

Proste PHP Class-Based Zapytania

przez [Brant Steen](#) 20 maja Trudność: Intermediate Długość: Średnie Języki: język angielski ▼
2010

PHP Web Development

Choć jest to zazwyczaj zaleca się użyć jakiegoś ramowej lub CMS, czasem projekt jest na tyle mała, tak że te opcje będą przytłaczać rozwój. Jednak nawet w mniejszych projektach, oddzielające elementy prezentacyjne z zaplecza zapytań nie powinny być ignorowane. Ten poradnik przeprowadzi Cię przez proces tworzenia podstawowy silnik odpytywanie klasy oparte na PHP i MySQL.

Krok 1. Konfiguracja projektu

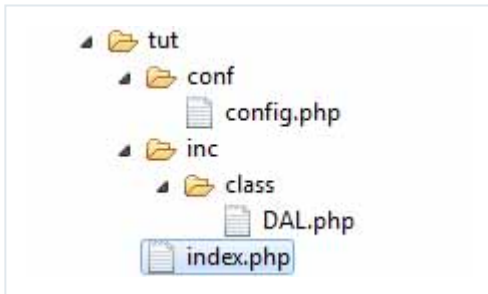
Pierwszą rzeczą, którą będą chcieli zrobić, to dokonać pewnych określonych plików i katalogów. Oto jak lubię konfiguracji moje projekty. Można, oczywiście, nie krępuj się zmienić nazwy i struktury do własnych upodobań. Tylko pamiętaj, aby zmienić wymagać w późniejszym czasie, gdy dobrze.

Dokonaj katalogów

Musimy nowy katalog do przechowywania wszystkiego. W tym przypadku nazwałem go *tut* . Wewnątrz tego, kładę moje pliki konfiguracyjne w katalogu o nazwie *conf* . Wtedy będę dokonać *inc* katalog (skrót) i umieścić to katalog „klasa” wewnątrz tego.

Dodaj pliki

Następnie wewnątrz / *conf* , dołożymy *config.php* . Wewnątrz / *INC* / *klasy* dołożymy *DAL.php* . Wreszcie, w katalogu głównym, dołożymy *index.php* .



DAL oznacza "Data Access Layer" lub "Data Access Link".

W wielowarstwowej architekturze, jest zasadniczo używany do przetłumaczenia wyniki kwerendy bazy danych do obiektów i vice-versa.

Krok 2. Konfiguracja bazy danych

Musimy uczynić bazę danych i wypełnić go z niektórych danych. Na potrzeby tego poradnika, będzie to po prostu być w bazie dwóch stół z jednym relacji jeden-do-wielu. To tylko tak możemy pokazać nasze Spanning silnik zapytań co najmniej jeden związek.

Tworzenie tabel

Tak więc, w bazie danych o nazwie „tut”, stwórzmy tabelę o nazwie *marki* i stół zwane *modele* . *Marki* stół będzie miał pola „id” i „name” oraz *modele* stół będzie miał pola „id”, „marka” oraz „name” .

✓ Table `tut`.`makes` has been created.




```
CREATE TABLE `tut`.`makes` (  
  `id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,  
  `name` VARCHAR( 100 ) NOT NULL  
) ENGINE = INNODB;
```



✓ Table `tut`.`models` has been created.

```
CREATE TABLE `tut`.`models` (  
  `id` INT( 11 ) NOT NULL AUTO_INCREMENT ,  
  `make` INT( 11 ) NOT NULL ,  
  `name` VARCHAR( 300 ) NOT NULL ,  
  PRIMARY KEY ( `id` )  
) ENGINE = INNODB;
```

Dodaj niektóre dane

Teraz możemy tylko dodać kilka marek (takich jak Ford, Chevy, etc.) jak dane w tabeli i sprawia, że w niektórych modelach, że ci producenci są odpowiedzialni.

			id	name
<input type="checkbox"/>			1	Ford
<input type="checkbox"/>			2	Chevy

			id	make	name
<input type="checkbox"/>			1	1	F-150
<input type="checkbox"/>			2	1	Explorer
<input type="checkbox"/>			3	1	Taurus
<input type="checkbox"/>			4	2	Tahoe
<input type="checkbox"/>			5	2	Silverado

Ten poradnik zakłada, masz jakąś wiedzę na temat baz danych i języka SQL, więc nie będę wchodzić w szczegóły na temat relacji klucza obcego / Setup.

Krok 3. Połączenie z bazą danych

Zazwyczaj nie lubię pracować z surowców stałych w PHP. Ja zazwyczaj określić grono rzeczy następnie dokonać niektóre funkcje, aby podłączyć do tych stałych. W tym przykładzie, po prostu zachować rzeczy proste i używać stałych.

Definiowanie zmiennych połączeń

W naszym / *conf* / *config.php* pliku, niech nasze zmienne konfiguracji połączenia z bazą danych. Podczas gdy jesteśmy w nim, niech rzucają include do naszego DAL.php skryptu.

```
01 <?php
02
03 // Include DAL
04 require_once(dirname(dirname(__FILE__)) . '/inc/class/DAL.php');
05
06 // Database
07 define ( 'DB_HOST', 'localhost' );
08 define ( 'DB_USER', 'root' );
09 define ( 'DB_PASSWORD', 'password1' );
10 define ( 'DB_DB', 'tut' );
11
12 ?>
```

Taka konfiguracja zakłada używasz MySQL na jego domyślnego portu.

Tworzenie funkcji połączenia

Teraz wewnątrz / *inc* / *klasa* / *DAL.php* , dołożymy funkcję będziemy używać do łączenia się w naszej bazie.

Połączenie, jak również wszystkie nadchodzące zapytania, będzie żył wewnątrz klasy o nazwie *DAL* . Owijanie wszystko zaangażowania bazy danych wewnątrz jednej klasy pozwala nam manipulować naszymi zapytań później bez konieczności

dotknąć skrypty biznesowych lub warstwa prezentacji. Ponadto, zapewnia pewien stopień udawanym przestrzeni nazw.

W przypadku tej klasy dodamy konstruktora, mimo że nie trzeba nic robić.

```
01 <?php
02
03 class DAL {
04
05     public function __construct(){}
06
07     private function dbconnect() {
08         $conn = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD)
09         or die ("<br/>Could not connect to MySQL server");
10
11         mysql_select_db(DB_DB,$conn)
12         or die ("<br/>Could not select the indicated database");
13
14         return $conn;
15     }
16
17 }
18
19 ?>
```

Należy zauważyć, że zakres *dbconnect* metody jest prywatny. To dlatego, że nie ma potrzeby łączenia się z bazą danych spoza naszego DAL. Zamiast tego, mamy metody zapytań publicznych, które będą dzwonić do *dbconnect* od wewnątrz DAL. Nieco mylące? Nie martw się, czytaj dalej.

Krok 4. Tworzenie podstawowe narzędzia Query

Do abstrakcyjnych nasze pytania tak, że możemy ponownie użyć krótkich fragmentów kodu, będziemy potrzebować dwóch rzeczy. Po pierwsze, będziemy potrzebować jakiegoś klasy „generic wyników zapytania”. Po drugie, musimy metoda rodzajowa odpytwanie wewnątrz naszego DAL.

Tworzenie klasy rodzajowe wynik zapytania

Celem tego wszystkiego jest w stanie konwertować zapytań SQL na przedmioty i zminimalizować wykorzystanie brzydkiego czas (*\$ wiersz = mysql_fetch_array (\$ wynik)*) pętli. Przedmioty są znacznie łatwiejsze w użyciu i pozwalają nam wykorzystać właściwości zamiast kluczy tablicy.

Krótko mówiąc, chcemy mieć klasę, która będzie tworzyć nazwy własności do danych latać i sklepów związanych z tymi właściwościami.

Dołożymy tej klasy wewnątrz naszego */ INC / klasa / DAL.php* skryptu. Ponieważ jest to nowa klasa, to będzie poza klasą DAL.

```
01 class DALQueryResult {
02
03     private $_results = array();
04
05     public function __construct(){}
06
07     public function __set($var,$val){
08         $this->_results[$var] = $val;
09     }
10
11     public function __get($var){
12         if (isset($this->_results[$var])){
13             return $this->_results[$var];
14         }
15         else{
16             return null;
17         }
18     }
19 }
```

```
}
```

Tworzenie kwerendy metoda rodzajowa

Teraz wewnątrz naszego *DAL* klasy, musimy uczynić metoda rodzajowa odpytanie że zamieni *SELECT* zapytania do *DALQueryResult* obiektów.

Zasadniczo, chcemy włączyć każdy zwrócony nazwę pola na własność *DALQueryResult* obiektu.

```
01 private function query($sql){
02
03     $this->dbconnect();
04
05     $res = mysql_query($sql);
06
07     if ($res){
08         if (strpos($sql,'SELECT') === false){
09             return true;
10         }
11     }
12     else{
13         if (strpos($sql,'SELECT') === false){
14             return false;
15         }
16         else{
17             return null;
18         }
19     }
20
21     $results = array();
22
23     while ($row = mysql_fetch_array($res)){
24
25         $result = new DALQueryResult();
26
27         foreach ($row as $k=>$v){
28             $result->$k = $v;
29         }
```

```

30     }
31
32     $results[] = $result;
33 }
34 return $results;
}

```

Oto prywatna funkcja, która akceptuje kwerendy SQL. To łączy się z bazą danych i uruchamia kwerendę. Następnie sprawdza, czy są jakieś wyniki. Jeśli nie ma żadnych wyników, zwraca wartość null na *SELECT* zapytania, fałszywe innych zapytaniach. Jeśli zapytanie było udane i zapytań nie było *SELECT* zapytania, zwróci prawdę. Jeśli było to *SELECT*, a następnie przetwarza się wyniki w tablicy obiektów DALQueryResult. To naśladuje wyniki, które można by normalnie z `mysql_query`.

Krok 5. Napisz konkretne zapytanie

Teraz jesteśmy gotowi, aby faktycznie napisać zapytanie SQL. Zapytania DAL powinny być bardzo specyficzne zarówno w nazwie i celu. Zrobimy taki, który wyszukuje wszystkie modele danej marki.

To będzie nasza pierwsza metoda publicznego.

```

1 public function get_models_by_make_name($name){
2     $sql = "SELECT models.id as id, models.name as name, makes.name as make FROM models JOIN makes ON models.make_id = makes.id WHERE makes.name = $name";
3     return $this->query($sql);
4 }

```

Tu są tylko pisanie kwerendy i zwracając wynik w postaci obiektów DALQueryResult. Nasz rodzajowy *kwerendy* metoda dba o iterations i podejmowania decyzji.

Ukończony DAL

W tym momencie nasza *DAL.php* skrypt jest gotowy. Powinien on wyglądać następująco.

```
01 <?php
02
03 class DALQueryResult {
04
05     private $_results = array();
06
07     public function __construct(){}
08
09     public function __set($var,$val){
10         $this->_results[$var] = $val;
11     }
12
13     public function __get($var){
14         if (isset($this->_results[$var])){
15             return $this->_results[$var];
16         }
17         else{
18             return null;
19         }
20     }
21 }
22
23 class DAL {
24
25     public function __construct(){}
26
27     public function get_models_by_make_name($name){
28         $sql = "SELECT models.id as id, models.name as name, makes.name as ma
29         return $this->query($sql);
30     }
31
32     private function dbconnect() {
33         $conn = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD)
34         or die ("<br/>Could not connect to MySQL server");
35
36         mysql_select_db(DB_DB,$conn)
```

```
37         or die ("<br/>Could not select the indicated database");
38
39     return $conn;
40 }
41
42 private function query($sql){
43
44     $this->dbconnect();
45
46     $res = mysql_query($sql);
47
48     if ($res){
49         if (strpos($sql,'SELECT') === false){
50             return true;
51         }
52     }
53     else{
54         if (strpos($sql,'SELECT') === false){
55             return false;
56         }
57         else{
58             return null;
59         }
60     }
61
62     $results = array();
63
64     while ($row = mysql_fetch_array($res)){
65
66         $result = new DALQueryResult();
67
68         foreach ($row as $k=>$v){
69             $result->$k = $v;
70         }
71
72         $results[] = $result;
73     }
74     return $results;
75 }
76 }
77
78 ?>
```

Reklama

Krok 6. Użyj DAL

A teraz w końcu udać się do naszego */index.php* skryptu i wyświetlić wyniki za pomocą DAL. Wszystko, co musimy zrobić, to to nasz */conf/config.php* pliku instancję Dal, i coś zrobić z danymi. Oto przykład.

```
01 <?php
02 // include configuration
03 require_once(dirname(__FILE__) . '/conf/config.php');
04
05 // instantiate a new DAL
06 $dal = new DAL();
07
08 // array of makes to check
```

```
09 $makes = array('Ford', 'Chevy', 'Honda');
10
11 // cycle through the makes
12 foreach ($makes as $make){
13     $results = $dal->get_models_by_make_name($make);
14     echo "<h1>Models by $make</h1>";
15
16     // check if there were any results
17     if ($results){
18         echo "<ul>";
19
20         // cycle through results
21         foreach ($results as $model){
22             echo "<li>$model->make $model->name (Database ID: $model->id)</li>";
23         }
24         echo "</ul>";
25     }
26     else{
27         // Display a message concerning lack of data
28         echo "<p>Sorry, we have no information regarding that manufacturer.</p>";
29     }
30 }
31 ?>
```

Jak widać, mamy teraz wyniki możemy nazwać nazw pól jako właściwości obiektu PHP.

Etap 7. Biorąc miejsca jeden krok dalej

Często zdarza się, że będzie użyteczny, aby przekształcić ogólny *DALQueryResult* do bardziej konkretnego obiektu. W tym przypadku, można napisać obiektów biznesowych, które akceptują *DALQueryResult* jako parametr konstruktora. Następnie, po prostu używać, aby zbudować nowy obiekt.

Oto przykład

```
01 <?php
02
03 class CarModel{
04     private $_id;
05     private $_make;
06     private $_name;
07
08     public function __construct(DALQueryResult $result){
09         $this->_id = $result->id;
10         $this->_make = $result->make;
11         $this->_name = $result->name;
12     }
13     public function __get($var){
14         switch ($var){
15             case 'id':
16                 return $this->_id;
17                 break;
18             case 'make':
19                 return $this->_make;
20                 break;
21             case 'name':
22                 return $this->_name;
23                 break;
24             default:
25                 return null;
26                 break;
27         }
28     }
29     public function __toString(){
30         return $this->_name;
31     }
32 }
33
34 ?>
```

Następnie wystarczy napisać kwerendę, aby zwracać tablicę tych obiektów zamiast tablicy generycznych *DALQueryResult* obiektów.

Pamiętaj, że zawsze wymienić zapytania bardzo specyficznie.

```
01 public function get_models_by_make_name_as_CarModel($name){
02     // Reuse existing query
03     $results = $this->get_models_by_make_name($sql);
04
05     // check for results
06     if (!$results){
07         return $results;
08     }
09     else{
10         // array to hold CarModel objects
11         $object_results = array();
12         // cycle through and convert to CarModel objects
13         foreach ($results as $result){
14             object_results[] = new CarModel($result);
15         }
16         // return array of CarModel objects
17         return object_results;
18     }
19 }
```

Budowy poszczególnych obiektów mogą stać się bardzo przydatne, gdy potrzebne są obliczenia wyodrębnić istotne dane z pól.

Mam nadzieję, że wszyscy jesteśmy zadowoleni z samouczka! Powodzenia.