ORM

Created	@May 11, 2022 8:27 PM
Class	
Type	
Materials	
Reviewed	☐

ORM stands for Object Relational Mapper. Here Object refers to : Object Oriented programming, Relational refers to: Relational Databases (like mysql) and Mapper refers to: Mapping of relational db syntax to an Object oriented syntax.

ORM acts as a layer between the developer and the databases, and it helps us to visualise and even code all the relational logic in the form of object oriented code. That means, every entity which becomes a table in relational database, can be treated as a blue print, and the entries of the tables can be treated as new objects. So it establishes a relation between tables and objects.

This concept is not only limited to NodeJs or Mysql, but other big frameworks like Ruby on Rails, Django etc by default support ORM. For our usecase i.e. for NodeJs we need to use a Third part ORM which is `Sequalize`.

So in order to setup a basic demo project

```
npm init
npm i express
npm i sequelize mysql2
npm i sequelize-cli
```

Express: to setup basic server

mysql2: to get dependent packages required to connect to a mysql db

sequelize: this is the actual sequalize package we are going to use to get functionality of the ORM

sequelize-cli: this is going to provide few command line tools to work with sequelize orm

How we connect to a mysql database ?

So the mysql database actually runs on a mysql server. And just like any server it will be responsible for handling request coming to it. So it also has a port number associated a url etc, but because of sequalize ORM we don't need to bother about the port number and url etc, Sequelize provides a very simple way for connecting to the mysql sever.

`EXPRESS SERVER` ——————————→ `MYSQL SERVER` and this connection and request cycle will be handled automatically by our ORM.

Schema

Schema is the blue print of your relational database. Example: We define that we will have one table named student, with 3 columns, name as varchar, age as int, marks as int.

Schema migration refers to any change to the schema being recorded.

Whenever we change something in the schema it acts as a new migration.

Development, test and production db

Development database will be your local database, residing in your own computer, so every developer will be having their own local development database.

Test db is setup for testing new features that we are going to release, and this is generally common for all developers.

Production db is the actual live database that your application interacts with when users do some actions on the application.

Setting up sequelize

So in order to setup basic sequelize we can manually create configurations and folders, but it automatically provides a CLI interface in order to get things done easily.

```
npx sequelize init
// this command will initialise basic folders and configuration setup
```

Then inside the `config/config.json` file we have our code that will actually connect to the mysql server. In that code all we have to do is just change the name of database, whatever we want to keep and the username and password of our mysql server we need to feed.

As soon as we are done with the changes run the following command.

```
npx sequelize db:create
// this will actually create a new database in mysql
```

Creating tables

So in order to create tables we can use the following command

```
npx sequelize model:generate --name Student --attributes name:string,age:integer
```

here the —name is mentioning the name of the table and the —attributes are mentioning the column and the types of the columns of the table.

The moment we execute this command we get a model file, and a migration file. Migration file contains the actual code that sequalize executes for creating the table, if we want to make some changes in the schema, we can change the migration file. Once we are sure about the change

we can execute

```
npx sequelize db:migrate
```

Querying with js

```
const express = require('express');
const app = express();
const Student = require('./models/index').Student;

console.log(Student);
app.listen(3000, async () => {
    console.log("server started");

    const new_student = await Student.create({name: "Suchitra", age: 23});
    console.log(new_student);
    const all_student = await Student.findAll();
    console.log(all_student);
});
```

So, in order to use the models to interact with the tables, we can get the index file from models folder `require('./models/index')` this will return an object that has access to the tables, so we can do `.Student` to access the student table.

Then we can use Student.create, Student.findAll etc to query on the tables.