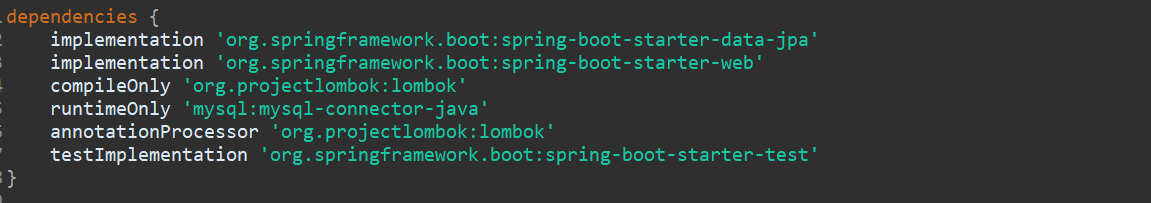
# Spring Boot, MySQL, JPA, Hibernate Restful CRUD API Example

Application- **spring-boot-crud-example**

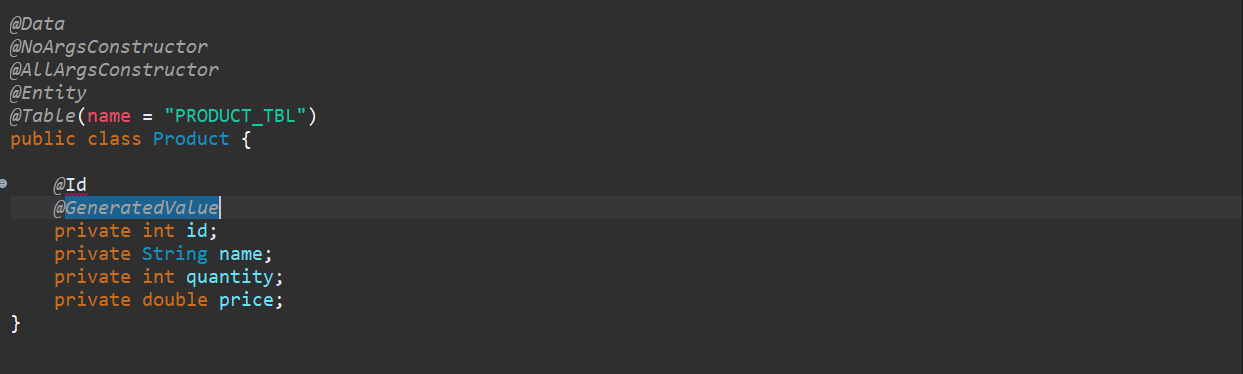
Dependencies- **Lombok** (avoid manually writing getter/setter), **Spring Web** (To expose REST API), **MySQL Driver** (MySQL Connector), **Spring Data JPA**.

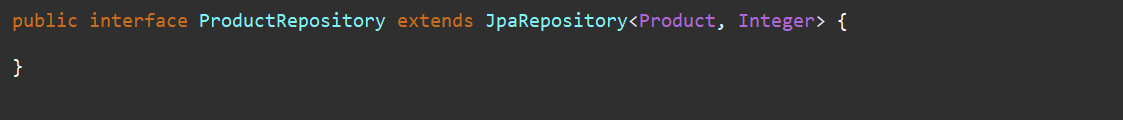


Let’s create a packages entity, repository, service and controller.

Now let’s create an Entity first as name given Product, we can say Model as well.

Now let’s use Lombok provided annotations. As this Product is my Model or Entity class so need to annotate with **@Entity**. To make id as PK and auto generated we need to use **@Id** & **@GeneratedValue**





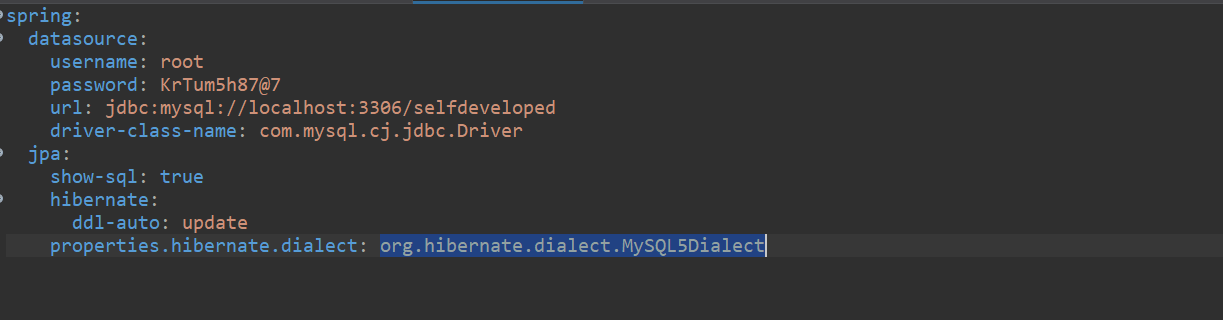
We have created our Entity and corresponding Repository.

Now we need to write logic to connect to database.

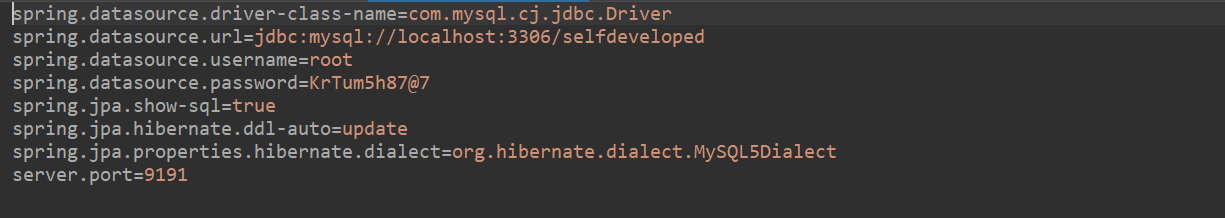
If u are using spring boot, then no need to write additional logic to connect to Database. all data base connection related configuration we are doing in our application.properties file.

If u want to see query generated by hibernate in ur console so u need to use **show-sql=true**. Schema name we have given as **selfdeveloped.**

Also, u can create one yml file. Create one **application.yml** file. Then we need to add the same inputs in those yml file.



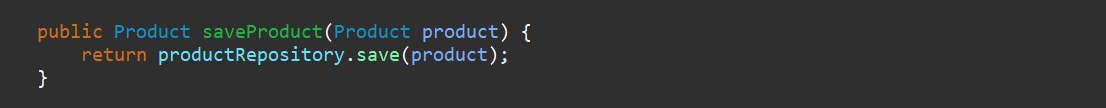
But I will use application. properties, u can use either of one.



Now let’s create the **Service** and **Controller**. This Product Service will talk to this Product Repository. So, we need to Inject this Repository to this Service.

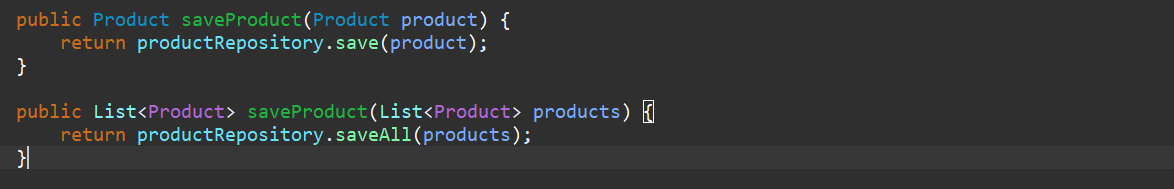
Let’s create a **ProductService** class and write all the Http Methods **GET, PUT, POST, DELETE.**

Let me first write POST method where we will save Product object to database.

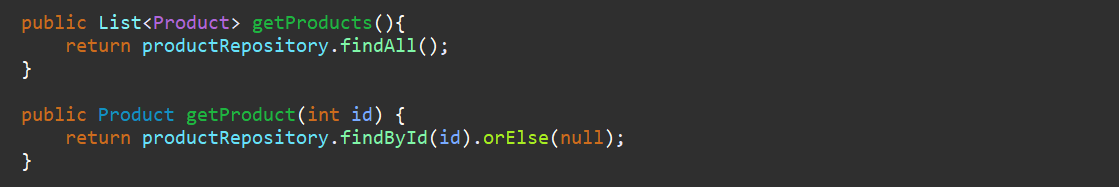


This will take argument as Product and it will save it.

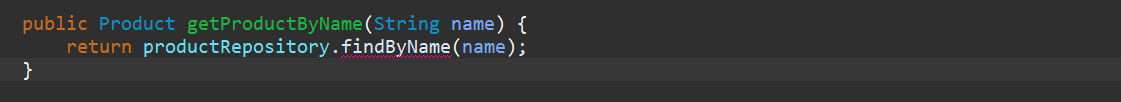
But if you want a List of Objects to save at same time… at that time you need to use **saveAll**() method given by Data JPA.



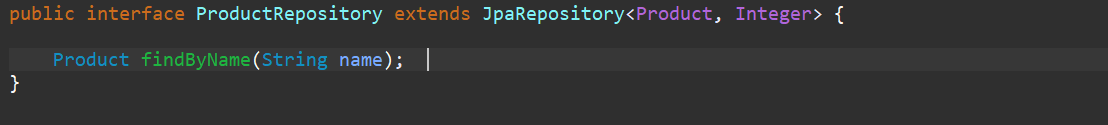
These 2 are the POST methods, lets write a method to fetch all Product object. And if u want to fetch object based on some field then write it separately. So **findById () will return as Optional** so if we won’t get the Object into DB based on id then it will return as NULL. It will never throw **NULL Pointer Exception**.



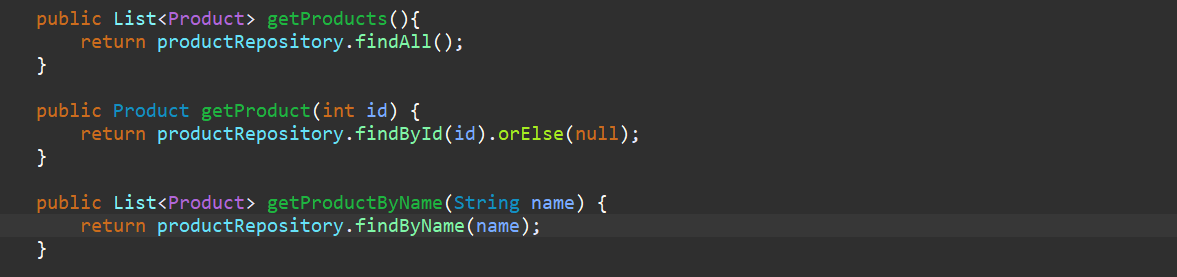
But instead of id if u want to fetch Product object based on some other entity fields like name, quantity and price. So, let’s customized our logic.

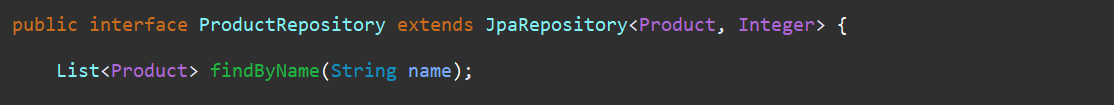


So **findBy** is the prefix and for which field you want to fetch the records that u need to pass here. I want to fetch based on name, so I specify **findByName ().** Now we need to create this method in our Repository.



This findByName () can return List<Product> you can try. So as of now we have written 2 **POST** method, 3 **GET** methods, now let’s write method for **PUT** and **DELETE**.



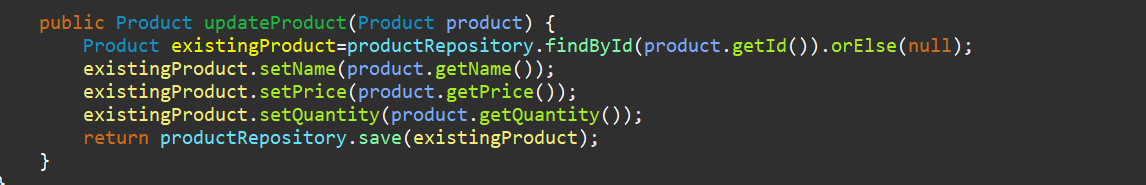


Let’s write a method to delete Product, so delete Product by id and it will return String value.



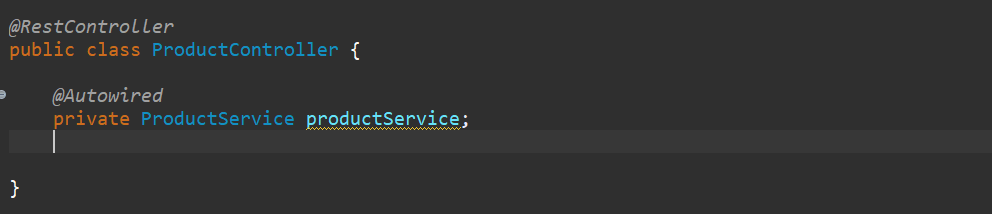
Now let’s write a method for update Product, I will return a Product object after update. Give the Product object which u want to update product. Then what we can do based on the id, because id can’t be modify. Id always would be readable field. Which will be read only. So first get the existing Product which is available in our database. then modify it.

So, 1st based on the id I want to fetch the existing Product, which is avail in our DB, then new then new Product value which we will give as a Postman or UI. Then all the fields I will populate to this Existing Product to update a Product. Because there is no update method in spring data jpa. All you need to play with save method.



Now we have written All the methods for ALL **GET, POST, PUT and DELETE.**

Now let’s write the Controller for it. So first we need to annotate this with **@RestController** , this controller class will talk to our Service class, so for that lets Inject Service class inside it.

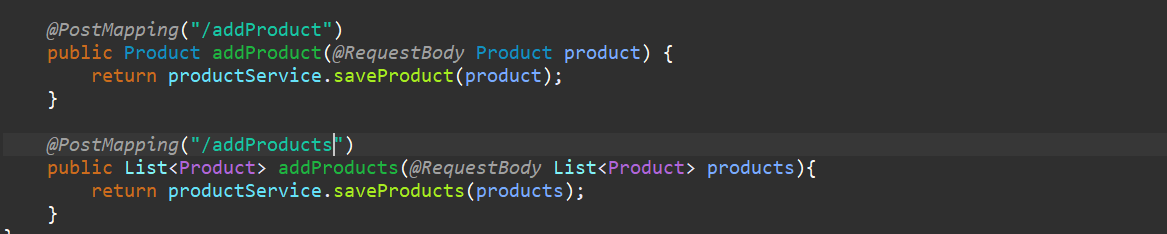


Now we need to write all the rest endpoint for all of the methods presents in our Product Service class.

First let’s write **POST** method. As this is the **POST** method and what Product is my Request so we need to add **@RequestBody,** so that input Json can be mapped to this Input Object. Similarly, there is one more method to save the list of Product objects. So, let’s write 2 **POST** methods first.

**We have written the POST API**

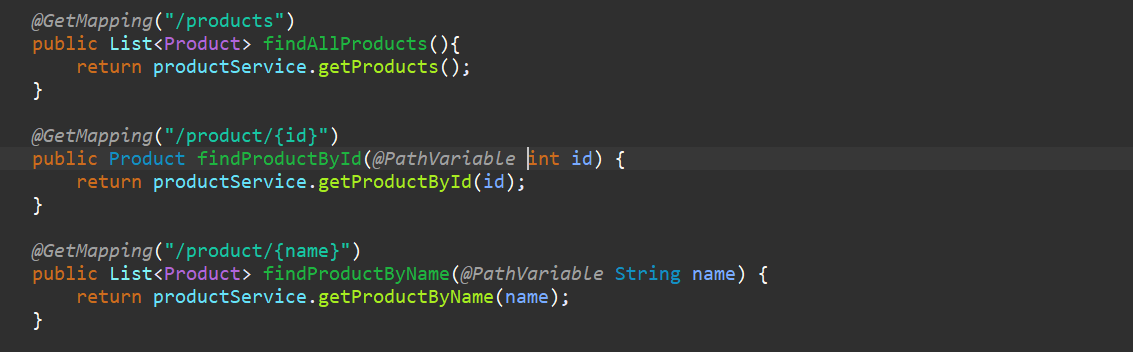
**--------------------------------------------**



**Let’s write the GET API**

------------------------------------

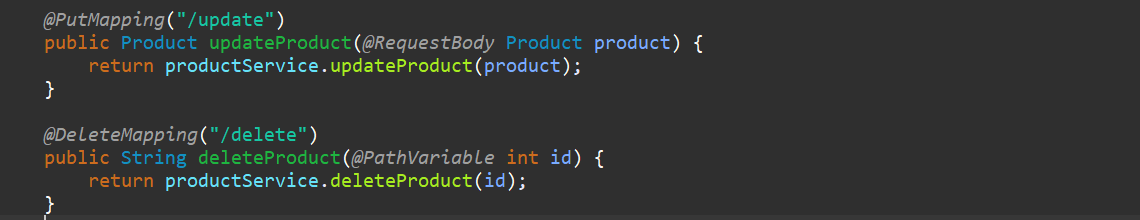
As we are passing id as part of Request Url to get Product Object so we need to use **@PathAvriable** also u can u **@RequestParam**.



Fine we wrote for **POST** and **GET.**

**Now we have to write for PUT and DELETE**

----------------------------------------------------------



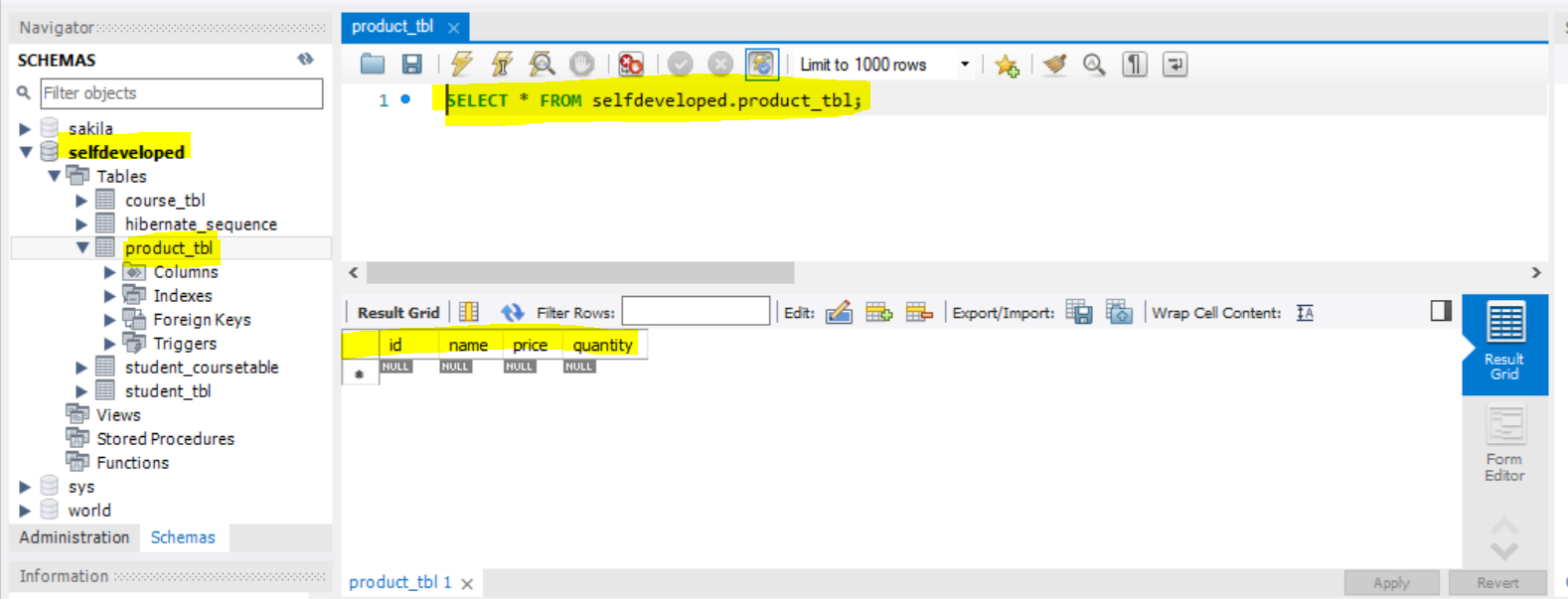
So, we wrote endpoints for all the HTTPS Methods. Let’s run and test through Postman.

Now let’s open MYSQL workbench and see.

Since we enabled **show-sql,** so we can see the query which is generated by Hibernateand our application is up on port 9191.

Hibernate: create table product\_tbl (id integer not null, name varchar (255), price double precision not null, quantity integer not null, primary key (id)) engine=MyISAM

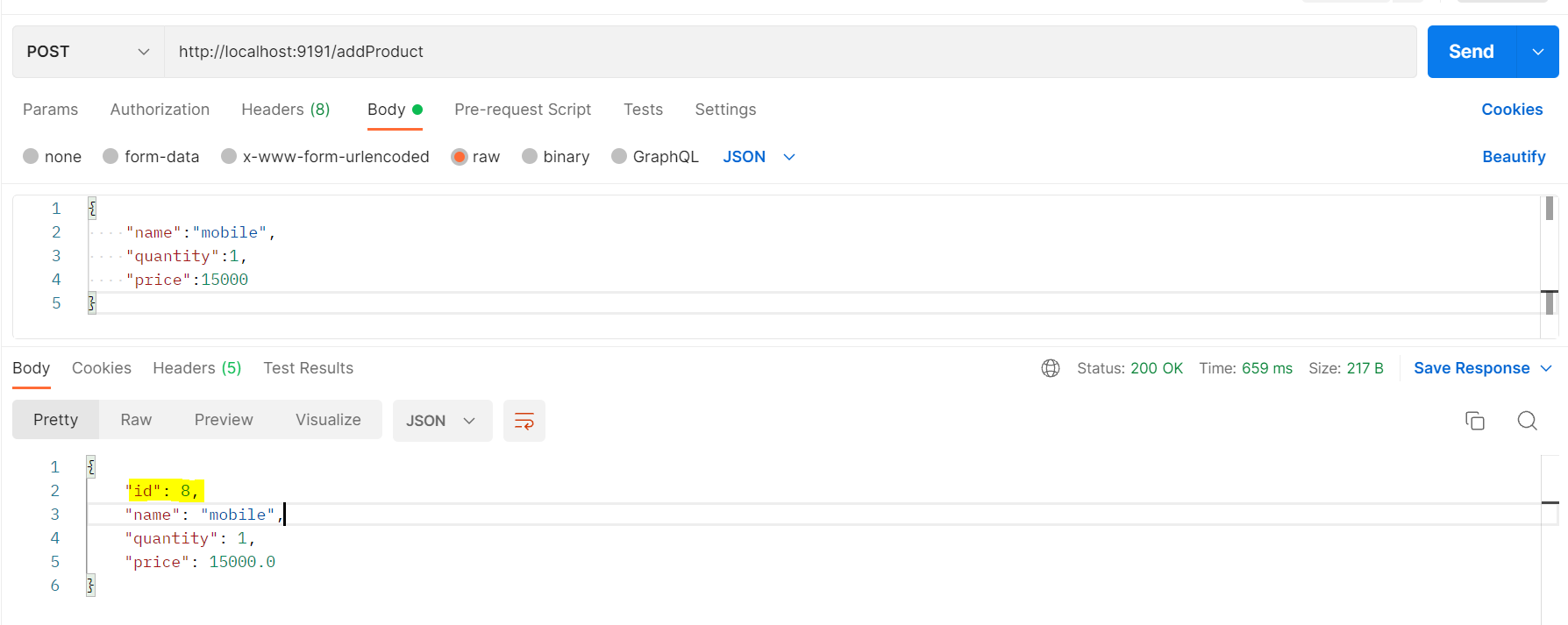
Then go to MySQL and you can see the **schema: selfdeveloped** and **Table: product\_tbl.** So, if u open the table and see there are no records because we didn’t performed any operations yet.



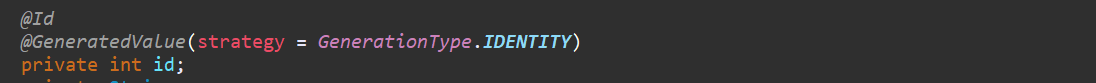
So, perform some Post api to persist some records in Product Table.

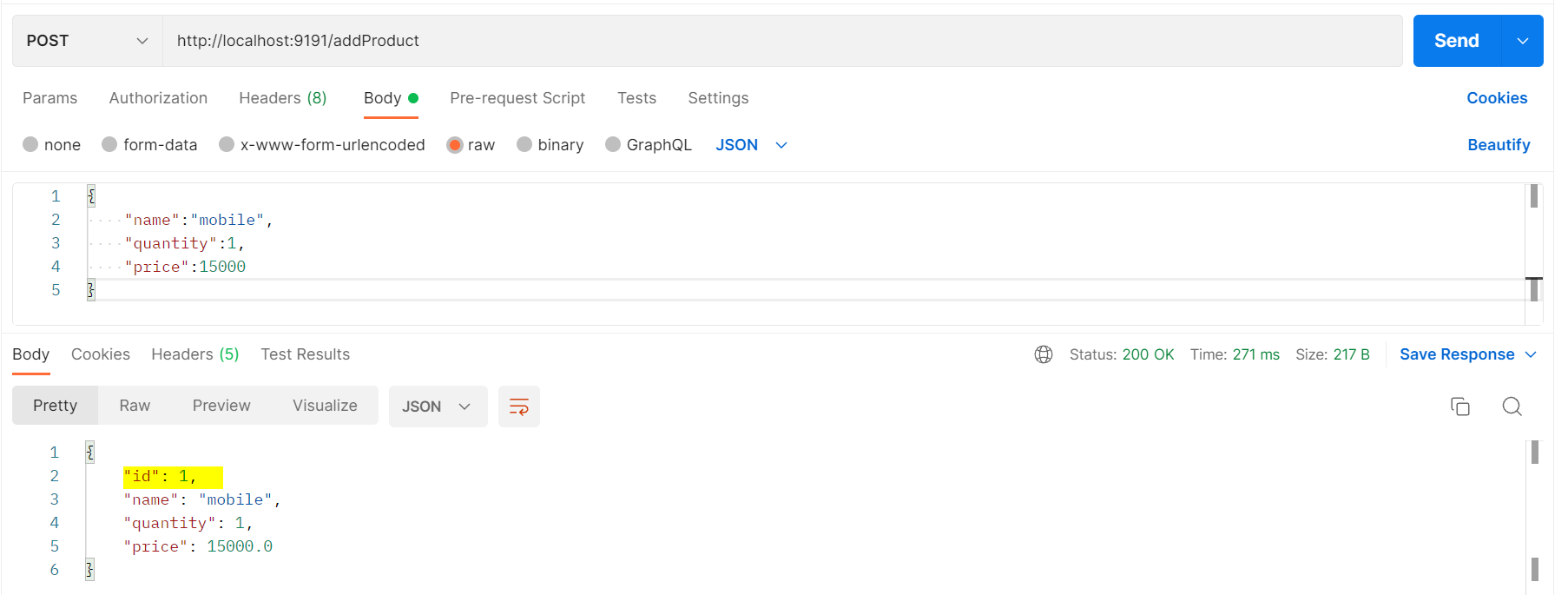
As id is auto generated so in input no need to give the id.

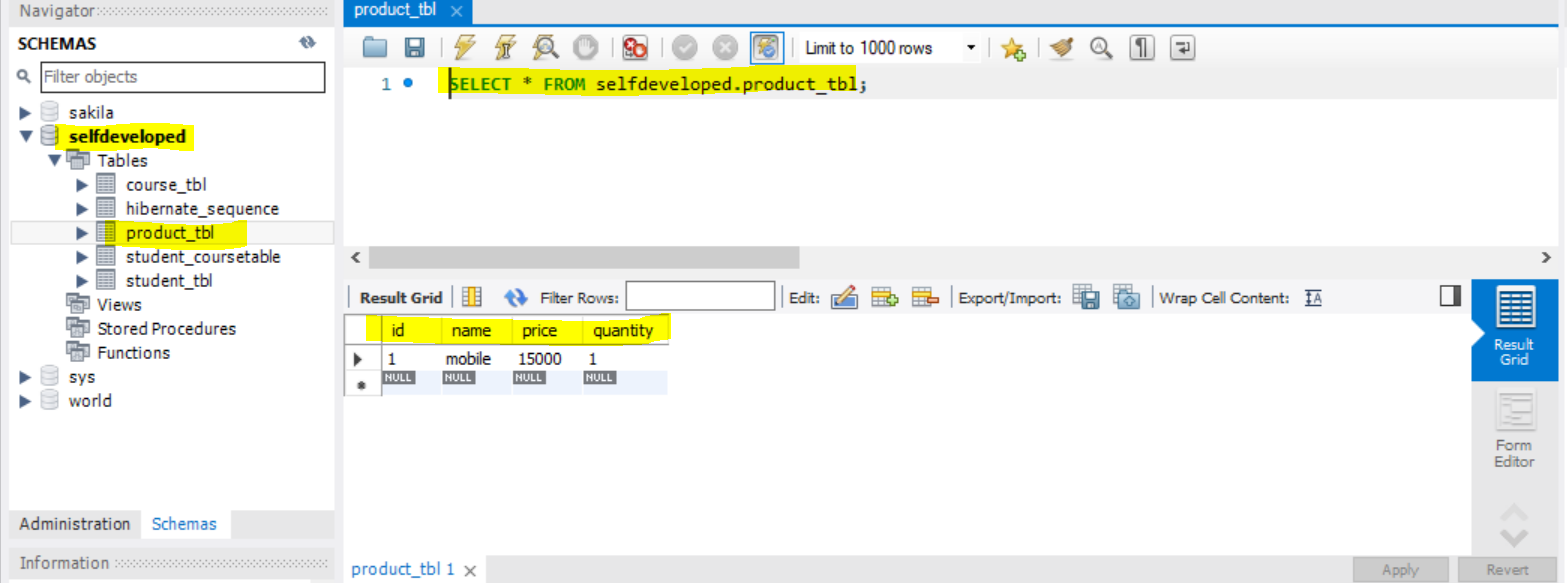
**POST:** [**http://localhost:9191/addProduct**](http://localhost:9191/addProduct)



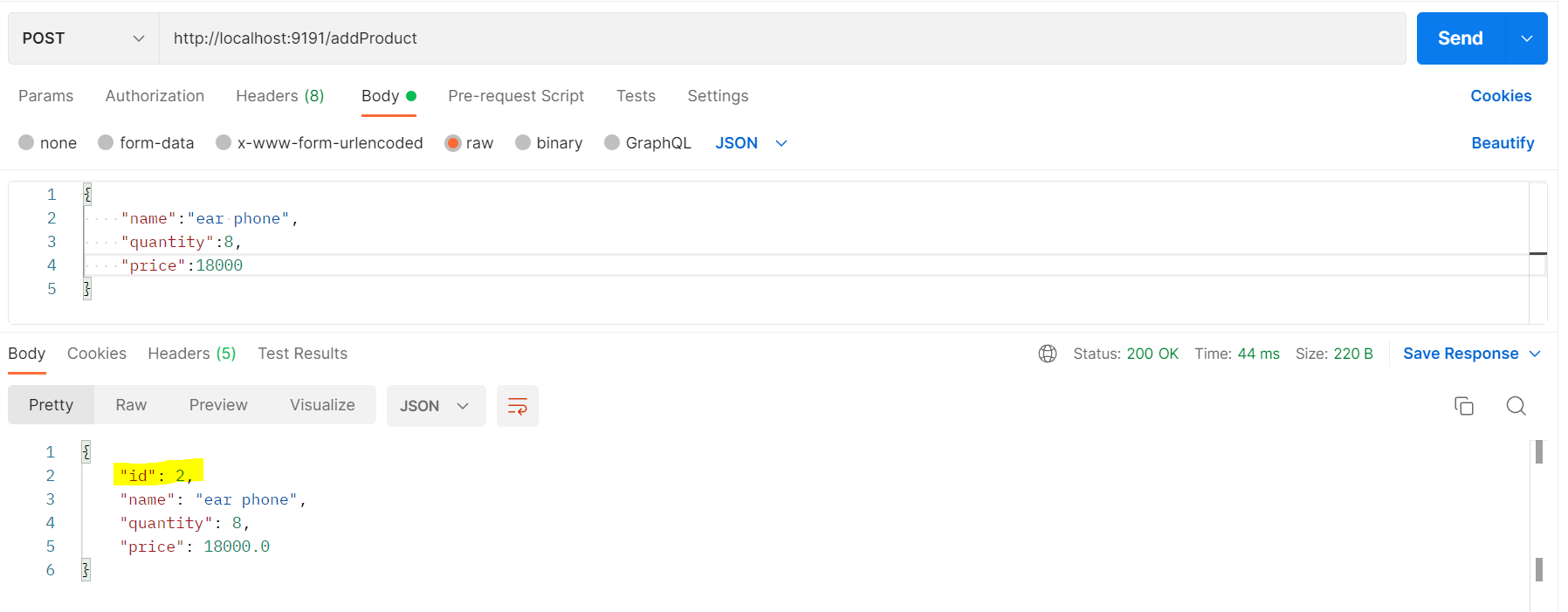
As it is generating any random id…so let’s change @GeneratedValue to IDENTITY



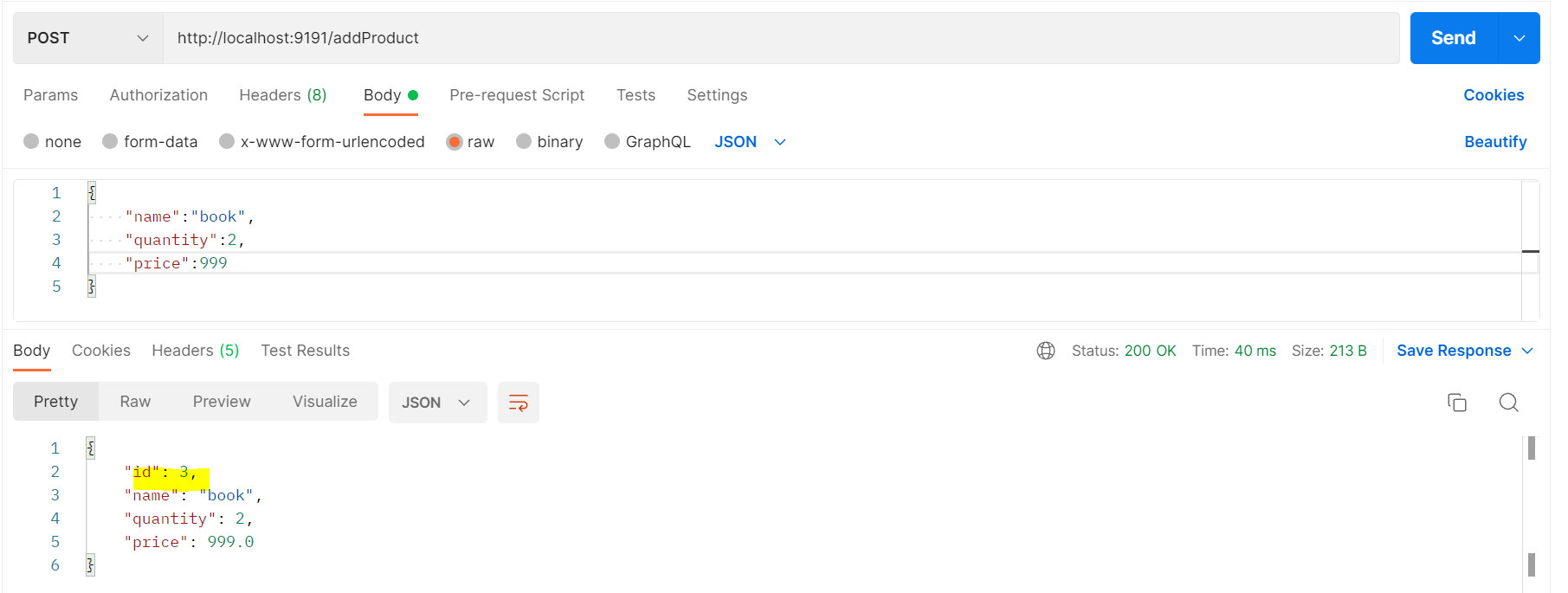




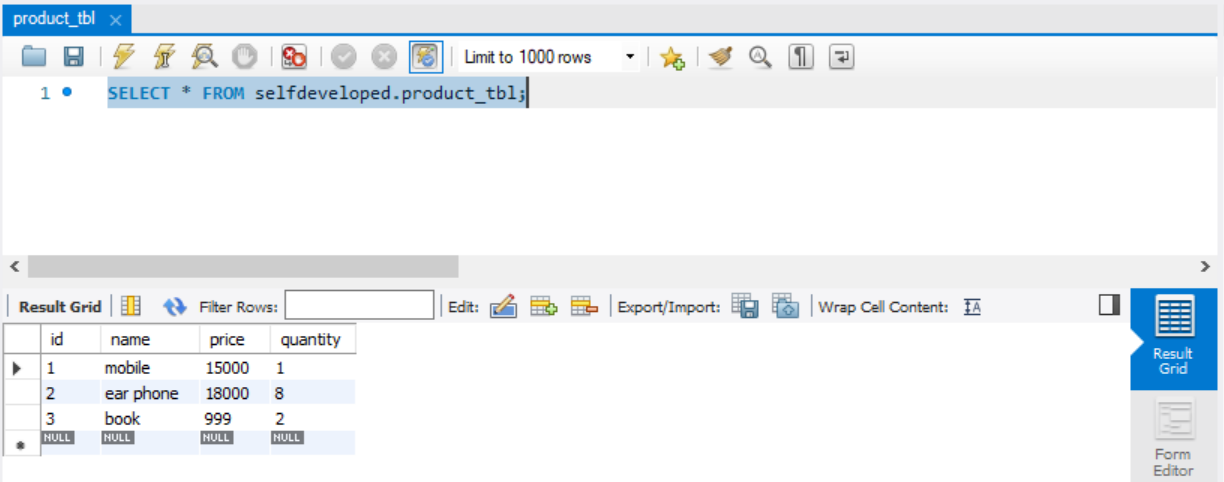
Let me add few more products…



As, u can see the id is auto generated to 2.



We have inserted 3 records. now let’s go to the DB.



So now we will save list of Product Object. Which is our 2nd API call. And instead o giving single product object we need to give list. Saved records with product id 4,5 and 6.

**POST:** [**http://localhost:9191/addProducts**](http://localhost:9191/addProducts)

**Request:**

**[**

**{**

**"name":"x",**

**"quantity":1,**

**"price":987**

**},**

**{**

**"name":"y",**

**"quantity":1,**

**"price":654**

**},**

**{**

**"name":"z",**

**"quantity":1,**

**"price":321**

**}**

**]**

**Response:**

**[**

**{**

**"id": 4,**

**"name": "x",**

**"quantity": 1,**

**"price": 987.0**

**},**

**{**

**"id": 5,**

**"name": "y",**

**"quantity": 1,**

**"price": 654.0**

**},**

**{**

**"id": 6,**

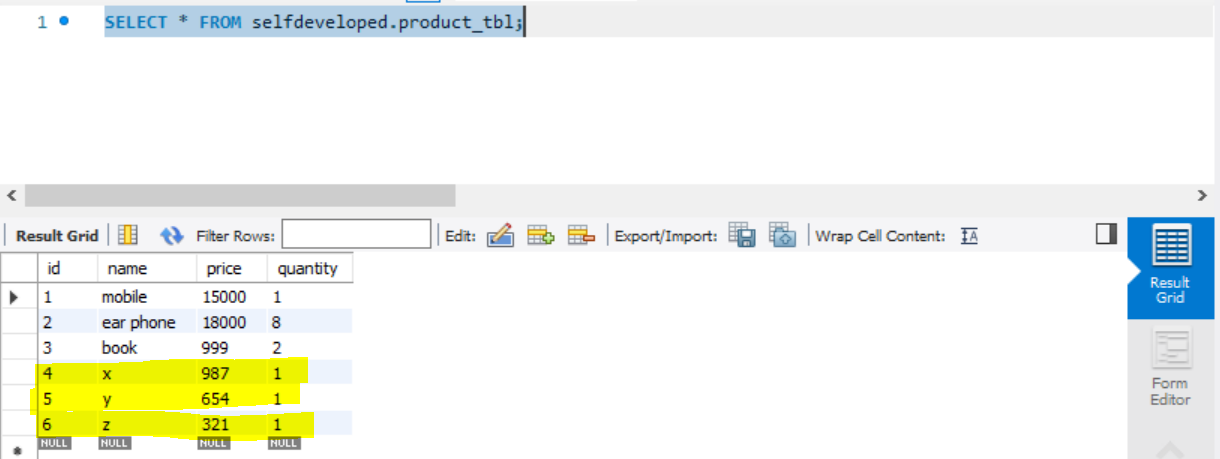
**"name": "z",**

**"quantity": 1,**

**"price": 321.0**

**}**

**]**



So, we verified **POST API** is working as expected.

Now let’s verify the **GET API**.

**GET:** [**http://localhost:9191/products**](http://localhost:9191/products)

**Response:**

**[**

**{**

**"id": 1,**

**"name": "mobile",**

**"quantity": 1,**

**"price": 15000.0**

**},**

**{**

**"id": 2,**

**"name": "ear phone",**

**"quantity": 8,**

**"price": 18000.0**

**},**

**{**

**"id": 3,**

**"name": "book",**

**"quantity": 2,**

**"price": 999.0**

**},**

**{**

**"id": 4,**

**"name": "x",**

**"quantity": 1,**

**"price": 987.0**

**},**

**{**

**"id": 5,**

**"name": "y",**

**"quantity": 1,**

**"price": 654.0**

**},**

**{**

**"id": 6,**

**"name": "z",**

**"quantity": 1,**

**"price": 321.0**

**}**

**]**

You can see All the Records.

Let’s fetch based on Id and name.

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

**Response:**

**{**

**"id": 1,**

**"name": "mobile",**

**"quantity": 1,**

**"price": 15000.0**

**}**

**GET:** [**http://localhost:9191/product/mobile**](http://localhost:9191/product/mobile)

**Response:**

**[**

**{**

**"id": 1,**

**"name": "mobile",**

**"quantity": 1,**

**"price": 15000.0**

**}**

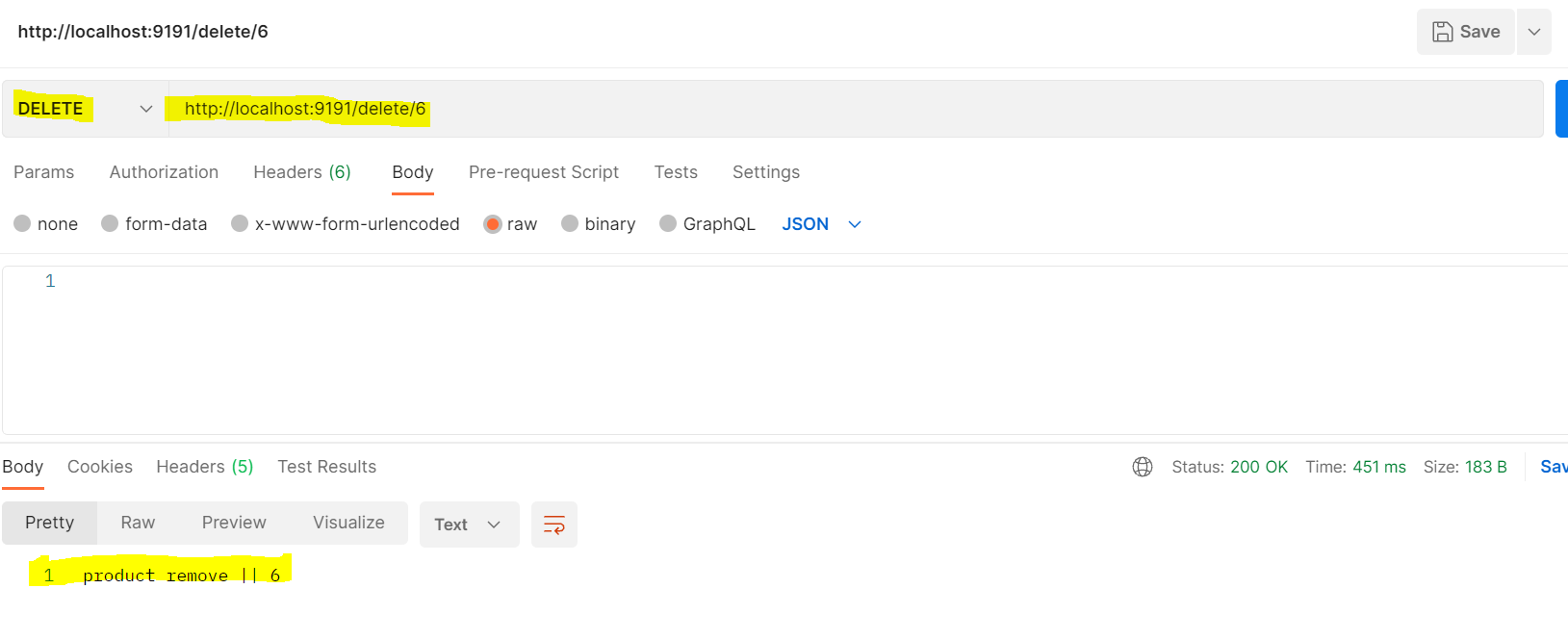
**]**

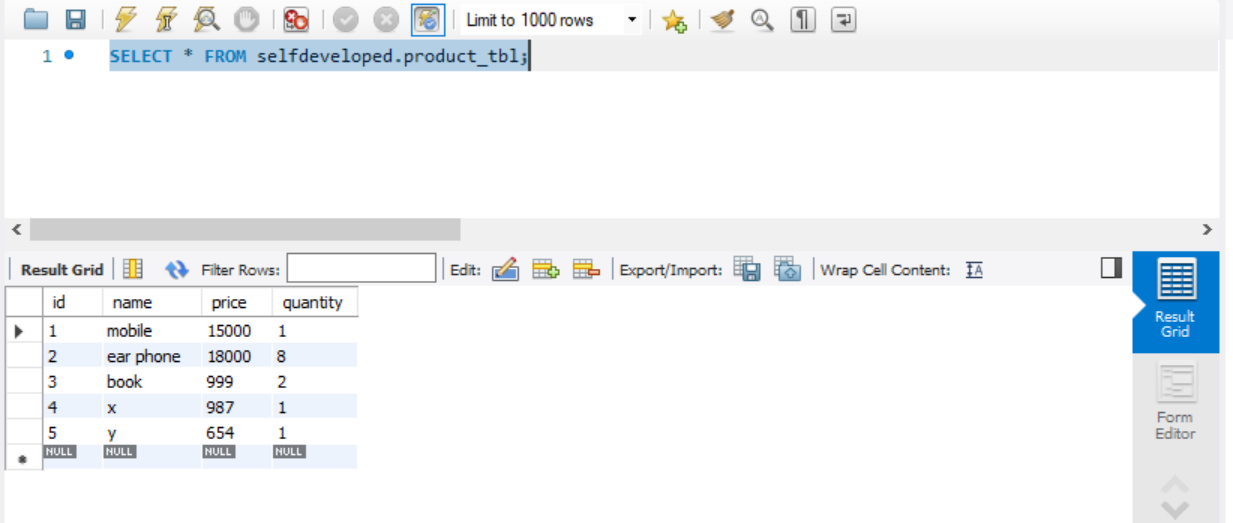
**POST and GET API** is working fine let’s try with **DELETE** one

I want to delete the last Record which we added (**id=6**)

**DELETE:** [**http://localhost:9191/delete/6**](http://localhost:9191/delete/6)







So, if u see DB record(id=6) is not present into our database.

Then go to any of the **POST API,** as we deleted the id=3 so let’s copy that JSON, which is not present in db.

**{**

**"name":"book",**

**"quantity":2,**

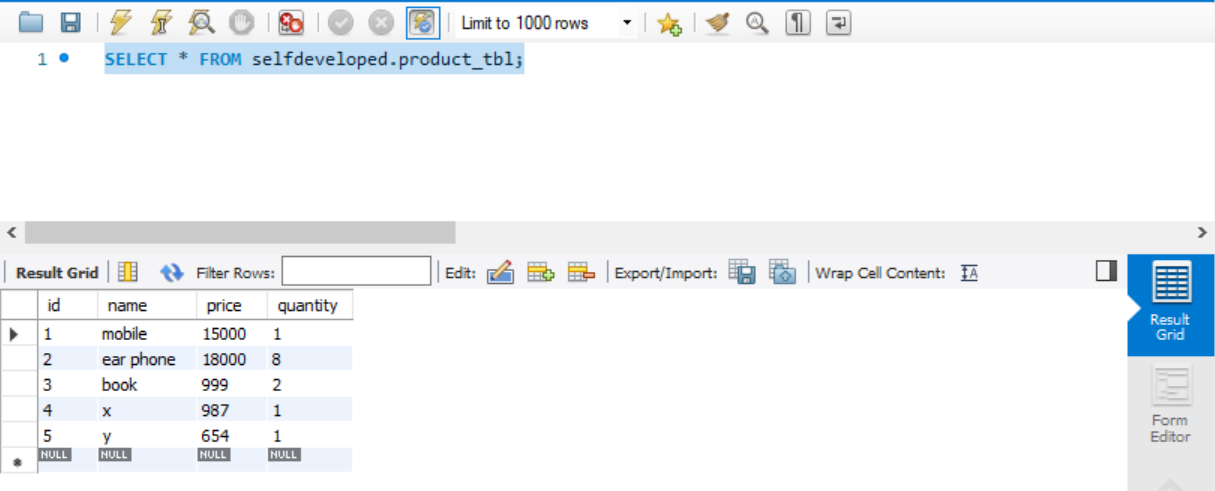
**"price":999**

**}**

Here u need to pass the id, because we are not going to create an entity, we are just going to Update It.

So, based on id we will find an Existing Object. Then will change the field of it.

Before Update our Records:



**PUT:** [**http://localhost:9191/update**](http://localhost:9191/update)

**Request:**

**{**

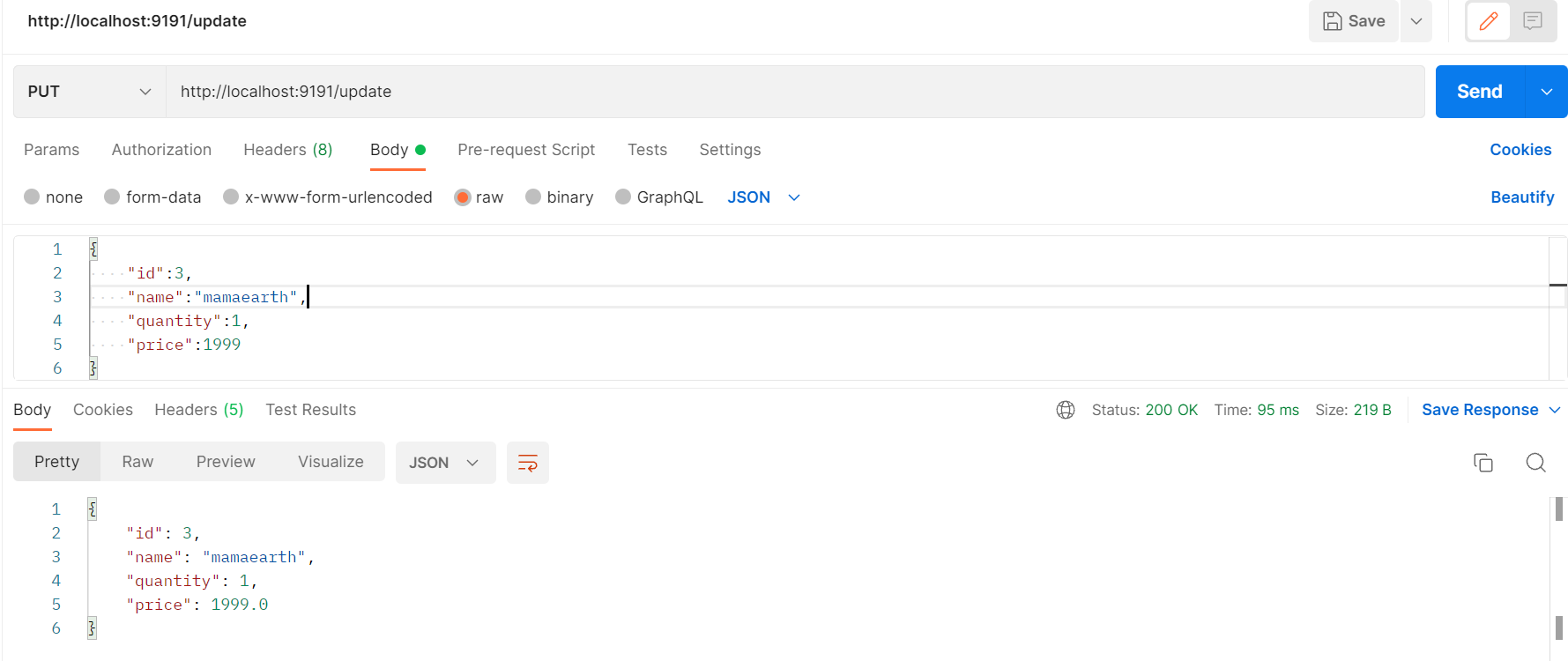
**"id":3,**

**"name":"mamaearth",**

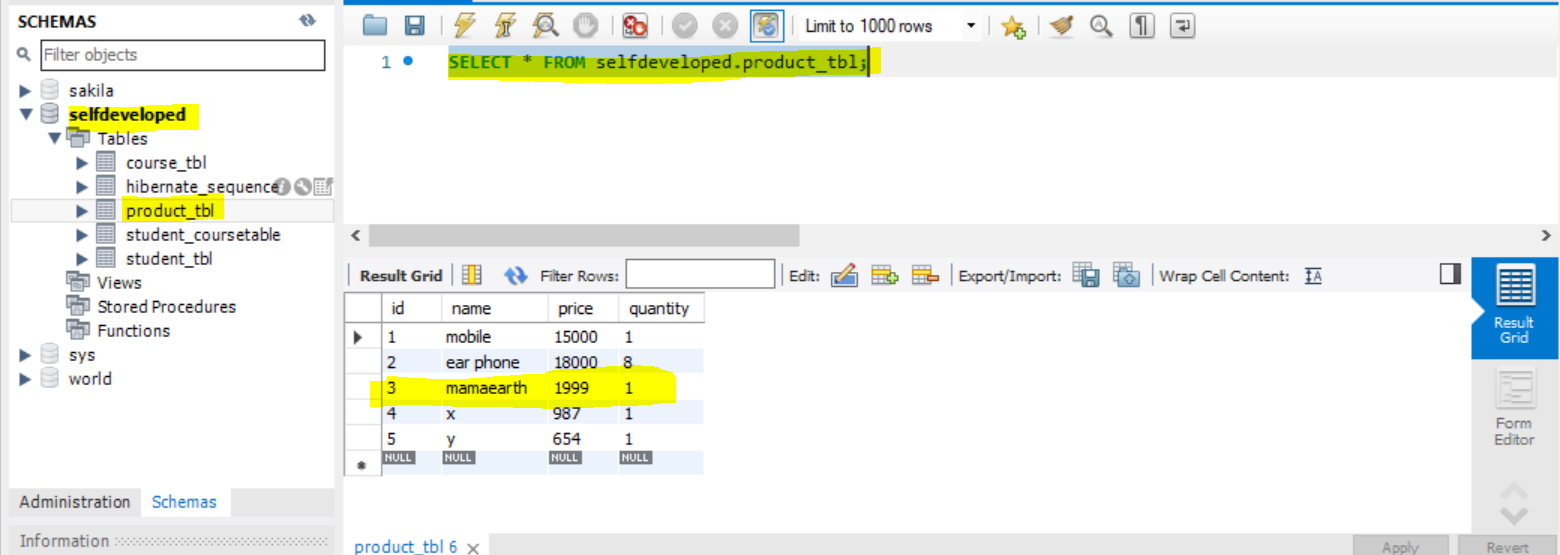
**"quantity":1,**

**"price":1999**

**}**



After PUT API call let’s see Updates Database



This is How We Can Play with Spring Crud Application Using Spring data JPA or you can use any NoSQL database. so already I covered for MongoDB, Cassandra and Neo4J. If Anyone Interested You Can Please Check My NoSQL Playlist.