# Lab: Workshop – MiniORM Part 2

This document defines the lab assignments for the ["DB Advanced" course @ Software](https://softuni.bg/courses/databases-advanced-hibernate) University.

Since you’ve come to this point it is considered that you have fully finished Part 1 of our Mini ORM. By following the guidelines of this document you will add more functionality to the framework like create, delete and so on.

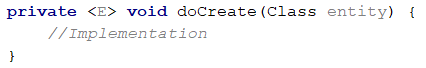
One of the things our framework cannot do to this moment is to create a table. In big applications schema manipulation is done with a default configuration of the framework, **depending on our development**. What does that mean?

When we have our app **running** and users actively **using** it, we rarely(almost never) want to lose or drop our data. Thus why complex ORMs like Hibernate or Entity, look for **changes in the models**(e.g. User) constantly and if set **update** **the** **structure** of the tables.

## Create Table

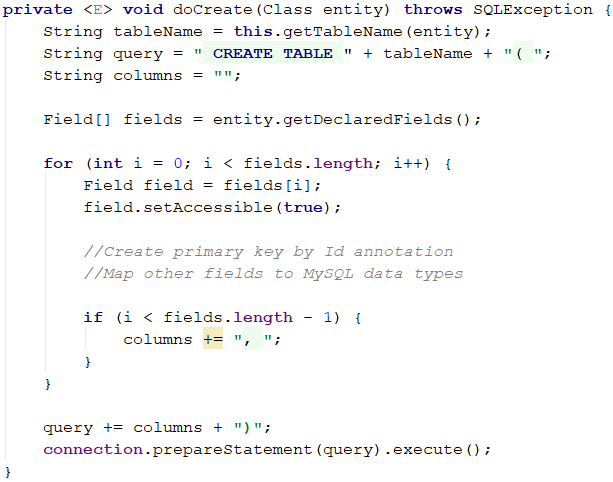
Creation of the tables will be done by the EntityManager object that we’ve already implemented. It should be able to parse the fields in our models to table columns – set appropriate data type, name(from Column annotation) and constraints.

Begin by creating such method doCreate(Class entity):

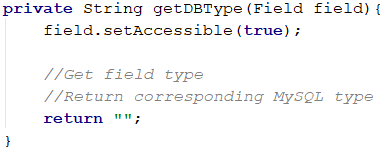


The method has several simple steps that has to perform:

* Get the table name that should be created
* Start building a query
* Scan model’s declared fields and map them to annotated names and MySQL data types



Data types mapping is very similar to the fillField method. You can do it by creating a helper method getDBType(Field field) and use it in the iteration of the entity’s fields.



### Hints

For example java int or Integer type is mapped to the “INT” SQL type, String to “VARCHAR(…)” and so on.

Consider when you should check the need of table creation. Sometimes we might need a simple alter and just add fields to already existing tables. If we drop and create new table we might lose important information about already existing users, no matter if the new fields are left blank after alter.

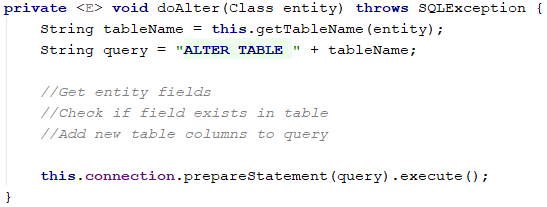
## Test Create

Create a new custom entity and try to persist it with the EntityManager

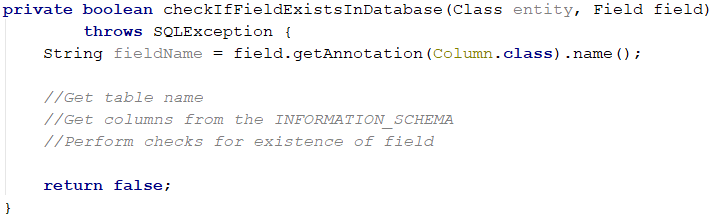
## Alter Table

In most cases when our application is running, we need to just **ALTER** the tables and add new model fields.

A new private method is needed:



And a helper one to let us know if the field that we will be trying to insert doesn’t exist already:



Information about the columns of a table can be retrieved from the information\_schema of the database server. It is an object where each MySQL instance stores information about all the other databases that the MySQL server maintains. Its is also referred to as the data dictionary and system catalog.

### Hints

* Get information about existing tables and their columns from information\_schema
* Use the ResultSet class to check if retrieved fields by COLUMN\_NAME

## Test Alter

Add new fields to already existing entity class(you can use the one made from 3.) and try to persist a new object.

## Delete

Try implementing the last CRUD operation – the delete.

### Hints

Create a method that receives database column and delete criteria as parameters, very similar to the find methods and delete records matching given criteria.