

# Doctrine To Use Or Not To Use

Benjamin Eberlei  
October 12, 2015



- ▶ Doctrine Project Team-Lead
- ▶ blogging [www.whitewashing.de](http://www.whitewashing.de)
- ▶ tweeting @beberlei



**We promote high quality code with  
trainings and consulting**

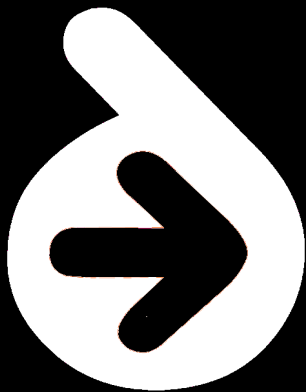
<http://qafoo.com>



**Profiling, Performance Monitoring, Error  
Tracking for PHP**

<https://tideways.io>







*I've often felt that much of the frustration with ORMs is about inflated expectations.* (Martin Fowler)

<http://martinfowler.com/bliki/OrmHate.html>

Mapping of relational data into memory is  
**really hard**

# Vilfredo Pareto (1848-1923)

---



Licensed under CC BY-SA 3.0 via Commons [https://commons.wikimedia.org/wiki/File:Vilfredo\\_Pareto.jpg](https://commons.wikimedia.org/wiki/File:Vilfredo_Pareto.jpg)





# Economics and Doctrine?

---

We built Doctrine for the 80% use case + \$some.

# Economics and Doctrine?

---

You can **implement 80%** of your use cases with  
Doctrine **in a sane way.**

Choosing Doctrine does not require you to go all in

# What is Doctrine good/bad at?

---

- ▶ Very good: CRUD (and Symfony Validator + Forms)
- ▶ Good: "DDD Lite" with coupling to Doctrine/DB
- ▶ Okish: Domain Driven Design (decoupling requires work)
- ▶ Bad: (Extremely-) high write throughput
- ▶ Bad: Analytics, Statistics, Reporting

Doctrine enables Domain-Driven Design  
**to some degree.**

**BUT**

to avoid a meltdown you  
**must still think in relations.**

# ORM Building Blocks: The 80%

---

- ▶ CRUD (find, persist, remove, flush)
- ▶ Associations
- ▶ Lazy Loading
- ▶ Association Fetching
- ▶ Doctrine Query Language
- ▶ Native Queries



## You are at risk of

- ▶ Unsolvable Edge Cases
- ▶ High Coupling
- ▶ Performance Problems
- ▶ Complexity

---

# 1. Inheritance



# 1. Inheritance: Lazy Loading

---

```
1 <?php
2
3 abstract class Product
4 {
5 }
6
7 class OrderItem
8 {
9     /**
10      * @ORM\ManyToOne(targetEntity="Product")
11      */
12     private $inventoryItem;
13 }
```

## 2. Inheritance

---

- ▶ Lazy Loading does not work on inheritance associations
- ▶ Joined Table Inheritance
  - ▶ Slow because of many (hidden) JOINS
  - ▶ Moving a field requires moving column between tables

# Alternative: The "nosql" column

---

```
1 <?php
2
3 class Product
4 {
5     /**
6      * @ORM\Column(type="json_array")
7      * @var array
8      */
9     private $attributes;
10 }
```

---

## 2. Events

## 2. Events

---

```
1 <?php
2
3 $product = $entityManager->find (Product::class , 1234);
4
5 $product->setTitle ("foo" );
6
7 $entityManager->flush () ;
```

## 2. Events

---

- ▶ Make interaction with UnitOfWork even more complex
- ▶ Run on **every** flush or load operation
- ▶ Don't run in context of the domain
- ▶ Some of the extensions on top of events are very inefficient

Move behaviour into  
entities, services or domain events

---

## 3. Batch Processing



## 3. Batch Processing

---

```
1 <?php
2
3 $dql = 'SELECT p FROM Product p';
4 $query = $entityManager->createQuery($dql);
5
6 $it = $query->iterate();
7 $i = 0;
8
9 foreach ($it as $product) {
10     $i++;
11
12     $product->setPrice(10);
13
14     if ($i % 20 == 0) {
15         $entityManager->flush();
16     }
17 }
```

## 3. Batch Processing

---

- ▶ ORMs are really bad at this
- ▶ Both memory and CPU bottleneck
- ▶ Highly coupled code

# Alternative: Model Changes

---

```
1 <?php
2
3 class UpdatePrice extends ProductChange
4 {
5     public $productId;
6     public $price;
7 }
8
9 class UpdateAvailability extends ProductChange
10 {
11     public $productId;
12     public $availability;
13 }
```

# Alternative: Model Changes

---

```
1 <?php
2
3
4 $update1 = new UpdatePrice ([
5     'productId' => 1234,
6     'price' => 10,
7 ] );
8
9 $update2 = new UpdateAvailability ([
10    'productId' => 4444,
11    'availability' => 120,
12 ] );
13
14 $repository ->applyChanges ([
15     $update1 ,
16     $update2
17 ] );
```

---

## 4. DQL, Fetch-Joins and Pagination

## 4. DQL, Fetch-Joins and Pagination

---

Order	Customer Name	Items	Total	State
<a href="#">#1234</a>	John Doe 53111 Bonn, Germany	4 <a href="#">Acme Super-Widget</a> and 3 more	120 €	Paid
<a href="#">#1234</a>	Petra Musterfrau 10111 Berlin, Germany	2 <a href="#">Acme Super-Widget</a> and one more	27 €	Paid

## 4. DQL, Fetch-Joins and Pagination

---

```
1 <?php
2
3 $dql = 'SELECT o, oi, p, da, sa, c
4       FROM Order o
5       JOIN o.orderItems oi
6       JOIN oi.product p
7       JOIN o.deliveryAddress da
8       JOIN o.shippingAddress sa
9       JOIN o.customer c';
10
11 $query = $entityManager->createQuery($query);
12 $paginator = new Paginator($query);
```

## 4. DQL, Fetch-Joins and Pagination

---

- ▶ Complex DQL queries with lots of fetches are slow
- ▶ Pagination applies some black magic that may not work
- ▶ Resulting entities are never exactly what you need to query
- ▶ Time-investment to extend DQL is often not worth it



# Alternative: View/Read Models

---

```
1 <?php
2
3 class OrderListEntry extends Struct
4 {
5     // all the columns
6     public $id;
7     public $orderNumber;
8
9     // aggregated and computed values
10    public $firstItem = array();
11    public $totalItems = 0;
12
13    // or re-use value objects
14    public $deliveryAddress;
15
16    // ...
17 }
```

# Alternative: View/Read Models

---

```
1 <?php
2
3 class OrderRepository extends EntityRepository
4 {
5     /**
6      * @return OrderList
7      */
8     public function listEntries($criteria)
9     {
10         $connection = $this->getEntityManager()
11                       ->getConnection();
12
13         // HERE BE SQL
14
15         return new OrderList();
16     }
17 }
```

# How-To build around the ORM in Symfony

---

The database connection is readily available:

- ▶ `$entityManager->getConnection()`
- ▶ `Service doctrine.dbal.default_connection`

# Doctrine DBAL APIs

---

```
1 <?php
2
3 class ProductRepository
4 {
5     private function applyPriceChange($change)
6     {
7         $connection = $this->getEntityManager()
8             ->getConnection();
9
10        $connection->update(
11            'products',
12            [ 'price' => $change->price ],
13            [ 'id' => $change->productId ]
14        );
15    }
16 }
```

# Doctrine DBAL APIs

---

```
1 <?php
2
3 class OrderRepository extends EntityRepository
4 {
5     public function listEntries($criteria)
6     {
7         $connection = $this->getEntityManager()
8                       ->getConnection();
9         $qb = $connection->createQueryBuilder();
10        $qb->from('orders', 'o')
11           ->innerJoin(
12               'o', 'customers', 'c',
13               'o.customer_id = c.id'
14           )
15           // ...
16     }
17 }
```

## You should avoid rolling your own

- ▶ Significant time investement
- ▶ hard to pull out
- ▶ Easier to use existing combined with custom SQL/code

## You can use NoSQL

- ▶ avoid the mapping complexity
- ▶ Not all data is relational

---

# Conclusions



# Rule of Thumb #1

---

Avoid complex mapping by using appropriate  
in-memory data-structures

## Rule of Thumb #2

---

You still have to think about the database

<https://joind.in/14976>



THANK YOU

Rent a quality expert  
[qafoo.com](https://qafoo.com)