NoSQL Databases

Laszlo SZATHMARY
University of Debrecen
Faculty of Informatics

- schema design

(last update: 2019-11-20 [yyyy-mm-dd])

2019-2020, 1st semester

Schema design

An important question while using MongoDB: should data be embedded, or should we store data in separate collection(s)?
RDBM systems prefer the 3rd normal form. However, in MongoDB, we should consider how we want to use data in the application. That is, the schema should be aligned to the application (*Application-Driven Schema*).

$$\left.\begin{array}{l}\text{What data are used together?}\\ \text{Which data do we read a lot?}\\ \text{Which data do we modify a lot?}\end{array}\right\}\ \text{consider these questions too}$$

Features of MongoDB:

- can use embedded data
- there is no join operation
- a collection has no declared schema (though documents in a collection are usually quite similar)

If your schema looks like an RDBMS schema, then it’s very likely that you don’t use MongoDB in a proper way.

Consider a blog. In an RDBM system, we would have a table called *post*, which would contain the blog posts. Possible fields: title, content, date of creation, etc. A post can have some keywords, but the tags would be stored in a separate table (*tags*). A post can also have comments, and these comments would go in a third table (*comments*). Thus, if we want to show a blog post with all its data, we should connect at least 3 tables with join operations.

In MongoDB, a post document can have several tags in a list. Similarly, comments can also be embedded in a post (usually, a blog post doesn’t have too many comments). Thus, to show a blog post with all its data, it would be enough to fetch just one document from the post collection. This one document contains every necessary data.

1:1 relation (One to One)

Example for a 1:1 relation: *Employee: Resume*. An employee has just one CV, and a CV belongs to exactly one employee.

We have two options:
1) Separate them in two collections. An Employee will have a *resume_id*, and a CV will have an *employee_id*.
2) Embedding. The CV gets embedded in the employee's document.

Which one to choose?
- The size of a document in MongoDB cannot exceed 16 MB. If the total size of the employee and the CV is more than 16 MB, then they cannot be together.
- If the CV is big (e.g. it contains images), and we need it rarely, but the employee is read often, then they can be separated.
- If the CV is updated very often, then it can go to a separate collection.
- However, the most natural choice for a 1:1 relation is embedding. If the previous special cases are not present, then the CV can be embedded in the employee's document.

1:N relation (One to Many / One to Few)

Example for a 1:N relation: *City: Person*. A lot of people live in a city, and a person has just one fix address.

Residents of a city could be stored in a list. However, the number of residents can be very high, thus this list could be really huge. In this case, separation would be a better choice.

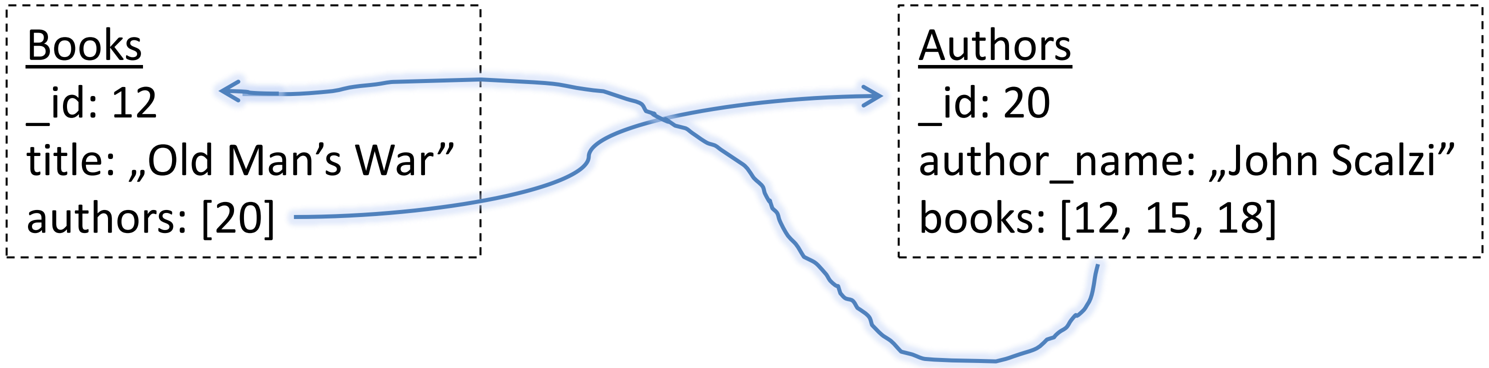
But what if the relation is not *One to Many*, but simply *One to Few*? That is, instead of a lot of elements, only some elements belong to a given element. Think of our blog post example: a post usually doesn't have too many comments. In this case, comments can be embedded in a blog post.

Summary:

One to Many:	using two collections seems to be a better idea
One to Few:	embedding

N:M relation (Many to Many)

Example for an N:M relation: *Books: Authors*. A book can have several authors, and an author can write several books.



Books
_id: 12
title: „Old Man's War"
authors: [20]

Authors
_id: 20
author_name: „John Scalzi"
books: [12, 15, 18]

Another way would be the embedding. However, it could lead to anomalies. For instance, let's suppose that we add books in Authors. In this case, the very same book would appear under several authors in the same way. What if we want to modify the data of a book? We should refresh the book's data everywhere…