

ORM in Java & Postgres

SWE3



September 24, 2021

FH Technikum – BIF5

Chahed Rajoub (if19b166)

Table of Contents

[Used languages: 2](#_Toc94366639)

[Functionality: 2](#_Toc94366640)

[Data Model: 2](#_Toc94366641)

[**Manual:** 4](#_Toc94366642)

ORM Functionality:

## Used languages:

* Java
* Postgres

## Functionality:

* A Good ORM can lead to much less code and much less repetition in a project
* Allows the changes in Database technologies
* Queries should be generated by the mapper without even thinking about Dialect or how the transaction is happening.
* This Design is simplified to approach a simple application of OR Mapper using Java, Postgres Container in Docker
* Database tables are already present, and the Mapper enables the queries on these tables
* The mapper is designed Code first style following these steps:
  + Configuring the connection with the Database as settings in persistence layer
  + Adding the classes as tables and the properties as columns in Entity layer
  + Adding different functions 'Select, delete, insert, update, drop' in Services layer
  + Adding transactions services 'join, find' in Service layer
  + Model layer is used to show a demo of the OR Mapper and uses annotations
  + Annotations layer to define the reflection function that will define the tables and the columns

## Data Model:

This ORM supports tables that has foreign keys to refer to other tables and connect to them. as an example, for 1-n relationship:

Table

Description automatically generated with medium confidence

as an example, for n-m relationship:

‘ Diagram

Description automatically generated

SQL Script could be like the following:

Text

Description automatically generated

**Manual:**

The following steps explain how to use this ORM:

1. Create a Postgres Database Docker Container. It could be #something like this:

*$ docker run --name Base -p 5432:5432 -e POSTGRES\_PASSWORD=if19b166 -d postgres*

1. Adding Url, username, password to Config.xml in Settings folder.
2. Create a new ORM object where it is needed.
3. Create classes as entities with annotations to connect to the tables in DB

@table -----------> for a class

@Column ----------> for a property with the name of the column

@PrimaryKey-------> for unique ids

@ForeignKey-------> for foreign ids with the entity related to it for safe choice

1. ORM can make the calls to our database and save them in lists of objects and cashed the values after first call.
2. The following functions can be used to fulfill the needs of this ORM:

|  |  |  |
| --- | --- | --- |
| **Function** | **Requires** | **Returns** |
| **setCash** | boolean to allow the cash | It is void Method to set the cash |
| **CreateTable** | Class Entity that is created by the user with proper annotations | Void method creates a table in postgres Container |
| **DropTable** | Class Entity that is created by the user with proper annotations | Void method drops a table in postgres Container |
| **SelectAllRows** | Class Entity that is created by the user with proper annotations | returns list of all objects of this class |
| **SelectAllColumns** | Class Entity that is created by the user with proper annotations | return list of Strings of columns names of the table mapped to the class |
| **SelectByID** | Class Entity with the id of the Primary key | returns an object of the class responding to this ID |
| **SelectByColumn** | Class Entity and the column and its value | returns list of objects that match the column value |
| **Argument** | create a neu argument with column name , value and operator | new argument to be used in the query |
| **SelectMultiArgQuery** | Class Entity , where operator (AND or OR), List of Arguments | List of Objects that match the required logic |
| **SelectOneArgQuery** | Class Entity and the argument | List of Objects that match the required logic |
| **JoinbyForiegnKey** | 1-n relation with the class entities and the id that matches the PK in one class with FK in the other one | Map of One Object and list of Objects related to it |
| **JoinAll** | 1-n relation with the class entities at any ID | Map of Objects and list of Objects related to them |
| **Insert** | New Object of the entity | returns new ID and updates the Object after saving |
| **UpdatebyID** | New Object and the ID | Void Method updates the row |
| **DeleteRowbyId** | Object and the ID | Delete the row by the ID |