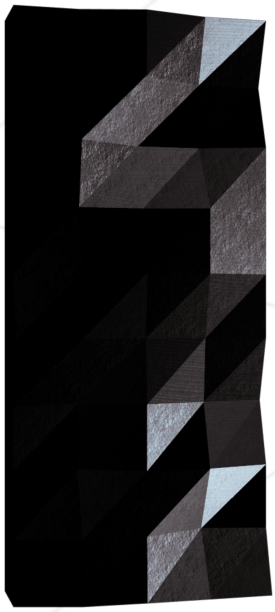

Formation PHP - Symfony

D05 - SQL vs ORM

Andrei Husar andrei.husar@pitechplus.com
Staff 42 bocal@staff.42.fr

Summary: Following 42 formation course, you will learn to use SQL and ORM (Object-Relational Mapping) in Symfony.

Contents

Chapter I

Foreword

One of the most common and challenging tasks for any application involves persisting and reading information to and from a database. Although the Symfony Framework doesn't integrate any component to work with databases, it provides tight integration with a third-party library called Doctrine. Doctrine's sole goal is to give you powerful tools to make database interactions easy and flexible.

Chapter II

General Rules

- This subject is the one and only trustable source. Don't trust any rumor.
- This subject can be updated up to one hour before the turn-in deadline.
- The assignments in a subject must be done in the given order. Later assignments won't be rated unless all the previous ones are perfectly executed.
- Be careful about the access rights of your files and folders.
- You must follow the `turn-in process` for each assignment. The url of your `GIT` repository for this day is available on your intranet.
- Your assignments will be evaluated by your Piscine peers.
- In addition to your peers evaluation, a program called the "Moulinette" should also evaluate your assignments. Fully automated, The Moulinette is tough and unforgiving in its evaluations. As a consequence, it is impossible to bargain your grade with it. Uphold the highest level of rigor to avoid unpleasant surprises.
- All shell assignments must run using `/bin/sh`.
- You must not leave in your turn-in repository any file other than the ones explicitly requested By the assignments.
- You have a question? Ask your left neighbor. Otherwise, try your luck with your right neighbor.
- Every technical answer you might need is available in the `mans` or on the Internet.
- Remember to use the Piscine forum of your intranet and also Slack!
- You must read the examples thoroughly. They can reveal requirements that are not obvious in the assignment's description.
- By Thor, by Odin! Use your brain!!!

Chapter III

Day-specific rules

- Use both annotations and configuration files for defining routes.
- Use controller-defined form types.
- Each displayed page must be properly formatted and should contain DocType, and HTML tags like `html`, `body`, `head`.
- The server used for this day is the one integrated in Symfony. It should be started and stoped using Symfony console commands.
- Only explicit request URLs should render a page without error. The not configured URLs should generate a 404 error.
- The requested URLs should work with and without trailing slash. E.g: both `/ex00` and `/ex00/` must work
- For solving each exercise, you will have to use only SQL, only ORM or both of them. If this condition is not met, no points will be given for your solution.
- Each exercise has to have its own table inside the database.
- The naming of database table should be explicit and correctly given. Eg: if you want to create a table for a model called **Person**, the database table should be named **persons**.

Chapter IV

Exercise 00

	Exercise 00
Exercise 00: Database table with SQL	
Turn-in directory : *ex*00/	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

This exercise will help you learn the basic of SQL usage inside of Symfony framework. The requirements of this exercise are simple:

- Create a bundle for this exercise, named ex00;
- Create the SQL query which will create a new table inside the database;
- Configure database connection;
- Create route which will call the generation of table;
- **No ORM is allowed in this exercise**;
- The structure of the table should correspond to the given one.

The result of this exercise has to be a page which contains a button/link to create the database table and a success/error message regarding the result of the creation.

The database table structure is the following:

- id - integer - primary key
- username - string - unique
- name - string
- email - string - unique

- enable - boolean
- birthdate - datetime
- address - long text

The call to create table should not fail in case the table is already created.

Chapter V

Exercise 01

	Exercise 01
Exercise 01: Database table with ORM	
Turn-in directory : *ex*01/	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

After introducing SQL with Symfony, it's time to move to ORM (Object relational mapping). In this exercise, you will have to create the same table as in **Exercise 00** The requirements of this exercise are simple:

- Create a bundle for this exercise, named ex01;
- Create the corresponding entity to the required structure;
- Create route which will call the generation of table;
- **No SQL is allowed in this exercise**.
- The structure of the table should correspond to the given one.

The result of this exercise has to be a page which contains a button/link to create the database table and a success/error message regarding the result of the creation.

The database table structure is the following:

- id - integer - primary key
- username - string - unique
- name - string
- email - string - unique
- enable - boolean

- birthdate - datetime
- address - long text

The call to create table should not fail in case the table is already created.

Hint: entity generation and database table creation can be made using doctrine ORM commands. The commands can be found by typing **php appconsole doctrine** inside the terminal. Also, these terminal commands can be called inside Symfony controllers.

Chapter VI

Exercise 02

	Exercise 02
Exercise 02: Insert and read with SQL	
Turn-in directory : $ex02/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Using your previously gained knowledge, you should now be able to make other operations from the CRUD packet. Lets start with Insert and Read.

This exercise will help you gain knowledge about mapping information from website forms to SQL queries. The requirements of this exercise are:

- Create a bundle for this exercise, named ex02;
- Create a separate table for this exercise, using the same structure as in the previous exercises;
- **This exercise will be solved using only SQL, no ORM allowed;**
- Create a Symfony form and a show form action in the controller;
- Create a controller endpoint for handling the information submitted from the form;
- Make sure the unique fields will not throw exceptions on insert;
- Create a webpage containing an HTML table in which to show the database content.

Neither the call to create table, nor the insert of data should fail in case the table is already created or the data is already present in the table.

Chapter VII

Exercise 03

	Exercise 03
Exercise 03: Insert and read with ORM	
Turn-in directory : $ex03/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Moving back to ORM, the operations presented in the last exercise can be implemented using only Symfony Doctrine ORM specific commands. For Insert and Read with ORM,the list of requirements are the following:

- Create a bundle for this exercise, named ex03;
- Create a new entity with its own independent table, using the same structure as in the previous exercises;
- **This exercise will be solved using only Doctrine ORM;**
- Create a Symfony form and map it to the entity created earlier;
- Create specific endpoints in the controller for Show form, handle inserted data and view the existing data from the database.
- Make sure the unique fields will not throw exceptions on insert;
- Create a webpage containing an HTML table in which to show the database content.
- The form should contain validation
- The ORM commands should not be handled in the controller.

Neither the call to create table, nor the insert of data should fail in case the table is already created or the data is already present in the table.

The final version of this exercise should allow the user to insert information into the database using the provided form and see the list of all entities from the database.

Hint: for doctrine commands, you can see the list by running **php appconsole doctrine** in the command line.

Chapter VIII

Exercise 04

	Exercise 04
Exercise 04: SQL - delete using request with condition	
Turn-in directory : *ex*04/	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Continuing with the CRUD specific operations, we get now to the point of Delete. For this exercise, using only SQL and PHP code, you will have to write a Symfony specific bundle which will handle this kind of operation. The delete should be made using an endpoint of the application which accepts a conditioned request, something like */delete/?id=* or */delete/{id}* , where the **id** would represent an unique identifier of the object to be deleted.

The requirements of this exercise are the following:

- Create a new bundle called ex 04;
- Create a separate table for this exercise;
- Create a controller method which has a route with parameter for deleting information;
- Create SQL code for deleting information from the database
- Check if an item exists in the database;
- Design an HTML page in which to show the existing information from the database in an HTML table and add the button for deleting each specific row;
- Use only SQL code for deleting information from the database, **no ORM allowed**
- Add a success/error message on the designed webpage for representing the obtained result of the operation

Chapter IX

Exercise 05

	Exercise 05
Exercise 05: ORM - delete using request with condition	
Turn-in directory : $ex05/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

For achieving CRUD in Symfony, ORM provides also methods for deleting entities from the database. This exercise will help you find the way of using delete option in Doctrine ORM. Following the requirements of the last exercise, you will have to implement the **Delete** option using only ORM. More precisely, the following requirements must be met:

- Create a new bundle called ex05;
- Create a new entity specialized for this exercise. You can take the structure from exercise 00 as an example;
- Create a controller method which has a route with a parameter for deleting information;
- Check if an item exists in the database;
- Design an HTML page in which to show the existing information from the database in an HTML table and add the button for deleting each specific row;
- Use only ORM commands for deleting information from the database;
- Add a success/error message on the designed webpage for representing the obtained result of the operation

Chapter X

Exercise 06

	Exercise 06
Exercise 06: SQL: update database information	
Turn-in directory : $ex06/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

The last element from the CRUD package is **Update**. For this exercise, using only SQL and PHP code, you will build, in Syfmony framework, a bundle that is capable of updating an existing entity. To be more precise, each entity is mapped to at least a table from the database, so, in conclusion, you will have to update the tables from the database. In order to solve this exercise, you will have to respect the following conditions:

- Create a bundle named ex06;
- Create a separate table in the database for this exercise. You can use as guideline the structure from ex00;
- Create a webpage for displaying the entities that exist in the database using a table;
- For each element from the webpage table, add a button for "Update", which will take the user to the edit page;
- Create a form which will be autocompleted with the entity details and will allow the user to change each of the fields;
- Design a controller with required methods for listing and updating entities from the database;
- Use only SQL code for handling database updates, no ORM allowed;
- No validation or conditions are required (unique fields, correct inserted values, etc);

- Display a success/error message on the list page, after updating an entity.

In the final version, this exercise will provide the option to update existing information from the database. By designing it, you will have completed the CRUD operations using SQL.

Chapter XI

Exercise 07

	Exercise 07
Exercise 07: ORM: update database information	
Turn-in directory : $ex07/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Since each of the CRUD operations were implemented using both SQL and ORM, this exercise will reflect the usage of Update in ORM. For solving this exercise, you will have to design a new Symfony bundle capable of displaying and updating information from the database. Due to the usage of ORM, the database is reflected in PHP code using entities. In conclusion, the next requirements must be satisfied in order to solve this exercise:

- Create a new bundle named ex07;
- Create a new Doctrine entity (you can use the structure from the previous exercises) mapped to a new table in the database;
- Design an HTML webpage table in which to show the existing information from the database;
- Develop a Symfony Form mapped to the entity created earlier;
- Create controller methods capable of handling the list, display and update of entities;
- No validations are required for this exercise;
- Only ORM commands must be used for modifying information in the database;
- Return a success/error message based on operation result;

Chapter XII

Exercise 08

	Exercise 08
Exercise 08: SQL - update table structure, create relations	
Turn-in directory : *ex*08/	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Congratulations! You have finished the CRUD operations using both SQL and ORM. Now you should be able to implement the CRUD operations using both options, without hesitation. But, due to the fact that project specification could change, a developer should also know how to change the database structure and, most important, how to create relations between database tables.

As you may already know, in relational databases, we have 3 types of relationships between tables:

- One-to-one, example: one wheel has one tyre and viceversa;
- One-to-many, example: one car has one owner, one owner could own several cars;
- Many-to-many, example: one order can have several products, one product could belong to several orders.

For this exercise, you have to design a Symfony bundle which is able to create a table with given structure, alter that table and add a new column and also create two relationships for two other tables (to be created) using foreign keys.

More detailed, the next requirements have to be met, in order to solve this exercise:

- Create a new bundle called ex08;
- Design the SQL code required for creating and altering a table and making relationships between tables;
- Create the initial table (named *persons*) using the structure from ex00, but don't

add the address field. The table creation should be made using a link in the homepage and a success/error message should be displayed;

- Create a new controller method which will add a new column to the designed table (ex: column *marital_status:ENUM(single, married, widower)*)
- Create a new controller method for creating two new tables: **bank_accounts** and **addresses**. You have the freedom of designing the structure of these two tables;
- Create a one-to-one relationship between **bank_accounts** and **person** table and a one-to-many relation;
- Use **only SQL** for creating tables and relations;
- For each operation (create table, alter tables, create relations between tables) return a success/error message;

Chapter XIII

Exercise 09

	Exercise 09
Exercise 09: ORM - update table structure, create relations	
Turn-in directory : *ex*09/	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

For achieving the scope of this day, one important exercise should be done using both SQL and ORM techniques. It is now the time to update an entity and create realtionships between entities using ORM. As stated in the previous exercise, there are 3 types of relations between entities (tables). One difference is that, in Symfony, these relations are made between entities and reflected in the database by automatically mapping foreign keys between talbes (or creating connection tables for the many-to-many relation).

In order to achieve the final result, you have to accomplish the following requirements:

- Create a new bundle named ex09;
- Create an initial entity using the structure defined in the previous exercises (Person entity structure);
- Add a new field (as required in the previous exercise) on the entity and update the database table;
- Create two new entities (BankAccount and Address) and make bidirectional relationships between those and Person entity;
- Use **only ORM** commands and instructions for handling the database structure;
- Create controller methods for handling the creation, update and relationships management.

Note: since Doctrine doesn't provide ways to alter the table and have a trace on it, you will have to use Doctrine migrations to add a new column on the table.

Chapter XIV

Exercise 10

	Exercise 10
Exercise 10: SQL & ORM - Insert into database from file	
Turn-in directory : $ex10/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Nowdays, the provider of the information to be stored can be of several types: human input, data received using calls to external web services, auto-generated data or even information read from files. In this exercise, the main focus is to store information obtained by reading files.

For this exercise, you will have to propose a solution that will use both SQL and ORM **at the same time**. The requirements for this exercise are:

- Create a new Symfony bundle named ex10;
- Create a new database table for SQL and a new entity (with related table) for ORM;
- Design a controller method which will read a file content and insert it into both SQL table and ORM related table;
- The controller method could be called from a link in a webpage. The information then should be visible into a table on a HTML page;
- Ensure that the information is correctly stored and no fields are lost during the reading - writing operation.

This exercise has to be solved using both SQL and ORM.

Hint: ensure that the file you want to read information from has the correct permission rights.

Chapter XV

Exercise 11

	Exercise 11
Exercise 11: SQL - Request with join, sort and condition	
Turn-in directory : $ex11/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Since these days, website are more complex than simple HTML pages, when using a database for providing information, sorting and conditional requests are very common. Also, due to database design principles, join between tables have to be made, in order for the user to have the required information. As an example, think of an e-commerce website: for a client to search for a specific product based on its type, category and color, the website backend has to make a join between the cateogry and product tables, but also filter the result based on product color and type. In addition, if the client wants to order the list based on price, in ascending order, a sorting of the information read from the database has to be made. Usually, all these joins, conditions and sorting are better to be made directly in the query language, since it would be faster and more efficient than storing all the information into variables and applying different filters or sorting after.

For this exercise, the requirements are the following:

- Create a new bundle named ex11;
- For this exercise, you can use the database tables designed in exercise 09, due to already existing relations (you can use one-to-one relations);
- Design an HTML webpage with a table in which to present the information stored in the database table;
- In the HTML table, add a form with filtering and sorting (eg: filter by date, sort by name);
- Create required methods in the controller for handling the listing and generating the result after applying the created request with filtering and sorting;

- Make sure that the information displayed in the HTML table is fetched from more than one table;
- Use **only SQL** for making calls to the database;
- For the request of information from the database, make sure to use **join, condition, sort**. If one is missing, no points will be given for this exercise!
- Add validations for parameters of the request.

Chapter XVI

Exercise 12

	Exercise 12
Exercise 12: ORM - Request with join, sort and condition	
Turn-in directory : $ex12/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

As you may already thought, since the last exercise represents an important part of a web application, the requirements of this exercise are based on the same idea. The big difference is that, this time, you will have to use **only ORM** methods for achieveing the required result.

In adition to the requirements of the last exercise, the following ideas have to be respected:

- Create a new bundle called ex12
- You can use the entities created in exercise 09, with one-to-one relationships between them;
- Make sure to create special repository queries for fetching desired information;
- Add validations for the parameters of the request.

Chapter XVII

Exercise 13

	Exercise 13
Exercise 13: ORM - Complete CRUD operations	
Turn-in directory : $ex13/$	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

Good job, you're almost at the end of this day. But, before you finish it, you will do two cool exercises. This exercise will reflect all the knowledge you gained by solving the previous exercises. The main idea is to implement all the operations represented in CRUD, using only ORM commands.

The requirements that have to be met for this exercise are the following:

- Create a new bundle called ex13
- Create a new entity of Employee type, which will respect the following structure:
  - id - integer
  - firstname - string
  - lastname - string
  - email - string
  - birthdate - datetime
  - active - boolean
  - employed_since - datetime
  - employed_until - datetime
  - hours - ENUM (8,6,4)

  - salary - integer
  - position - ENUM(manager, account_manager, qa_manager, dev_manager, ceo, coo, backend_dev, frontend_dev, qa_tester)

- Create a one-to-many relationship between Employee entity and itself, representing the manager of an employee (each employee should have a manager - a person responsible);
- Implement a read endpoint, edit endpoint, delete endpoint and creation endpoint for the Employee entity;
- The form for creating/editing an entity should contain different types of validations, based on the field type (ensure that some of them are required, unique, email has correct format, etc);
- Success/error messages should be created for each operation;
- The application should not crash when requests are not correct (eg: deleting inexistent information, trying to insert duplicates, etc).

Chapter XVIII

Exercise 14

	Exercise 14
Exercise 14: SQL - Functional injections	
Turn-in directory : *ex*14/	
Files to turn in : `Files and folder from your application`	
Allowed functions : `All methods`	
Notes : `n/a`	

As promised, this is the last exercise of this day, but it will be one of the most interesting. Maybe you have heard of the term *SQL injection*, but you never tried to create one. This exercise will prove the voulnerability of SQL and HTML forms without validations inside a website. An SQL injection represents a technique used to attack websites and run different SQL queries (get specific field values, change information, drop information, drop database) over the database, using unprotected form fields.

For this exercise, the following requirements have to be met:

- Create a new bundle named ex14;
- Create a controller method which will insert a new table into the database, if the table does not exist already;
- The table status should be reflected in the webpage using a specific message (”Table exists” vs ”Table does not exist”)
- Create an HTML form in a webpage, mapped to a controller action, to insert information into the database;
- Add a functional SQL injection using Javascript, on the form submit event;
- The result of the SQL injection should be visible in the homepage or, if altering of information is done, the result should be visible in a listing of the database information.