**Spring JPA Tutorial**

**Introduction**

We use JDBC (An API from Java) to connect to our database. In JDBC, we try to fetch all the records from the database and convert them into objects which is not an ideal way.

An ideal way would be that our classes and objects that are defined represent certain records in the database. There are many frameworks available which give us this functionality called ***ORM (Object Relationship Mapping).***

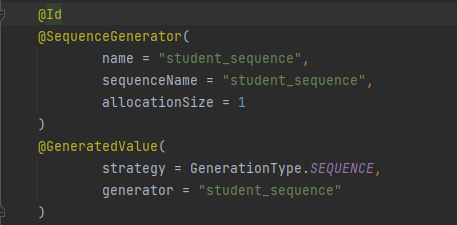
There are frameworks like ***Hibernate*** which provide this functionality. But these frameworks (or ORM providers) have their own sets of implementation for it. With this, it becomes very difficult for us to switch from one framework to another.

That’s why Java created a standard specification (***Java Persistence API or JPA)*** that is used by every ORM provider so that it becomes easy to switch if we want to.

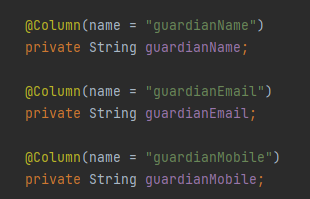
JPA is just a specification that is used by the frameworks (ORM providers)

**Creating Tables**

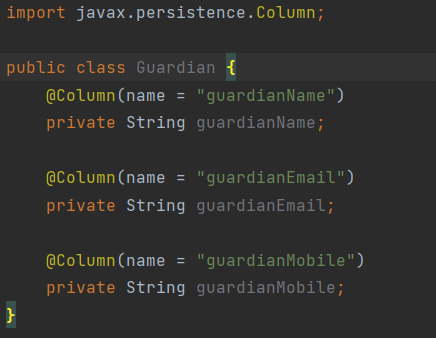
Primary key should be auto-increment when adding a new record in the table. For this (in production), we use ***@SequenceGenerator*** for this.



With ***@SpringBootTes***t, when we test our application, then it will impact our database. With ***@DataJpaTest***, it will not impact our application since it flushes the data after program execution.



Now the information above, should be stored in the separate database but not the ***student*** database.



We don’t want to create a separate class for ***Guardian,*** we just want to embed this class in the ***Student*** class.

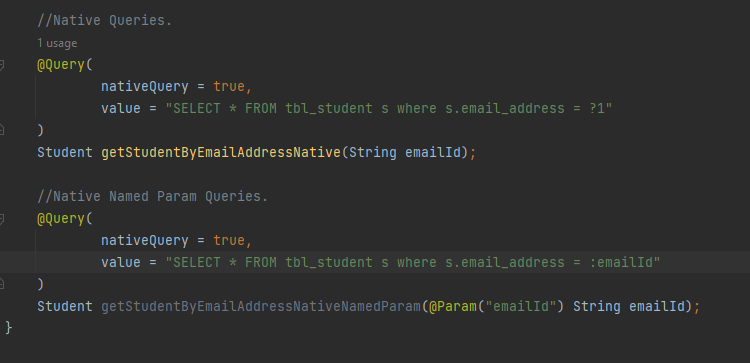
For this, we can use ***@Embeddable*** annotation to the class we want to embed. Then we can use ***@Embedded*** annotation in whichever class we want ***Guardian*** class to be embedded in.

To see the available methods that we can use to fetch the records, go through:: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation>

**Query Annotation**

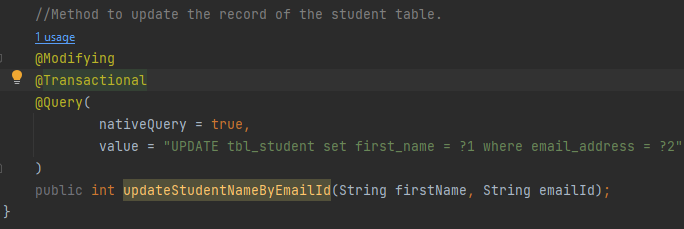
The queries are written on the basis of class names that are there in our ***entities*** folder. It is not written on the basis of the table names.

JPQL queries are not SQL queries (***Native queries***).



First query is the ***native query*** (simple SQL query). But in this for any query we have to ? sign followed by the position of the query in the method argument.

Second query is the ***native named parameter query*** where inside of mentioning the position of the parameter, we give it a name.



Whenever we want to modify the record in the table like updating or deleting to be specific, then we have to use ***@Modifying*** annotation. Also for the same operations, we have to create a transaction for it. When the method completes its execution, the transaction gets committed to the database.

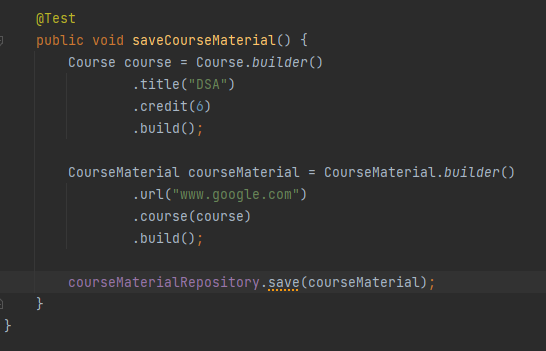
To do that, we can use ***@Transactional*** annotation. The main use of transaction is that ***we can execute multiple queries from the same method.***

**Different Relationships in JPA**

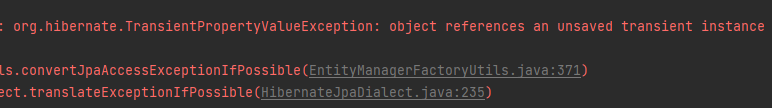
1. **One-To-One Relationships**

***“Uni-directional relationships can be used when we need to have the information on one side.***

***Bi-directional relationships are used when we want to access both the entity and the information about the other entity.”***

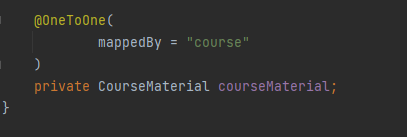


When we run the above code, it gives the error.



This is because in the database, we don’t have any ***courses*** at the moment. Although we are creating one, it has to be saved in the database also. This is where the concept called ***Cascading*** comes into action.

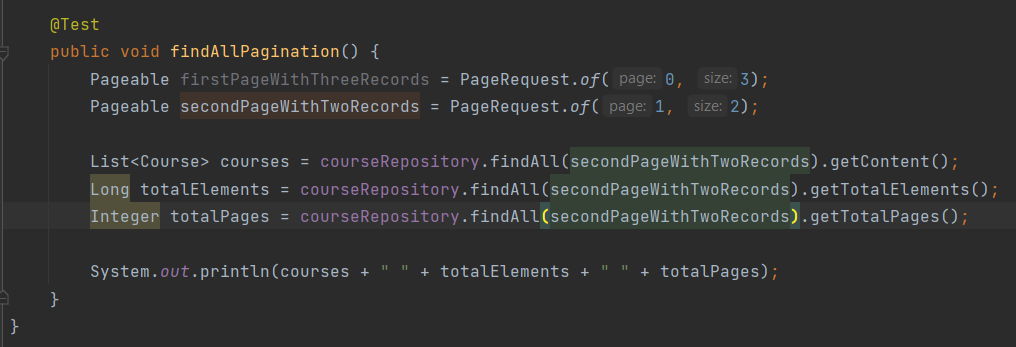
Here ***CourseMaterial*** is the parent & ***Course*** is the child. When we want to fetch the ***CourseMaterial,*** we have the option to either get the ***Course*** data (***EAGER***) as well along with the parent data or just get the parent data (***LAZY***).



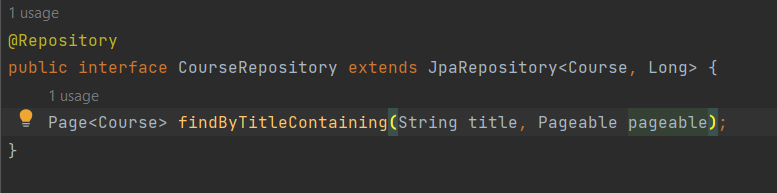
As we have already joined the two tables using ***@JoinColumn*** inside the ***CourseMaterial*** class, here in Course class, we can’t use the same annotation again. Instead here we have to say that this attribute is already mapped by some other attribute of ***CourseMaterial*** class using the ***mappedBy*** property.

We can use ***optional*** property to specify that a ***CourseMaterial*** cannot be formed if a ***Course*** is not created. By default the optional property is set to true.

**Pagination & Sorting**

****

If we want to fetch some data with pagination then we can do the following:

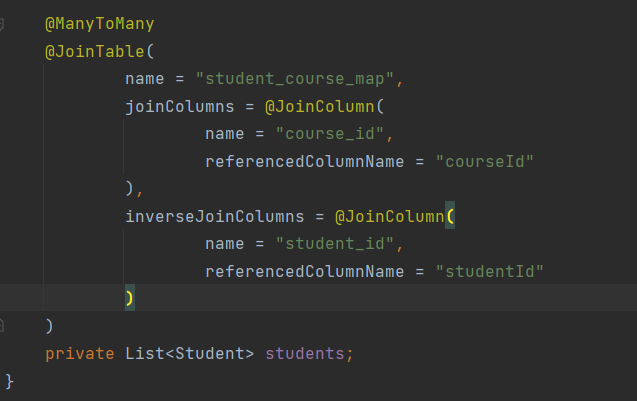


**Many-To-Many Relationship**

Here there represents a ***third table*** which contains the mapping between the two tables.

***Article Link for bi-directional @OneToMany & @ManyToOne::*** <https://www.coderscampus.com/hibernate-onetomany-bidirectional-relationship/>

***Avoid using unidirectional one-to-many relationships.***



Here we don’t use ***@JoinColumn*** since we will be creating a third table that will contain the mapping of the two tables. That’s why we use ***@JoinTable*** for this.

***Article Link for Cascading::*** <https://howtodoinjava.com/hibernate/hibernate-jpa-cascade-types/>