## Настройка DbUnit для функциональных тестов

#### **Зачем DbUnit?**

Для тестирования взаимодействия Вашего приложения с базой данных Вам надо дополнительно проделать следующие шаги:

* Учесть структуру БД и таблиц
* Произвести вставку первоначальных данных при необходимости
* Провести проверку состояния БД после выполнения каких-либо операций бизнес-логики
* Очистить базу и повторить заново для каждого теста (иначе последующие тесты будут зависеть от предыдущих)

Если писать такое вручную с использованием SQL-запросов, то довольно скоро начинаешь проклинать юнит-тестирование в принципе. Кроме того, это не соответствует одному из главных принципов юнит-тестирования — тесты должны быть минимально сложны и максимально читабельны.

#### **По порядку**

Итак, как же правильно должен проходить тест взаимодействия с базой?

1. **Очистка базы**. При первом запуске мы не знаем в каком состоянии находится БД, поэтому мы обязаны «начать с чистого листа»;
2. **Вставка начальных данных (фикстур)**. Обычно приложению нужны какие-либо начальные данные, которые оно извлекает из базы для последующей обработки. именно их и надо вставить в только что очищенную базу;
3. **Собственно выполнение тестов и проверка результатов**. Без комментариев.

#### **PHPUnit Database Test Case**

Если в случае обычного тест-кейса в PHPUnit Вы должны просто унаследовать класс PHPUnit\_Framework\_TestCase, то в случае с тестированием БД все несколько сложнее:

**require\_once** "PHPUnit/Extensions/Database/TestCase.php";

**class** **MyTest** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getConnection**()

{

$pdo = **new** PDO('mysql:host=localhost;dbname=testdb', 'root', '');

**return** $this->createDefaultDBConnection($pdo, 'testdb');

// или используя настройки из XML файла

// $pdo = **new** PDO($GLOBALS[‘DB\_DSN’], $GLOBALS[‘DB\_USER’], $GLOBALS[‘DB\_PASSW’] );

// **return** $this->createDefaultDBConnection($pdo, ($GLOBALS[‘DB\_DBNAME’]);

}

**public** **function** **getDataSet**()

{

**return** $this->createFlatXMLDataSet(dirname(**\_\_FILE\_\_**).'/\_files/guestbook-init.xml');

}

}

Вы должны реализовать два абстрактных метода — getConnection() и getDataSet(). Первый необходим для установления соединения с базой, второй для заполнения базы таблицами и заполнения собственно таблиц.  
Важно заметить, что getConnection() должен использовать PDO для подключения к базе, но Ваше приложение не обязано использовать PDO для запросов к базе. Соединение, устанавливаемое методом getConnection() используется лишь для подготовки БД к тестам и ассертов.  
Начальное содержимое базы абстрагируется с помощью интерфейсов PHPUnit\_Extensions\_Database\_DataSet\_IDataSet и PHPUnit\_Extensions\_Database\_DataSet\_IDataTable. Метод getDataSet() вызывается методом setUp() для получения и вставки фикстур. В примере мы использовали фабричный метод createFlatXMLDataSet() для получения датасета из XML-представления.

#### **DataTables & DataSets**

Итак, что же это такое? Это ключевые понятия рассматриваемого расширения. DataTable и DataSet — это абстракция для таблиц и записей в реальной БД. Довольно несложный механизм позволяет скрыть реальную БД за объектами, которые в свою очередь могут быть реализованы различными способами.  
Такая абстракция необходима для сравнения ожидаемого контента базы и реального. Ожидаемый контент может быть представлен в различных видах благодаря абстракции — например, XML, CSV, массивы PHP. Интерфейсы DataTable и DataSet позволяют проводить сравнение данных из источника ожидаемых с реальными из БД.  
Также DataSet и DataTable используются для задания начального состояния базы данных перед выполнением теста.   
Ниже рассмотрим различные варианты датасетов.

#### **Flat XML DataSet**

Это наиболее простой вид датасета. Каждый элемент внутри корневого представляет собой одну запись из БД. Имя элемента должно соответствовать имени таблицы, а атрибуты и значения — поля и значения полей соответственно, например:

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<post post\_id="1" title="My First Post" date\_created="2008-12-01 12:30:29" contents="This is my first post" rating="5" />

<post post\_id="2" title="My Second Post" date\_created="2008-12-04 15:35:25" contents="This is my second post" />

</dataset>

Это эквивалентно таблице post в БД с 2 записями

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **post\_id** | **title** | **date\_created** | **contents** | **rating** |
| 1 | My First Post | 2008-12-01 12:30:29 | This is my first post | 5 |
| 2 | My Second Post | 2008-12-04 15:35:25 | This is my second post | *NULL* |

В общем-то, довольно просто и понятно.  
Пустая таблица эквивалента пустому элементу, например, пустая таблица current\_visitors:

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<current\_visitors />

</dataset>

NULL-значения для записи представляются как отсутствие соответствующего атрибута (см. пример с blog, поле rating), однако, тут следует учесть один момент. Для flat XML DataSet структуру таблицы определяет первый элемент, т.е. если в первом элементе нет каких-либо атрибутов, а в последующих элементах для той же таблицы они есть, то эти атрибуты будут проигнорированы. Например, если в примере с таблице blog из первого элемента убрать атрибут date\_created с его значением, то во втором элементе этот атрибут не будет учитываться и в таблице не будет поля date\_created.  
Использование с помощью метода createFlatXmlDataSet():

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getDataSet**()

{

**return** $this->createFlatXmlDataSet('myFlatXmlFixture.xml');

}

}

#### **XML DataSet**

Данный вариант представления в XML лишен недостатков Flat XML, но и несколько сложнее:

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<table name="post">

<column>post\_id</column>

<column>title</column>

<column>date\_created</column>

<column>contents</column>

<column>rating</column>

<row>

<value>1</value>

<value>My First Post</value>

<value>2008-12-01 12:30:29</value>

<value>This is my first post</value>

<value>5</value>

</row>

<row>

<value>2</value>

<value>My Second Post</value>

<value>2008-12-04 15:35:25</value>

<value>This is my second post</value>

<null />

</row>

</table>

</dataset>

Таблица полностью представляется элементом <table>, в который вложены <column> для определения полей таблицы и <row> для представления записей. В свою очередь, в <row> могут быть вложены <value> для представления значащих полей и <null /> для NULL-значений.  
  
Пустая таблица представляется как таблица без <row> элементов:

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<table name="current\_visitors">

<column>current\_visitors\_id</column>

<column>ip</column>

</table>

</dataset>

Использование с помощью метода createXMLDataSet():

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getDataSet**()

{

**return** $this->createXMLDataSet('myFlatXmlFixture.xml');

}

}

#### **CSV Data Set**

Представление таблицы в формате CSV (Comma Separated Values — простейший формат для хранения таблиц). Все довольно понятно:  
post\_id,title,date\_created,contents,rating  
1,My First Post,2008-12-01 12:30:29,This is my first post,5  
2,My Second Post,2008-12-04 15:35:25,This is my second post,  
  
Использование несколько сложнее чем в случае XML:

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getDataSet**()

{

$dataSet = **new** PHPUnit\_Extensions\_Database\_DataSet\_CsvDataSet();

$dataSet->addTable('post', 'post.csv');

**return** $dataSet;

}

}

Для использования нам необходимо создать объект класса PHPUnit\_Extensions\_Database\_DataSet\_CsvDataSet. Конструктор принимает три аргумента, которые определяют формат CSV:

**public** **function** **\_\_construct**($delimiter = ',', $enclosure = '"', $escape = '"'){}

После этого добавляем таблицы в датасет методом addTable — один файл — одна таблица.

#### **PHP массивы**

На данный момент нет стандартной реализации датасетов с помощью массивов, но ее нетрудно реализовать ;)  
  
Предположим, что нам надо хранить датасеты в таком формате:

**array**(

'post' => **array**(

**array**(

'post\_id' => 1,

'title' => 'My First Post',

'date\_created' => '2008-12-01 12:30:29',

'contents' => 'This is my first post',

'rating' => 5

),

**array**(

'post\_id' => 2,

'title' => 'My Second Post',

'date\_created' => '2008-12-04 15:35:25',

'contents' => 'This is my second post',

'rating' => **null**

),

),

)

Реализация:

**require\_once** 'PHPUnit/Extensions/Database/DataSet/AbstractDataSet.php';

**require\_once** 'PHPUnit/Extensions/Database/DataSet/DefaultTableIterator.php';

**require\_once** 'PHPUnit/Extensions/Database/DataSet/DefaultTable.php';

**require\_once** 'PHPUnit/Extensions/Database/DataSet/DefaultTableMetaData.php';

**class** **ArrayDataSet** **extends** **PHPUnit\_Extensions\_Database\_DataSet\_AbstractDataSet**

{

**protected** $tables = **array**();

**public** **function** **\_\_construct**(array $data)

{

**foreach** ($data **as** $tableName => $rows) {

$columns = **array**();

**if** (**isset**($rows[0])) {

$columns = array\_keys($rows[0]);

}

$metaData = **new** PHPUnit\_Extensions\_Database\_DataSet\_DefaultTableMetaData($tableName, $columns);

$table = **new** PHPUnit\_Extensions\_Database\_DataSet\_DefaultTable($metaData);

**foreach** ($rows **as** $row) {

$table->addRow($row);

}

$this->tables[$tableName] = $table;

}

}

**protected** **function** **createIterator**($reverse = FALSE)

{

**return** **new** PHPUnit\_Extensions\_Database\_DataSet\_DefaultTableIterator($this->tables, $reverse);

}

}

Немного комментариев — для своего датасета мы наследуем абстрактный датасет (который наследуют flat XML, XML, CSV и другие). В конструктор мы передаем оговоренный ранее массив. Как и в случае с flat XML струткура таблицы определяется первой записью, но в данном случае это не критично, потому что мы имеем возможность явно указать NULL-значения. Структура, кстати, определяется с помощью создания объекта PHPUnit\_Extensions\_Database\_DataSet\_DefaultTableMetaData. После этого создаем собственно таблицу, передав в нее структуру и добавляем записи в таблицу с помощью метода addRow(). Так же нам необходимо реализовать абстрактный метод createIterator, но в этом нет ничего сложного :)  
  
Использование:

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getDataSet**()

{

**return** **new** ArrayDataSet(**array**(

'post' => **array**(

**array**(

'post\_id' => 1,

'title' => 'My First Post',

'date\_created' => '2008-12-01 12:30:29',

'contents' => 'This is my first post',

'rating' => 5

),

**array**(

'post\_id' => 2,

'title' => 'My Second Post',

'date\_created' => '2008-12-04 15:35:25',

'contents' => 'This is my second post',

'rating' => **null**

),

),

));

}

}

#### **Query/Database Dataset**

Для ассертов нам потребуются не только ожидаемые датасеты, но и реальные из базы данных. В этом нам поможет QueryDataSet

$ds = **new** PHPUnit\_Extensions\_Database\_DataSet\_QueryDataSet($this->getConnection());

$ds->addTable('post');

или с явным использованием запроса:

$ds->addTable('post', 'SELECT \* FROM post ORDER BY post\_id');

Также можно использовать существующее подключение для автоматического получения датасетов из существующих таблиц с помощью метода PHPUnit\_Extensions\_Database\_DB\_DefaultDatabaseConnection::createDataSet() (это объект, создаваемого в getConnection()). Если не передавать параметр в createDataSet(), то будет создан датасет из всех существующих таблиц. Если передать в качестве параметра массив с именами таблиц в базе, то датасет будет создан только из этих таблиц.

#### **Replacement DataSet**

Я уже упоминал о проблеме NULL-значений для flat XML датасета (для CSV проблема та же — невозможно явно задать NULL-значение в фикстуре). Это можно решить с помощью специального декоратора — ReplacementDataSet:

**require\_once** 'PHPUnit/Extensions/Database/DataSet/ReplacementDataSet.php';

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getDataSet**()

{

$ds = $this->createFlatXmlDataSet('myFlatXmlFixture.xml');

$rds = **new** PHPUnit\_Extensions\_Database\_DataSet\_ReplacementDataSet($ds);

$rds->addFullReplacement('##NULL##', **null**);

**return** $rds;

}

}

Теперь мы можем использовать ##NULL## в XML для обозначения NULL-значения:

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<post post\_id="1" title="My First Post" date\_created="2008-12-01 12:30:29" contents="This is my first post" rating="5" />

<post post\_id="2" title="My Second Post" date\_created="2008-12-04 15:35:25" contents="This is my second post" rating="##NULL##" />

</dataset>

#### **Фильтрация датасетов**

В случае больших датасетов можно применить фильтрацию с помощью DataSetFilter:

**require\_once** 'PHPUnit/Extensions/Database/DataSet/ReplacementDataSet.php';

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **testIncludeFilteredPost**()

{

$dataSet = $this->getConnection()->createDataSet();

$filterDataSet = **new** PHPUnit\_Extensions\_Database\_DataSet\_DataSetFilter($dataSet);

$filterDataSet->addIncludeTables(**array**('post'));

$filterDataSet->setIncludeColumnsForTable('post', **array**('post\_id', 'title'));

*// ..*

}

**public** **function** **testExcludeFilteredPost**()

{

$dataSet = $this->getConnection()->createDataSet();

$filterDataSet = **new** PHPUnit\_Extensions\_Database\_DataSet\_DataSetFilter($dataSet);

$filterDataSet->addExcludeTables(**array**('foo', 'bar', 'baz'));

$filterDataSet->setExcludeColumnsForTable('post', **array**('date\_created', 'rating'));

*// ..*

}

}

В первом случае мы оставили в датасете только таблицу post и содержимое ее записей только для полей post\_id и title. Во втором — мы исключили из датасета таблицы 'foo', 'bar' и 'baz', а из записей таблицы post убрали значения для полей 'date\_created' и 'rating'.

#### **Композиция датасетов**

Мы можем соединять несколько датасетов в один. В случае, если датасеты имеют одинаковые таблицы, то записи в них будут добавлены, например:  
dataset-1.xml

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<post post\_id="1" title="My First Post" date\_created="2008-12-01 12:30:29" contents="This is my first post" rating="5" />

</dataset>

dataset-2.xml

**<?xml version="1.0" encoding="UTF-8" ?>**

<dataset>

<post post\_id="2" title="My Second Post" date\_created="2008-12-04 15:35:25" contents="This is my second post" />

</dataset>

Аггрегируем их:

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **getDataSet**()

{

$ds1 = $this->createFlatXmlDataSet('dataset-1.xml');

$ds2 = $this->createFlatXmlDataSet('dataset-2.xml');

$compositeDs = **new** PHPUnit\_Extensions\_Database\_DataSet\_CompositeDataSet();

$compositeDs->addDataSet($ds1);

$compositeDs->addDataSet($ds2);

**return** $compositeDs;

}

}

#### **Asserts**

Часто бывает необходимо проверить количество записей в таблице. Это можно сделать с помощью обычного assertEquals:

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **testAddEntry**()

{

$this->assertEquals(2, $this->getConnection()->getRowCount('post'));

$blog = **new** Blog();

$blog->addPost("My third post.", "This is my third post.");

$this->assertEquals(3, $this->getConnection()->getRowCount('post'));

}

}

Метод getRowCount() возвращает количество записей в указанной таблице.  
  
Для сравнения таблиц используется метод assertTablesEqual():

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **testTables**()

{

$queryTable = $this->getConnection()->createQueryTable('post', 'SELECT \* FROM post');

$expectedTable = $this->createFlatXmlDataSet("myFlatXmlFixture.xml")->getTable("post");

$this->assertTablesEqual($expectedTable, $queryTable);

}

}

Необходимо помнить о том, что тест может сфейлится при проверке даты — если у вас в фикстуре есть установленная дата, а в базу записывается текущее время, то Вы получите фейл, если эти даты не совпадают. Поэтому зачастую из ожидаемого результат убирают даты и, соответственно, изменяют получение реального датасета:

$queryTable = $this->getConnection()->createQueryTable('post', 'SELECT post\_id, title, date\_created, contents, rating FROM post');

И наконец, можно сравнивать непосредственно датасеты с помощью assertDataSetsEqual():

**class** **MyTestCase** **extends** **PHPUnit\_Extensions\_Database\_TestCase**

{

**public** **function** **testTables**()

{

$dataSet = $this->getConnection()->createDataSet(**array**('post'));

$expectedDataSet = $this->createFlatXmlDataSet("myFlatXmlFixture.xml");

$this->assertDataSetsEqual($expectedDataSet, $dataSet);

}

}