# Database abstraction layers in PHP. PDO versus DBAL

~ Gonzalo Ayuso

45 Votes

I normally use PDO in my PHP projects. I like it because it's a PHP extension easy to use and shares the same interface between all databases. Normally I use PostgreSQL but if I change to mySql or Oracle I don't need to use different functions to handle the database connections.

PHP has a great project called Doctrine2. Doctrine2 is a ORM and it uses its own database abstraction layer called DBAL. In fact DBAL isn't a pure database abstraction layer. It's built over PDO. It's a set of PHP classes we can use that gives us features not available with 'pure' PDO. If we use Doctrine2 we're using DBAL behind the scene, but we don't need to use Doctrine2 to use DBAL. We can use DBAL as a database abstraction layer without any ORM. Obiously this extra PHP layer over our PDO extension needs to pay a fee. I will have a look to this fee in this post. I will take one of my old post about PDO and I will do the same with DBAL to see the performance differences. Let's start:

**The PDO version**:

```
1 error_reporting(-1);
2 $time = microtime(TRUE);
3 $mem = memory_get_usage();
4
5 $dbh = new PDO('pgsql:dbname=mydb;host=localhost', 'gonzalo',
6 'password');
7 $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
8
9 $dbh->beginTransaction();
10
11 $smtp = $dbh->prepare('INSERT INTO test.tbl1 (id, field1) values
12 (:ID, :FIELD1)');
13
14 for ($i=0; $i<1000; $i++) {
15     $smtp->execute(array('ID' => $i, 'FIELD1' => "field {$i}"));
16 }
17
18 $dbh->commit();
19
20 $stmt = $dbh->prepare('SELECT * FROM test.tbl1 limit 10000');
21 $stmt->execute();
22
23 $i=0;
24 while ($row = $stmt->fetch()) {
25     $i++;
```

```
}
echo '<h1>PDO</h1>';
echo "<strong>{$i} </strong>";

print_r(array('memory' => (memory_get_usage() - $mem) / (1024 *
1024), 'seconds' => microtime(TRUE) - $time));

$dbh->beginTransaction();
$smtp = $dbh->prepare('delete from test.tbl1');
$smtp->execute();
$dbh->commit();
```

**The DBAL version**:

```
error_reporting(-1);
$time = microtime(TRUE);
$mem = memory_get_usage();

use Doctrine\DBAL\DriverManager;

$connectionParams = array(
    'dbname' => 'mydb',
    'user'  => 'gonzalo',
    'password' => 'password',
    'host'  => 'localhost',
    'driver' => 'pdo_pgsql',
    );

$dbh = DriverManager::getConnection($connectionParams);

$dbh->beginTransaction();

$smtp = $dbh->prepare('INSERT INTO test.tbl1 (id, field1) values
(:ID, :FIELD1)');

for ($i=0; $i<1000; $i++) {
    $smtp->execute(array('ID' => $i, 'FIELD1' => "field {$i}"));
}

$dbh->commit();

$stmt = $dbh->prepare('SELECT * FROM test.tbl1 limit 10000');
$stmt->execute();

$i=0;
while ($row = $stmt->fetch()) {
    $i++;
}
echo '<h1>DBAL</h1>';

```

```
    echo "<strong>{$i} </strong>";

    print_r(array('memory' => (memory_get_usage() - $mem) / (1024 *
    1024), 'seconds' => microtime(TRUE) - $time));
```

As we can see DBAL is slower than pure PDO (obiously). Anyway the most of the extra time of DBAL is the time we need to include php classes (remember PDO is a PHP extension and we dont need to include any file). If we take times excluding the include time, the memory usage is almost the same and the execution time a little slower.

Autoload for DBAL version:

```
spl_autoload_register(function ($class) {
        $class = str_replace('\\', '/', $class) . '.php';
        require_once($class);
    }
);
```

or hardcoded includes for this example

```
require_once('Doctrine/DBAL/Driver.php');
require_once('Doctrine/DBAL/Driver/Connection.php');
require_once('Doctrine/DBAL/Platforms/AbstractPlatform.php');
require_once('Doctrine/DBAL/Driver/Statement.php');

require_once('Doctrine/DBAL/DriverManager.php');
require_once('Doctrine/DBAL/Configuration.php');
require_once('Doctrine/Common/EventManager.php');
require_once('Doctrine/DBAL/Driver/PDOPgSql/Driver.php');
require_once('Doctrine/DBAL/Driver.php');
require_once('Doctrine/DBAL/Connection.php');
require_once('Doctrine/DBAL/Driver/Connection.php');
require_once('Doctrine/DBAL/Query/Expression/
ExpressionBuilder.php');
require_once('Doctrine/DBAL/Platforms/PostgreSqlPlatform.php');
require_once('Doctrine/DBAL/Platforms/AbstractPlatform.php');
require_once('Doctrine/DBAL/Driver/PDOConnection.php');
require_once('Doctrine/DBAL/Driver/PDOStatement.php');
require_once('Doctrine/DBAL/Driver/Statement.php');
require_once('Doctrine/DBAL/Events.php');
require_once('Doctrine/DBAL/Statement.php');
```

# Outcomes of the tests:

With pure PDO:

- memory: 0.0044288635253906
- seconds: 0.24748301506042

With DBAL and autoload:

- memory: 0.97610473632812
- seconds: 0.29042816162109

With DBAL and hardcoded requires:

- memory: 0.97521591186523
- seconds: 0.31192088127136

With DBAL bypassing the include part:

- memory: 0.0099525451660156
- seconds: 0.30333304405212

The fee we paid for using DBAL gives us some extra features. OK we don't need DBAL to get those features. If we code a bit we can get them (remember DBAL is nothing but a PHP extra layer). But DBAL has a great interface a well documented. Now I'm going to list a few extra features from DBAL very interesting, at least for me:

# Transactional mode

I really like it. It allows us to create scripts like that:

```
 $dbh->transactional(function($conn) {
1    $smtp = $conn->prepare('INSERT INTO wf.tbl1 (id, field1)
2values (:ID, :FIELD1)');
3
4    for ($i=0; $i<1000; $i++) {
5        $smtp->execute(array('ID' => $i, 'FIELD1' => "field
6{$i}"));
7    }
 });
```

A simple closure will make the code more concise and it will commit/rollback our transaction for us. In fact I borrowed this function in my PDO projects to use this interface. I love Open source.

Snippet from DBAL library:

```
1 /**
2  * Executes a function in a transaction.
3  *
4  * The function gets passed this Connection instance as an
5 (optional) parameter.
6  *
7  * If an exception occurs during execution of the function or
8 transaction commit,
9  * the transaction is rolled back and the exception re-thrown.
10 *
11 * @param Closure $func The function to execute transactionally.
12 */
```

```
  public function transactional(Closure $func)
{
    $this->beginTransaction();
    try {
        $func($this);
        $this->commit();
    } catch (Exception $e) {
        $this->rollback();
        throw $e;
    }
  }
```

## Types conversion

Really useful, at least for when I work with dates:

```
$date = new \DateTime("2011-03-05 14:00:21");
$stmt = $conn->prepare("SELECT * FROM articles WHERE publish_date
> ?");
$stmt->bindValue(1, $date, "datetime");
$stmt->execute();
```

## List of Parameters Conversion

It's a cool feature too available in DBAL since Doctrine 2.1

```
$dbh->executeQuery('SELECT * FROM wf.tbl1 WHERE id IN (?)',
    array(array(1, 2, 3, 4, 5, 6)),
    array(\Doctrine\DBAL\Connection::PARAM_INT_ARRAY));
```

Bind parameters with IN clause with PDO is a bit ugly. We need to create a series of bind parameters depending on our list to map them within the SQL. It's possible but DBAL interface is smarter.

## Transaction Nesting

Another cool feature:

```
 $dbh->beginTransaction();
 try {
     $dbh->beginTransaction();
     try {
         $smtp = $dbh->prepare('INSERT INTO wf.tbl1 (id, field1)
 values (:ID, :FIELD1)');

         for ($i=0; $i<1000; $i++) {
             $smtp->execute(array('ID' => $i, 'FIELD1' => "field
{$i}"));
```

```
        }

        } catch (Exception $e) {
            $dbh->rollback(); //transaction marked for rollback
only
            throw $e;
        }
    $smtp = $dbh->prepare('INSERT INTO wf.tbl1 (id, field1)
values (:ID, :FIELD1)');

    for ($i=0; $i<1000; $i++) {
        $smtp->execute(array('ID' => $i, 'FIELD1' => "field
{$i}"));
    }

    $dbh->commit(); // real transaction committed
} catch (Exception $e) {
    $dbh->rollback(); // transaction rollback
    throw $e;
}
```

This piece of code with PDO will throw the following error:
*There is already an active transaction*
but it works with DBAL. If we need to do this kind of things with PDO we need to use savepoints and things like that. DBAL does the ugly part for us.

https://gonzalo123.com/2011/07/11/database-abstraction-layers-in-php-pdo-versus-dbal/