

Docs

ORM Project

Doctrine ORM and DBAL configured from an array, and the repositories of Derafu over Doctrine entities. A proof of concept without a stable API

Anonymous · 26/10/2026 · #php

Table of Contents

ORM Project 3

Introduction 4

Entity Manager 6

Repositories 8

Migrations 9

Symfony Integration 13

Docs » Data Handling Category » ORM Project

ORM Project

Doctrine ORM and DBAL configured from an array, and the repositories of Derafu over Doctrine entities. A proof of concept without a stable API

Anonymous · 26/10/2026 · #php

Derafu ORM

Last updated on 26/10/2026

Introduction

Doctrine ORM and DBAL configured from an array, and the repositories of Derafu over Doctrine entities. A proof of concept without a stable API

Anonymous · 26/10/2026

Models and Entities



Since 2006, I participated in FLISoL. In 2014, I presented SowerPHP at UTEM (Santiago, Chile). Years before, I presented MiPaGiNa at the University of Chile. On that occasion, someone in the audience said to me “Oh, you built an ORM,” and I said, “What is that?” That day, during the presentation I was giving, I learned what an ORM was. This project compiles the ideas that have appeared in my mind over the years. Always with the goal of making everything “automagical.”

– Esteban De La Fuente Rubio

derafu/orm does not implement an ORM: it uses Doctrine ORM and DBAL. It is a library, not a framework bundle, and its API is not stable yet.

Installation

PHP 8.5 or later is required.

```
composer require derafu/orm
```

Components

Class	Purpose
EntityManagerFactory	Create an EntityManager from an array of options (EntityManagerFactoryInterface)
ConnectionParametersFactory	Create the parameters of the connection, with its DSN (ConnectionParametersFactoryInterface)
DoctrineRepositoryFactory, DoctrineRepository	Give the entities of Doctrine through the repository manager of derafu/repository

Class	Purpose
<code>DoctrineCliConfigurationFactory</code>	Create what the CLI of Doctrine Migrations needs, from a DSN or an array

Guides

- [Entity manager](#): the `dbal` and `orm` options, with the types, the caches and the errors.
- [Repositories](#): one repository manager for the entities of Doctrine and the data sources of `derafu/repository`.
- [Migrations](#): the configuration of the CLI of Doctrine Migrations, without a framework.
- [Symfony integration](#): wire the entity manager and the repository factory with Symfony DependencyInjection.

Last updated on 26/10/2026

Entity Manager

Create an entity manager from an array, with the options of Doctrine DBAL and ORM

Anonymous · 26/10/2026

Entity Manager

`EntityManagerFactory` creates an `EntityManager` from an array of options. It implements `Derafu\Orm\Contract\EntityManagerFactoryInterface`, so it can be a service that is injected and replaced.

```
use Derafu\Orm\Factory\EntityManagerFactory;

$entityManager = (new EntityManagerFactory())->create([
    'dbal' => [
        'url' =>
        'mysql://app:secret@127.0.0.1:3306/app?serverVersion=8.0&charset=utf8mb4',
    ],
    'orm' => [
        'paths' => [__DIR__ . '/src/Entity'],
        'dev_mode' => false,
        'naming_strategy' => Doctrine\ORM\Mapping\UnderscoreNamingStrategy::class,
    ],
]);
```

The configuration has two sections, `dbal` and `orm`, with the options that Doctrine has. The entities are mapped with PHP attributes, and Doctrine's native lazy objects are always enabled.

The parameters of the connection are created by a `ConnectionParametersFactoryInterface`, which `EntityManagerFactory` receives in its constructor. `ConnectionParametersFactory` is the default one.

dbal

Every parameter of the connection that DBAL's `DriverManager` understands, with its official name (`driver`, `path`, `memory`, `host`, `port`, `user`, `password`, `dbname`, `charset`, `serverVersion`, `defaultTableOptions`, `driverOptions`...). Doctrine checks them. The section also has these options:

Option	Value
<code>url</code>	A DSN, read with DBAL's <code>DsnParser</code> . The schemes are <code>mysql</code> , <code>mysqli</code> , <code>pgsql</code> , <code>postgresql</code> , <code>postgres</code> , <code>sqlite</code> , <code>sqlserver</code> and <code>mssql</code> . The parameters of the DSN win over the other ones

Option	Value
types	Custom types, as name => class of a Doctrine\DBAL\Types\Type. The uuid type is always registered
schema_filter	A regular expression of the tables that Doctrine keeps

A SQLite connection needs a path, unless it is memory. A DSN like `sqlite:///memory:` is in memory, and a relative path in a SQLite DSN is relative to the current directory.

The DBAL type registry is global. A type that is already registered is replaced, so the same configuration can be used more than once.

orm

Option	Value
paths	List of directories with the entities
dev_mode	true or false
naming_strategy	A class, or an instance, of Doctrine\ORM\Mapping\NamingStrategy
resolve_target_entities	interface => class of the entity that stands for the interface
dql	Custom functions of DQL: string_functions, numeric_functions and datetime_functions, each as name => class of a FunctionNode
metadata_cache, query_cache, result_cache, hydration_cache	A PSR-6 pool each

Without the caches, `ORMSetup` of Doctrine chooses one: in memory in development mode, and APCu, Memcached or Redis if they are available otherwise.

Errors

The options are read with [derafu/config](#), through the typed readers of `derafu/array`, so one with another type raises their translatable error, naming the option (for example The key `"orm.paths"` must be of type `list<string>`, string given.). `DatabaseConfigurationException` is for what the package itself checks: a database that is not configured, a DSN that is not valid or has no known driver, a SQLite connection without a file, a regular expression that is not valid, a class that is not of the expected type, and a map whose keys or values are not strings.

Last updated on 26/10/2026

Docs » Data Handling Category » ORM Project » Repositories

Repositories

One repository manager for the entities of Doctrine and the data sources of derafu/repository

Anonymous · 26/10/2026

Repositories of Derafu over Doctrine

`DoctrineRepositoryFactory` creates a repository of `derafu/repository` for a class that Doctrine maps. Registered in the repository manager under those classes, one manager gives the repository of every class, whether its data is in a table or in a data source:

```
use Derafu\Orm\Factory\DoctrineRepositoryFactory;
use Derafu\Repository\Service\DataProvider;
use Derafu\Repository\Service\RepositoryManager;

$manager = new RepositoryManager(
    dataProvider: new DataProvider([App\Entity\Country::class => __DIR__ .
'/countries.php']),
    factories: [App\Entity\Invoice::class => new
DoctrineRepositoryFactory($entityManager)],
);

$invoices = $manager->getRepository(App\Entity\Invoice::class); // Doctrine.
$countries = $manager->getRepository(App\Entity\Country::class); // Data source.
```

Both are a `RepositoryInterface`, with the same queries (see [querying](#)). An entity in the database can refer to one that is not by its identifier, in a plain column, and get it through the manager.

The factory takes any `EntityManagerInterface`, not only the one of `EntityManagerFactory`. It rejects an identifier that is not a class that Doctrine maps with a `ManagerException`, and each class has to be registered, as the manager does not look for the classes that Doctrine maps by itself.

`DoctrineRepository` gives the queries of `RepositoryInterface` over a Doctrine repository. The lock arguments of `find()` reach Doctrine, which requires a transaction for a pessimistic lock, as when it is used directly. The methods that are only of Doctrine's repository, such as `createQueryBuilder()`, are available through the entity manager.

Last updated on 26/10/2026

Migrations

The configuration of the CLI of Doctrine Migrations from a DSN or an array, without a framework

Anonymous · 26/10/2026

Migrations

The CLI of [Doctrine Migrations](#), `vendor/bin/doctrine-migrations`, has to know the database that it connects to and the directory of the migrations. In an application with `doctrine-migrations-bundle` the bundle gives it, and its commands are used instead. Without a framework, the convention of Doctrine is a `cli-config.php` file that returns a `DependencyFactory`. Without that file the command stops with `It was not possible to locate any configuration file.`

`DoctrineCliConfigurationFactory` creates that `DependencyFactory` from a DSN or from an array, so the file is a single line.

The CLI looks for a file called `cli-config.php` in the directory where the command runs, and then in its `config/` subdirectory, so it can be in the root of the project or in `config/`. The file returns the factory:

```
use Derafu\Orm\Factory\DoctrineCliConfigurationFactory;

return (new
DoctrineCliConfigurationFactory())->create('postgresql://app:secret@127.0.0.1:5432/app?serverVersion=16&charset=utf8');
```

The credentials do not have to be in the file: `create((string) getenv('DATABASE_URL'))` reads the DSN from the environment.

DSN

Database	DSN
PostgreSQL	<code>postgresql://app:secret@127.0.0.1:5432/app?serverVersion=16&charset=utf8</code>
MySQL, MariaDB	<code>mysql://app:secret@127.0.0.1:3306/app?serverVersion=8.0&charset=utf8mb4</code>
SQLite	<code>sqlite:///var/data.db</code> , relative to the current directory, or <code>sqlite:///path/to/data.db</code> , absolute

The DSN is read with DBAL's `DsnParser`, as the `url` option of the [entity manager](#), so the other schemes and the rules are the same: `pgsql` and `postgres` are the same driver as `postgresql`, the

parameters after `?` are parameters of the connection (`serverVersion`, `charset`, `sslmode...`), and a special character in the user or the password is written URL-encoded (`p@ss` is `p%40ss`).

Array

With an array, which has the sections `dbal` and `orm` of the entity manager and `migrations`:

```
return (new DoctrineCliConfigurationFactory())->create([
    'migrations' => [
        'paths' => ['App\\Migrations' => __DIR__ . '/migrations'],
        'all_or_nothing' => true,
        'transactional' => true,
        'check_database_platform' => true,
    ],
    'dbal' => ['driver' => 'pdo_sqlite', 'path' => __DIR__ . '/var/data.db'],
    'orm' => ['paths' => [__DIR__ . '/src/Entity']],
]);
```

In `config/cli-config.php`, `__DIR__` is the `config/` directory, so the paths are built from `dirname(__DIR__)`.

The three options of `migrations` are enabled by default. With a DSN, the migrations are in `migrations/` and the entities in `src/Entity/`, relative to the current directory, so the commands are run from the root of the project. The directories of the migrations, the entities and the SQLite file are created if they do not exist. The commands are those of Doctrine: `vendor/bin/doctrine-migrations diff`, `migrate` and `status`.

Usage

The commands are run from the root of the project, with the file in place:

1. `vendor/bin/doctrine-migrations generate` creates an empty migration class in the directory of the migrations, with the `up()` and `down()` methods to complete.
2. `vendor/bin/doctrine-migrations diff` creates the migration by comparing the database with the entities of `orm.paths`: its `up()` has the SQL that makes the database like the entities (a `CREATE TABLE` for an entity whose table does not exist).
3. `vendor/bin/doctrine-migrations status` shows the database that it uses and the executed and available migrations.
4. `vendor/bin/doctrine-migrations migrate` executes the pending migrations and ends with `[OK] Successfully migrated to version: App\Migrations\Version20260101000000`.
5. `vendor/bin/doctrine-migrations up-to-date` tells if there is nothing left to execute: `[OK] Up-to-date! No migrations to execute.`

The executed migrations are recorded in the table `doctrine_migration_versions` of the database,

which Doctrine creates. No entity maps that table, so `diff` leaves it out of the comparison and never proposes to drop it, and the other commands always see it, even if the `schema_filter` of `dbal` would leave it out.

`migrate` always asks for a confirmation, and `diff` asks for one when there are migrations that are not executed. Add `--no-interaction` to run them in a script or in a CI, where nobody can answer.

Command	Purpose
<code>migrations:generate</code> (<code>generate</code>)	Create an empty migration
<code>migrations:diff</code> (<code>diff</code>)	Create a migration from the differences between the entities and the database
<code>migrations:migrate</code> (<code>migrate</code>)	Execute the migrations up to a version, or the latest
<code>migrations:execute</code> (<code>execute</code>)	Execute one or more versions, up or down
<code>migrations:status</code> (<code>status</code>)	Show the configuration and the state of the migrations
<code>migrations:list</code> (<code>list-migrations</code>)	List the migrations and their state
<code>migrations:current</code> (<code>current</code>), <code>migrations:latest</code> (<code>latest</code>)	Show the current version and the latest one
<code>migrations:up-to-date</code> (<code>up-to-date</code>)	Tell if the database is up to date
<code>migrations:version</code> (<code>version</code>)	Add or remove versions in the table of versions, without executing them
<code>migrations:rollup</code> (<code>rollup</code>)	Replace the recorded versions with a single one
<code>migrations:dump-schema</code> (<code>dump-schema</code>)	Write the schema of the database as a migration
<code>migrations:sync-metadata-storage</code> (<code>sync-metadata-storage</code>)	Update the table of versions to the latest format

Tables that no entity maps

`diff` makes the database like the entities, so it proposes to drop the tables of the database that no entity maps (a `DROP TABLE legacy_x` for a table of another application, for example). To keep it away from them, limit the tables that Doctrine reads with the `schema_filter` of the `dbal` section, a regular expression of the tables to keep:

```
'dbal' => [
    'url' =>
    'postgresql://app:secret@127.0.0.1:5432/app?serverVersion=16&charset=utf8',
    'schema_filter' => '~^(?!legacy_)~',
```

```
],
```

The `--filter-expression` option of `diff` does the same for one run. When both are used, a table has to pass both.

With SQLite, a table whose identifier is a single integer without a generated value is proposed again by every `diff`, as Doctrine reads that table back as an auto-increment key. It is a behavior of Doctrine with SQLite, not of the factory.

Last updated on 26/10/2026

Docs » Data Handling Category » ORM Project » Symfony Integration

Symfony Integration

Wire the entity manager and the repository factory with Symfony DependencyInjection

Anonymous · 26/10/2026

Symfony Integration

For an application that uses `symfony/dependency-injection`, `resources/config/orm-services.yaml` defines these services:

Service	Notes
<code>EntityManagerFactory</code> and <code>ConnectionParametersFactory</code>	With their contracts, <code>EntityManagerFactoryInterface</code> and <code>ConnectionParametersFactoryInterface</code> , as aliases. An application replaces a factory by pointing the alias to its own service
<code>Doctrine\ORM\EntityManagerInterface</code>	Created by the factory of the entity manager, from the parameters <code>doctrine.dbal</code> and <code>doctrine.orm</code>
<code>DoctrineRepositoryFactory</code>	To register in the repository manager, see repositories
The provider of the translations of the package	Tagged <code>derafu_translation.resource_provider</code>

Import it from the file of services of the application and set the parameters:

```
imports:
  - { resource: '../vendor/derafu/orm/resources/config/orm-services.yaml' }

parameters:
  doctrine.dbal:
    url: '%env(DATABASE_URL)%'
  doctrine.orm:
    paths: ['%kernel.project_dir%/src/Entity']
    naming_strategy: Doctrine\ORM\Mapping\UnderscoreNamingStrategy
```

Their defaults are a SQLite file in `var/app.db` and the entities in `src/Entity`. An application can ignore the file and define its own services, with its own configuration or with Doctrine directly.

Last updated on 26/10/2026