# Exercise: Web Security and Database Drivers

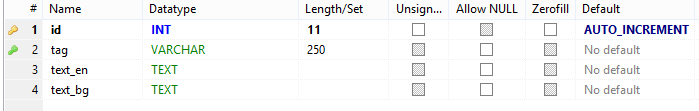
This document defines the in class assignments from the ["Web Development Basics" Course @ Software University](https://softuni.bg/courses/web-development-basics/).

## Localization system with database

Dynamic data stored directly in memory could be one of the fastest ways for searching and traversing, yet very ineffective when it comes about maintenance. Big nested arrays and other data structures might become unmaintainable and hardly portable. As our localization system grows and we will hire interpreters for this job we’d need an interface where these guys to enter their translations

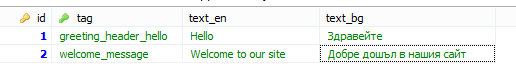
### Task 1. Move the nested array in a database table

Let’s imagine our nested array as a MySQL table. We need to find for certain tag, the value of certain language. What we need is a column for the tag, and column for each language. As we will be filtering for tag, we may also need our tag column to be key indexed for faster searching. We may use variable character for the tag as we don’t expect it to be literally big text and for the language columns a textual type. Our table (let’s call it `translations`) should look something like:



### Task 2. Populate the table with some data

Let’s move some of the hardcoded translations to the table

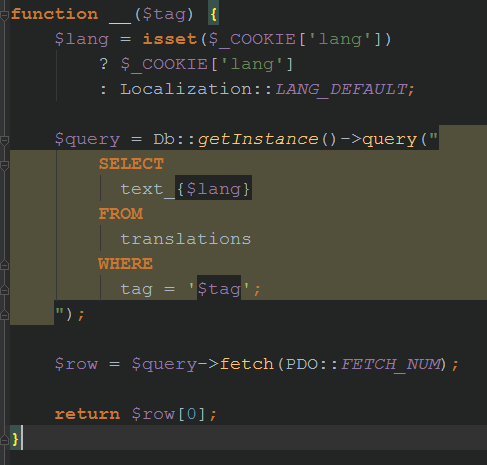


### Task 3. Start using the database instead the array

* Connect to the database via PDO

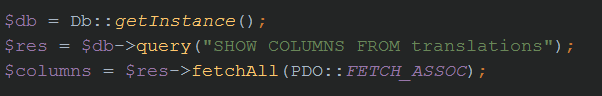


* Make a singleton class returning the PDO instance to ensure you will connect to the db only once and the connection can be accessed globally.
  + Call the class **Db**
  + Use the convention for singletons and call the instantiating method **getInstance()**
* Reusing the logic for the default/cookie language we need to use the database in our translation function

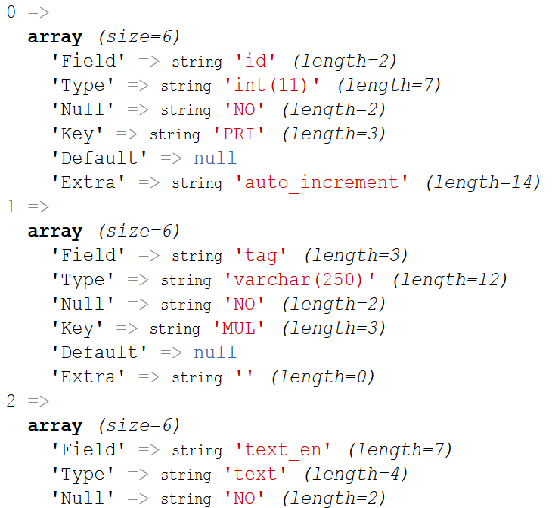


### Task 4. Dynamically take the possible languages

We are still using the constants with the languages and the default language constants. It’s not that bad, but if we need to add new language we need to alter the db table as well as add new constant. We can extract the languages from the table columns. All columns starting with text\_ are the language columns.



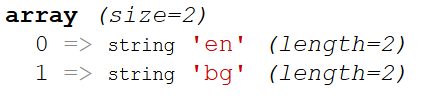
Our **$columns** variable now looks like



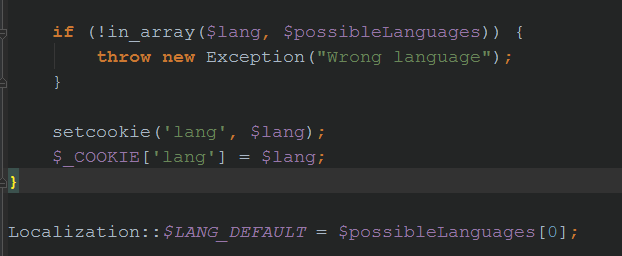
Now you need to extract all entries which “Field” key starts with “text\_” and convert the result into new array called $possibleLanguages. Hint: Use **array\_map(), array\_filter(), array\_values(), strpos() and str\_replace()** functions (similar to **Where** and **Select** extension methods, **IndexOf() and Replace()** string methods in C# and **Values** property for Dictionaries).



Finally the $**possibleLanguages** variable should look like



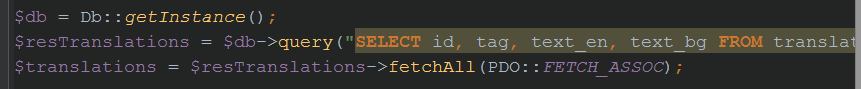
The check afterwards will be using in\_array(). And will be using the first index as a default language. Thus we need a static field instead of constant in order to assign it the value.



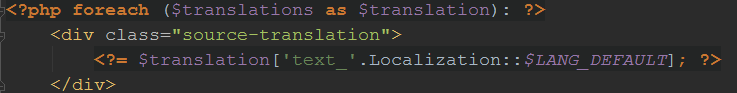
### Task 5. Administration for the translators

Our newly hired interpreters should have an interface where they will translate from. Let’s create a page that lists all English translations and provides textareas for translations in Bulgarian. If there is already a translation in Bulgaria, show it in the textarea. Then a Save button will commit all translations to the database.

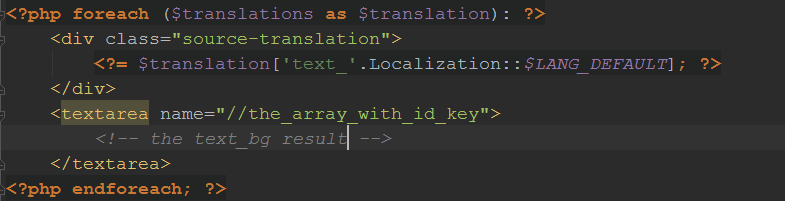
* Let’s extract all data from translations



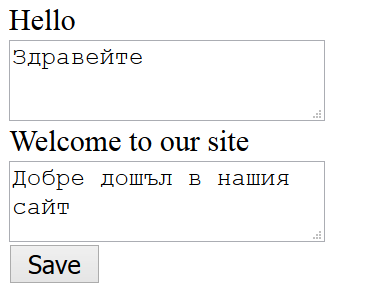
* Now in the presentation (let’s say admin.php for example) go through the resulted associative array and print the default language (source). We need to traverse the returned array and print each text\_en. We will produce the EN suffix from the $LANGUAGE\_DEFAULT static field



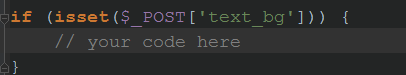
* Then in a textarea print the text\_bg content. It will be very useful if you name each textarea as an array called text\_bg, but putting each result ID as an array key e.g. text\_bg[6] for the 6-th row, so you will lately UPDATE the lang\_bg WHERE id = 6.



* Wrap all the form with FORM that that is doing a POST request. After the last textarea put a button for submitting the form. The it should result in something like:



* In the backend handle the POST request and for each textarea make a query that updates the lang\_bg for the given ID



### Task 6. Protect from SQL Injection

The problem here is that we are concatenating strings to build queries so far. If you have followed the pattern to build the query to UPDATE text\_bg (e.g. “UPDATE translations SET text\_bg = “$input” WHERE id = $inputId”), if the user enters for $input “**1', text\_en = 'bla**” one will update your text\_en column, because will inject SQL code in the value.

In order to prevent this, use fully the features of the PDO extension. One of the best ways to achieve this is to use parameterized queries / prepared statements with binding parameters.

Take a look at PDO’s official documentation: <http://php.net/manual/en/pdo.prepared-statements.php>