Separating code from DB functionality

Asked 16 years, 1 month ago Modified 16 years, 1 month ago Viewed 573 times

Part of PHP Collective

8

I'm developing an object-oriented PHP website right now and am trying to determine the best way to abstract database functionality from the rest of the system. Right now, I've got a DB class that manages all the connections and queries that the system uses (it's pretty much an interface to MDB2). However, when using this system, I've realized that I've got a lot of SQL query strings showing up everywhere in my code. For instance, in my User class, I've got something like this:

```
function checkLogin($email,$password,$remember=false){
    $password = $this->__encrypt($password);
    $query = "SELECT uid FROM Users WHERE email=? AND pw=?";

    $result = $this->db->q($query,array($email,$password));

    if(sizeof($result) == 1){
       $row = $result->fetchRow(MDB2_FETCHMODE_ASSOC);
       $uid = $row['uid'];
    }else{
       return false;
    }

    /* Rest of the login script */
}
```

What I would like to do is find out the best technique for reducing the amount of inline SQL. I understand that one way to do this would be to write functions within User for each of the queries that User makes use of (something like the following), but that could lead to quite a few functions.

```
function checkLogin($email,$password,$remember=false){
    $password = $this->__encrypt($password);
    $uid = $this->do_verify_login_query($email,$password);

    /* Rest of the login script */
}

function do_verify_login_query($email,$encpw){
    $query = "SELECT uid FROM Users WHERE email=? AND pw=?";
    $result = $this->$db->q($query,array($email,$encpw));

    if(sizeof($result) == 1){
       $row = $result->fetchRow(MDB2_FETCHMODE_ASSOC);
       return $row['uid'];
    }else{
       return false;
```

```
    }
}
```

So...my question. What is the best technique for managing the large amount of queries that a typical database application would use? Would the way I described be the proper way of handling this situation? Or what about registering a list of queries within the DB class and associating with each a unique ID (such as USER_CHECKLOGIN) that is passed into the DB's query function? This method could also help with security, as it would limit the queries that could be run to only those that are registered in this list, but it's one more thing to remember when writing all the class functions. Thoughts?

PHP php database

Share
Improve this question
Follow

edited Nov 13, 2008 at 16:54

Mat Nadrofsky
8,274 ● 8 ● 51 ● 73

asked Nov 13, 2008 at 3:33

cmptrgeekken
8,082 ● 3 ● 31 ● 35

5 Answers

Sorted by: Highest score (default)

7

Having the SQL pulled out into separate functions is a decent start. Some other things you can do:

- Create separate classes for database access code. This will help make sure you don't have SQL functions scattered around in all of your PHP files.
- Load the SQL from external files. This completely separates your SQL code and your PHP code, making both more maintainable.
- Use stored procedures when you can. This removes the SQL from your PHP code altogether, and helps improve your database security by reducing the risk that external SQL will get executed.

Share Improve this answer Follow

answered Nov 13, 2008 at 3:47

Bill the Lizard
405k ● 211 ● 572 ● 889

Wouldn't stored procedures require inline SQL statements to execute them, since they only exist in the database? Or am I not understanding what you mean when you say to use stored procedures? – cmptrgeekken Nov 13, 2008 at 3:52

MySQL actually just added stored procedures, and I've read the syntax isn't as flexible as it is in SQL Server. IMO, stored procedures are really useless unless you need to write an extensive or complicated query. – Kevin Nov 13, 2008 at 4:03

You call stored procedures with a function call, passing the name and the query parameters. macronimous.com/resources/… – Bill the Lizard Nov 13, 2008 at 4:09

Nice response Bill. I like all of the listed approaches. – Mat Nadrofsky Nov 13, 2008 at 16:51

4

You might want to look into implementing the ActiveRecord Pattern. Using a design pattern such as this provides some consistency in how you work with data from your tables. There can be some downsides to these sorts of approaches, mainly performance for certain types of queries but it can be worked around.

Share Improve this answer Follow

answered Nov 13, 2008 at 3:49

D-Rock

2,676 ● 1 ● 21 ● 27

I suppose one advantage to using this sort of pattern would be that you can use the same structure for representing queries, and not have to worry about database-specific syntax in generating your SQL, thus adding yet another layer of abstraction. – cmptrgeekken Nov 13, 2008 at 3:57

4

Another option can be the use of an ORM, for PHP the most powerful are:

- Propel
- Doctrine

Both allow you to access your database using a set of objects, providing a simple API for storing and querying data, both have their own query language, that is converted internally to the targeted DBMS native SQL, this will ease migrating applications from one RDBMS to another with simple configuration changes. I also like the fact that you can encapsulate datamodel logic to add validation for example, only by extending your model classes.

Share Improve this answer Follow

answered Nov 13, 2008 at 4:05

Christian C. Salvadó

826k ● 184 ● 927 ● 843

3

Since you say you're doing this as OO PHP, then why do you have SQL scattered through all the methods in the first place? More common models would be:

1. Use an ORM and let that handle the database.
2. Give your classes one or more 'load' methods which use a single query to pull all of an object's data into memory and a 'save' method which uses a single query to update everything in the database. All the other methods only need to do in-memory manipulation and the database interactions are confined to the load/save methods.

The first option will generally be more robust, but the second may run faster and will probably feel more familiar compared to the way you're used to doing things, if either of those are concerns.

For your login example, the way I would do it, then, would be to simply load the user by email address, call `$user->check_password($entered_password)`, and throw an exception/return false/whatever if `check_password` fails. Neither `check_password` nor any of the login handling code need to concern themselves with the database, or even with knowing that a database is where the user gets loaded from.

Share Improve this answer Follow

answered Nov 13, 2008 at 16:47

Dave Sherohman

46.1k ● 14 ● 66 ● 103

0

Another option is to think of the queries as data and store them in the database. For instance, you can create one table that stores the query with a name and another table that stores the parameters for that query. Then create a function in PHP that takes the name of the query and an array of params and executes the query, returning any results. You could also attach other metadata to the queries to restrict access to certain users, apply post-functions to the results, etc.

Share Improve this answer Follow

answered Nov 13, 2008 at 17:57

Unlabeled Meat

254 ● 1 ● 4