Talk to your database with Doctrine

PHP Conference Argentina 2013

Benjamin Eberlei, @beberlei

5th October 2013

Qafoo
passion for software quality

About me

passion for software quality

Helping people to create high quality web applications.

`http://qafoo.com`

- Doctrine Developer
- Symfony Contributor
- Twitter `@beberlei` and `@qafoo`

What is Doctrine?

The Doctrine Project is the home of a selected set of PHP libraries primarily focused on databases and related functionality.

Databases?

- Relational databases
  - MySQL
  - PostgreSQL
  - Oracle
  - SQL Server
  - Sqlite
  - Experimental: Drizzle, DB2, Sybase
- Non-relational databases
  - Document: CouchDB, MongoDB, JCR
  - Graph: OrientDB
  - Caches: Riak, Redis, Memcache and many more

Doctrine DBAL

- Provides
  - Driver abstraction
  - SQL dialect abstraction (Both DML and DDL)
  - Convenience APIs for database access
  - SQL type abstraction
  - Database schema abstraction
- Indepedent of the ORM

Driver abstraction

- No need to use driver APIs directly in your code
- API is similar to PDO
- Supported (stable) drivers
  - PDO
  - mysqli
  - oci8
  - sqlsrv

Driver abstraction

```php
<?php

class PostTable
{
    private $pdo;

    public function connect()
    {
        $this->pdo = new PDO('mysql:dbname=blog;host=127.0.0.1', 'root', '');
        $this->pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    }

    public function listAll()
    {
        $sql = 'SELECT *
                  FROM posts
                 WHERE status = "PUBLISHED"
              ORDER BY publish_date DESC
                 LIMIT 0,20';
        $stmt = $this->pdo->query($sql);

        return $stmt->fetchAll();
    }
}
```

passion for software quality

Driver abstraction

```php
<?php

$table = new PostTable();
$table->connect();

$posts = $table->listAll();

$twig->render('posts.html.twig', array('posts' => $posts));
```

passion for software quality

Driver abstraction

```php
<?php
use Doctrine\DBAL\DriverManager;

class PostTable
{
    private $conn;

    public function connect()
    {
        $this->conn = DriverManager::getConnection(array(
            'driver'   => 'pdo_mysql',
            'dbname'   => 'blog',
            'user'     => 'root',
        ));
    }

    public function listAll()
    {
        $sql = 'SELECT *
                  FROM posts
                 WHERE status = "PUBLISHED"
              ORDER BY publish_date DESC
                 LIMIT 0,20';
        $stmt = $this->conn->query($sql);

        return $stmt->fetchAll();
    }
}
```

passion for software quality

SQL dialect abstraction

- Concept: Platform
- APIs to control different SQL styles and names
- Examples:
  - LIMIT queries
  - usual SQL functions

SQL dialect abstraction

```php
<?php

class PostTable
{
    /**
     * @var \Doctrine\DBAL\Platform\AbstractPlatform
     */
    private $platform;

    public function connect()
    {
        // ...
        $this->platform = $this->conn->getDatabasePlatform();
    }
}
```

SQL dialect abstraction

```php
<?php

class PostTable
{
    public function listAll()
    {
        $sql = $this->platform->modifyLimitQuery(
                 'SELECT *
                    FROM posts
                   WHERE status = "PUBLISHED"
                ORDER BY publish_date DESC',
                0, 20
        );

        $stmt = $this->conn->query($sql);
        return $stmt->fetchAll();
    }
}
```

Convenience APIs

- Doctrine Connection has methods to
  - `insert($table, array $data)`
  - `update($table, array $data, $where)`
  - `delete($table, $where)`
- SQL QueryBuilder

Convenience APIs: Insert

```php
<?php

class PostTable
{
    public function insert(array $post)
    {
        $this->conn->insert('posts', $post);
    }
}
```

Convenience APIs: Insert

```
<?php

$table = new PostTable();
$table->connect();
$table->insert(array(
    'title'          => 'Hello World!',
    'content'        => 'This is my first post',
    'publish_status' => 'PUBLISHED',
    'publish_date'   => date('Y-m-d H:i:s'),
));
```

Convenience APIs: Update

```php
<?php

class PostTable
{
    public function update($id, array $post)
    {
        $this->conn->update('posts', $post, array('id' => $id));
    }
}
```

Convenience APIs: Delete

```php
<?php

class PostTable
{
    public function delete($id)
    {
        $this->conn->delete('posts', array('id' => $id));
    }
}
```

Convenience APIs: SQL QueryBuilder

```php
<?php

class PostTable
{
    public function listAll()
    {
        $query = $this->conn->createQueryBuilder();
        $query->select('*')
              ->from('posts')
              ->where('status = "PUBLISHED"')
              ->orderBy('publish_date', 'DESC')
              ->setFirstResult(0)
              ->setMaxResults(20);

        $stmt = $query->execute();
        return $stmt->fetchAll();
    }
}
```

passion for software quality

Create simple abstractions

```php
<?php

abstract class Table
{
    abstract public function getName();

    public function insert(array $data)
    {
        $this->conn->insert($this->getName(), $data);
    }

    public function update($id, array $data)
    {
        $this->conn->update($this->getName(), $data, array('id' => $id));
    }

    public function createQueryBuilder()
    {
        $query = $this->conn->createQueryBuilder();
        $query->select('*')
              ->from($this->getName());

        return $query;
    }
}
```

Create simple abstractions

```php
<?php

class PostTable extends Table
{
    public function getName()
    {
        return 'posts';
    }

    public function listAll()
    {
        $stmt = $this->createQueryBuilder()
            ->where('status = "PUBLISHED"')
            ->orderBy('publish_date', 'DESC')
            ->setFirstResult(0)
            ->setMaxResults(20)
            ->execute()
        ;

        return $stmt->fetchAll();
    }
}
```

SQL Type abstraction

- ANSI SQL does not standardize types
- Each vendor has lots of different types and semantics
- Doctrine `Type` API is abstraction for
  - Converting PHP to SQL with `convertToDatabaseValue()`
  - Converting SQL to PHP with `convertToPhpValue()`
- Supported types
  - String and long texts
  - Numbers, decimals, floats
  - Date, Time and Datetime, with and without TZ offsets
  - Blob
  - Booleans

SQL Type abstraction

```php
<?php

use Doctrine\DBAL\Types\Type;

$now = new DateTime('now');
$type = Type::getType('datetime');

$sqlValue = $type->convertToDatabaseValue($now, $platform);
// Formatted 2013-10-05 15:28:27

$phpValue = $type->convertToPhpValue($sqlValue, $platform);
// DateTime instance again
```

SQL Type abstraction

```php
<?php

use Doctrine\DBAL\Types\Type;

abstract class Table
{
    public function insert(array $data)
    {
        $this->conn->insert($this->getName(), $this->convertToSqlValues($data));
    }

    protected function convertToSqlValues(array $data)
    {
        $columnTypes = $this->getColumnTypes();

        foreach ($data as $columnName => $phpValue) {
            if (isset($columnTypes[$columnName])) {
                $type = Type::getType($columnTypes[$columnName]);
                $data[$columnName] =
                    $type->convertToDatabaseValue($phpValue, $this->platform);
            }
        }

        return $data;
    }

    abstract protected function getColumnTypes();
}
```

SQL Type abstraction

```php
<?php

class PostTable extends Table
{
    protected function getColumnTypes()
    {
        return array('publish_date' => 'datetime');
    }
}

$table = new PostTable();
$table->connect();
$table->insert(array(
    'title'          => 'Hello World!',
    'content'        => 'This is my first post',
    'publish_status' => 'PUBLISHED',
    'publish_date'   => new DateTime('now'),
));
```

Database Schema Abstraction

- API to fetch current state of database schema
- Object-Oriented Graph of
  - Tables
  - Columns
  - Indices
  - Foreign Keys
  - Sequences
- Compare different schema graphs
  - Beware: Database diffs are not perfect!

Database Schema Abstraction

```php
<?php

use Doctrine\DBAL\Schema\Table;

class PostTable extends Table
{
    public function getDefinition()
    {
        $table = new Table();
        $table->addColumn('id', 'integer');
        $table->addColumn('title', 'string');
        $table->addColumn('content', 'text');
        $table->addColumn('publish_status', 'string');
        $table->addColumn('publish_date', 'datetime');

        $table->setPrimaryKey(array('id'));
        $table->addIndex(array('publish_status', 'publish_date'));

        return $table;
    }
}
```

Qafoo
passion for software quality

Database Schema Abstraction

```php
<?php

abstract class Table
{
    public function createTable()
    {
        $schemaManager = $this->conn->getSchemaManager();
        $tableDefinition = $this->getDefinition();

        $schemaManager->createTable($tableDefinition);
    }
}
```

Database Schema Abstraction

```php
<?php
use Doctrine\DBAL\Schema\Comparator;

abstract class Table
{
    public function updateTable()
    {
        $schemaManager = $this->conn->getSchemaManager();
        $tableDefinition = $this->getDefinition();

        $current = $schemaManager->listTableDetails($this->getName());

        $comparator = new Comparator();
        $tableDiff = $comparator->diffTable($current, $tableDefinition);

        $sqls = $this->platform->getAlterTableSQL($tableDiff);

        foreach ($sqls as $sql) {
            $this->conn->exec($sql);
        }
    }
}
```

passion for software quality

Doctrine ORM

- Provides
  - Mapping PHP Objects to Database
  - Manages Association between objects
  - Creates SQL Schema from PHP Objects
- ORMs are leaky abstraction
  - Knowledge of underyling SQL is highly recommended
  - Mapping between SQL and Objects has performance penalty
  - Not always the best solution for all problems

Defining PHP Objects

- PHP Objects managed with Doctrine are called Entities
- Doctrine uses DataMapper pattern:
  - No base class or interface required for your entities
  - Usage of the constructor is allowed
- Configuration of the mapping is necessary

1. Maps PHP Objects to DB Tables

```php
<?php
class Post
{
    protected $id;
    protected $title;
    protected $body;
}
```

```sql
CREATE TABLE Post (id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    body TEXT
);
```

2. Metadata Mapping with Annotations, XML, Yaml

```php
<?php
/** @Entity **/
class Post
{
    /** @Id @GeneratedValue @Column(type="integer") **/
    protected $id;
    /** @Column(type="string") **/
    protected $title;
    /** @Column(type="text") **/
    protected $body;
}
```

Defininig Associations

- Doctrine manages foreign keys by looking at object references
    - No explicit foreign key management necessary
- Reference to a single object is N:1 or 1:1
- Reference to a collection is 1:N or M:N

3. Object-References map to Foreign Keys

```php
<?php
/** @Entity **/
class Post
{
    /**
     * @ManyToOne(targetEntity="User")
     **/
    protected $author;

    public function __construct(User $user)
    {
        $this->author = $user;
    }
}

$user = new User();
$post = new Post($user);
```

passion for software quality

4. ”Collections” contain many object references

```php
<?php
use Doctrine\Common\Collections\ArrayCollection;

class Post
{
    /**
     * @OneToMany(targetEntity="Comment", mappedBy="post",
     *    cascade={"persist"})
     **/
    protected $comments;

    public function __construct()
    {
        $this->commments = new ArrayCollection();
    }

    public function addComment($text)
    {
        $this->comments[] = new Comment($this, $text);
    }
}
```

passion for software quality

EntityManager

- The `EntityManager` is facade to all Doctrine APIs
- Allows to add and remove objects from the database
- Seperation of notification and actual transaction
  - `persist` and `remove` methods
  - `flush` batches SQL operations in single transaction

5. EntityManager has to know about objects

```php
<?php

$entityManager->persist($post);
$entityManager->persist($user);
```

passion for software quality

6. EntityManager#flush() batches SQL operations

```php
<?php

$entityManager->flush();
```

Finding Objects

- Using simple finders
  - By ID
  - By Key=Value Conditions
- Using `Criteria`
  - Object-Oriented API
  - Allows more comparison operators
- Doctrine Query Lanuage (DQL)

7. Find by ID

```php
<?php

$post = $entityManager->find("Post", $id);
```

8. Find by Criteria

```php
<?php

$authorRepository = $entityManager->getRepository("Author");
$author = $authorRepository->find($authorId);

$postRepository = $entityManager->getRepository("Post");
$post = $postRepository->findOneBy(array("title" => "Hello World!"));

$posts = $postRepository->findBy(
    array("author" => $author),
    array("title" => "ASC")
);
```

Doctrine Query Language

- DQL is not SQL (its own Object Query Language)
- Classes and fields instead of tables and columns
- Real (cachable) parser manually constructed from EBNF
- Uses Runtime Metadata Information

9. Find with DQL

```php
<?php

$dql = "SELECT p AS post, count(c.id) AS comments " .
       "FROM Post p JOIN p.comments c GROUP BY p";
$results = $entityManager->createQuery($dql)->getResult();

foreach ($results as $row) {
    echo $row['post']->getTitle() . " (" . $row['comments'] . ")";
}
```

passion for software quality

Qafoo

passion for software quality

THANK YOU

Rent a quality expert

qafoo.com