##### Lab 7 – Large DBs, Pagination, AJAX

In academia, we often create a database schema and populate the tables with a few dozen fake records. But we should have much more fake data that resembles real data when we are testing our apps. One way of populating our test database is with [Faker](https://packagist.org/packages/fzaninotto/faker).

[Faker](https://github.com/fzaninotto/Faker) is an open-source php library that lets us specify the different data that we require for many different scenarios. It is available through [Composer](https://getcomposer.org/); Composer is the open source defacto dependency manager used in PHP. Composer allows you to install packages that you require; Composer uses a json file to list all your project dependencies, and it will find and install them. [Packagist](https://packagist.org/) is the package repository used by Composer.

Once we have seeded the database, we will create an application that displays the results using pagination.

**Part 0 - Solving intermittent 502 Bad Gateway problems**

Do this part only if you find that you have intermittent 502 Bad Gateway responses from nginx:

Go to /etc/php/7.0/fpm/pool.d/ , edit www.conf (use sudo nano) and edit the line listen = /run/php/php7.0-fpm.sock to /**var**/run/php/php7.0-fpm.sock

Exit and save the file, then run sudo service php7.0-fpm restart

**Part 1 – Install Faker using Composer**

Composer is installed in the Vagrant php.box.

Create a new lab7 folder for this project

In Vagrant:

* cd to the lab7 folder
* write the following command:

composer require fzaninotto/faker 1.6

This will install version 1.6 (latest stable release of Faker) in a vendor folder. It will also update a composer.json file so that Composer can track this project dependency.

**Part 2- Create a table in MySQL**

Create a customer table with the following columns and obvious data types in either Vagrant or waldo2:

id (auto-incrementing)

firstname

lastname

birthdate (use MySQL DateTime)

salary

civicnumber

street

city

state

postcode (note: this follows the American standard, so leave about 12 char in your db)

**Part 3 – Populate the database**

Create a class to represent a customer, and a DAO class.

You will now write a command-line application that uses Faker to seed the database. Faker documentation can be found at <https://github.com/fzaninotto/Faker>

In order to use Faker, you will need to load the required files, get a Faker instance, and generate data by accessing properties. Code to get an instance:

require "vendor/autoload.php"; //assumes your code is lab7 folder

//your other requires or autoload function for Customer and DAO

$faker = Faker\Factory::create(); //factory method to get an instance

Note the backslash syntax: this is a qualified class name (Factory) with its namespace (Faker).

Once you have an instance, you generate data by accessing Faker “properties” and methods. These properties are called “[formatters](https://github.com/fzaninotto/Faker#formatters)” and are actually generated random results. Some of the formatters available:

* firstName //generates a randomly chosen gender American name
* firstName($gender) //generates an American name, $gender can be ‘male’ or ’female’
* lastName
* numberBetween($min, $max) //random int generator
* randomFloat($nbMaxDecimals, $min, $max) //random float generator with decimal digits
* dateTimeBetween(‘-65 years’, ‘now’) //DateTime between
* buildingNumber
* streetName
* city
* stateAbbr
* postcode

Using Faker:

$faker->firstName will return a randomly generated American first name, like ‘Maynard’.

$faker->streetName will return a random street name like ‘Keegan Trail’

Note: the random numbers created by Faker are not any different from random numbers you would create using PHP functions. You’ll notice that the salaries don’t really reflect reality: you would ideally like a function to return a random salary that fit a normal distribution. This [post](http://www.eboodevelopment.com/php-random-number-generator-with-normal-distribution-bell-curve/) has a nice function to create normalized random numbers. If you use it, be sure to give credit to the author.

Note: when working with datetime Faker formatters, the result is returned as a DateTime php object. You cannot use a DateTime object directy in an SQL commend (SQL parameters can be strings, int, or null). So you will have to convert the DateTime object into string representation with the date function. The syntax:

$stringDate = $datetime->format('Y-m-d H:i:s');

In a loop, generate 1000 Customer objects and save them to the database (create 1000 records if you are using the Homestead database, I’m worried about space limits on Waldo2).

Note: it may take a few seconds to write the 1000 records into your database. Make sure you write that as a command-line application (not a web application), since we don’t want to have a maximum execution time limit. If you are using the local Homestead database, you can experiment with even more records!

**Part 4: Web application**

Create a php script index.php that gives the user a form where they can choose to find customers and display their full names based on:

- living within a certain postcode (allow incomplete postcode entry)

- salary range

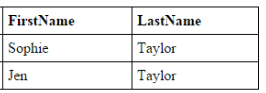
- age range (this is tricky since we store birthdates: you need to find the age. MySQL [documentation](http://dev.mysql.com/doc/refman/5.7/en/date-calculations.html) has an answer using TIMESTAMPDIFF function.

Your results page should use pagination (20 results / page).

**Part 5: Ajax application**

Now create a php script name.php. You will use Javascript to make an AJAX GET request to ajax.php on every key up, and display the records that are returned. You will practice with a simple query: the user enters the first letters in a lastname, the PHP response provides all employees whose lastname starts with the given letters.





Make sure that your form is not submitting - simply don’t use a FORM tag just a text input field.

Your ajax PHP script should send back an array of Customers using JSON encoding. Here are some (untested) code snippets:

//build the 2-D array in a DAO class

$response=null; //to make sure it is empty, unsetting does something similar, but you can’t json\_encode it

//create a prepared statement

//Add the following line so the fetch mode is into a Customer object

$stmt->setFetchMode(PDO::FETCH\_CLASS|PDO::FETCH\_PROPS\_LATE, ‘Customer’);

//execute the statement

$response = $stmt->fetchAll();

Recall: you will be able to fetch into an object only if the class’s properties have the same names as the columns. If not, use aliases in your SQL query or fetch as an associated array

Recall: you can only json encode public attributes. In order not to lose all encapsulation, make the Customer class implement JsonSerializable, and add a jsonSerialize() function that returns an associative array with the attributes that you want to serialize:

function jsonSerialize() {

return [

'firstname' => $this->firstname,

'lastname' => $this->lastname

];

}

Recall that the SQL LIKE query with wildcards is tricky in a prepared statement because of the wildcard needing to be appended. You must use bindValue if you are not passing by reference (e.g., passing a calculated value or a literal, nor a direct variable):

$stmt = $dbh->prepare("SELECT \* FROM `customer` WHERE `lastname` LIKE :lastname");

$stmt->bindValue(':lastname', $lastname . '%');

// in a presentation layer class:

header('Content-Type: application/json');

//echo the json

echo json\_encode($response);

The JSON data is an array of objects, but it can be null if the result set is empty. In JS, you will need to iterate through the json data and do something:

//convert resultText into a JSON object

//if the jsonArr is not null

for(var i = 0; i < jsonArr.length; i++) {

//do something with jsonArr[i] object

}

//handle the situation if jsonArr is null

Once you get the resultText in JS, you will have to display the data in a table. In pure JS, you would use something like innerHTML. With JQuery, you would use .html(htmlString) function to set the HTML contents in your div/table. Note that you’ll have an easier time clearing and rebuilding the table if you use a <tbody> tag.

WARNING: Debugging php when it is sending JSON or XML is tricky since any error message gets written into the JSON/XML and is not shown on the browser. The browser developer tools are your best friends! Learn how to put breakpoint in your JS code (this is very helpful in debugging JS, much better than alerts!) and how to look at the network http messages going back and forth.