**Spring Boot | Spring Data JPA Many to Many Bidirectional Relationship Example**

This Many-to-Many is one of the most commonly used association mapping in Real time Because in real time while working with Entity we always need to maintain their Mapping to not loose any information. To make it more simpler let me walk you through one happy scenario which is perfect example to demonstrate Many-to-Many association mapping.



So, if u can see 2 Entities something called Students and Courses. So, if you think from real time perspective student can purchase many courses.



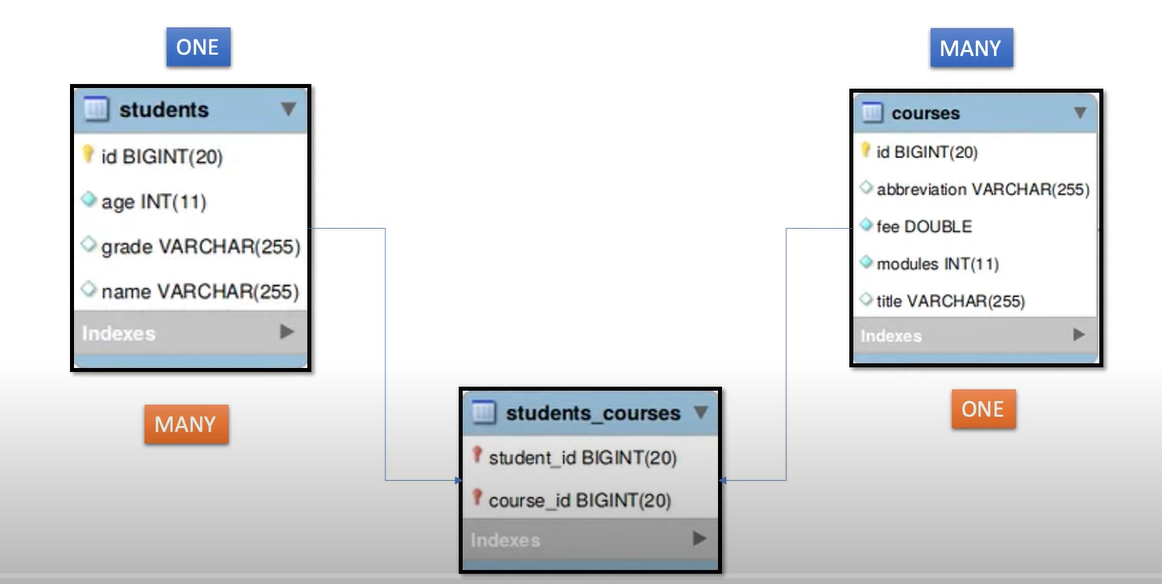
Similarly, if u can think in the reverse direction a single course can be purchased by multiple students.



So, if u observed carefully from student to course it is **one to many** like one student can purchase multiple courses. Similarly, a single course can be purchased by multiple students. So, in this case if I use one to many in both the cases then for me it would be difficult for me to maintain primary and foreign key in both the cases.

So, to overcome this kind of conflicts rather than use one to many in both the place we can directly use Many-to-Many in association Mapping.

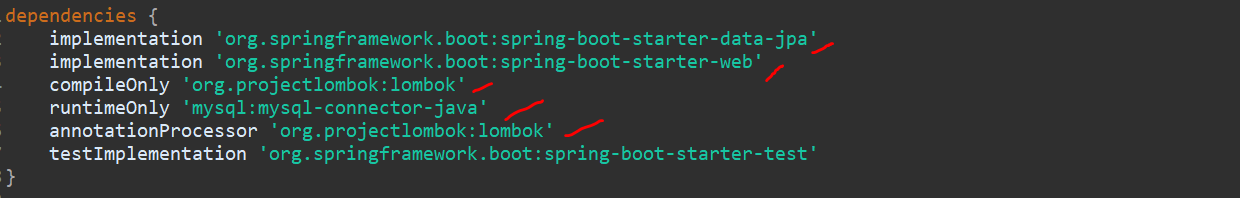
so, while using Many to Many association mappings behind the scene it will create an additional database table to hold the primary key of both the relationship table like Student and Course. Which makes schema mapping quite easy and understandable.



Let’s start the Implementation---

Application - **jpa-manytomany**

Dependencies- **Spring Web, Lombok, Spring Data JPA, MySQL Driver**



Now I will just create couple of packages…

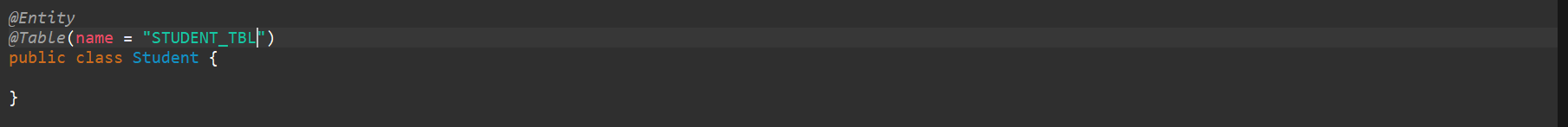
**Entity/Dao or Repository/ controller**

Now I am not creating the service better I will directly inject the repo but it’s a good practice to create a service layer. Since I am just demonstrating about the mapping so I ll just focus on mapping part.

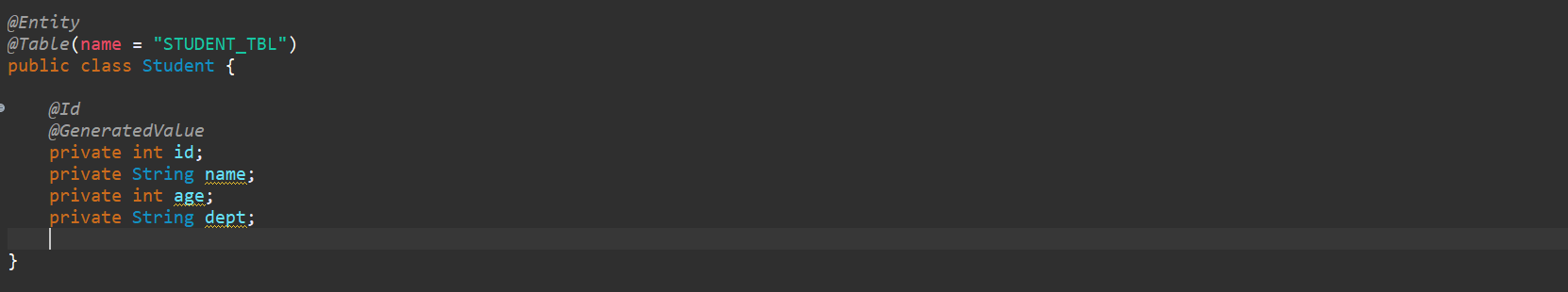
So, if u observed the flow diagram, we need to create 2 entities **User** and **Course.**

Now from Student I need to write association mapping so that Student can map to multiple Course. Similarly, from Course I need to write on mapping a single Course can be mapped to multiple Student.

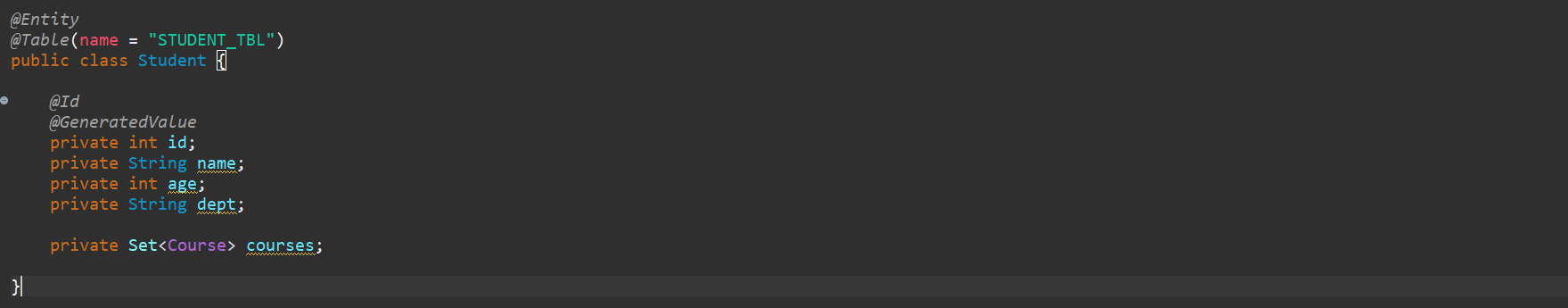
So, to make Student as an Entity first annotate class as **@Entity.** Then I can annotate **@Table,** you can give table name.



Now you can define the couple of fields for Student Entity. I just want id to be autogenerate so using **@Id** and **@GeneratedValue.**



Now from Student I want to map list of Course or Set of Course.



Since I am using Lombok so no need to do getter and setter manually… instead of writing @Getter and @Setter we can write @Data, but there is some issue. If I will write @Data in multiple mapping json or Jackson is unable to bind those data. There is keep looping the **stackoverflow** error I faced. So, its good practice to use @Getter and @Setter if u is working with entity mapping.

Now this Course I need to tell to the Jpa this is the Student I want to inject multiple Course. So as per the flow diagram u just create another table and map primary key of Student and Course. So that is what I need to define using Annotation. So just use **@ManytoMany** and specify the required parameters.

Cascade type u can make Persis but let’s keep ALL.



Now the main important thing here is u need to define here Join Table. You need to tell to the JPA to create one additional table with the information I am giving into this annotation.

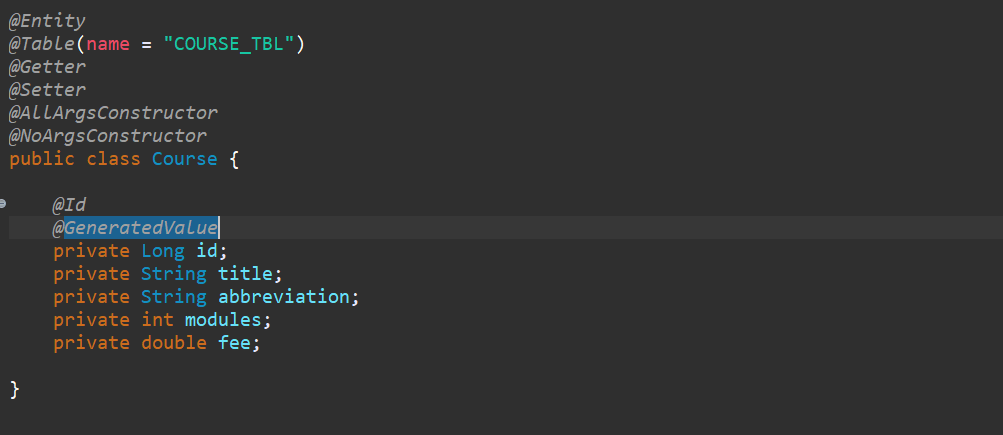
So, you need to define one annotation called **@JoinTable.** Instead of **ManyToMany** in other association mapping you can directly use **@JoinColumn** but in case of **ManyToMany** you want to hibernate or jpa to create new additional table for you. So, you need to define table information.

So, I will tell hibernate or JPA to create a table with name **STUDENT\_COURSE\_TABLE,** now next what you need to do you also need to define the **Join Column** Information. so, I will tell to JPA this is what the Table create at Runtime and just use below mapping. When I say below mapping you can say **@JoinColumns.** then you need to define Join Column to keep inside this table. So, we need to define **@JoinColumn**. the **name** of the column is **student\_id** and then **referencedColumnName** is the id of Student.

If you observed this Student Entity **STUDENT\_COURSE\_TABLE.** this table will create by JPA I am telling him to keep this particular **id** of the Student with column name **student\_id.** So, **student\_id** be the column name and it will point or refer to the **id** of my **Student.** Similarly, also I need to tell about the Course details. So, I will define another Join Column for Course. So, I will just add **@InverseJoinColumn** then I will define annotation **@JoinColumn.** Then I will tell to the JPA use the column name which is **course\_id** and refer to the column name **id** and this id is the primary key of Course table. So now this table **STUDENT\_COURSE\_TABLE** contains primary key of Student table and primary key of Course table. That’s it all about Many to Many Association Mapping.



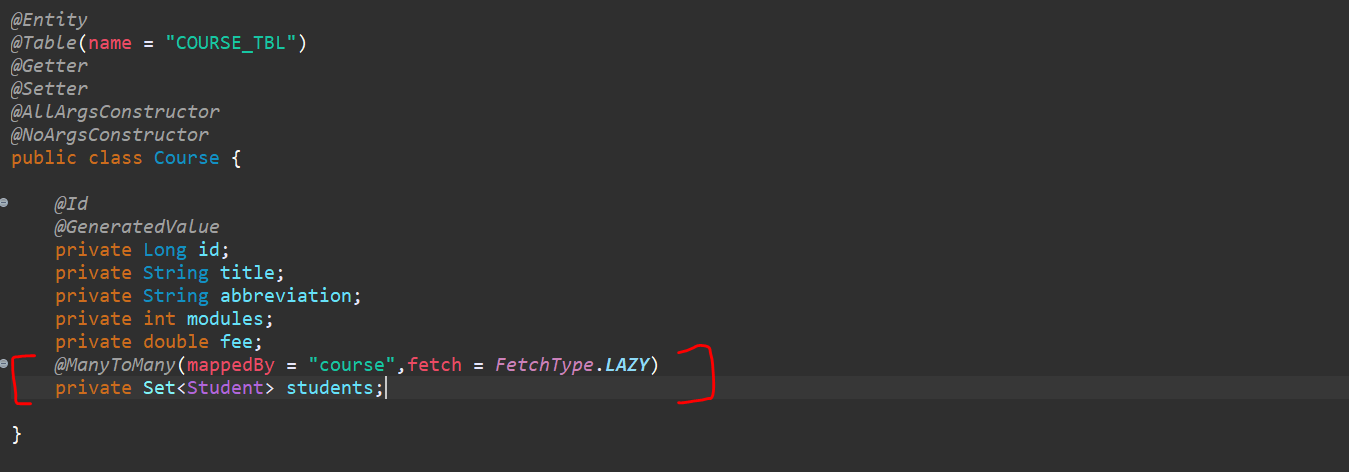
Now we just did from Student similarly we need to do from Course end.



Now next the way to define Student can use set of Course, similarly from Course we need to define a Course can be purchased by multiple Students or a set of Students. And here also we need to define the annotations. Here also need to define annotation **@ManytoMany** but no need to define **@Jointable** information in particular Course table because Student is my Parent, or the owner table is the **Student** with the help of Student I am going to Map the particular course. So no need to define the all information’s into the Child table which is not a Owner Entity.

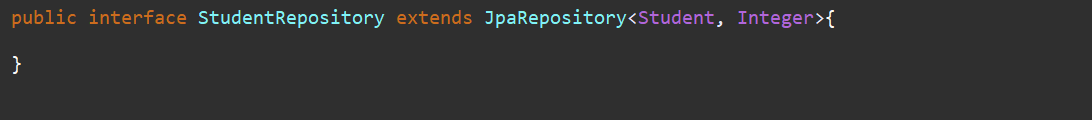
So, to tell to the JPA the Owner is Student what I will use there is an attribute called **mappedBy** or I need to tell to the Spring or JPA that this is not the owner entity the owner Entity is different something called Student. That is what we defined here.

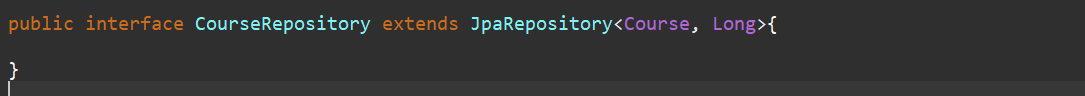
Then I also define the fetch , **FetchType.LAZY.**



mappedBy attribute indicate the Entity that owns the Bidirectional Relationship. In a bi-directional relationship we use ManyToMany annotations defined in both the entity but only one Entity owns the Relationship, and we have picked that Student.

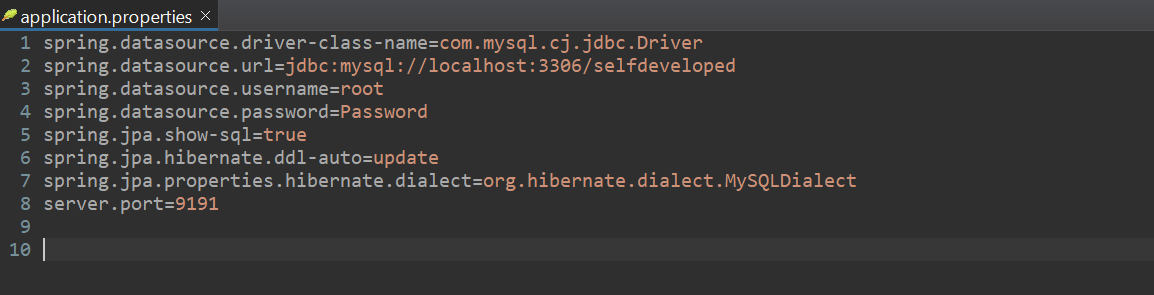
Since we have 2 Entities, so we need to create 2 different Repositories…since we are using Spring data Jpa, so we need to extend these 2 interfaces with JpaRepository…it contain 2 parameter first a Entity name and second a datatype of Primary key.





Now we have created an Entity and its corresponding Repositories. Now next step to make a connection from my Java application to db MySQL.

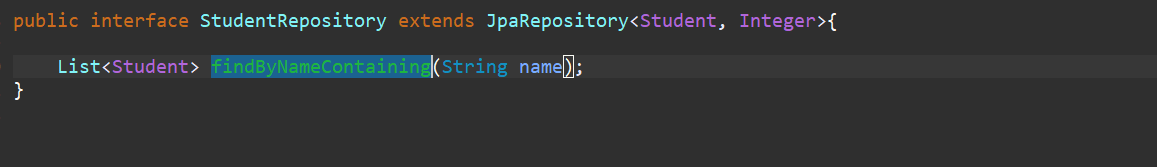
So, I need to define the **data source properties** inside this **application. properties** file. So, I have those key and value handy with me let me add it here.



If are u aware with Spring Data JPA… let’s add few methods in Jpa Repository…

So, In Student Repository I just want to write one method to find the Student if that input contains the Specific Fields…

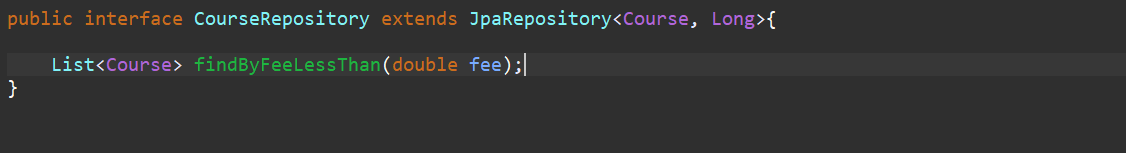
**findByNameContaining (String name),** so instead of name u can use any other fields from ur entity class, and it will return me the List of Student object.



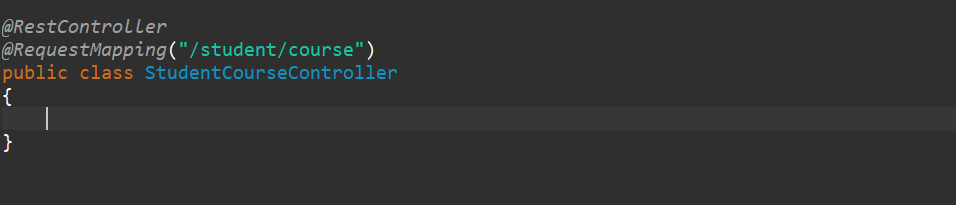
So, this is pretty simple Guys. So **findBy** is the **prefix** and next any field name from entity based on what u want to do the operation. If the input exists what we are providing, then it will give the List of Students.

Similarly, we will write a method in Course Repository, which will return me the List of Course based on the amount what I will give if that is less than the price.

We will use **findByFeeLessThan (double fee) is** like an aggregate function, it also returns as a List of Course.

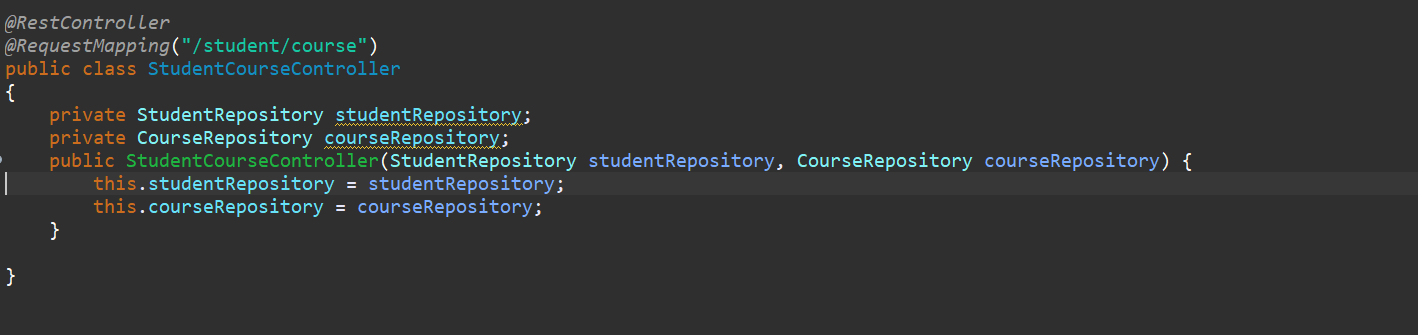


Now we need to save the Students along with the Course, so for that lets create an endpoint in our Controller. And let’s give root url as **/student/course.**



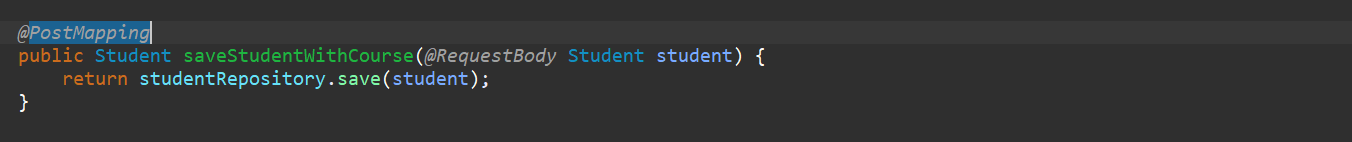
Note: - First thing I want to Inject both the Repositories here again I will repeat that this is not a good practice to Inject your Repositories directly to your Controller. You should write one separate layer which is Service Layer. Since this is demo so I am avoiding to writing a service layer. That is the reason I am injecting repo directly in the Controller.

And I am injecting both using Constructor Injection.



Now let’s define our endpoint, it will take Student Object from the postman along with course and will save in the Student Repository.

So here I need to give input as a Student because Student Contains set of Course. So, while building the Student Object you need to pass the Set of Course from the Postman. Declared this method with **@PostMapping** not passing any url since we have a root url.

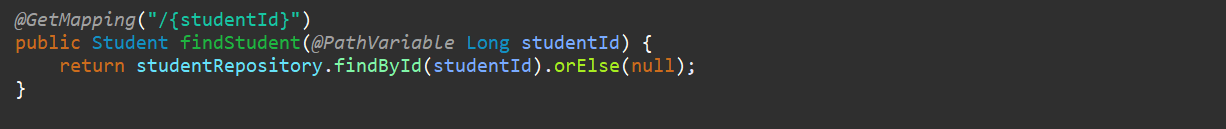


Now I will write a method which will return me the List of Student, once I will fetch list of Student it should display me the List of Course purchased by the particular Student.

Simple guys if I fetch the Student object, I can get the set of Course as well. Because Cascade type I set All and Fetch Type is LAZY.



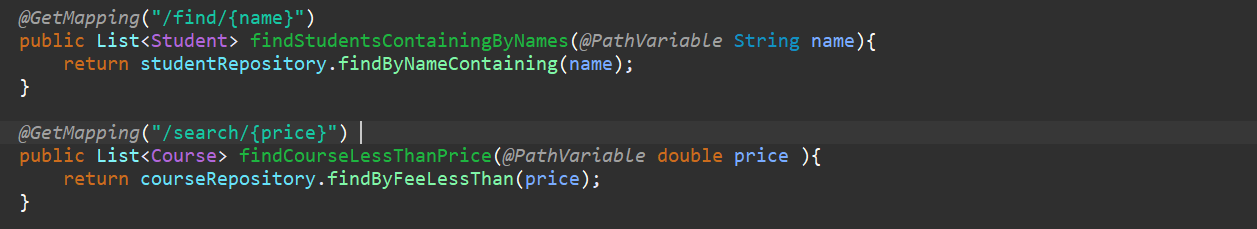
Another endpoint will return me the Student, particular Student object based on the student id what I will give. I am defining student id as a Path Variable so that I can pass id as part of url. Now if u observed this will return as an Optional. So, I will use get the object if not then return me the null.



Now let’s call these 2 methods what we defined in Student Repo and Course Repo.

Fetch Student based on name passing as in put field. Fetch Courses based on the price what I ll give…

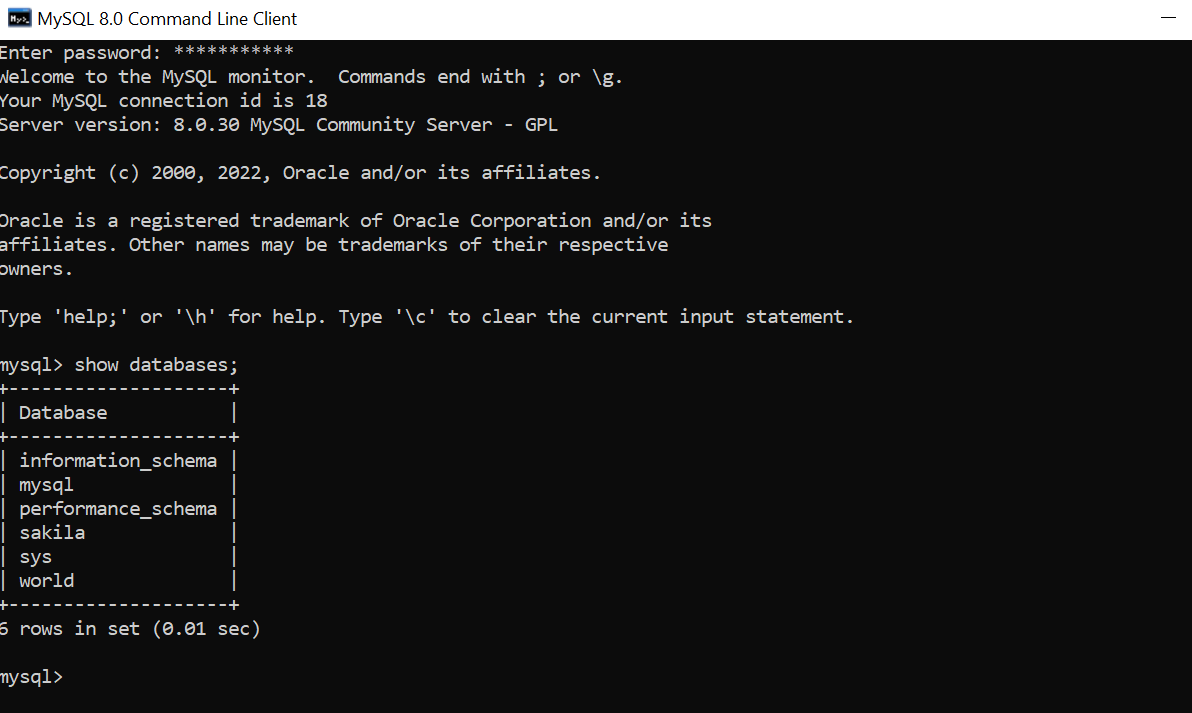
So, fetching courses whose price is less than the price what I am giving…





We are good now we defined all the endpoints.

Let’s start the application…



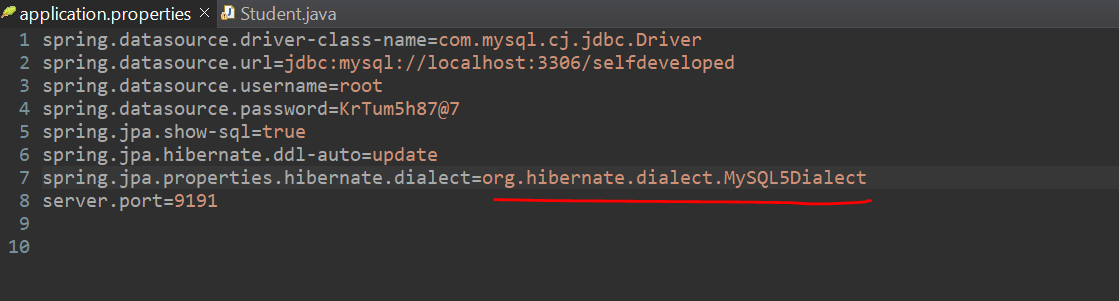
**The problem is that - in Hibernate 5.x and earlier - the dialect org.hibernate.dialect.MySQLDialect is for MySQL 4.x or earlier. The fragment TYPE=MYISAM that is generated by this dialect was deprecated in MySQL 4.0 and removed in 5.5.**

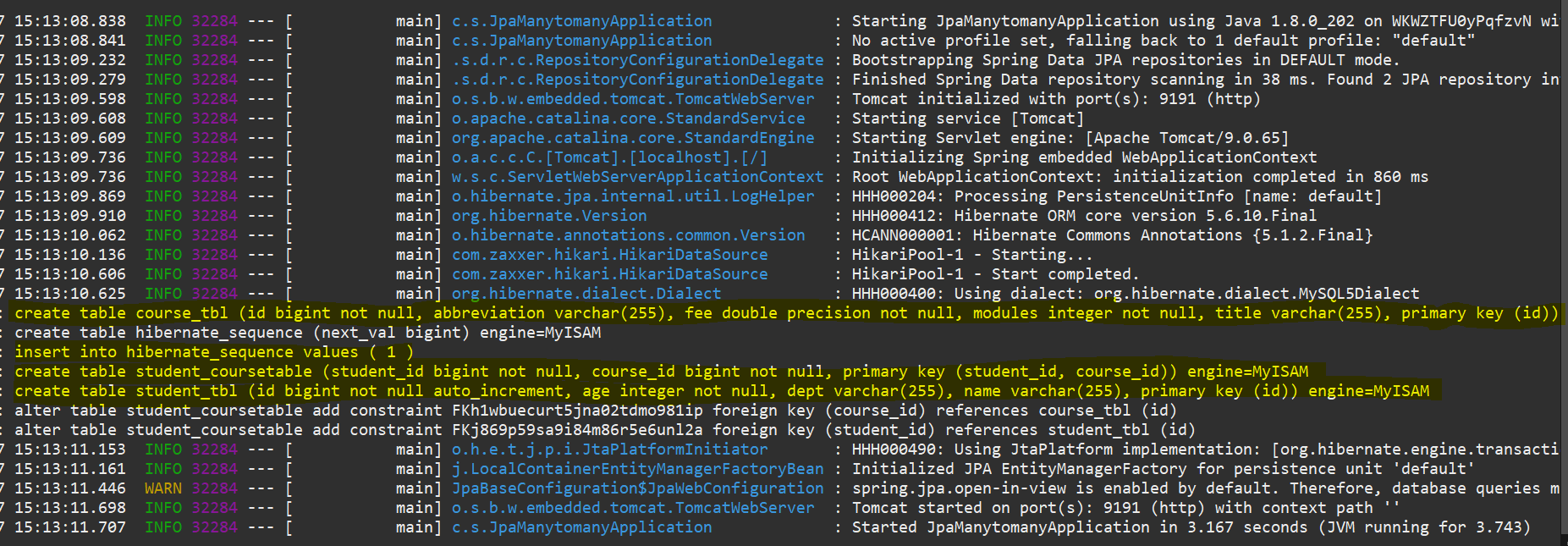
**Given that you use MariaDB, you need to use (depending on the version of MariaDB and - maybe - the version of Hibernate) one of:**

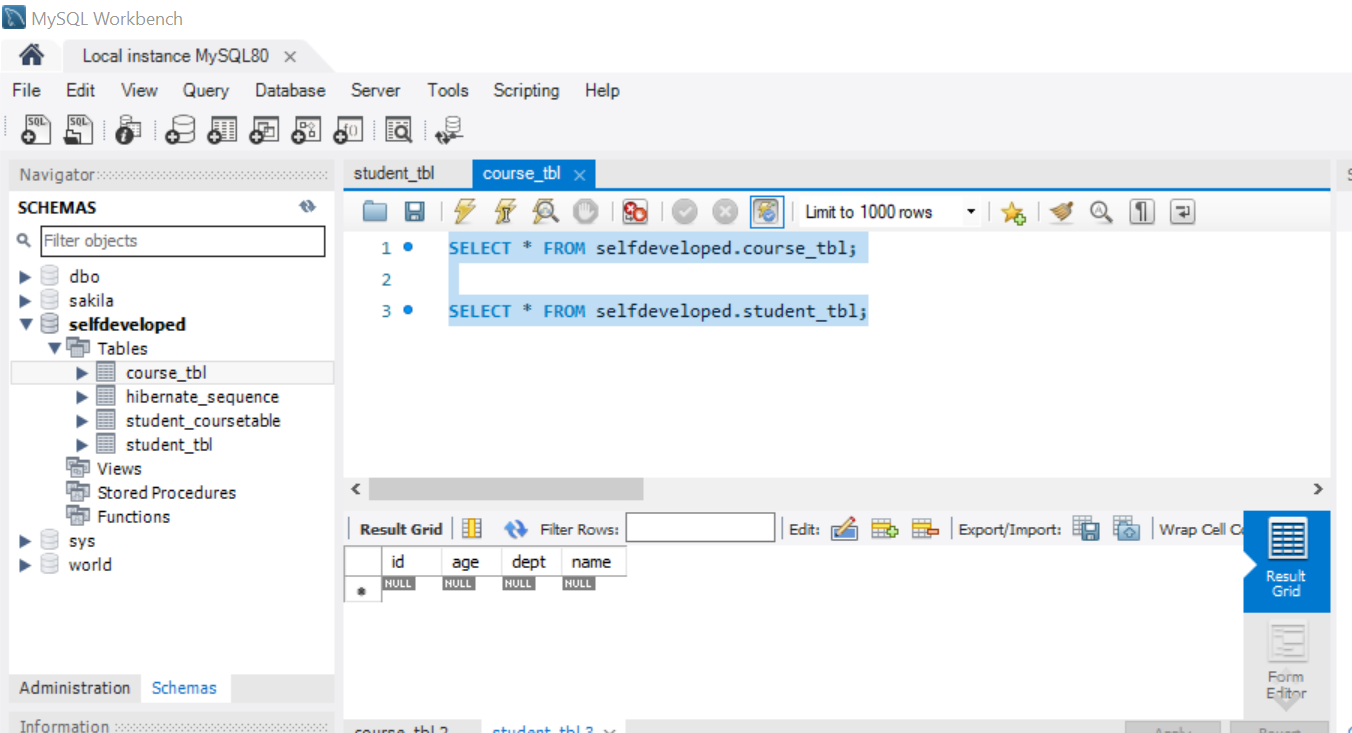
* **org.hibernate.dialect.MariaDBDialect**
* **org.hibernate.dialect.MariaDB53Dialect**

**If you are using MySQL, or if the above two dialects for MariaDB don't exist in your version of Hibernate:**

* **org.hibernate.dialect.MySQL5Dialect**
* **org.hibernate.dialect.MySQL55Dialect**
* **org.hibernate.dialect.MySQL57Dialect**







So, 4 tables have been created 2 for entity 1 for hibernate sequence and 1 for hibernate mapping table.

Now let’s go to the postman and try to hit the rest end point…

Try for **POST** because we want to add the Student object.

**POST** <http://localhost:9191/student/course>

Header- Content-Type application/json

Payload:-

**{**

**"name": "santosh",**

**"age": 33,**

**"dept": "UI",**

**"courses": [**

**{**

**"title": "Angular",**

**"abbreviation": "ng",**

**"modules": 12,**

**"fee": 5000**

**},**

**{**

**"title": "React JS",**

**"abbreviation": "RJS",**

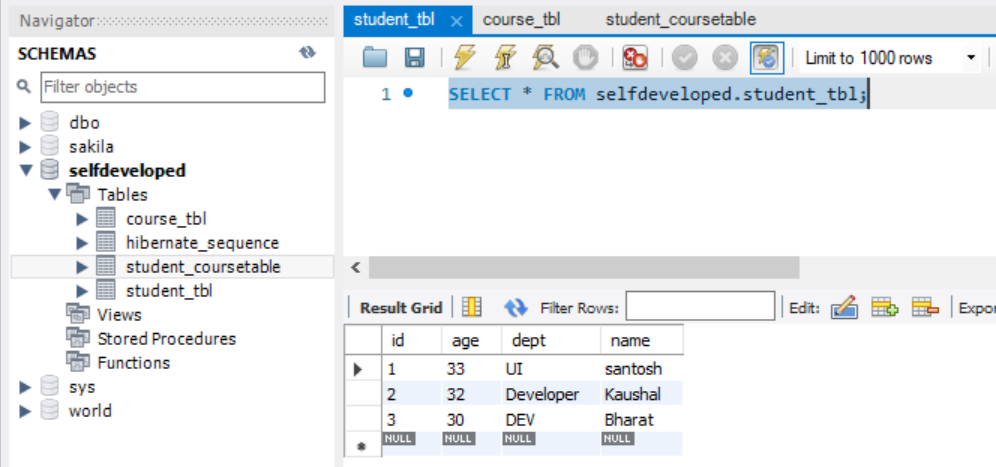
**"modules": 11,**

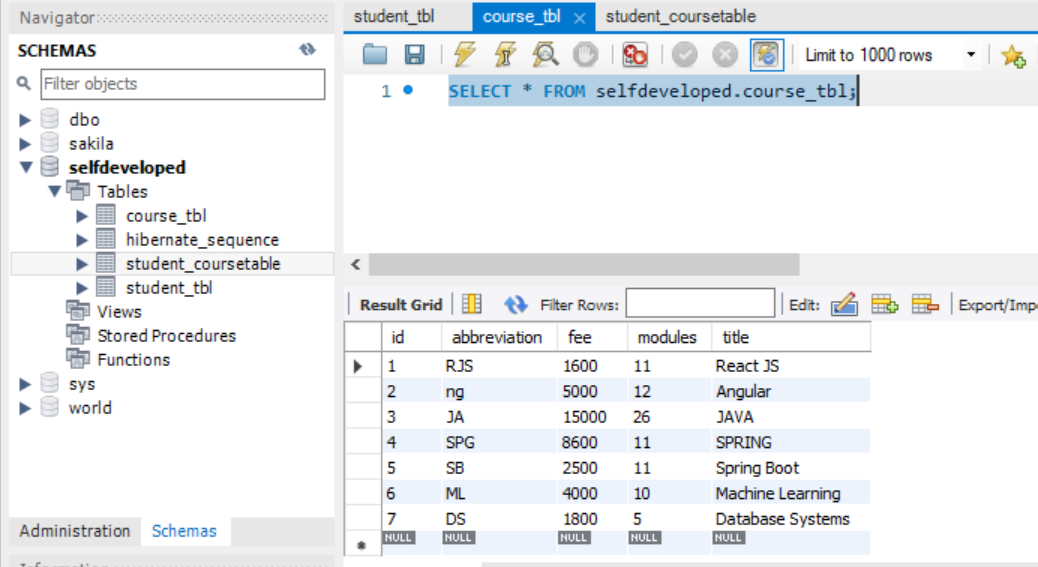
**"fee": 1600**

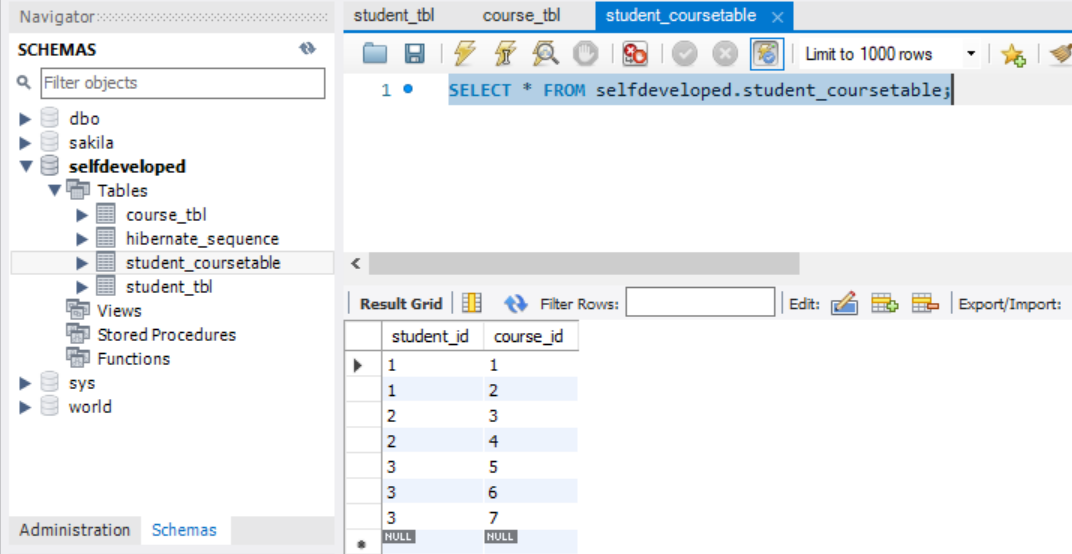
**}**

**]**

**}**







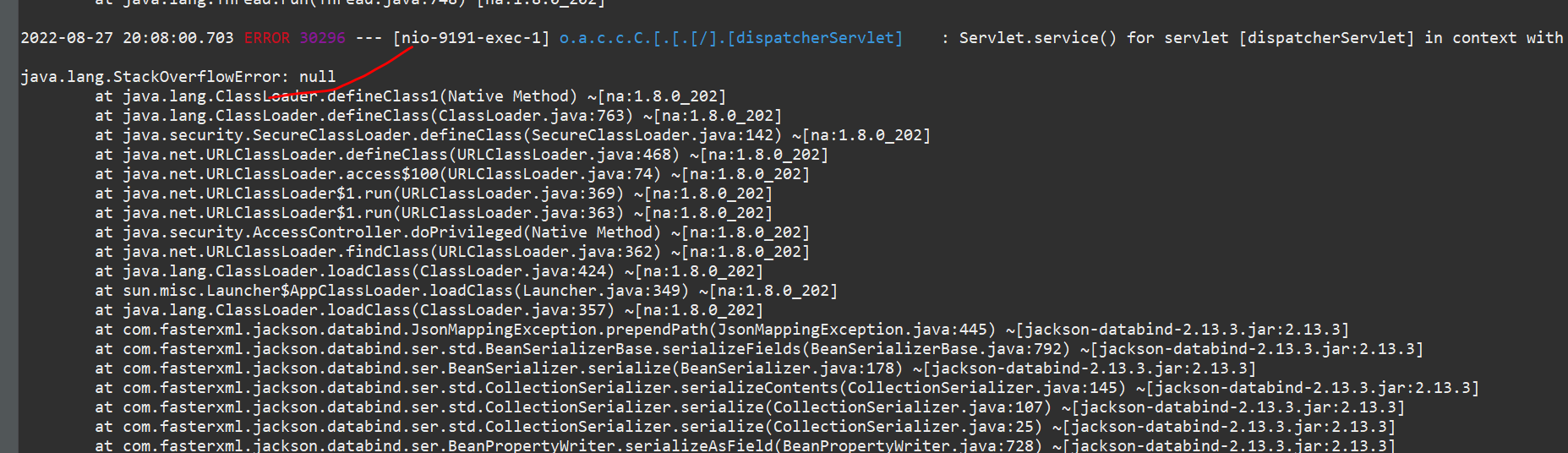
**The id is auto incremented this is because of Generation Type-IDENTITY**

The Student Santhosh who purchased the 2 course, the Student Kaushal who purchased the 2 course and the Student Bharat who purchased the 3 course.

So now using Post Mapping its logging the Student and Course data and adding one additional table for keeping the all primary key mapping information.

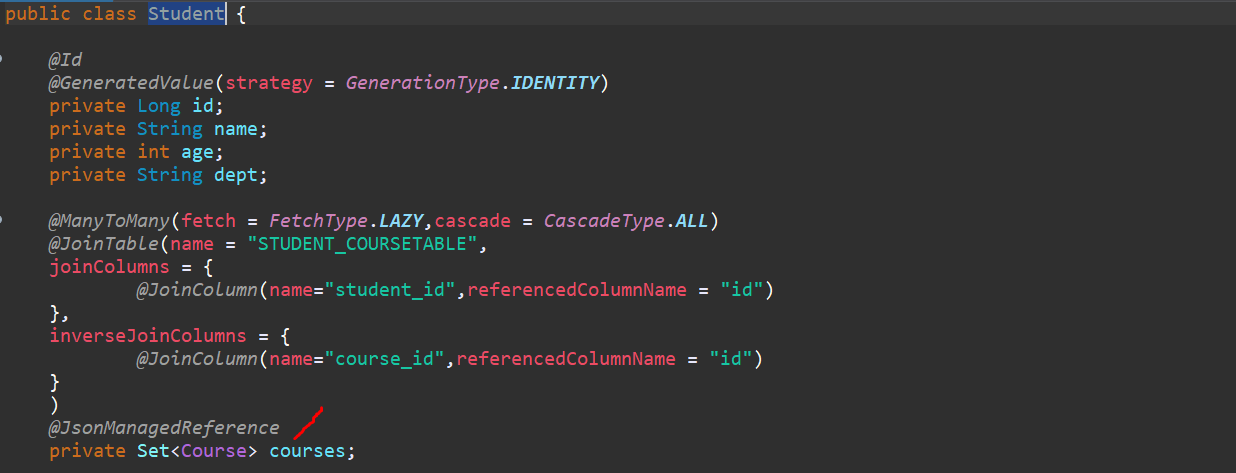
**>>> Now let’s fetch the all the Students**

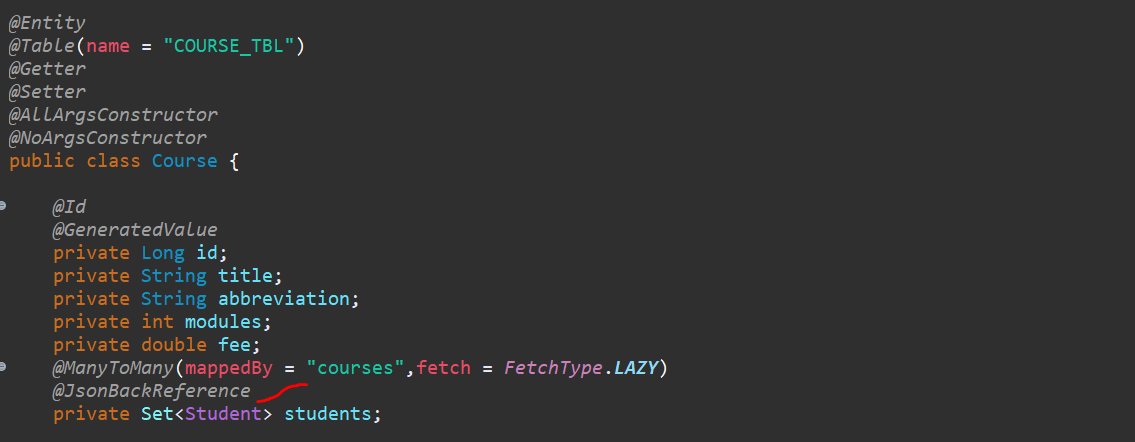
GET <http://localhost:9191/student/course>



So, this is what I was talking about if u remember very beginning, we use **@Getter and @Setter** annotation rather than **@Data.**

Student having list of Course and again Course having list of Student that is bit Conflicting for Jackson Databinding to Map to the Object. So, there are 2 annotations let me use that **@JsonManagedReference.**





**This is what using the Json we are telling the or we are just define into the Jackson Bind use these 2 annotations to Bind the JSON to the Objects.**

Let me restart it.

**GET** [**http://localhost:9191/student/course**](http://localhost:9191/student/course)

**Response:**

**[**

**{**

**"id": 1,**

**"name": "santosh",**

**"age": 33,**

**"dept": "UI",**

**"courses": [**

**{**

**"id": 2,**

**"title": "Angular",**

**"abbreviation": "ng",**

**"modules": 12,**

**"fee": 5000.0**

**},**

**{**

**"id": 1,**

**"title": "React JS",**

**"abbreviation": "RJS",**

**"modules": 11,**

**"fee": 1600.0**

**}**

**]**

**},**

**{**

**"id": 2,**

**"name": "Kaushal",**

**"age": 32,**

**"dept": "Developer",**

**"courses": [**

**{**

**"id": 3,**

**"title": "JAVA",**

**"abbreviation": "JA",**

**"modules": 26,**

**"fee": 15000.0**

**},**

**{**

**"id": 4,**

**"title": "SPRING",**

**"abbreviation": "SPG",**

**"modules": 11,**

**"fee": 8600.0**

**}**

**]**

**},**

**{**

**"id": 3,**

**"name": "Bharat",**

**"age": 30,**

**"dept": "DEV",**

**"courses": [**

**{**

**"id": 5,**

**"title": "Spring Boot",**

**"abbreviation": "SB",**

**"modules": 11,**

**"fee": 2500.0**

**},**

**{**

**"id": 7,**

**"title": "Database Systems",**

**"abbreviation": "DS",**

**"modules": 5,**

**"fee": 1800.0**

**},**

**{**

**"id": 6,**

**"title": "Machine Learning",**

**"abbreviation": "ML",**

**"modules": 10,**

**"fee": 4000.0**

**}**

**]**

**}**

**]**

**>>> Now let me fetch Student based on his Id (fetch id=1)**

**GET** [**http://localhost:9191/student/course/1**](http://localhost:9191/student/course/1)

**Response-**

**{**

**"id": 1,**

**"name": "santosh",**

**"age": 33,**

**"dept": "UI",**

**"courses": [**

**{**

**"id": 2,**

**"title": "Angular",**

**"abbreviation": "ng",**

**"modules": 12,**

**"fee": 5000.0**

**},**

**{**

**"id": 1,**

**"title": "React JS",**

**"abbreviation": "RJS",**

**"modules": 11,**

**"fee": 1600.0**

**}**

**]**

**}**

**>>> Find by Name Containing -> I will give ka**

**GET** [**http://localhost:9191/student/course/find/ka**](http://localhost:9191/student/course/find/ka)

Response-

**[**

**{**

**"id": 2,**

**"name": "Kaushal",**

**"age": 32,**

**"dept": "Developer",**

**"courses": [**

**{**

**"id": 4,**

**"title": "SPRING",**

**"abbreviation": "SPG",**

**"modules": 11,**

**"fee": 8600.0**

**},**

**{**

**"id": 3,**

**"title": "JAVA",**

**"abbreviation": "JA",**

**"modules": 26,**

**"fee": 15000.0**

**}**

**]**

**}**

**]**

You can see here the name starts with or containing the character I am getting the response.

**>>>> If I will add something s**

**GET** [**http://localhost:9191/student/course/find/s**](http://localhost:9191/student/course/find/s)

**[**

**{**

**"id": 1,**

**"name": "santosh",**

**"age": 33,**

**"dept": "UI",**

**"courses": [**

**{**

**"id": 2,**

**"title": "Angular",**

**"abbreviation": "ng",**

**"modules": 12,**

**"fee": 5000.0**

**},**

**{**

**"id": 1,**

**"title": "React JS",**

**"abbreviation": "RJS",**

**"modules": 11,**

**"fee": 1600.0**

**}**

**]**

**},**

**{**

**"id": 2,**

**"name": "Kaushal",**

**"age": 32,**

**"dept": "Developer",**

**"courses": [**

**{**

**"id": 3,**

**"title": "JAVA",**

**"abbreviation": "JA",**

**"modules": 26,**

**"fee": 15000.0**

**},**

**{**

**"id": 4,**

**"title": "SPRING",**

**"abbreviation": "SPG",**

**"modules": 11,**

**"fee": 8600.0**

**}**

**]**

**}**

**]**

I can see Santhosh and even Kaushal is there because it contains s.

Similarly let’s try another endpoint

**>>>> To find out the Course based on the Price or Price Comparison**

So, all the courses whose price is less than 2000 would be display here.

**GET** [**http://localhost:9191/student/course/search/2000**](http://localhost:9191/student/course/search/2000)

**[**

**{**

**"id": 1,**

**"title": "React JS",**

**"abbreviation": "RJS",**

**"modules": 11,**

**"fee": 1600.0**

**},**

**{**

**"id": 7,**

**"title": "Database Systems",**

**"abbreviation": "DS",**

**"modules": 5,**

**"fee": 1800.0**

**}**

**]**

**So here are only 2 courses under 2000**

**So, let me increase the price 15000**

**GET** [**http://localhost:9191/student/course/search/15000**](http://localhost:9191/student/course/search/15000)

**[**

**{**

**"id": 1,**

**"title": "React JS",**

**"abbreviation": "RJS",**

**"modules": 11,**

**"fee": 1600.0**

**},**

**{**

**"id": 2,**

**"title": "Angular",**

**"abbreviation": "ng",**

**"modules": 12,**

**"fee": 5000.0**

**},**

**{**

**"id": 4,**

**"title": "SPRING",**

**"abbreviation": "SPG",**

**"modules": 11,**

**"fee": 8600.0**

**},**

**{**

**"id": 5,**

**"title": "Spring Boot",**

**"abbreviation": "SB",**

**"modules": 11,**

**"fee": 2500.0**

**},**

**{**

**"id": 6,**

**"title": "Machine Learning",**

**"abbreviation": "ML",**

**"modules": 10,**

**"fee": 4000.0**

**},**

**{**

**"id": 7,**

**"title": "Database Systems",**

**"abbreviation": "DS",**

**"modules": 5,**

**"fee": 1800.0**

**}**

**]**

So, this is just Aggregate function to just use the Less Than Spring Data JPA.

So, this is Just What about Many To many Relationship. So, I believe maximum scenario being developed using Many to Many Mapping.