Tirth Shah
Fullstack Developer

Mongoose: Modelling Relationships between Connected Data

So, in this section, you learned that:

- To model relationships between connected data, we can either reference a document or embed it in another document.

- When referencing a document, there is really no relationship between these two documents. So, it is possible to reference a non-existing document.

- Referencing documents (normalization) is a good approach when you want to enforce data consistency. Because there will be a single instance of an object in the database. But this approach has a negative impact on the performance of your queries because in MongoDB we cannot JOIN documents as we do in relational databases. So, to get a complete representation of a document with its related documents, we need to send multiple queries to the database.

- Embedding documents (denormalization) solves this issue. We can read a complete representation of a document with a single query. All the necessary data is embedded in one document and its children. But this also means we'll have multiple copies of data in different places. While storage is not an issue these days, having multiple copies means changes made to the original document may not propagate to all copies. If the database server dies during an update, some documents will be inconsistent. For every business, for every problem, you need to ask this question: "can we tolerate data being inconsistent for a short period of time?" If not, you'll have to use references. But again, this means that your queries will be slower.

```
// Referencing a document
const courseSchema = new mongoose.Schema({
    author: {
        type: mongoose.Schema.Types.ObjectId,

        ref: ‘Author’
    }
})
```

```
// Referencing a document
const courseSchema = new mongoose.Schema({
    author: {
        type: new mongoose.Schema({
            name: String,
            bio: String
        })
    }
})
```

- Embedded documents don’t have a save method. They can only be saved in the context of their parent.

```
// Updating an embedded document
const course = await Course.findById(courseId);
course.author.name = ‘New Name’;

course.save();
```

- We don't have transactions in MongoDB. To implement transactions, we use a pattern called "Two Phase Commit". If you don't want to manually implement this pattern, use the **Fawn** NPM package:

```
// Implementing transactions using Fawn

 try {
   await new Fawn.Task()
       .save('rentals', newRental)
       .update('movies', { _id: movie._id }, { $inc: numberInStock: -1 }})
       .run();
}
catch (ex) {
    // At this point, all operations are automatically rolled back
}
```

- **ObjectIDs** are generated by MongoDB driver and are used to uniquely identify a document. They consist of 12 bytes:
    - 4 bytes: timestamp
    - 3 bytes: machine identifier
    - 2 bytes: process identifier
    - 3 byes: counter

- ObjectIDs are *almost* unique. In theory, there is a chance for two ObjectIDs to be equal but the odds are very low (1/16,000,000) for most real-world applications.

```
// Validating ObjectIDs
mongoose.Types.ObjectID.isValid(id);
```

- To validate ObjectIDs using **joi**, use **joi-objectid** NPM package.