# SpringBoot: Spring Data JPA - One to Many & Join Query



In this tutorial we will discuss how to do Entity Relationship Mapping using Spring Data JPA. And we will also discuss how to perform JOIN operation using Spring data JPA.

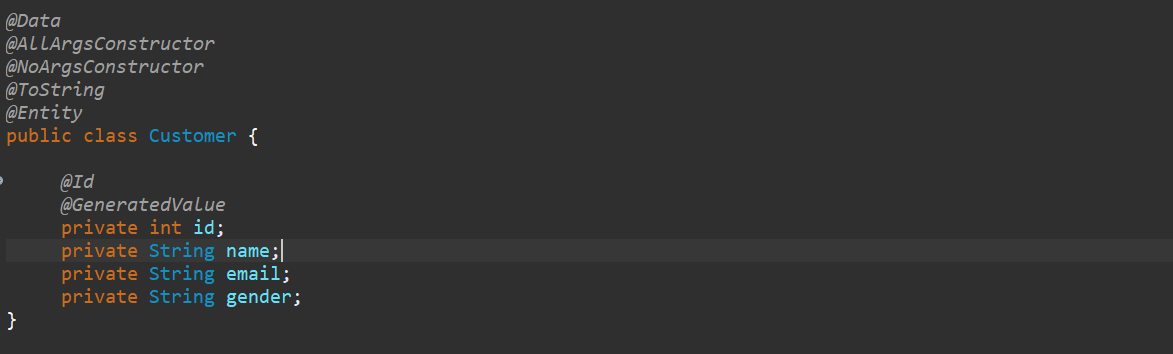
Let’s consider an Example which we will build today. If u can see this diagram, there is one customer deals with multiple products. **Product-1**, **product-2** and **product-3**.

So, by considering this scenario we can make a relationship between customer and product as a **One to Many**, it means one customer can purchase multiple products. Let’s start In Action –

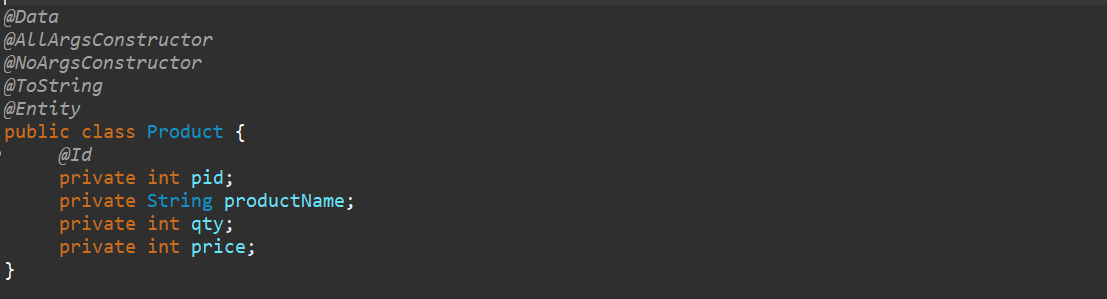
**Application- spring-data-jpa-mapping**

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

So, lets create 2 Entity **Customer** and **Product**.



Similar way let’s create another entity called Product



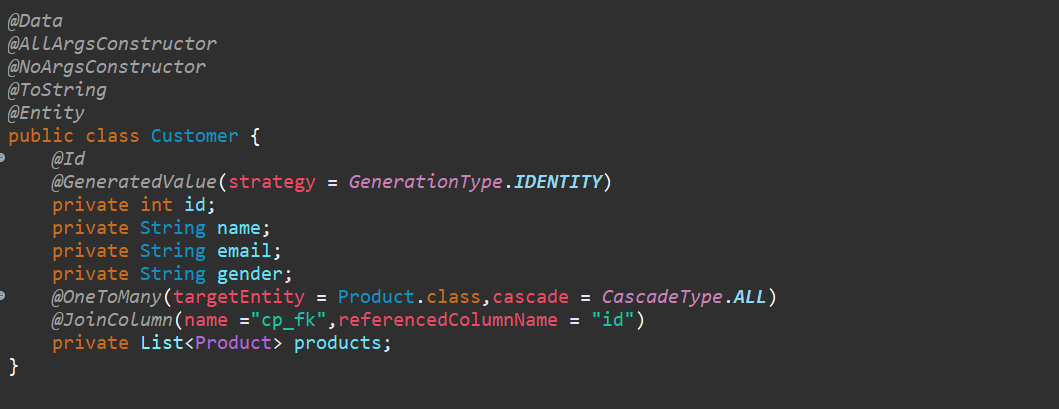
So, In Customer I want Id to be auto generated. But in Product I am going to add it manually.

Now we have created 2 Entity so, now we need to maintain the relationship between these 2.

So here a Customer can have multiple Product. So, in Customer I can directly injects the List of Product.

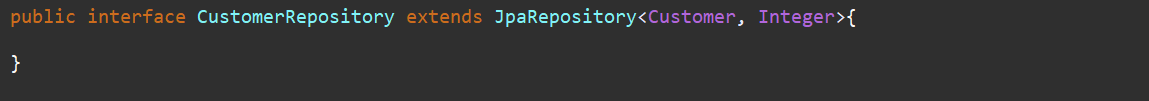
So, this is like a one to many. So, there is one annotation **@OneToMany.** In one-to-many u need to specify which is ur target Entity, so my target entity is **Product.class.** then we need to specify the cascade, here **CascadeType** is **ALL**. Then we need to specify the Join Column. Give the name of your Join Column. Let me give name something like **customer product foreign key(cp\_fk)** and give the referenced column name which id u want to map as a foreign key so I want to map a customer id(**id**).

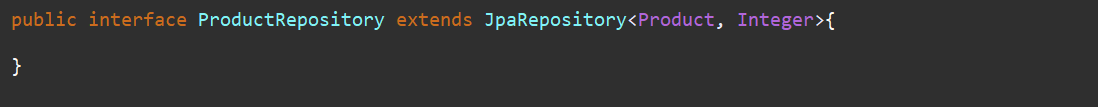
**Now, whatever the Customer Object you will create, and you will add the list of Product. This Customer id will be act as a foreign key in your Product table.**



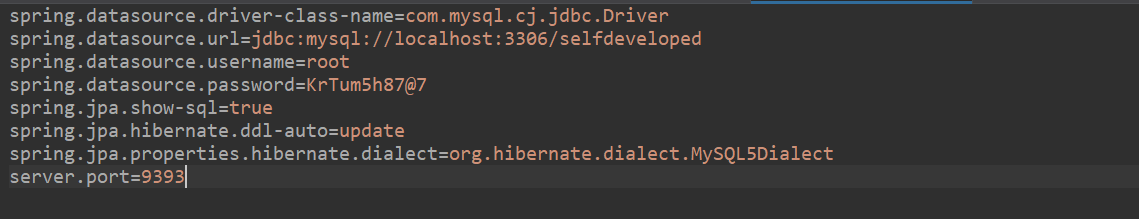
So, this is what all about one-to-many mapping in hibernate.

Now let’s create a repository called Customer Repository.



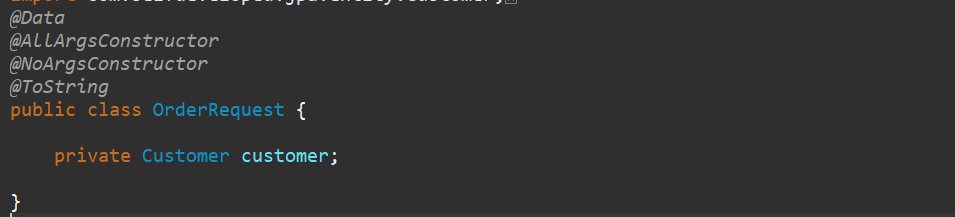


Now let’s provide datasource information in **application. properties** file.



Let’s go ahead and create our controller class---

Now what I will do instead of playing with this Entity object I will create one more package which will be **DTO**. In our dto I will create one class **OrderRequest.** So now in my controller I will pass this **OrderRequest** as a request**.**



In controller, lets write a method to save Customer object so once we will save the Customer object so it will save Customer as well as List of Product belongs to that particular customer.

That’s what we have created one DTO class OrderRequest, from OrderRequest we can extract Customer object. And from Customer object we can extract List of Product. So, lets write a method for it…

***@PostMapping*("/placeOrder")**

**public Customer placeOrder(*@RequestBody* OrderRequest request) {**

**return customerRepository.save(request.getCustomer());**

**}**

**Now write one more rest endpoint which will return a List of Customer Object. So now let’s test theses 2 Rest API, If the data is stored into Database, then we will perform Join Operation.**



So, let’s Run our application, and hit request from Postman ->

**POST -** [**http://localhost:9393/placeOrder**](http://localhost:9393/placeOrder)

So as part of Request body we need to pass the Customer as well as List of Products….

So already I created the Request template let me add that….

Payload:

**{**

**"customer": {**

**"name": "Kaushal",**

**"email": "kk@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 201,**

**"productName": "Mobile",**

**"price": 10000,**

**"qty": 1**

**},**

**{**

**"pid": 399,**

**"productName": "Laptop",**

**"price": 20000,**

**"qty": 3**

**}**

**]**

**}**

**}**

**Response-**

**{**

**"id": 1,**

**"name": "Kaushal",**

**"email": "kk@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 201,**

**"productName": "Mobile",**

**"qty": 1,**

**"price": 10000**

**},**

**{**

**"pid": 399,**

**"productName": "Laptop",**

**"qty": 3,**

**"price": 20000**

**}**

**]**

**}**

**Similarly let’s insert the 2nd record…**

Request-

**{**

**"customer": {**

**"name": "Bharat",**

**"email": "b2@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 334,**

**"productName": "Watch",**

**"qty": 1,**

**"price": 5000**

**},**

**{**

**"pid": 712,**

**"productName": "Clothes",**

**"qty": 4,**

**"price": 27000**

**}**

**]**

**}**

**}**

**Response-**

**{**

**"id": 2,**

**"name": "Bharat",**

**"email": "b2@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 334,**

**"productName": "Watch",**

**"qty": 1,**

**"price": 5000**

**},**

**{**

**"pid": 712,**

**"productName": "Clothes",**

**"qty": 4,**

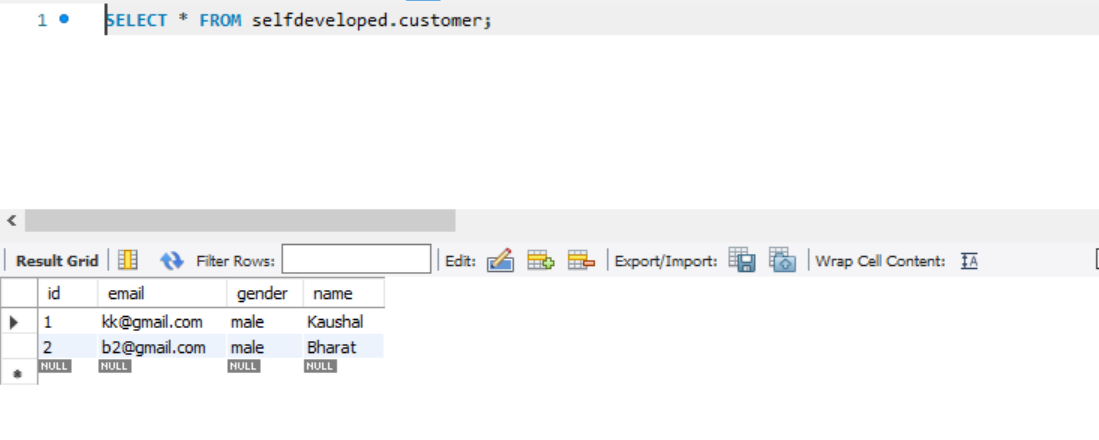
**"price": 27000**

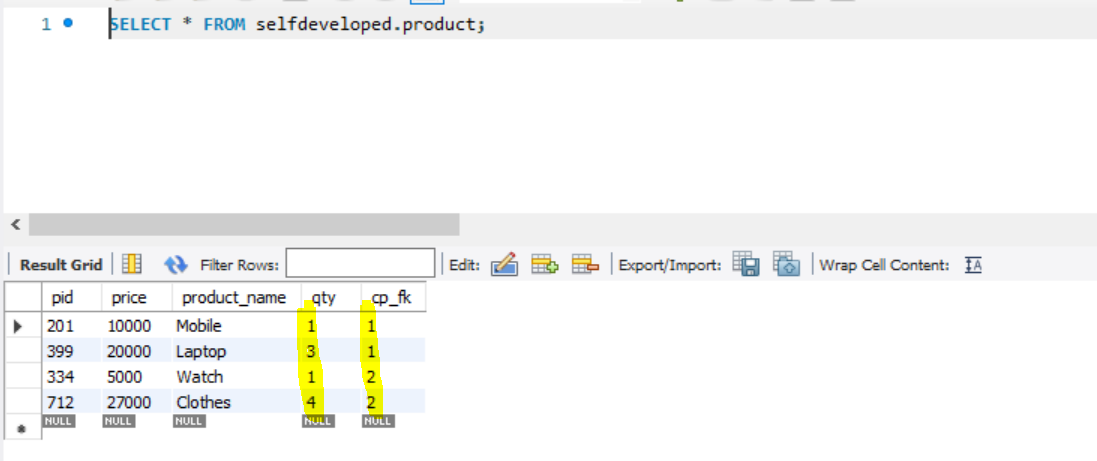
**}**

**]**

**}**

**Let’s check the database records---**





So, if you observed here cp\_fk which is our foreign key getting mapped from our customer. So first 2 products were purchased by customer 1, that’s why the primary key added as a foreign key here.

So, we inserted 2 records let’s verify using GET API call…

**GET** [**http://localhost:9393/findAllOrders**](http://localhost:9393/findAllOrders)

**Response-**

**[**

**{**

**"id": 1,**

**"name": "Kaushal",**

**"email": "kk@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 201,**

**"productName": "Mobile",**

**"qty": 1,**

**"price": 10000**

**},**

**{**

**"pid": 399,**

**"productName": "Laptop",**

**"qty": 3,**

**"price": 20000**

**}**

**]**

**},**

**{**

**"id": 2,**

**"name": "Bharat",**

**"email": "b2@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 334,**

**"productName": "Watch",**

**"qty": 1,**

**"price": 5000**

**},**

**{**

**"pid": 712,**

**"productName": "Clothes",**

**"qty": 4,**

**"price": 27000**

**}**

**]**

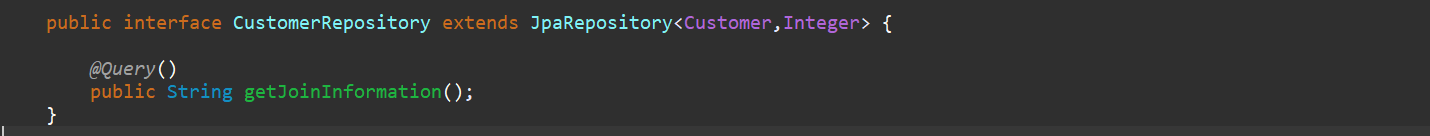
**}**

**]**

**Now let’s do Join Operation to fetch couple of fields from Customer table and from Product table.**

So, from Customer table we will fetch Customer name and from Product table we will fetch Product name using JOIN Query. Then just go to our Repository class and there we need to write our JOIN Query.

**Imp->**



So, on top of this method, we need to provide one annotation **@Query**. And inside this query we need to write a Join statement.

So, to fetch customer name and product name simply we can write

***@Query* ("SELECT c.name, p.productName FROM Customer c JOIN c.products p")**

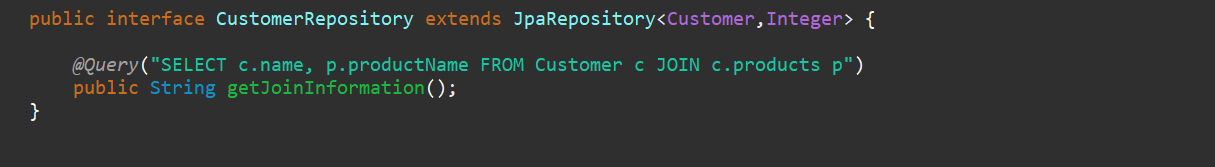
**public String getJoinInformation();**

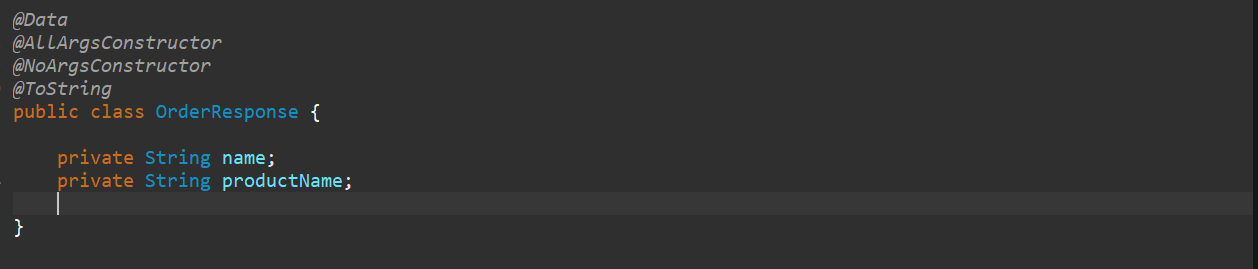
where c will be the alias name here.

So, give the ALIAS here and we need to give the JOIN here, then in Customer table only we do have a list of Products. So, get the Project object from Customer itself. This is how we are writing query in Spring data JPA. Its nowhere related to SQL Join Query. This query statement related to JPA. So, specify **c.getProducts** and then p this is what the Alias we specify.

So, this query will work as expected. But there is a problem so in this query we are fetching only 2 fields and if u observed here the return type is String so either it should be String Array, or we need to create a class with these 2 fields. So that we can tell hibernate or Spring Data Jpa these are the 2 fields please Map to my Domain or map to my DTO which I created.

As I am getting 2 fields so let me create one new DTO something like **OrderResponse**.





**So, we created the Response DTO.**

So, in our Repository method in stead of returning String we can return **List<OrderResponse>.**

**public interface CustomerRepository extends JpaRepository<Customer,Integer> {**

***@Query*("SELECT c.name, p.productName FROM Customer c JOIN c.products p")**

**public List<OrderResponse> getJoinInformation();**

**}**

So, this is what the expected response. we are expecting name and product name that’s what we have created OrderResponse. But how Spring data will know that these 2 fields I need to map to OrderResponse. That’s what we need to specify to this query. And need to pass 2 fields as part of constructor.

So, SELECT new **com.selfdeveloped.jpa.dto. OrderResponse (c.name, p.productName)**

**public interface CustomerRepository extends JpaRepository<Customer,Integer> {**

***@Query*("SELECT new com.selfdeveloped.jpa.dto.OrderResponse(c.name, p.productName) FROM Customer c JOIN c.products p")**

**public List<OrderResponse> getJoinInformation();**

**}**

So, if u observed this query, we are just expecting these 2 fields, to be pass as part of my constructor, which I created the OrderResponse class. So, using JOIN query it will fetch these 2 values then it will be mapped to our OrderResponse object. That’s what we create the new OrderResponse object inside the query itself.

So, this is how we can write Join Query using Spring data Jpa. So, let’s verify either this is working or not what we need to do we just need to call this method from controller.

***@GetMapping*("/getInfo")**

**public List<OrderResponse> getJoinInformation(){**

**return customerRepository.getJoinInformation();**

**}**

So, let’s restart our application. To verify Join statement let’s verify the below rest endpoint url…

**GET** [**http://localhost:9393/getInfo**](http://localhost:9393/getInfo)

**Response-**

**[**

**{**

**"name": "Kaushal",**

**"productName": "Mobile"**

**},**

**{**

**"name": "Kaushal",**

**"productName": "Laptop"**

**},**

**{**

**"name": "Bharat",**

**"productName": "Watch"**

**},**

**{**

**"name": "Bharat",**

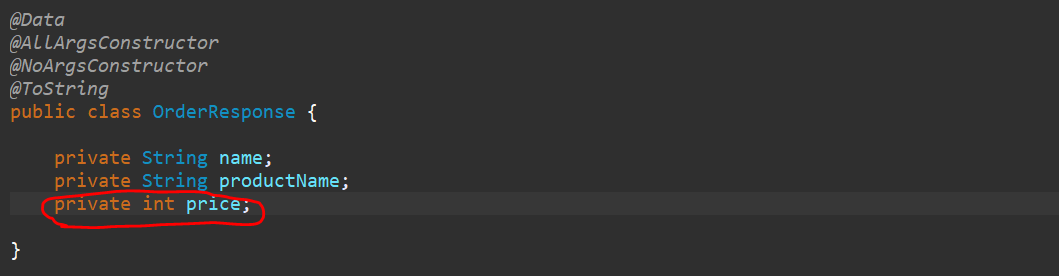
**"productName": "Clothes"**

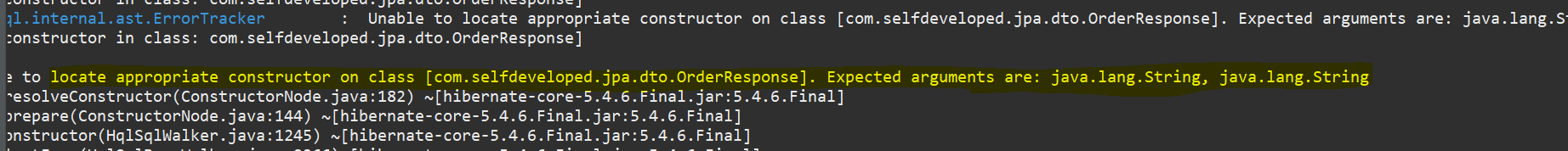
**}**

**]**

We can see the Join Statement Result. So, we are just getting the customer name from customer table and product name from Product table. As we have 4 records, we are able to see those entry. So, Join operation is working as expected.

But there is some disadvantages let me show you that, go to our DTO class here I have 2 field and in query also we do have only 2 fields that’s why its working. But if I will add one more field in our OrderResponse then it will not work it will throw an error. Add one field price as well. Let’s re-run it we will get the **RuntimeException.**





I will tell u why and how to resolve also.so the error is

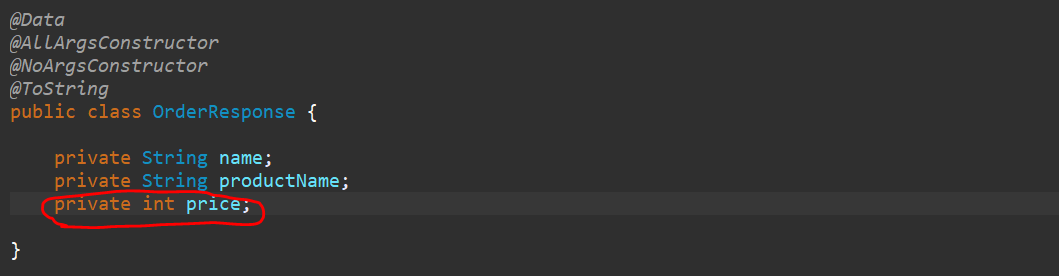
Unable to locate appropriate constructor on class [com.selfdeveloped.jpa.dto.OrderResponse]. Expected arguments are: java.lang.String, java.lang.String

Bcz its expecting only 2 fields bcz as we written in our Query name and productName only 2 elements not a third elements. So this OrderResponse Constructor, this is constructor only right …package name, class name and with 2 fields right this is one only the constructor.

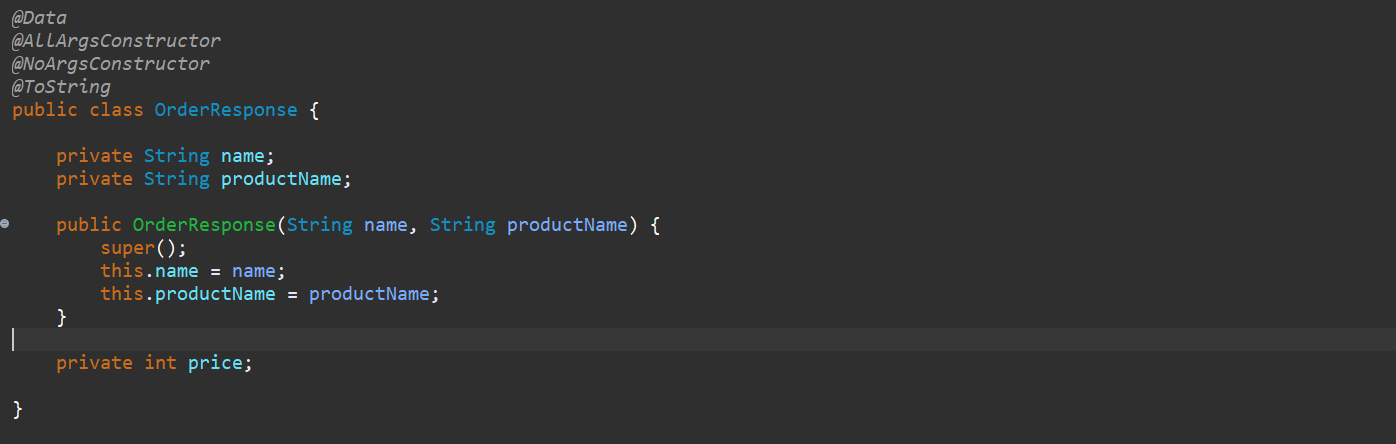
***@Query*("SELECT new com.selfdeveloped.jpa.dto.OrderResponse(c.name, p.productName) FROM Customer c JOIN c.products p")**

So, this Jpa Query will always expect the constructor with these 2 arguments. So, what can we do to avoid such kind of Exception.

Go to your OrderResponse as u added All Argument Constructor it will create constructor with these 3 fields, right??



So, we just need to manually add the constructor. So just specify the 2 fields which is mentioned in our query.



Now if I run this application I will not get any exception.

**GET** [**http://localhost:9393/getInfo**](http://localhost:9393/getInfo)

**Response-**

**[**

**{**

**"name": "Kaushal",**

**"productName": "Mobile",**

**"price": 0**

**},**

**{**

**"name": "Kaushal",**

**"productName": "Laptop",**

**"price": 0**

**},**

**{**

**"name": "Bharat",**

**"productName": "Watch",**

**"price": 0**

**},**

**{**

**"name": "Bharat",**

**"productName": "Clothes",**

**"price": 0**

**}**

**]**

If you observed, we are getting **name** and **productName,** but we are getting price by default. Bcz as part of join query we are not getting price value. So, u can ignore using **JsonIgnore** property or using **JsonInclude**. So let me do that as well.

***@Data***

***@AllArgsConstructor***

***@NoArgsConstructor***

***@ToString***

***@JsonIgnoreProperties*(ignoreUnknown = true)**

***@JsonInclude*(*JsonInclude*.*Include*.*NON\_DEFAULT*)**

**public class OrderResponse {**

**private String name;**

**private String productName;**

**public OrderResponse(String name, String productName) {**

**super();**

**this.name = name;**

**this.productName = productName;**

**}**

**private int price;**

**}**

So, we don’t want to see the default value null for String or 0 for integer as part of my response. So, this is how we can avoid it but if you want to add more field as part of your query, so u just need to specify that number of constructors in respective DTO class. Let’s say if it is returning 4 values then u need to add constructor with those 4 values. So that u won’t get any issue.

So, let’s re-run this application.

**GET** [**http://localhost:9393/getInfo**](http://localhost:9393/getInfo)

**[**

**{**

**"name": "Kaushal",**

**"productName": "Mobile"**

**},**

**{**

**"name": "Kaushal",**

**"productName": "Laptop"**

**},**

**{**

**"name": "Bharat",**

**"productName": "Watch"**

**},**

**{**

**"name": "Bharat",**

**"productName": "Clothes"**

**}**

**]**

Now we are not able to see the default values. So this is how we can avoid it.

**This is How we can perform Join and Association Mapping using Spring Data JPA.**

**JPQL & Native Query (Custom Query)**

**1. GET** [**http://localhost:9393/customerByName/Kaushal**](http://localhost:9393/customerByName/Kaushal)

**Response:**

**[**

**{**

**"id": 1,**

**"name": "Kaushal",**

**"email": "kk@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 201,**

**"productName": "Mobile",**

**"qty": 1,**

**"price": 10000**

**},**

**{**

**"pid": 399,**

**"productName": "Laptop",**

**"qty": 3,**

**"price": 20000**

**}**

**]**

**}**

**]**

**2. GET** [**http://localhost:9393/customer/b2@gmail.com**](http://localhost:9393/customer/b2@gmail.com)

**[**

**{**

**"id": 2,**

**"name": "Bharat",**

**"email": "b2@gmail.com",**

**"gender": "male",**

**"products": [**

**{**

**"pid": 334,**

**"productName": "Watch",**

**"qty": 1,**

**"price": 5000**

**},**

**{**

**"pid": 712,**

**"productName": "Clothes",**

**"qty": 4,**

**"price": 27000**

**}**

**]**

**}**

**]**

**3. GET** [**http://localhost:9393/native/b2@gmail.com**](http://localhost:9393/native/b2@gmail.com)

**{**

**"timestamp": "2022-09-06T13:41:37.246+0000",**

**"status": 500,**

**"error": "Internal Server Error",**

**"message": "could not extract ResultSet; SQL [n/a]; nested exception is org.hibernate.exception.SQLGrammarException: could not extract ResultSet",**

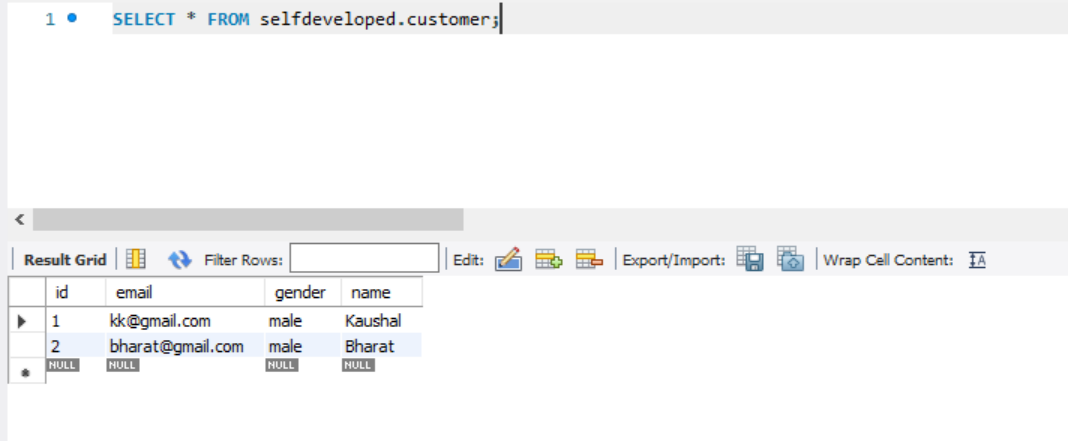
**"path": "/native/b2@gmail.com"**

**}**

**4. PUT http://localhost:9393/update\_customer\_data/2/bharat@gmail.com**

**Response**

**New Email for customer-id2 is -** [**bharat@gmail.com**](mailto:bharat@gmail.com)



**5. PUT** [**http://localhost:9393/change\_customer\_data/1/kaushal@gmail.com**](http://localhost:9393/change_customer_data/1/kaushal@gmail.com)

**Response-**

**New Email for customer-id1 is - kaushal@gmail.com**

