# DB Practical Work 2:
# The Post model

December 20, 2017

**Abstract**

The following subject aims at implementing the data handling for posts in a twitter-like web-application. Implementations are to be done in the file `model_student/post.php`

## Contents

# 1 Requirement

To fulfill this work, you will need the following elements:

- A working environment with db connection to both app and test databases (see `0setup.pdf`).
- At least the tables modeling user related tables and **posts**, **mentions** and **likes** in `sql/schemas.sql`.

# 2 Work to do

You have to fill out the functions defined in the file `model_student/post.php`

These functions are used in the application to get access to the database. Therefore, these functions must observe some rules about both input data (the formats of the parameters of the functions) and output data (the returned values).

In the functions, you can access to the PDO object by using the following instruction:

```
$db = \Db::dbc();
```

Then, you can perform queries using `$db` like a PDO object:

```
$db = \Db::dbc();
$result = $db->query('SELECT * FROM post');
```

When you completed all the functions, you can check them by using the available unit tests.

# 3 The post entity

## 3.1 Presentation

The Post entity represents a post and its properties:

- the date and time when it was published
- the message itself

The post must be linked to the user who has written it and, if it was to respond to another post, must be linked to it.

## 3.2 Creating and getting a post

### 3.2.1 `get($id)`

`get` is the functions that returns the post ids. It must return null if no post with the given id were found.

Here is an example of what `get` should return:

```
(object) array(
"id" => 1337,
"text" => "Text #test",
"date" => new \DateTime('2011-01-01T15:03:01'),
"author" => \Model\User\get(2)
)
```

### 3.2.2 `create($author_id, $text, $response_to=null)`

`create` saves a post object. It does a rather important number of things (see comments in source file).

For now, let's ignore both hashtags and mentions parsing.

`create` must return the id of the post. It is appreciated if a transaction is used.

## 3.3 Handling responses

### 3.3.1 `get_with_joins($id)`

`get_with_joins` is a function which includes (within the post object) joined elements which are : a) the post object to which the post responds to, b) the objects for every users that liked the post and c) the hashtag it includes.

For now, let's ignore the last two elements : likes and hashtag. (the attributes can be set to an empty array, with `array()`). `responds_to` must in the contrary return a post object.

Here is an example of what should return `get_with_joins`:

```
(object) array(
"id" => 1337,
"text" => "Ima writing a post !",
"date" => new \DateTime('2011-01-01T15:03:01'),
"author" => \Model\User\get(2),
"likes" => array(),
"hashtags" => array(),
"responds_to" => null
)
```

#### 3.3.2 `get_responses($id)`

`get_responses` returns an array of post objects which respond to the occuring post.

### 3.4 Deleting a post

#### 3.4.1 `destroy($id)`

`destroy` takes care of deleting post objects.

**Important!**
The deletion behaviour must be allowing deleting posts which are liked or which mention users. **To see and modify the deletion policy in phpMyAdmin, you must go in the table view → Structure → Relation view.**

### 3.5 Listing and searching for posts

#### 3.5.1 `list_all($date_sorted=false)`

`list_all` returns the list of every posts sorted according to the passed parameter:

- `$date_sorted="ASC"` means sorting along publication time (oldest to most recent).
- `$date_sorted="DESC"` means sorting along publication time (most recent to oldest).
- `$date_sorted=false` means no sorting.

#### 3.5.2 `list_user_posts($id, $date_sorted="DESC")`

`list_user_posts` returns the list of posts from a given user.
`$date_sorted` handling is exactly the same as in `list_all`

#### 3.5.3 `search($string)`

`search` returns a list of posts which texts include a given string.

## 4 Mentioning users

### 4.1 Presentation

Mentioning a user is useful to draw his/her attention. By including `@the_username` to the message, a user is mentioned. If the message is "Hello @alice and @bob",

two users are mentioned (if they exist): the one with username alice and the one with username bob.

### 4.2 `mention_user($pid, $uid)`

`mention_user` creates a mention association.

### 4.3 `get_mentioned($pid)`

`get_mentioned` returns the list of the mentioned users in a given post.

### 4.4 `create($author_id, $text, $response_to=null)`

Now that you coded the functions that takes care about mentions, you must use them in `create` so that mentions are added when creating a post.

`create` must handle making the mention associations while creating a post. To help, the function `extract_mentions` (in `model/post.php`) is provided.

## 5 Liking posts

### 5.1 Presentation

A user can like a post. This association has to be modeled accordingly.

### 5.2 `like($uid, $pid)`

`like` creates a like association.

### 5.3 `unlike($uid, $pid)`

`unlike` removes a like association.

### 5.4 `get_likes($pid)`

`get_likes` returns the user objects who liked the post.

### 5.5 `get_with_joins($id)`

It is time to make `get_with_joins` handle likes. The `likes` attribute must be a list of user objects.