**Understanding Mongoose**

A very good place to learn how to use mongoosse <https://attacomsian.com/blog/nodejs-mongodb-local-connection> but, to use it we should first understand it’s schema, and for that we go to <https://attacomsian.com/blog/mongoose-schemas>

Mongoose is described as an ORDBMS api for MongoDB, so let’s turn to Wikipedia.

<https://en.wikipedia.org/wiki/Object%E2%80%93relational_database#:~:text=An%20object%E2%80%93relational%20database%20(ORD,and%20in%20the%20query%20language>.

An **object–relational database** (**ORD**), or **object–relational database management system** (**ORDBMS**), is a [database management system](https://en.wikipedia.org/wiki/Database_management_system) (DBMS) similar to a [relational database](https://en.wikipedia.org/wiki/Relational_database), but with an [object-oriented database model](https://en.wikipedia.org/wiki/Object_database): objects, classes and inheritance are directly supported in [database schemas](https://en.wikipedia.org/wiki/Database_schema) and in the [query language](https://en.wikipedia.org/wiki/Query_language). In addition, just as with pure relational systems, it supports extension of the [data model](https://en.wikipedia.org/wiki/Data_model) with custom [data types](https://en.wikipedia.org/wiki/Data_type) and [methods](https://en.wikipedia.org/wiki/Method_%28computer_science%29).



Example of an object-oriented database model[[1]](https://en.wikipedia.org/wiki/Object%E2%80%93relational_database#cite_note-USDT01-1)

An object–relational database can be said to provide a middle ground between relational databases and [object-oriented databases](https://en.wikipedia.org/wiki/Object_database). In object–relational databases, the approach is essentially that of relational databases: the data resides in the database and is manipulated collectively with queries in a query language; at the other extreme are OODBMSes in which the database is essentially a persistent object store for software written in an [object-oriented programming language](https://en.wikipedia.org/wiki/Object-oriented_programming_language), with a programming API for storing and retrieving objects, and little or no specific support for querying.

At one point, some people thought that object-oriented databases were going to cure all the world’s problems, but they have not gained traction. Whether this was due to the large number of professionals who know and love the very well established RDBMS (MySQL, SQL-Server, etc.) or to people getting more interested in NoSQL databases, is a anyone’s guess. In any case, despite the importance of OOP in programming, object oriented databases have not taken over.

You should, however, draw on your knowledge of classes and objects in JavaScript and python to see what mongoose schemas are doing.

The schema definition is like a class definition --- but not yet one which can be instantiated.
Look at the paragraph ***Defining a Schema*** on the attacomsian site.

The first two lines:

const mongoose = require('mongoose')

const { Schema } = mongoose

bring in the mongoose module (no surprise) and then set const Schema = mongoose.Schema;

 Schema is a constructor function – as you will see when you look at the line

 const postSchema = new Schema({…….});

So we turn to the definition of postSchema. It should remind you of how we defined a JSON object in JS. Please notice that at this point it has no methods.

Now that we have a schema we want to use it – but there is an extra step, because so far it is just a definition, with has no methods, and therefore no way to interact with our mongo database.

The **mongoose.model( )** basically takes your schema and turns it into a constructor function with some bells and whistles. Why with bells and whistles? Because in addition to being able to use it to Create a new document, you will also be able to use it for all the CRUD functions.

Let’s look in more detail at paragraph ***Creating a Model***.

The website says “*A*[*Mongoose model*](https://attacomsian.com/blog/mongoose-models)*is a compiled version of the schema definition that maps directly to a single document in the collection.”*You should understand “maps” here in the same way the we say “this function maps x to 2x”. In other words, *after we compile the schema into a model* we will have a function which may be used to create a new document.

We obtain the model with:

To create a model in Mongoose, you use the **mongoose.model()** method with the model name as the first parameter and the schema definition as the second parameter:

const Post = mongoose.model('Post', postSchema)

Post is our new model, but it is this souped-up constructor function, as you can see in the next line:

const post = new Post({

 title: 'Mongoose Introduction',

 author: 'Atta'

})

Why souped-up? Because *“Now the****Post****model is ready for querying, creating, updating, and removing documents from the****posts****collection in MongoDB:”*

Please notice the capitalization and final ‘s’ here. The model is **Post** and it interacts with a collection **posts**. A particular document might be named **post**.

Finally, because a schema is really an object we can (woo-hoo) add methods to it. Look at the **Instance methods** paragraph. I hope that this is reminiscent of adding methods to the prototype of a class. (MongoDB is written in bison, which is similar to json.)

Looking at the paragraph on ***Static Methods***, you can see that there are a lot of methods which you automatically get with any model, and some ***Query Helpers***.

At this point we can turn to the official documentation for mongoose at <https://mongoosejs.com/docs/guide.html> There are two interesting and new features here – ***Virtuals***, (see also ***Views***) which don’t persist in the database, and ***Middleware.*** This is a very rich site and we could easily spend more time here.

**Thought questions:** Both the mongo and mongoose modules allow you to acess a db and collection and to use all the MongoDB functions. What is different about them? What does mongoose add and limit in terms of data models and functionality? Which one would you prefer to use, underwhat circumstances and why?