Doctrine ORM

Reza Salehi
CLOUD CONSULTANT

@zaalion linkedin.com/in/rezasalehi2008

Overview

Introducing Doctrine

Doctrine libraries

- Doctrine ORM

Setting up Doctrine

- Composer

Using Doctrine ORM with PHP

Demo: Doctrine ORM

Introducing Doctrine PHP

Doctrine (PHP)

"A set of PHP libraries primarily focused on providing persistence services and related functionality."

en.wikipedia.org/wiki/Doctrine_(PHP)

https://www.doctrine-project.org/

Doctrine: PHP Open Source Proje

doctrine-project.org

Projects Development Events Consulting Partners Blog

Edit Translate Search

Doctrine

The Doctrine Project is the home to several PHP libraries primarily focused on database storage and object mapping. The core projects are the Object Relational Mapper (ORM) and the Database Abstraction Layer (DBAL) it is built upon.

Get Started View Projects

Featured Partner

stickermule

Sticker Mule is the fastest and easiest way to buy custom printed products. Thousands of people trust us to make kick ass stickers, labels,...

View More

Doctrine has been downloaded a total of **1,669,377,829** times!

Why use Doctrine?

- Around since 2006 with very stable, high-quality codebase.
- Extremely flexible and powerful object-mapping and query features.
- Support for both high-level and low-level database programming for all your use-cases.
- Large Community and integrations with many different frameworks (Symfony, Laravel, Zend Framework and more)

Latest Blog Posts

- Released doctrine/migrations 3.0-alpha
- Doctrine MongoDB ODM 1.3.0 and 2.0.0-RC2 released

Projects

- Annotations
- Cache
- Coding Standard
- Collections
- Common
- Database Abstraction Layer
- Event Manager
- Inflector
- Instantiator

https://www.doctrine-project.org/projects.html

Annotations

Docblock Annotations Parser

Docs GitHub

Cache

PHP Doctrine Cache library is a popular cache implementation that supports many different drivers such as redis, memcache, apc, mongodb and others.

Docs GitHub

Coding Standard

The Doctrine Coding Standard is a set of PHPCS rules applied to all Doctrine projects.

Docs GitHub

Collections

PHP Doctrine Collections library that adds additional functionality on top of PHP arrays.

Docs GitHub

Common

PHP Doctrine Common project is a library that provides additional functionality that other Doctrine projects depend on such as better reflection support, persistence interfaces, proxies, event system and much more.

Docs GitHub

Database Abstraction Layer

Powerful PHP database abstraction layer (DBAL) with many features for database schema introspection and management.

Docs GitHub

https://www.doctrine-project.org/projects.html

Event Manager

The Doctrine Event Manager is a simple PHP event system that was built to be used with the various Doctrine projects.

Docs GitHub

Inflector

PHP Doctrine Inflector is a small library that can perform string manipulations with regard to upper/lowercase and singular/plural forms of words.

Docs GitHub

Instantiator

A small, lightweight utility to instantiate objects in PHP without invoking their constructors

Docs GitHub

Lexer

PHP Doctrine Lexer parser library that can be used in Top-Down, Recursive Descent Parsers.

Docs GitHub

Migrations

PHP Doctrine Migrations project offer additional functionality on top of the database abstraction layer (DBAL) for versioning your database schema and easily deploying changes to it. It is a very easy to use and a powerful tool.

Docs GitHub

MongoDB Abstraction Layer

PHP Doctrine MongoDB project is a library that provides a wrapper around the native PHP Mongo PECL extension to provide additional functionality.

Docs GitHub

https://www.doctrine-project.org/projects.html

Docs GitHub

Docs GitHub

Reflection

The Doctrine Reflection project is a simple library used by the various Doctrine projects which adds some additional functionality on top of the reflection functionality that comes with PHP. It allows you to get the reflection information about classes, methods and properties statically.

Docs GitHub

RST Parser

PHP library to parse reStructuredText documents and generate HTML or LaTeX documents.

Docs GitHub

Skeleton Mapper

The Doctrine SkeletonMapper is a skeleton object mapper where you are 100% responsible for implementing the guts of the persistence operations. This means you write plain old PHP code for the data repositories, object repositories, object hydrators and object persisters.

Docs GitHub

https://www.doctrine-project.org/projects.html

Docs GitHub

Docs GitHub

MongoDB Object Document Mapper

PHP Doctrine MongoDB Object Document Mapper (ODM) provides transparent persistence for PHP objects to MongoDB.

Docs GitHub

Object Relational Mapper

PHP object relational mapper (ORM) that sits on top of a powerful database abstraction layer (DBAL). One of its key features is the option to write database queries in a proprietary object oriented SQL dialect called Doctrine Query Language (DQL). This provides developers with a powerful alternative to SQL that maintains flexibility without requiring unnecessary code duplication.

Docs GitHub

Persistence

The Doctrine Persistence project is a set of shared interfaces and functionality that the different Doctrine object mappers share.

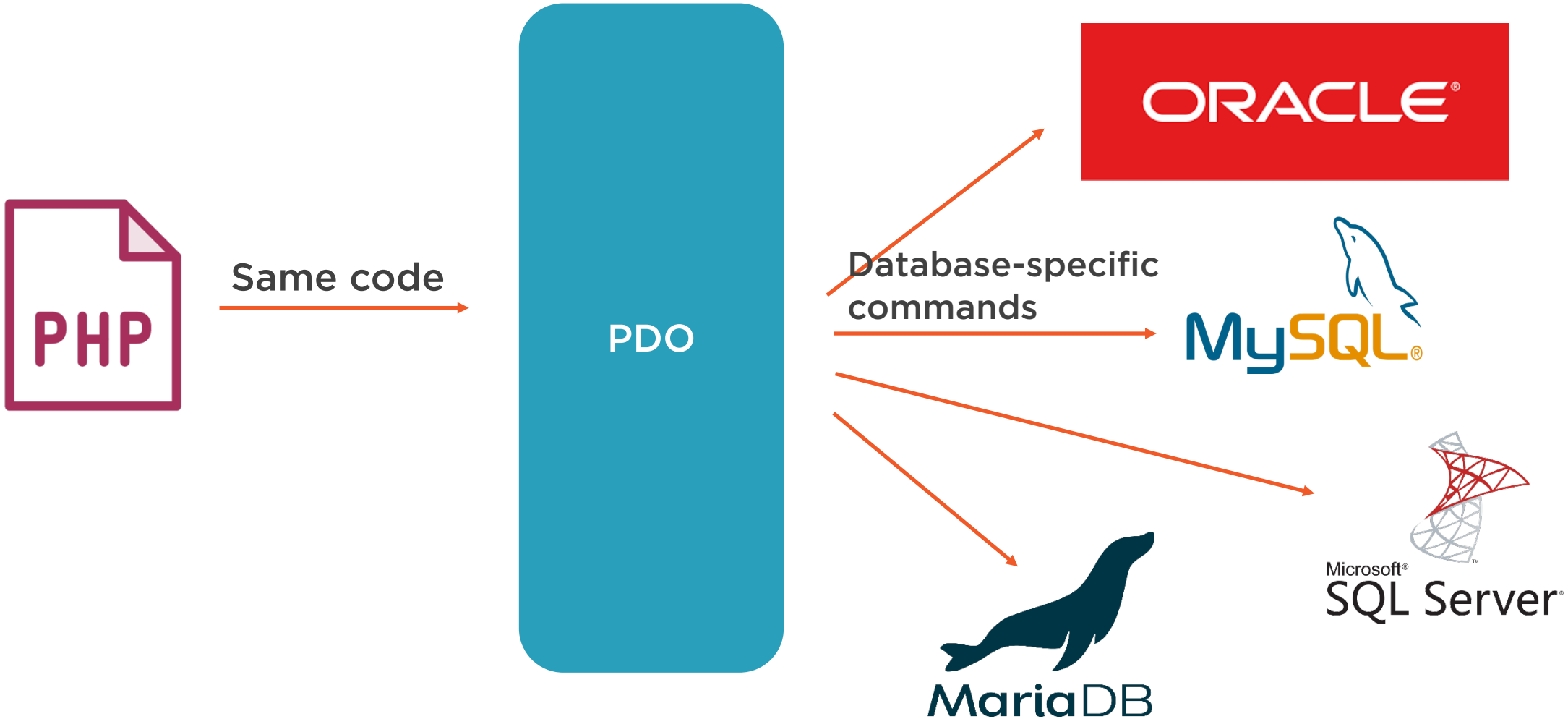
Docs GitHub

PHPCR ODM

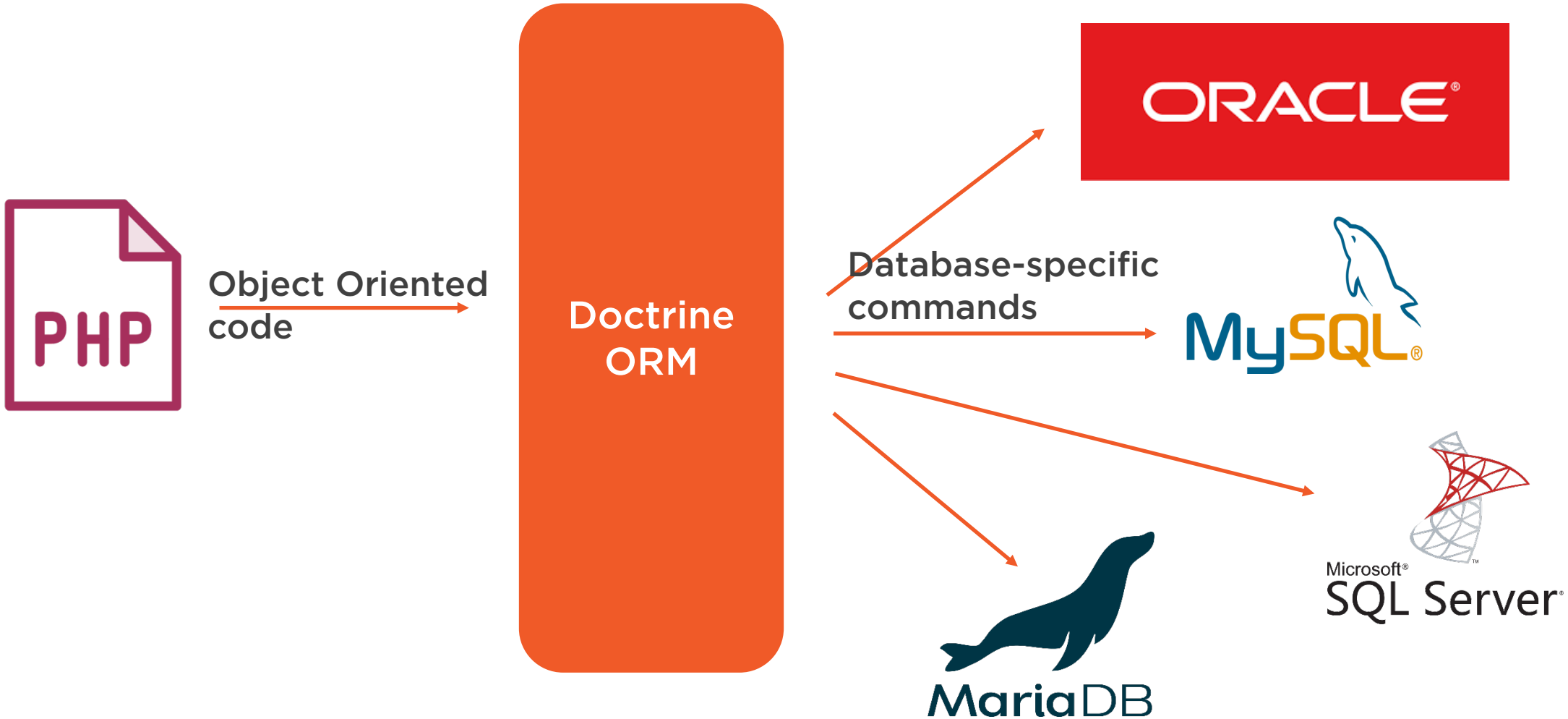
PHP Doctrine Content Repository Object Document Mapper (ODM) provides transparent persistence for PHP objects.

Docs GitHub

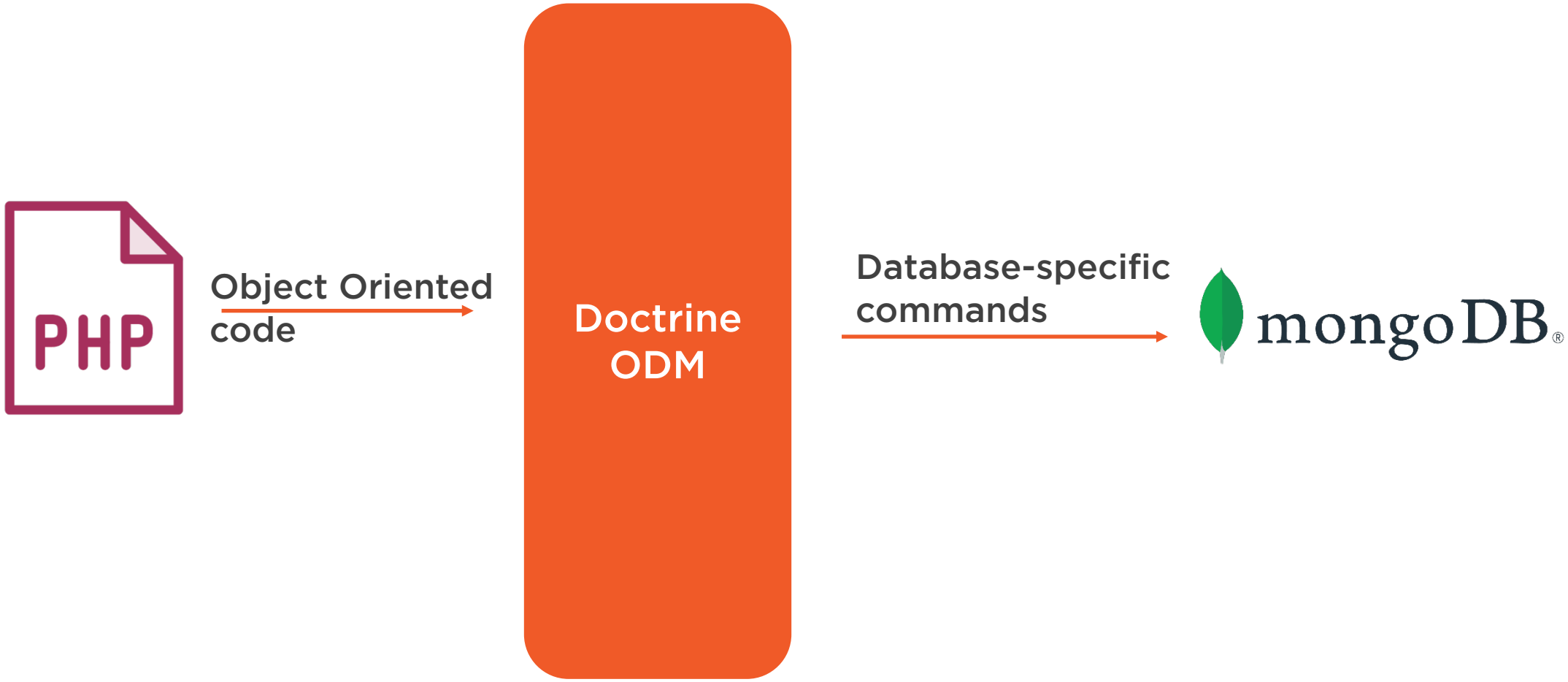
PDO Data-access Abstraction



Doctrine ORM



Doctrine ORM



Doctrine ORM

The project started in 2006

Write database queries in Doctrine Query Language (DQL)

Works with several relational databases

Doctrine ORM 2.0 is available for PHP 7.1+

Setting up Doctrine ORM

Object Relational Mapper - Doct

https://www.doctrine-project.org/projects/orm/2.7.html

doctrine-project.org/projects/orm/2.7.html

Projects Development Events Consulting Partners Blog Edit Translate Search ORM

Projects / ORM / 2.7

Object Relational Mapper

2.7.3 stable

PHP object relational mapper (ORM) that sits on top of a powerful database abstraction layer (DBAL). One of its key features is the option to write database queries in a proprietary object oriented SQL dialect called Doctrine Query Language (DQL). This provides developers with a powerful alternative to SQL that maintains flexibility without requiring unnecessary code duplication.

Docs GitHub

Install

```
$ composer require doctrine/orm:2.7.3
```

Releases

2020-05-27	2.7.3	stable	
2020-03-21	2.7.2	stable	
2020-02-15	2.7.1	stable	
2019-11-19	2.7.0	stable	

Use Composer to install Doctrine.

Composer

https://getcomposer.org/

getcomposer.org

A Dependency Manager for PHP

Latest: **1.10.10** (changelog)

A preview release for our next major version is available!
Try out **2.0.0-alpha3** (changelog) now using *composer self-update --preview*

Getting Started | Download

Documentation | Browse Packages

Issues | GitHub

Authors: Nils Adermann, Jordi Boggiano and many community contributions

Sponsored by:

Logo by: WizardCat

Composer

https://getcomposer.org/download/

getcomposer.org/download/

Home | Getting Started | Download | Documentation | Browse Packages

Download Composer

Latest: v1.10.10

Windows Installer

The installer will download composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

Download and run Composer-Setup.exe - it will install the latest composer version whenever it is executed.

Command-line installation

To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation, use the guide on installing Composer programmatically.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === '8a6138e2a05a8c28539c9f0fb361159823655d7ad2deecb371b6
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

This installer script will simply check some `php.ini` settings, warn you if they are set incorrectly, and then download the latest `composer.phar` in the current directory. The 4 lines above will, in order:

- Download the installer to the current directory
- Verify the installer SHA-384, which you can also cross-check here
- Run the installer
- Remove the installer

WARNING: Please do not redistribute the install code. It will change with every version of the installer. Instead, please link to this page or check how to install Composer programmatically.

Installer Options

--install-dir

You can install composer to a specific directory by using the `--install-dir` option and providing a target directory. Example:

```
php composer-setup.php --install-dir=bin
```

Getting Started with Doctrine

This guide covers getting started with the Doctrine ORM. After working through the guide you should know:

- How to install and configure Doctrine by connecting it to a database
- Mapping PHP objects to database tables
- Generating a database schema from PHP objects
- Using the `EntityManager` to insert, update, delete and find objects in the database.

Guide Assumptions

This guide is designed for beginners that haven't worked with Doctrine ORM before. There are some prerequisites for the tutorial that have to be installed:

- PHP (latest stable version)
- Composer Package Manager (`Install Composer <https://getcomposer.org/doc/00-intro.md>`_)

The code of this tutorial is available on Github.

> This tutorial assumes you work with **Doctrine 2.6** and above. Some of the code will not work with lower versions.

What is Doctrine?

Doctrine 2 is an object-relational mapper (ORM) for PHP 7.1+ that provides transparent persistence for PHP objects. It uses the Data Mapper pattern at the heart, aiming for a complete separation of your domain/business logic from the persistence in a relational database management system.

The benefit of Doctrine for the programmer is the ability to focus on the object-oriented business logic and worry about persistence only as a secondary problem. This doesn't mean persistence is downplayed by Doctrine 2, however it is our belief that there are considerable benefits for object-oriented programming if persistence and entities are kept separated.

- Bugs can be paginated through a list-view.

Project Setup

Create a new empty folder for this tutorial project, for example `doctrine2-tutorial` and create a new file `composer.json` inside that directory with the following contents:

```
{
    "require": {
        "doctrine/orm": "^2.6.2",
        "symfony/yaml": "2.*"
    },
    "autoload": {
        "psr-0": {"": "src/"}
    }
}
```

Install Doctrine using the Composer Dependency Management tool, by calling:

```
$ composer install
```

This will install the packages Doctrine Common, Doctrine DBAL, Doctrine ORM, into the `vendor` directory.

Add the following directories:

```
doctrine2-tutorial
|-- config
|   `-- xml
|   `-- yaml
`-- src
```

The YAML driver is deprecated and will be removed in version 3.0. It is strongly recommended to switch to one of the other mappings.

Top

Obtaining the EntityManager

Doctrine's public interface is through the `EntityManager`. This class provides access points to the complete lifecycle management for your entities, and transforms entities from and back to persistence. You have to configure and create it to use your entities with Doctrine 2. I will show the configuration steps and then discuss them step by step:

```php
<?php
// bootstrap.php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for Annotations
$isDevMode = true;
$proxyDir = null;
$cache = null;
$useSimpleAnnotationReader = false;
$config = Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"), $isDevMode, $proxyDi
// or if you prefer yaml or XML
//$config = Setup::createXMLMetadataConfiguration(array(__DIR__."/config/xml"), $isDevMode);
//$config = Setup::createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"), $isDevMode);

// database configuration parameters
$conn = array(
    'driver' => 'pdo_sqlite',
    'path' => __DIR__ . '/db.sqlite',
);

// obtaining the entity manager
$entityManager = EntityManager::create($conn, $config);
```

The YAML driver is deprecated and will be removed in version 3.0. It is strongly recommended to switch

Top

Metadata for an Entity can be configured using DocBlock annotations directly in the Entity class itself, or in an external XML or YAML file. This Getting Started guide will demonstrate metadata mappings using all three methods, but you only need to choose one.

PHP XML

```php
<?php
// src/Product.php

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="products")
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue
     */
    protected $id;
    /**
     * @ORM\Column(type="string")
     */
    protected $name;

    // .. (other code)
}
```

The YAML driver is deprecated and will be removed in version 3.0. It is strongly recommended to switch to one of the other mappings.

```yaml
# config/yaml/Product.dcm.yml
Product:
```

Top

```
    // .. (other code)
}
```

The YAML driver is deprecated and will be removed in version 3.0. It is strongly recommended to switch to one of the other mappings.

```yaml
# config/yaml/Product.dcm.yml
Product:
  type: entity
  table: products
  id:
    id:
      type: integer
      generator:
        strategy: AUTO
  fields:
    name:
      type: string
```

The top-level `entity` definition specifies information about the class and table name. The primitive type `Product#name` is defined as a `field` attribute. The `id` property is defined with the `id` tag. It has a `generator` tag nested inside, which specifies that the primary key generation mechanism should automatically use the database platform's native id generation strategy (for example, AUTO INCREMENT in the case of MySql, or Sequences in the case of PostgreSql and Oracle).

Now that we have defined our first entity and its metadata, let's update the database schema:

```
$ vendor/bin/doctrine orm:schema-tool:update --force --dump-sql
```

Specifying both flags `--force` and `--dump-sql` will cause the DDL statements to be executed and then printed to the screen.

Now, we'll create a new script to insert products into the database:

Setting up Doctrine ORM

Install Doctrine using PHP dependency manager (Composer)

Include Composer autoload file in your project

Obtain an *EntityManager* object

Create your entities (maps to your database tables)

Create Your Entities

Code first

Start with developing Objects and then map them onto your database

Model first

Model using tools (e.g. UML), then generate DB schema and PHP code from this model

Database first

Already have a DB schema and generate corresponding PHP code from it

Coding with Doctrine ORM

Doctrine ORM allows interacting with databases without SQL queries.

```
$person = new Person();
$person->setFirstname("Reza");
$person->setLastname("Salehi");

$entityManager->persist($person);
$entityManager->flush();
```

Insert a New Record

```
$person = $entityManager->find('Person’, 12);
$person->setWight(152);
$entityManager->flush();
```

Update a Record

```
$person = $entityManager->find('Person’, 12);
$entityManager->remove($person);
$entityManager->flush();
```

Delete a Record

```
$personRepository = $entityManager-> getRepository('Person');
$persons = $personRepository->findAll();
```

Find All Records

Demo

Setting up Doctrine ORM

- Installing Composer
- Installing Doctrine

Demo

Using Doctrine ORM with PHP

- Configuring the PHP project

Summary

Introducing Doctrine

Doctrine ORM

Setting up Doctrine

- Composer

Using Doctrine ORM with PHP

Demo:

- Setup Doctrine ORM
- Using Doctrine in PHP