many” side.

   Usually, referencing in the “many” side

4. One-to-many Relationship

We can embed each document from the many sides into the document on the one side.

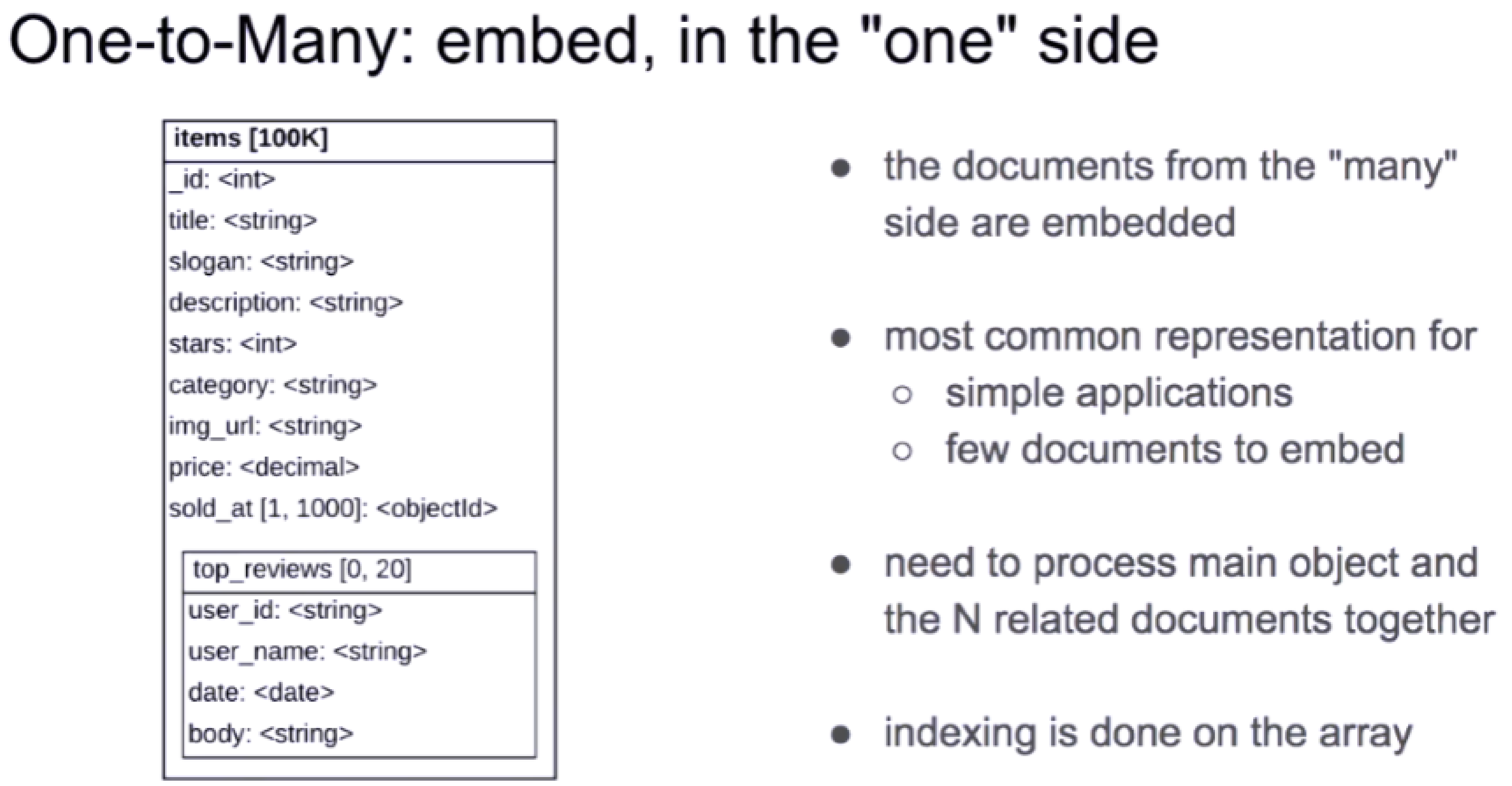
Or vice versa, we can embed the document from one side into each document on the many side.

Instead of using a single collection and embedding the information in it, we keep the documents in two separate collections and reference documents from one collection in documents of the other collection.

let's have a look at all of these representations of the one-to-many relationship

4. One-to-many Relationship

The first representation embeds the end documents as an array of sub documents.



One-to-Many: embed, in the "one" side

items [100K]
_id: <int>
title: <string>
slogan: <string>
description: <string>
stars: <int>
category: <string>
img_url: <string>
price: <decimal>
sold_at [1, 1000]: <objectId>

top_reviews [0, 20]
user_id: <string>
user_name: <string>
date: <date>
body: <string>

- the documents from the "many" side are embedded
- most common representation for
  - simple applications
  - few documents to embed
- need to process main object and the N related documents together
- indexing is done on the array

4. One-to-many Relationship

In our product catalog, we keep the top reviews of an item within the item itself because we want to display these reviews once the item gets retrieved from the database.

In other words, items are the main objects we access.

And when we do so, we want to bring in all the associated top reviews with them.

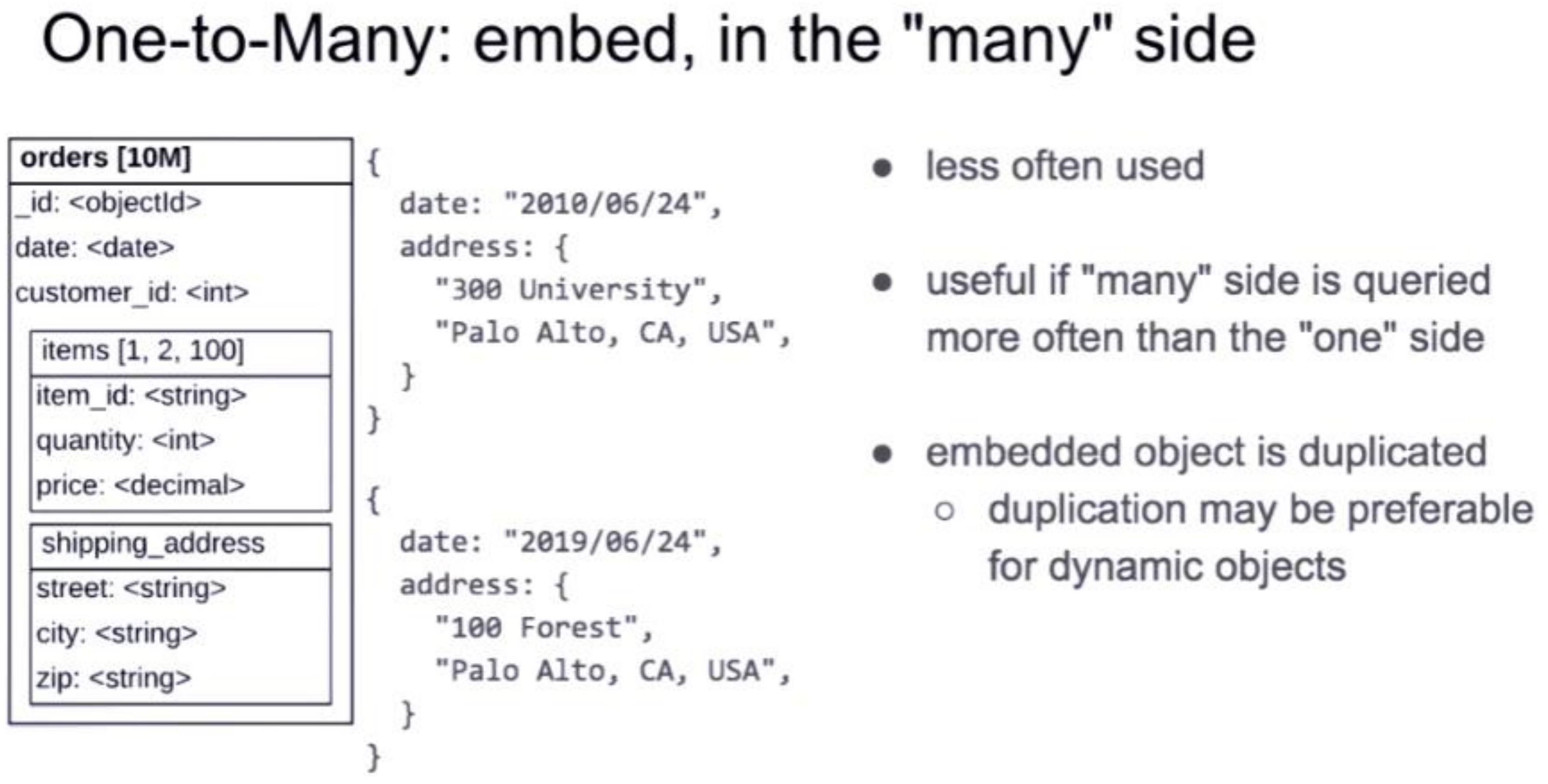
For simple applications where the number of embedded documents is small, this is the most common representation.

The information that is needed together stays together.

As for quartering on the many side, which are now embedded, we use multi-key indexes, which are designed for indexing values in array fields.

4. One-to-many Relationship

The second representation of one-to-many relationships is to embed the document from the one side in each of the documents associated with it from the many side.



One-to-Many: embed, in the "many" side

orders [10M]
_id: <objectId>
date: <date>
customer_id: <int>

items [1, 2, 100]
item_id: <string>
quantity: <int>
price: <decimal>

shipping_address
street: <string>
city: <string>
zip: <string>

```
{
  date: "2010/06/24",
  address: {
    "300 University",
    "Palo Alto, CA, USA",
  }
}

{
  date: "2019/06/24",
  address: {
    "100 Forest",
    "Palo Alto, CA, USA",
  }
}
```

- less often used
- useful if "many" side is queried more often than the "one" side
- embedded object is duplicated
  - duplication may be preferable for dynamic objects

4. One-to-many Relationship

A good example will be an address and the order's delivery to the address.

Over time, because many orders were delivered to the same address, we have a one-to-many relationship between address and orders.

Application are more likely to handle orders than trying to figure out everything shipped to a given location.

The access pattern's focus is on orders, not addresses.

As a result, it makes more sense for us to store the address, the one side of the relationship, on every order, the many sides of the relationship, rather than the other way around.

This representation is less often used.

4. One-to-many Relationship

The main disadvantage of this representation is that the embedded object must be duplicated in many locations.

However, the nature of the embedded information is static hardly ever changes such as the shipping address of the order.

It is, in fact, better to store it in the order document itself.

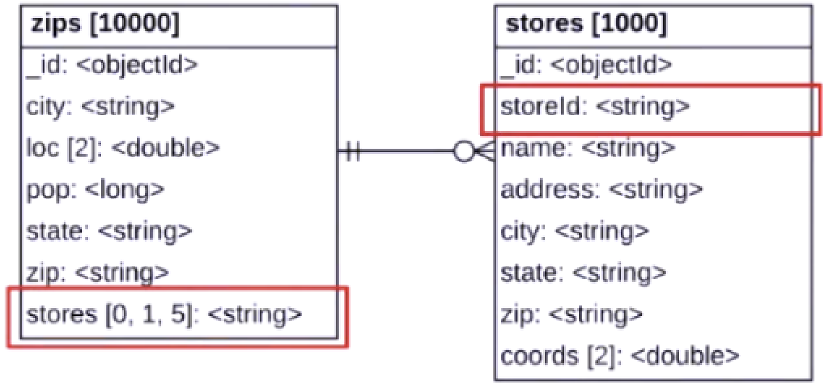
If the address for a given customer changes in the future, it does not that affect all the previous orders and where they got delivered.

4. One-to-many Relationship

The third representation is to have two collections.

From the one side, we reference the many side. To do so, we need an array of references.

One-to-Many: reference, in the "one" side



```
{
  ...
  stores:
     [ "store1", "store408", "store441" ],
  ...
}
```

- Array of references
- Allows for large documents and a high count of these
- List of references available when retrieving the main object
- Cascade deletes are not supported by MongoDB and must be managed by the application

4. One-to-many Relationship

In our zips collection, which contain our zip codes, we create an array of stores where each element in the array is a store ID value that identifies documents in the store's collection.

The referencing representations are great.

If we can't or don't need to embed the remaining entity information, we get to know all the references, the stores, without making a second query on the store's collection.

If the entries in the array are descriptive enough, we are saving ourselves some queries.

4. One-to-many Relationship

However, in this representation, we need to keep in sync the area references and their reference documents.

If we don’t need the reference document, we must also delete the reference to this document.

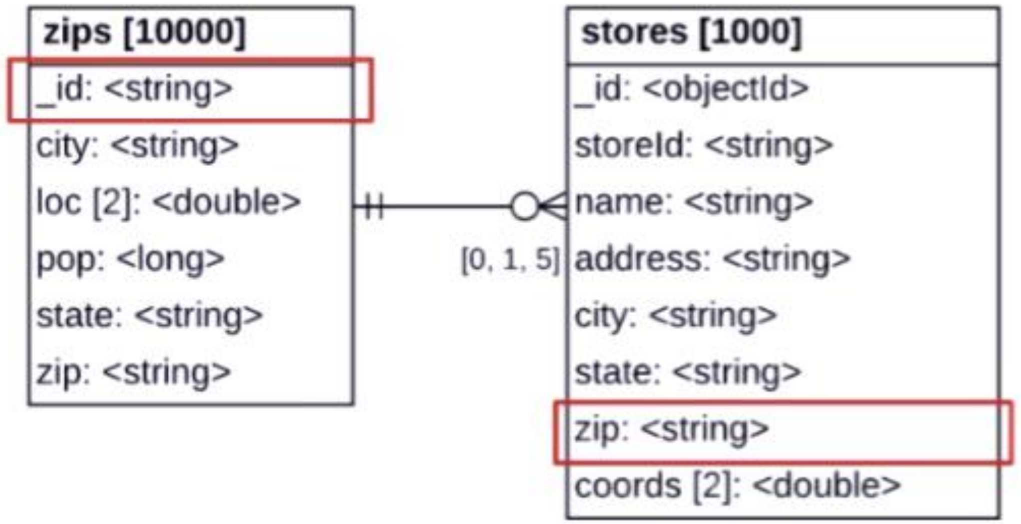
MongoDB does not currently support foreign keys or cascade deletes, therefore there is no way for all references to automatically be removed on document deletion.

The application must perform all of this sort of data management.

4. One-to-many Relationship

More commonly, references are stored in the documents on the many side of a one to many relationship

One-to-Many: reference, in the "many" side



- preferred representation using references
- Allows for large documents and a high count of these
- No need to manage the references on the "one" side.

4. One-to-many Relationship

For example, we may have a collection of zip codes, each zip code adding possibly many stores in it.

By adding a single field called zip in each of my stores documents, I can reference the document in the zips collection.

Compared to the previous representation, if we <u>delete a store, there is no additional reference to remove</u> because the reference is inside the document we are removing.

4. One-to-many Relationship

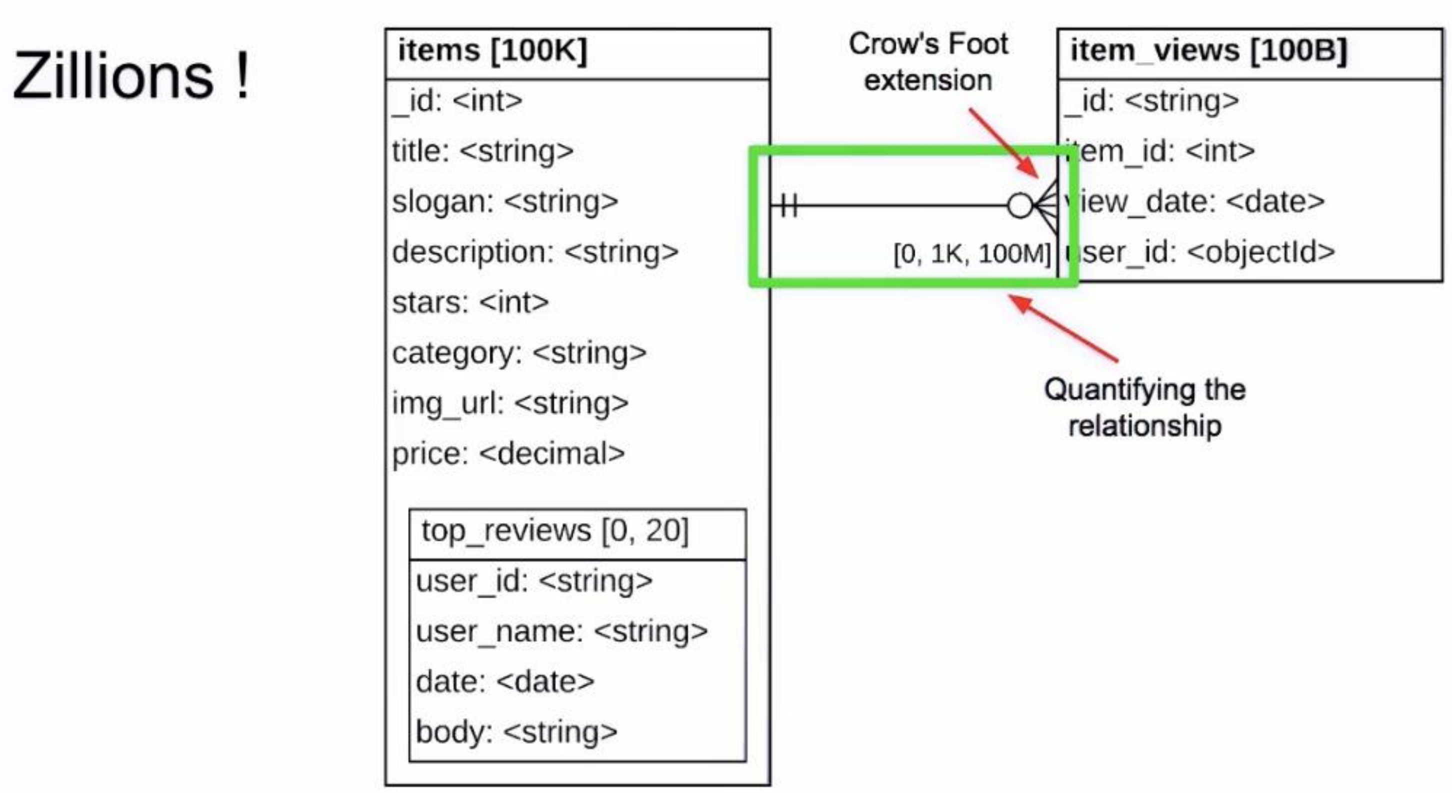
Recap

- ✓ There are a lot of choices: embed or reference and choose the side between "one" and “many”.
- ✓ Duplication may occur when embedding on the “many” side. However, it may be OK, or even preferable
- ✓ Prefer embedding over referencing for simplicity, or when there is a small number of referenced documents as all related information is kept together
- ✓ Embed on the side of the most queried collection
- ✓ Prefer referencing when the associated documents are not always needed with the most often queried documents

5. One-to-zillions Relationship

In previous lessons, we used the word zillions, and introduced a graphical notation for representing the one-to-zillions relationships.

We extended the crow's foot notation by adding fingers the foot in order to easily see those zillions sides.



5. One-to-zillions Relationship

We should also use the notation used in this course for cardinalities.

One thing is to identify a relationship as one to zillions, but better still if you can quantify that same relationship.

What is the minimum number of associated document? The most likely number and the maximum number. The maximum number is what we care the most about this relationship.

Zillions means something is humongous, out of proportion, so watch out for it.

The one-to-zillions relationship is not that different relationship, per se. It is a subcase of the one-to-many relationship.

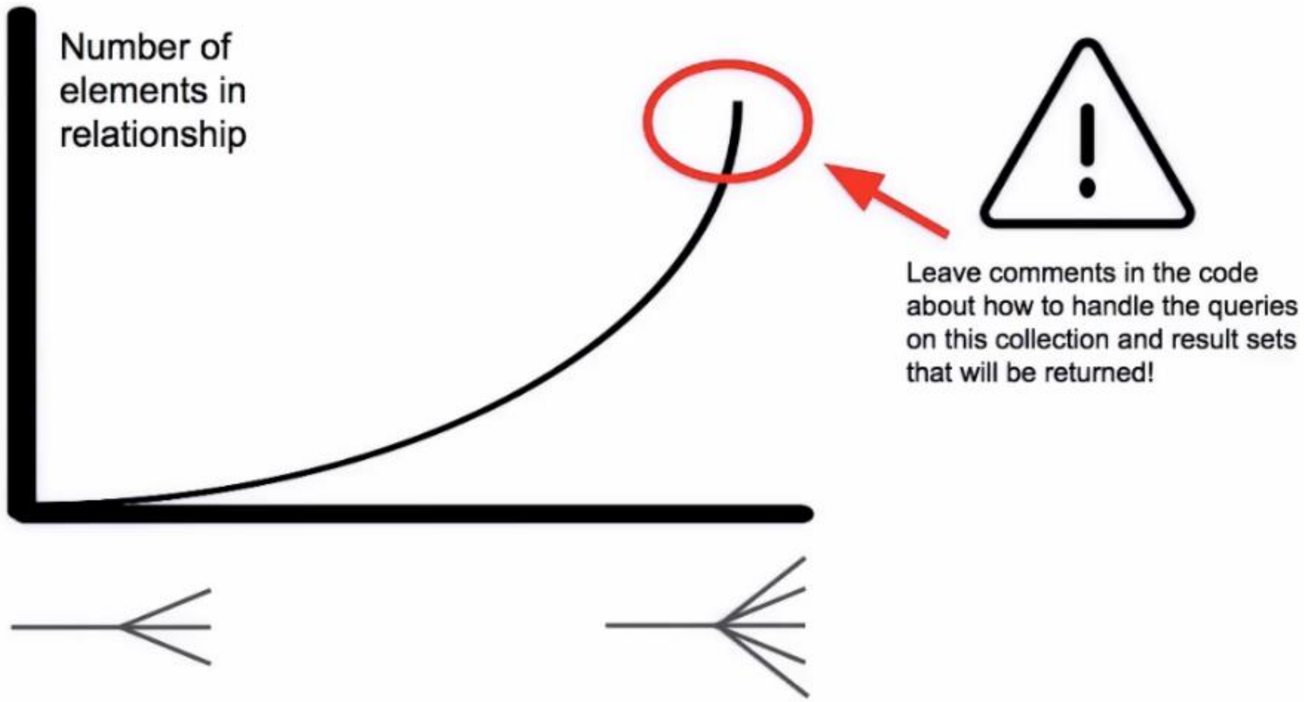
If we have a one-to-many relationship and the many identified as 10,000 or more, we call that relationship one to zillion.

5. One-to-zillions Relationship

The last thing you want the application to do is to retrieve a document and its zillions associated documents, then process the complete results set.

So comment those sections of the code with directives on how to handle or reduce the result set that comes back.

Given the cardinality of these relationships and the pressure on computational resources to process them, you need to be on the lookout for very large arrays of subdocuments or unbound arrays of references.


Number of elements in relationship
Leave comments in the code about how to handle the queries on this collection and result sets that will be returned!

5. One-to-zillions Relationship

1. **Embed**
   - ~~*in the “one” side;*~~
   - ~~*in the “many” side;*~~

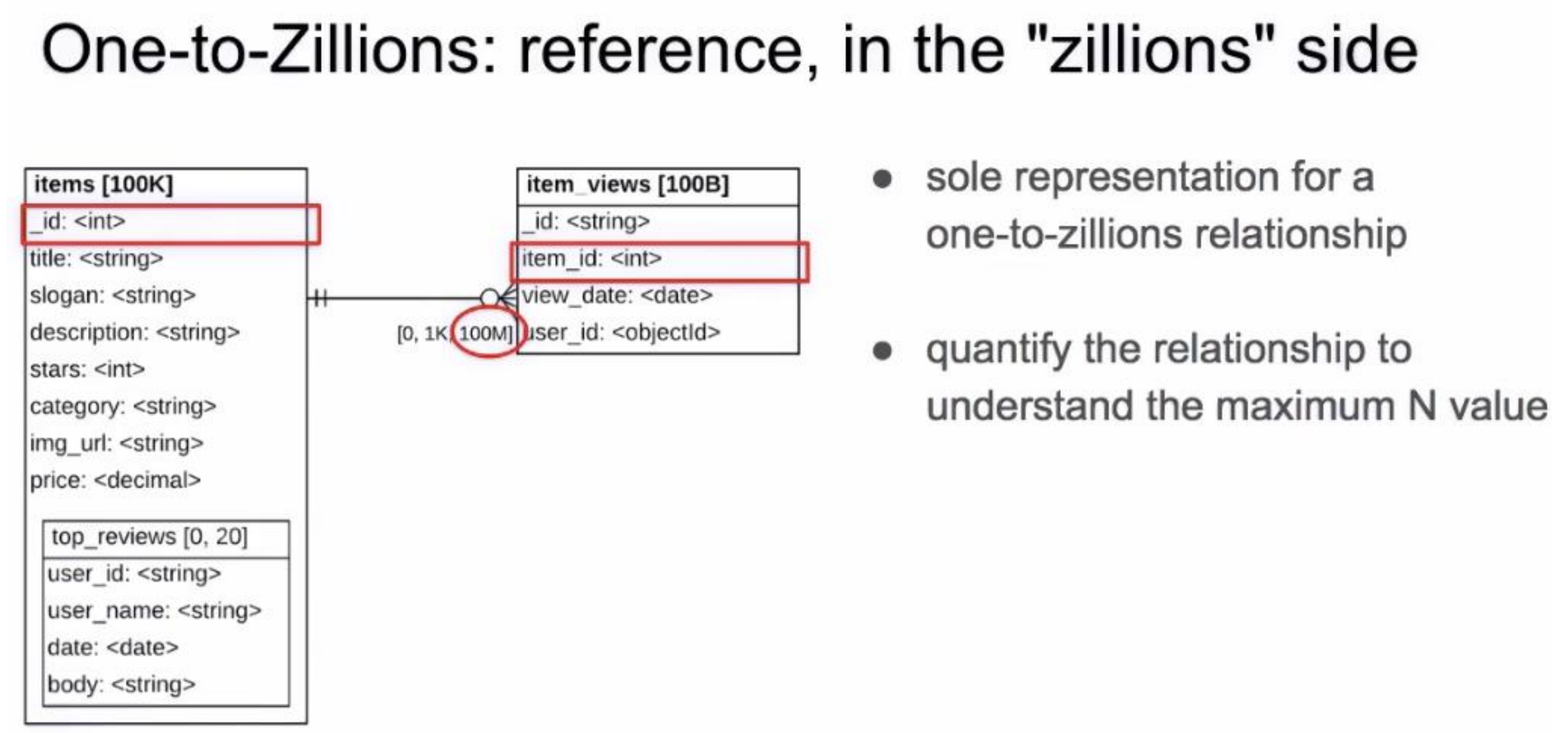
2. **Reference**
   - ~~*in the “one” side;*~~
   - in the “many/zillions” side.

Looking back at the representation for the one-to-many relationships, we have a single one left the representation where we referenced the document on the one side of the relationship from the many or zillion side.

5. One-to-zillions Relationship

As you model this relationship, make an effort to quantify it.

This will become important, so that you can understand the impact of the maximum value on your resources.



One-to-Zillions: reference, in the "zillions" side

- sole representation for a one-to-zillions relationship
- quantify the relationship to understand the maximum N value

5. One-to-zillions Relationship

Recap

- ✓ Its a particular case of the one-to-many relationship.
- ✓ The only available representation is to reference the document on the “one” side of the relationship from the “zillion" side.
- ✓ Pay extra attention to queries and code that handle "zillions" of documents.

6. Many-to-Many Relationship

Here's the most complicated relationship.

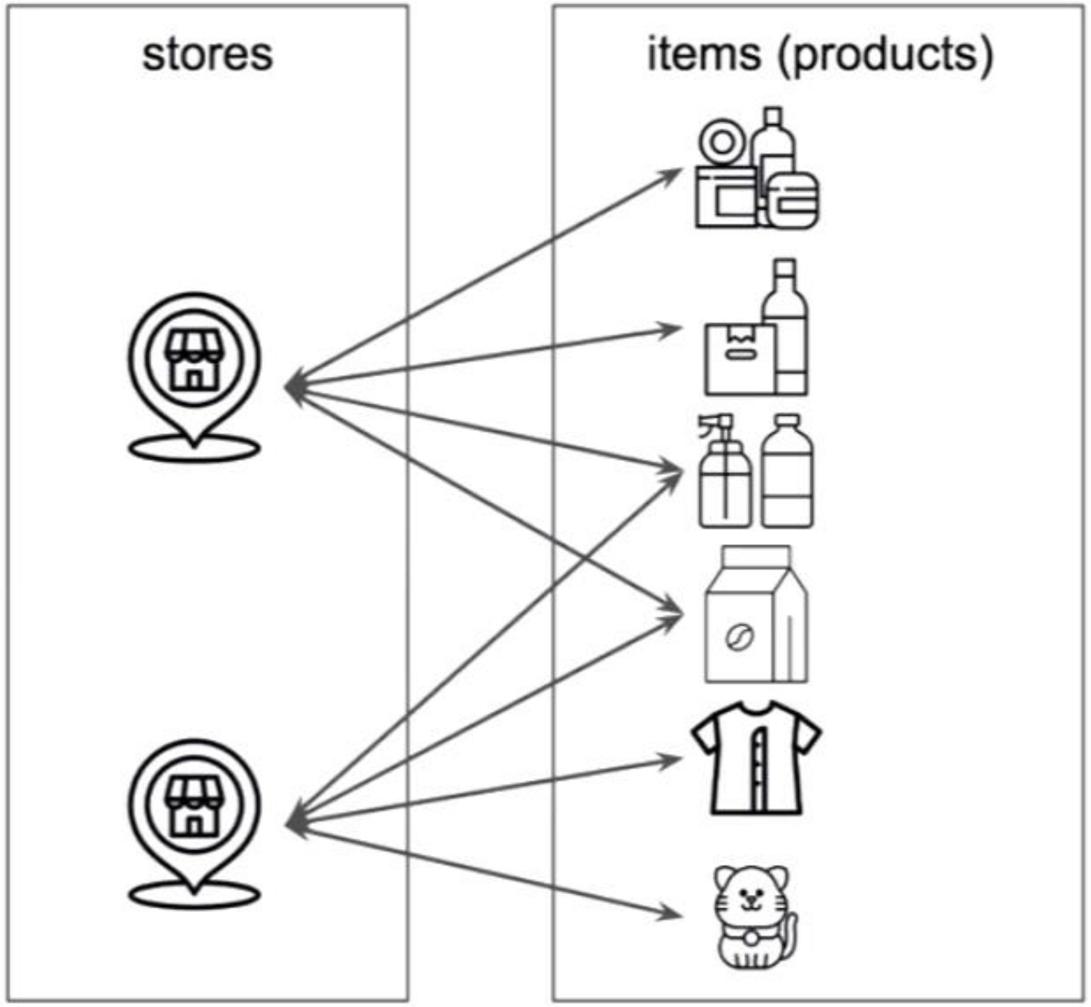
The many-to-many relationship is identified by documents on the first side being associated with many documents on the second side, and documents on the second side being associated with many documents on the first side.

6. Many-to-Many Relationship

- The many-to-many relationship is identified by documents on the first side being associated with many documents on the second side, and documents on the second side being associated with many documents on the first side.

- Looking at products sold in our stores, we can see that the given store sells many items, and each item is sold in many stores.

- This relationship can trick you into thinking it is a one-to-many relationship, if you only consider one side of the relationship.



6. Many-to-Many Relationship

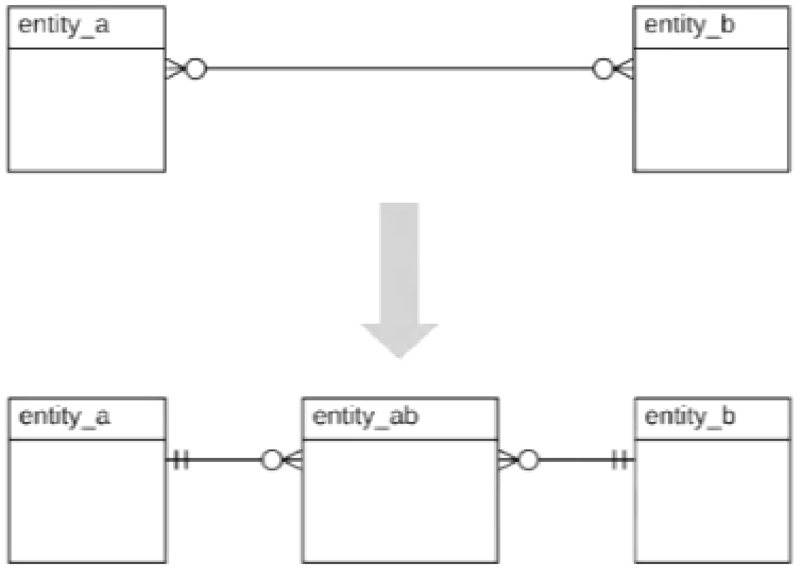
In a normalized relational model, you can't actually link two tables as many-to-many. However, some design tools can do the extra work for you.

Under the hood, an additional relationship table needs to be created to define this relationship, sometimes referred as a jump table.

It results in breaking up the many-to-many relationship into two one-to-many relationships linked together by our extra third table.

6. Many-to-Many Relationship

Many-to-Many => 2 x One-to-Many



actor_id	actor_name
actor1	Marlon Brando
actor2	Al Pacino
actor3	Robert DeNiro

movie_id	actor_id
movie1	actor1
movie1	actor2
movie2	actor2
movie2	actor3

movie_id	movie_title
movie1	The Godfather
movie2	The Godfather 2

6. Many-to-Many Relationship

For example, keeping track of the many-to-many relationship between actors and movies requires this table in between our actors and movies.

Misidentifying these types of relations means creating new tables and incurring data migration costs.

This type of modification, it's just not fun with traditional databases.

MongoDB's flexible schema easily allows for this type of schema modification, and is more forgiving, as scatter field can easily be transformed into an array field.

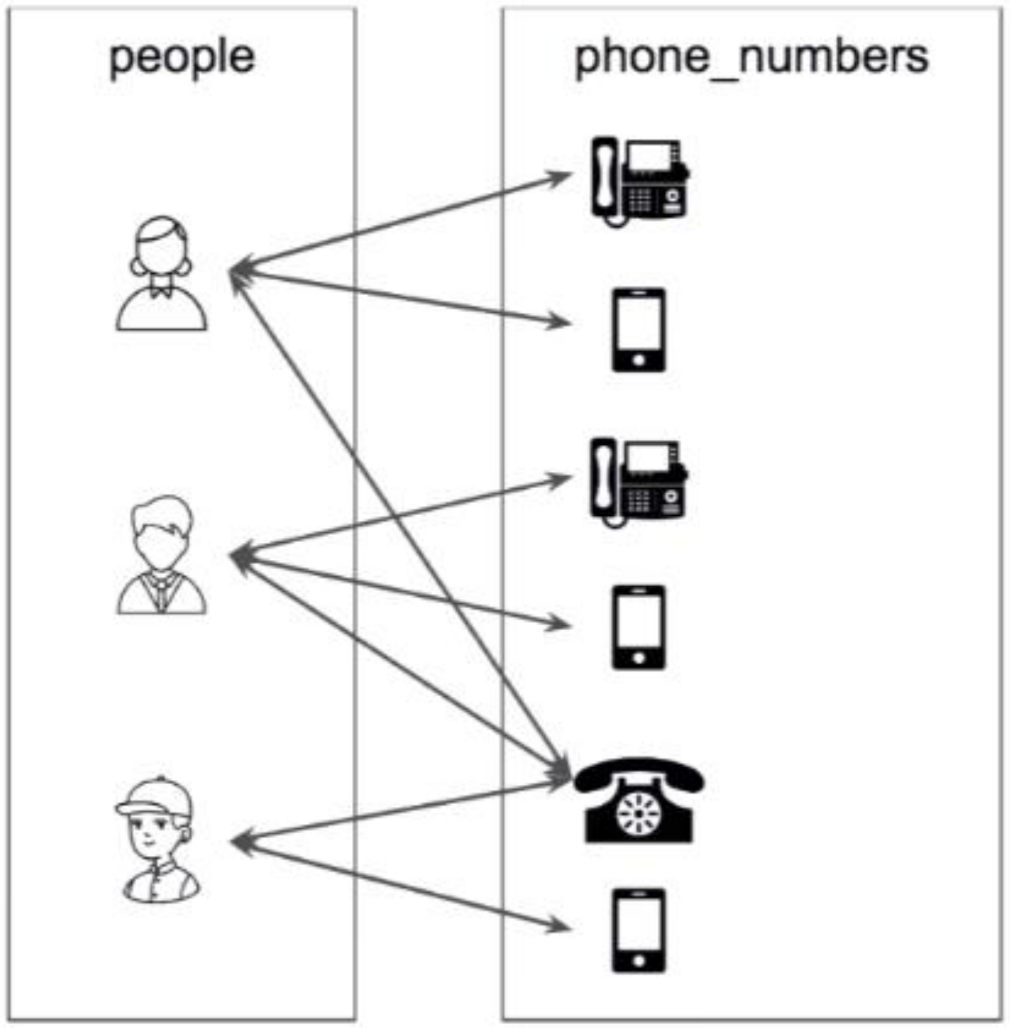
6. Many-to-Many Relationship

Let's take another example people and phone numbers. Someone may have a few phone numbers.

Some of these phone numbers are exclusive, and some are not.

A family's shares the home phone number.

So a person can have many phone numbers, and the phone number may be owned by many people, resulting in a many-to-many relationship.



6. Many-to-Many Relationship

We can treat the phone number for a home as uniquely owned by each member of the family by making copies of it.

Now, we have a one-to-many relationship instead, which removes complexity.

Consequences of this duplication, if the family moves, we must modify each family's members phone numbers separately.

Performing the same update multiple times may not sound like the right design, however, considered the previous design, where we store only one telephone number value in the database.

6. Many-to-Many Relationship

If someone moves like the child, for example and updates their phone number, the update will apply to all the members of the family.

We don't get the option to choose between doing multiple updates or one update that applies to all records.

Duplication is better, in some occasions, especially when you want to keep control of your data and when you want to avoid joining data.

What are the different ways in which we can model this many-to-many relationship? Similarly to the other relationships, we can embed the data from the other collection, or we can reference the other collection.

6. Many-to-Many Relationship

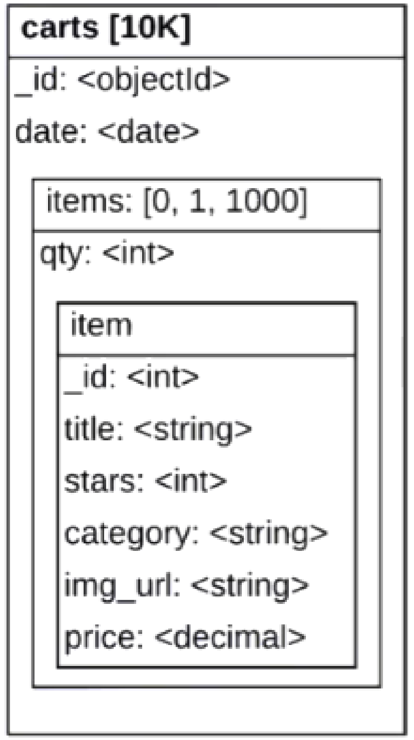
1. **Embed**
   a. array of subdocuments in the “many” side;
   b. array of subdocuments in the other “many” side;

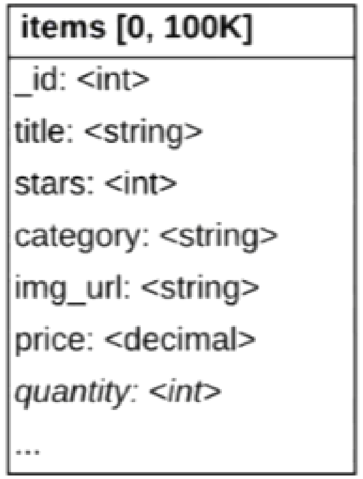
Usually, only the most queried side is considered

2. **Reference**
   a. array of references in one “many” side;
   b. array of references in the other one “many” side.

6. Many-to-Many Relationship

Let's use the carts and items.





- the documents from the less queried side are embedded
- results in duplication
- keep "source" for the embedded documents in another collection
- indexing is done on the array

6. Many-to-Many Relationship

The main entity is the cart in which we want to find the items, not the reverse.

We embed the items in the cart, because we always retrieve this information together.

Having copies of items in the carts period is usually fine, because they represent the state of those items at the time they were added to the cart.

The same applies to addresses and orders. The address used for that order at the time of the order creation should be duplicated.

Even when we embed the items in the carts, these items require a collection to hold their definition for several reasons.

There will be several other access patterns in your application that utilize items without carrying or needing the information on orders they have been added to.

6. Many-to-Many Relationship

Item documents have different life cycle than cart documents.

An item may exist without being in any carts, like an actor may be listed before being in a movie.

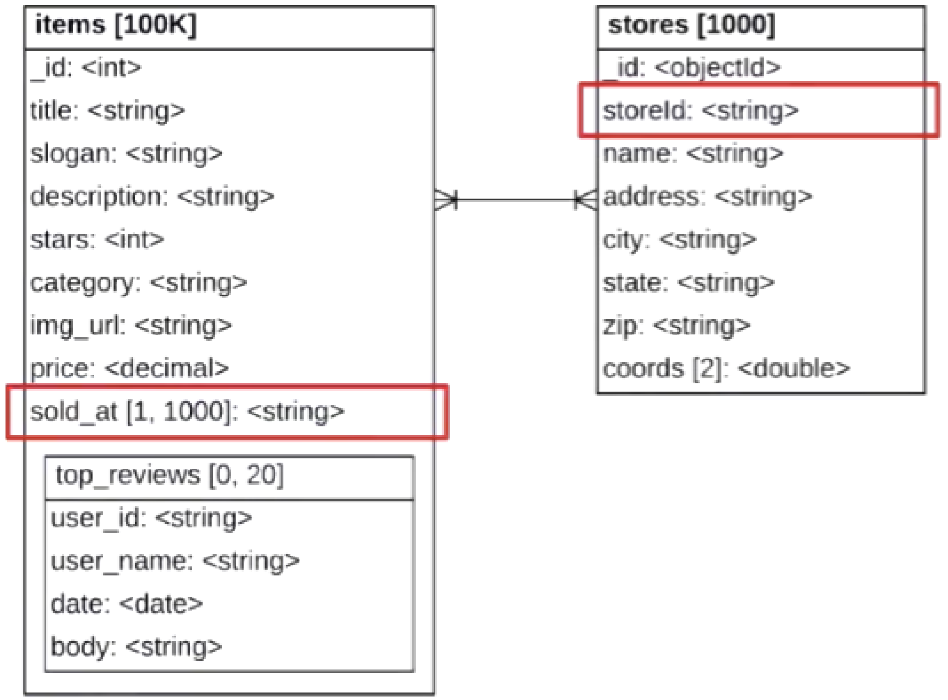
This requirement of keeping a source of input applies only to this specific representation in the many-to-many relationship.

In a one-to-many relationship, if a user is deleted with their phone numbers, we don't care about not having this phone number in any other document, because the phone number only belonged to this deleted user.

6. Many-to-Many Relationship

Instead of embedding, we can also use references. Let's start with a representation that keeps the references on the side that is the most often queried.

Many-to-Many: reference, in the main side



- array of references to the documents of the other collection
- references readily available upon first query on the "main" collection

```
db.stores.find({_id: {$in: ["store1", "store2", ...]}})
```

6. Many-to-Many Relationship

When a document is retrieved, we will get the list of all the associated object's unique identifiers.

Those are expressed in array field. If we need to add the information on these, we can use a second query, like the list of IDs.

Or we can use the dollar lookup operator to effectively do a join between the object ID for stores and the corresponding documents in the stores collection.

This is very similar to join operations in the SQL language.

6. Many-to-Many Relationship

The second representation of the many-to-many relationship that uses references is the one where we keep as we may have guessed the reference in another collection.

Because this representation created one use arrays of references, they are very similar. However, there are few differences.

Here, each store has a field called item sold that carries a list of references to the items sold in the given store. When we retrieve an item, we still don't know where it is sold.

We need a second query to get this information, which was another case in the previous representation.

6. Many-to-Many Relationship

A query like the following will return to stores in which the item with an ID of 10 or green MongoDB T-shirt has sold.

So which of those two representation that use references should we use?

The representation that fits our queries and also works well with the way we have the documents in our system.

When we add an item, which happens more often than adding a store, is it better to add the list of store in which it is sold or to individually update all stores to add the item?

How we answer this question will help us pick the right representation.

6. Many-to-Many Relationship

Recap

- ✓ Ensure it is a “many-to-many” relationship that should not be simplified;
- ✓ A “many-to-many” relationship can be replaced by two “one-to-many” relationships but does not have to with the document model;
- ✓ Prefer embedding on the most queried side;
- ✓ Prefer embedding for information that is primarily static over time and may profit from duplication;
- ✓ Prefer referencing over embedding to avoid managing duplication