Modelling Relationships:

In the real world applications we deal with various entities like Course, Author etc. These entities have some kind of association between them.

Eg. Every Course has author associated with it. But, then, Author is more than just a name. It is more than just a simple string.

That means, we can have a collection of authors inside which we store various author documents and in each author document we can have properties like name, website, image and so on…

So, how to work with Related Objects like this?

There are basically 2 approaches to deal with such situation:

1. Using References (Normalization)
2. Using Embedded Documents (Denormalization)

1. Using References (Normalization):

So, with the first approach our objects look like this,

```
let author = {
        name: 'Amit'
        // other properties
}

let course = {
        author: 'id'  // the id is a reference to author document in authors collection
}
```

2. Using Embedded Documents (Denormalization)

So, with the second approach our objects look like this,

```
let course = {
        author = {
                name: 'Amit'
                // other properties
        }
}
```

Which approach is better?

Each approach has its strengths and weaknesses. It depends on application and its querying requirement.

So, basically you need to do a trade-off between Query Performance vs Consistency.

Eg. In the 1st approach we've a single place to define an author. So, if tomorrow we decide to change the name of the author from 'Amit' to 'Amit Sahani' then there's a single place that we need to modify and all courses that are referencing to that author will immediately see the updated author.

"This is called – Consistency in a data"

However, every time we wanna query a course we need to do an extra query to load the related author. Now, sometimes that extra query may not be a big deal but in certain situations, you wanna make sure that your query runs as fast as possible. If this is the case, then we need to look at 2nd approach i.e. "Embedded Document"

With this approach, we can load a course object and it's authors with a single query. We don't have to do an additional query to load the authors because author is inside our course object or course document.

"2nd approach gives us – Performance"

However, with 2nd approach if tomorrow we decide to change the name of author from 'Amit' to 'Amit Sahani' then chances are there are multiple course documents that need to be updated and if our update operation doesn't complete successfully then it is possible that we'll have some course documents that are not updated and we will end up with **inconsistent data.**

That's why we need to do trade-off between consistency and performance. We can't have both of them at the same time.

3rd Approach – Hybrid Approach

Now, imagine that each author has 50 properties and we don't want to duplicate all those properties inside every course in our database. So, we can have a separate collection of authors but instead of using author reference in a course document, we can embed an author document inside of a course document but not the complete representation of that author. Perhaps, we only want a name property.

```
Eg. let author = {
        name: 'Amit'
        // 50 other properties
}
```

```
let course = {
    author = {
        id: 'ref',
        name: 'Amit'
    }
}
```

When to use Hybrid Approach?

If we wanna have a snapshot of our data at a point in time then we use the hybrid approach.

Eg. Imagine that you're designing an E-Commerce application where we will have collections like Orders, Products, Shopping Carts and so on…

In each order, we need to store a snapshot of a product because we wanna know the price of that product at a given point in time. So, that's where we will use hybrid approach.

So, which approach we use really depends on the application we're building. There's nothing right or wrong.