

Getting started with Doctrine

Doctrine Installation SVN

Download any version of Doctrine from the SVN server `http://svn.doctrine-project.org`

Check out a specific version:

```
svn co http://svn.doctrine-project.org/branches/1.2 .
```

To update Doctrine execute the following command from your terminal:

```
svn update
```

SVN Externals

Navigating to your checked out project:

```
cd /var/www/my_project
```

Setup Doctrine as an SVN external:

```
svn propedit svn:externals lib/vendor
```

The above command will open editor. Place the following text inside and save:

```
doctrine http://svn.doctrine-project.org/branches/1.2/lib
```

Install Doctrine by doing an svn update:

```
svn update
```

It will download and install Doctrine at the following path: `/var/www/my_project/lib/vendor/doctrine`

Doctrine Installation with PEAR package

Install Doctrine with the following command:

```
pear install pear.doctrine-project.org/Doctrine-1.2.x
```

For more details [check the documentation](#).

Doctrine implementation

Include Doctrine libraries

Find the `Doctrine.php` file containing the core class in the lib folder where you downloaded Doctrine.

Move doctrine libraries to a folder in your project:

```
mkdir lib mkdir lib/vendor mkdir lib/vendor/doctrine mv /path/to/doctrine/lib doctrine
```

Or you can use externals:

```
svn co http://svn.doctrine-project.org/branches/1.2/lib lib/vendor/doctrine
```

Add it to your svn externals:

```
svn propedit svn:externals lib/vendor
```

This will open up your editor. Place the following inside and save:

```
doctrine http://svn.doctrine-project.org/branches/1.2/lib
```

When you do SVN update you will get the Doctrine libraries updated:

```
svn update lib/vendor
```

Require Doctrine Base Class

Create a file named `bootstrap.php` and place the following code in the file:

```
// bootstrap.php /* Bootstrap Doctrine.php, register autoloader specify configuration attributes and load models. */ require_once(dirname(__FILE__) . '/lib/vendor/doctrine/Doctrine.php');
```

Register Autoloader

Register the class `:php:class:`Doctrine`` autoloader function in the bootstrap file:

```
// bootstrap.php spl_autoload_register(array('Doctrine', 'autoload'));
```

Create the singleton `:php:class:`Doctrine_Manager`` instance and assign it to a variable named `$manager` :

```
// bootstrap.php $manager = Doctrine_Manager::getInstance();
```

For more details [check the documentation](#).

Doctrine basic use

Generating Doctrine schema files from database

Modify `bootstrap.php` to use the MySQL database:

```
// bootstrap.php $conn = Doctrine_Manager::connection('mysql://root:mys3cr3et@localhost/doctrinetest',  
'doctrine');
```

You can use the `:php:meth:`$conn->createDatabase`` method to create the database if it does not already exist.

Storage file for created models:

```
mkdir doctrine_example/models
```

Generate models using PHP script:

```
// test.php Doctrine_Core::generateModelsFromDb( 'models', array('doctrine'), array('generateTableClasses'  
=> true) );
```

You can place custom functions inside the classes to customize the functionality of your models. Below are some examples:

```
// models/User.php class User extends BaseUser { public function setPassword($password) { return $this-  
>_set('password', md5($password)); } }
```

In order for the above password accessor overriding to work properly you must enable the `auto_accessor_override` attribute in your `bootstrap.php` file like done below:

```
// bootstrap.php $manager->setAttribute(Doctrine_Core::ATTR_AUTO_ACCESSOR_OVERRIDE, true);
```

Now when you try and set a users password it will be md5 encrypted.

Autoload models from directory

```
// bootstrap.php Doctrine_Core::loadModels('models');
```

Generate Schema Files from Models

Use PHP script:

```
// example.php Doctrine_Core::generateYamlFromModels('schema.yml', 'models');
```

Execute the example.php script:

```
php example.php
```

File named schema.yml is created in the root of the project directory.

For more details [check the documentation](#).

Creating database tables from Doctrine schema file

To re-generate everything and make sure the database is reinstantiated each time prepare script:

```
// generate.php require_once('bootstrap.php'); Doctrine_Core::dropDatabases();  
Doctrine_Core::createDatabases(); Doctrine_Core::generateModelsFromYaml('schema.yml', 'models');  
Doctrine_Core::createTablesFromModels('models');
```

Regenerate the database from the schemafiles running the command:

```
php generate.php
```

For more details [check the documentation](#).

Using Doctrine with Symfony 2

Installation

For installation with Symfony 2.1 and newer select one of the following methods:

Dependencies and bin/vendors

Add the following snippets to “deps” files:

```
[doctrine-mongodb] git=http://github.com/doctrine/dbal.git [doctrine-mongodb-odm]  
git=http://github.com/doctrine/doctrine2.git [DoctrineBundle]  
git=http://github.com/doctrine/DoctrineBundle.git target=/bundles/Doctrine/Bundle/DoctrineBundle
```

Composer

Add the following dependencies to your projects composer.json file:

```
"require": { # .. "doctrine/doctrine-bundle": "~1.2" # .. }
```

Basic use of Doctrine with Symfony 2

Writing Data Fixtures

The times of using YAML for data fixtures is no longer. Instead, you are only required to use plain PHP for loading your data fixtures.

```
// data/fixtures/fixtures.php $em = $this->getEntityManager(); $admin = new \Models\User(); $admin->username = 'admin'; $admin->password = 'changeme';
```

Building Doctrine

Now you're ready to build everything. The following command will build models, forms, filters, database and load data fixtures.

```
$ php symfony doctrine:build --all --and-load
```

Updating Schema

If you change your schema mapping information and want to update the database you can easily do so by running the following command after changing your mapping information.

```
$ php symfony doctrine:build --all-classes --and-update-schema
```

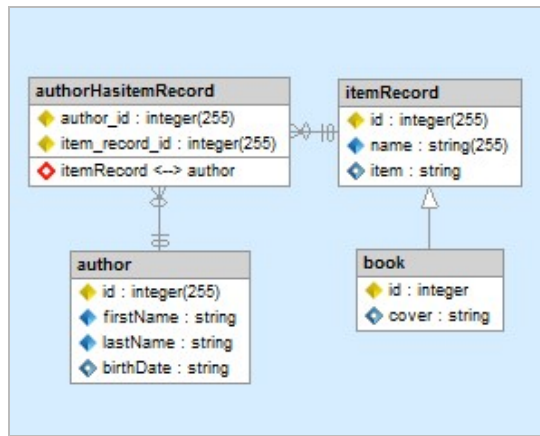
Doctrine Model Elements

Schema file structure

Doctrine2 uses one `*.dcm.xml` schema file for each entity. The file is structured like this:

YML

```
book: columns: id: cover: inheritance: extends: itemRecord itemRecord: columns: id: name: item: relations:
Authors: author: columns: id: firstName: lastName: birthDate: relations: ItemRecords: authorHasitemRecord:
columns: author_id: item_record_id:
```



Generated by [Skipper](#)

You can go and see how to [do this in few clicks](#).

Entity

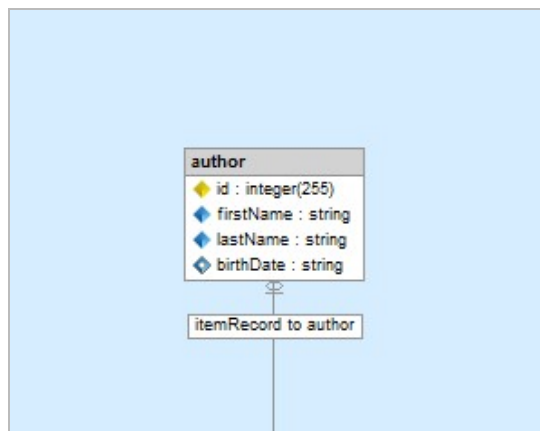
Simple entity

Simple entity with a primary key and several fields:

YML

```

author: columns: id: unique: true primary: true type: integer(255) notnull: true firstName: type: string
notnull: true lastName: type: string notnull: true birthDate: type: string relations: ItemRecords:
  
```



Generated by [Skipper](#)

Entity with all options defined

Entity with all options defined:

YML

```
itemRecord: options: charset: utf8 type: INNODB collate: utf8_unicode_ci tableName: item_record attributes:
export: none columns: id: unique: true primary: true type: integer(255) publisherId: type: integer(255)
eanId: unique: true type: integer name: unique: true type: string(255) item: default: item_new type:
string(255) indexes: ix_name_ean: fields: [name, eanId] type: unique ix_name: fields: [name] relations:
publisher: ean: authors:
```



Generated by [Skipper](#)

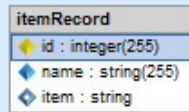
You can go and see how to [do this in few clicks](#).

Id

Primary key definition:

YML

```
id: unique: true primary: true type: integer(255) notnull: true autoincrement: true
```

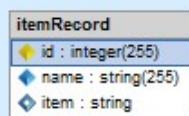


Generated by [Skipper](#)

Primary key with all base properties set:

YML

```
columns: id: unique: true primary: true type: integer(255) notnull: true autoincrement: true collation: utf8_unicode_ci nospace: true notblank: true unsigned: true
```



Generated by [Skipper](#)

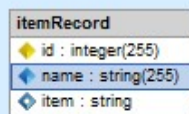
You can go and see how to [do this in few clicks](#).

Field

Regular field definition:

YML

```
name: unique: true type: string(255) notnull: true
```

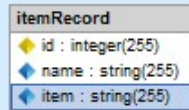


Generated by [Skipper](#)

Regular field with all options set:

YML

```
columns: name: unique: true type: string(255) notnull: true notblank: true fixed: true collation: utf8_unicode_ci
```



Generated by [Skipper](#)

Field validators

List of all validators:

YML

```
column: item: notnull: true fixed: true notblank: true country: false creditcard: false date: false email:
false future: false htmlcolor: false ip: false minlength: 1 nospace: true past: false readonly: false
regexp: item_* range: 0,999 scale: 2 unsigned: true usstate: false
```

You can go and see how to [do this in few clicks](#).

Index

Indexes

Non-unique index:

YML

```
author: indexes: ix_name_last: fields: [lastName]
```

author	
id	integer(255)
firstName	string
lastName	string
birthDate	string
awards	string

Generated by [Skipper](#)

Unique index definition:

YML

```
author: indexes: ix_first_name_last_name_date: fields: [firstName, lastName, birthDate] type: unique
```

author	
id	integer(255)
firstName	string
lastName	string
birthDate	string
awards	string

Generated by [Skipper](#)

You can go and see how to [do this in few clicks](#).

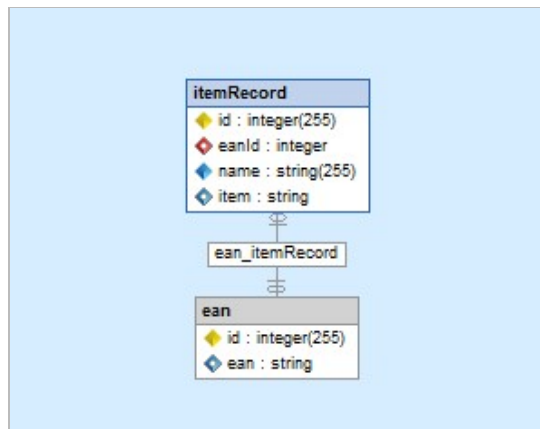
Association

One to One

One to one owner side:

YML

```
itemRecord: columns: id: unique: true primary: true eanId: unique: true type: integer ean: class: ean
foreignAlias: itemRecord local: eanId foreign: id
```

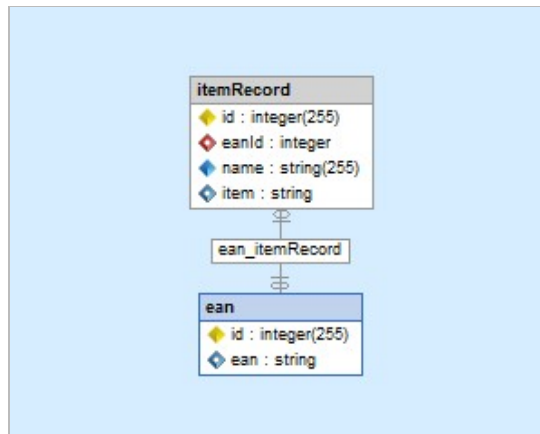


Generated by [Skipper](#)

One to one inverse side:

YML

```
ean: columns: id: unique: true primary: true unique: true type: string
```



Generated by [Skipper](#)

Many to One

Many to one owner side:

YML

```

itemRecord: columns: id: unique: true primary: true publisherId: type: integer(255) notnull: true
relations: publisher: class: publisher foreignAlias: itemRecord local: publisherId foreign: id
  
```



Generated by [Skipper](#)

Many to one inverse side:

YML

```
publisher: columns: id: unique: true primary: true
```



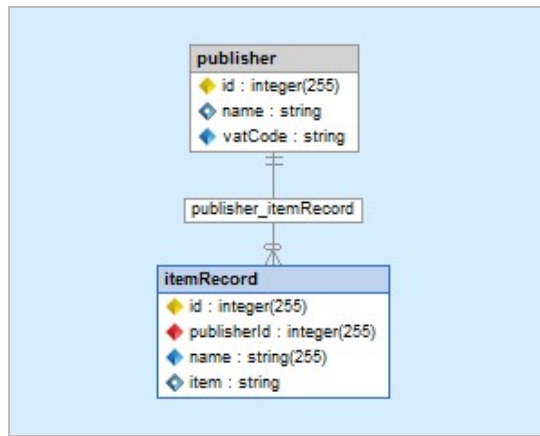
Generated by [Skipper](#)

Association with all options enabled

Many to one owner side with all properties:

YML

```
relations: publisher: class: publisher foreignAlias: itemRecord onDelete: SET NULL onUpdate: CASCADE
cascade: [(NULL)] type: one foreignType: many local: publisherId foreign: id
```



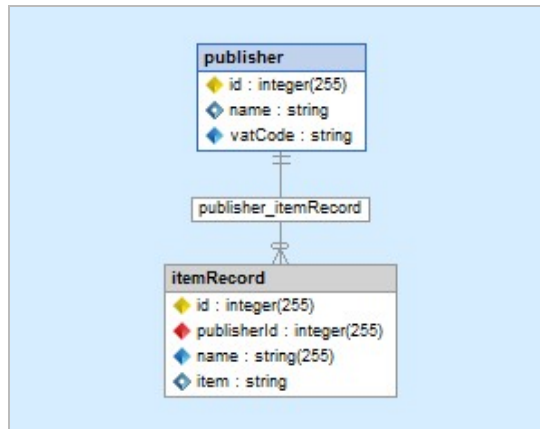
Generated by [Skipper](#)

Many to one inverse side with all properties:

YML

```

relations: itemRecord: class: itemRecord foreignAlias: publisher onDelete: SET NULL onUpdate: CASCADE
cascade: [(NULL)] type: many foreignType: one local: id foreign: publisherId
  
```



Generated by [Skipper](#)

Many to One using non-PK foreign key

Many to one owner side using non-PK foreign key:

YML

```

itemRecord: columns: id: unique: true primary: true type: integer(255) publisher_name: type: string
publisher_vat_code: type: string relations: Publisher: class: publisher foreignAlias: ItemRecord local:
  
```

publisher_name foreign: name



Generated by [Skipper](#)

Many to one inverse side using non-PK foreign key:

YML

```
publisher: columns: id: unique: true primary: true type: integer(255) notnull: true autoincrement: true
name: type: string vatCode: type: string notnull: true
```




Generated by [Skipper](#)

You can go and see how to [do this in few clicks](#).

MN Association

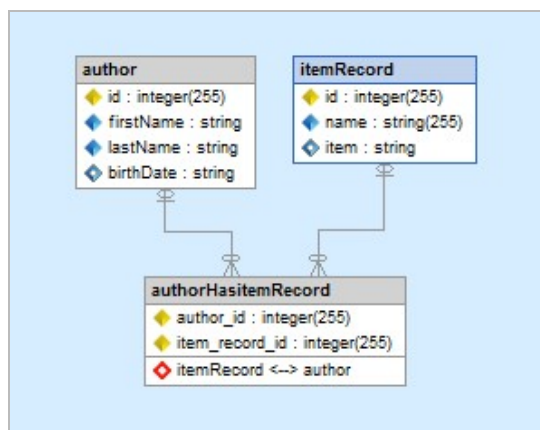
Many to Many

Many to many owner side:

YML

```

itemRecord: columns: id: unique: true primary: true relations: Authors: class: author refClass:
authorHasitemRecord local: item_record_id foreign: author_id
  
```

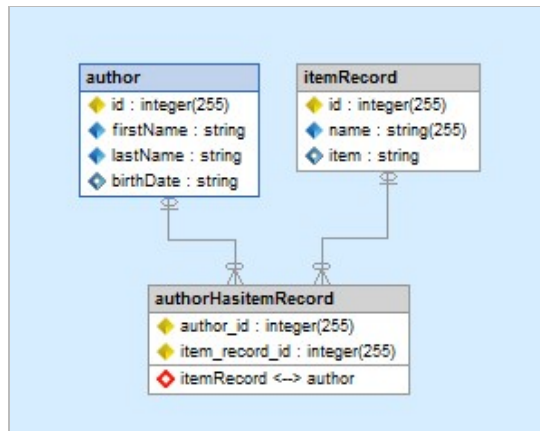


Generated by [Skipper](#)

Many to many inverse side:

YML

```
author: columns: id: unique: true primary: true relations: ItemRecords: class: itemRecord refClass:
authorHasitemRecord local: author_id foreign: item_record_id
```



Generated by [Skipper](#)

Many to many with all options enabled:

YML

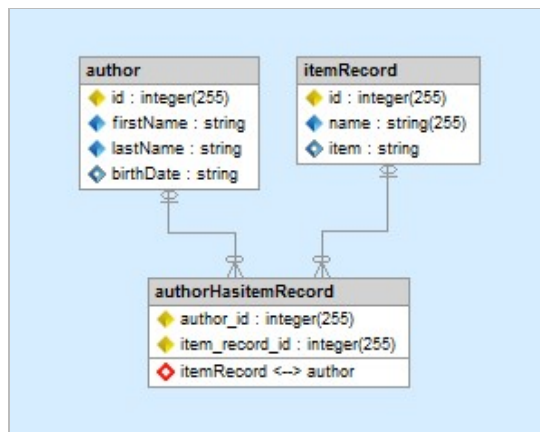
```
authorHasitemRecord: columns: author_id: primary: true type: integer(255) notnull: true item_record_id:
primary: true type: integer(255) notnull: true relations: author: class: author onDelete: SET NULL
onUpdate: CASCADE foreignAlias: itemRecord local: author_id foreign: id
```

MN Entity

Many-to-many entity.

YML

```
authorHasitemRecord: columns: author_id: primary: true type: integer(255) notnull: true item_record_id:
primary: true type: integer(255) notnull: true relations: author: class: author foreignAlias: itemRecord
local: author_id foreign: id
```



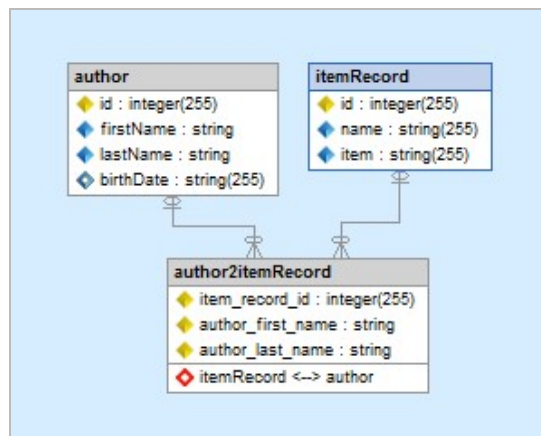
Generated by [Skipper](#)

Many to Many using non-PK foreign key

Many to many owner side using non-PK foreign keys:

YML

```
itemRecord: columns: id: unique: true primary: true publisher_name: type: string publisher_vat_code: type:
string name: unique: true type: string(255) item: default: item_new type: string(255) relations: Author:
class: author refClass: author2itemRecord local: item_record_id foreign: author_first_name
```

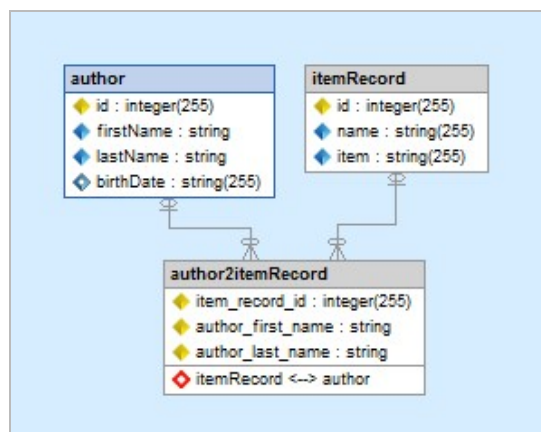


Generated by [Skipper](#)

Many to many inverse side using non-PK foreign key:

YML

```
author: columns: id: unique: true primary: true firstName: type: string notnull: true lastName: type:
string notnull: true relations: ItemRecord: class: itemRecord refClass: author2itemRecord local:
author_first_name foreign: item_record_id
```

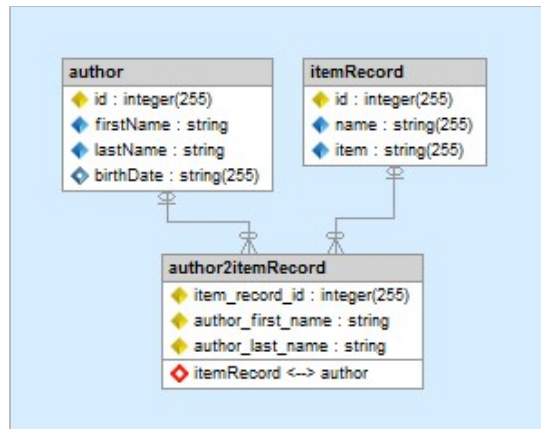
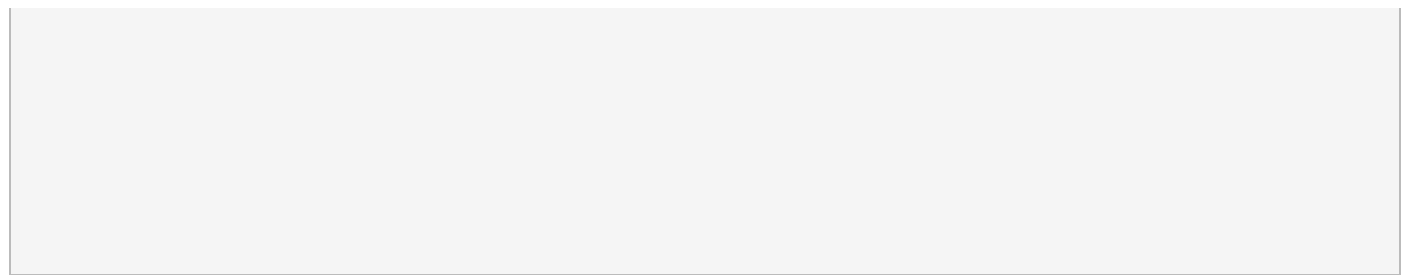


Generated by [Skipper](#)

Many to many entity using non-PK foreign keys:

YML

```
author2itemRecord: columns: item_record_id: primary: true type: integer(255) notnull: true
author_first_name: primary: true type: string notnull: true author_last_name: primary: true type: string
notnull: true
```



Generated by [Skipper](#)

You can go and see how to [do this in few clicks](#).

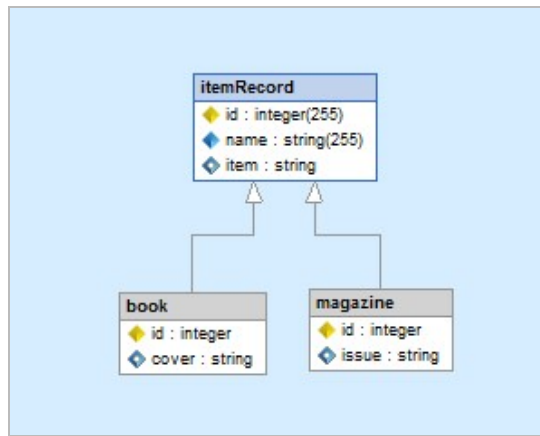
Inheritance

Simple inheritance

Simple table inheritance parent:

YML

```
itemRecord: columns: id: unique: true primary: true
```

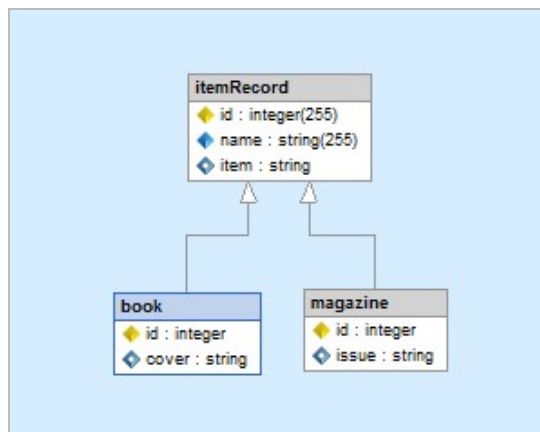


Generated by [Skipper](#)

Simple table inheritance child:

YML

```
book: columns: id: unique: true primary: true inheritance: extends: itemRecord type: simple keyField: item
keyValue: book
```



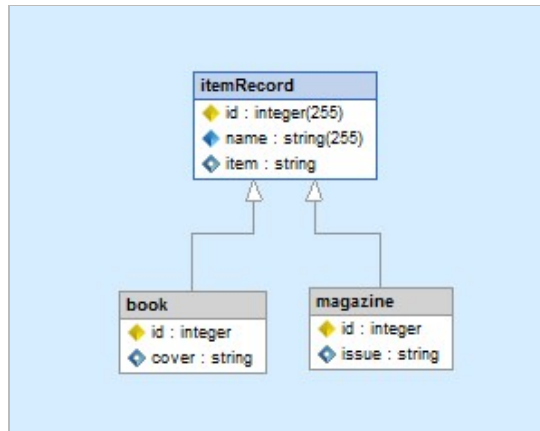
Generated by [Skipper](#)

Concrete inheritance

Concrete inheritance parent:

YML

```
itemRecord: columns: id: unique: true primary: true
```

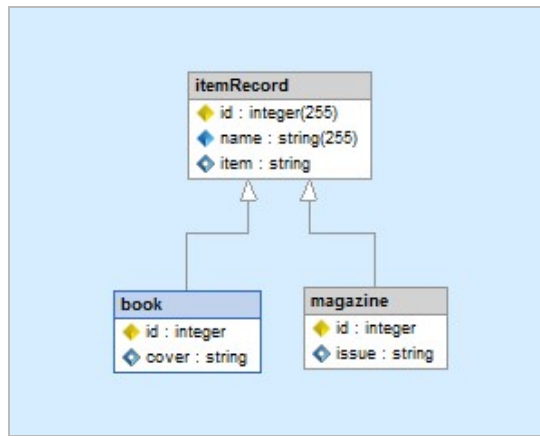


Generated by [Skipper](#)

Concrete inheritance child:

YML

```
book: columns: id: unique: true primary: true inheritance: extends: itemRecord type: concrete keyField:
item keyValue: book
```



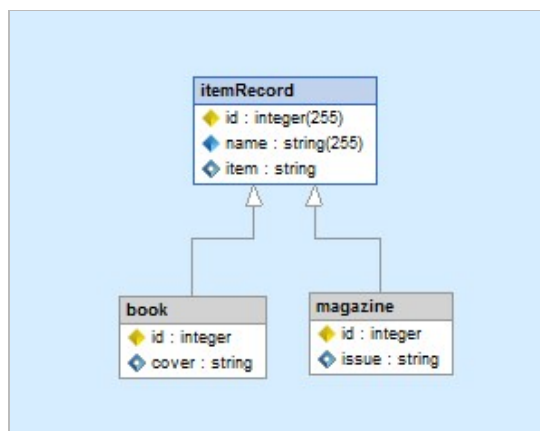
Generated by [Skipper](#)

Column aggregation inheritance

Column aggregation inheritance parent:

YML

```
itemRecord: columns: id: unique: true primary: true
```

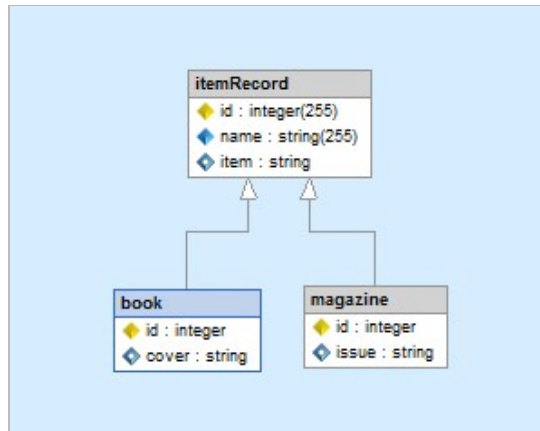


Generated by [Skipper](#)

Column aggregation inheritance child:

YML

```
book: columns: id: unique: true primary: true inheritance: extends: itemRecord type: column_aggregation
keyField: item keyValue: book
```

Generated by [Skipper](#)

You can go and see how to [do this in few clicks](#).

ActAs Behaviors

Blameable

Blameable behavior will automate the update of username or user reference fields on your Entities or Documents.

YML

```
actAs: Blameable: columns: created: disabled: false length: 8 name: created_by alias: alias type: integer
updated: alias: alias disabled: false length: 8 name: updated_by onInsert: true type: integer blameVar:
userId default: false listener: Doctrine_Template_Listener_Blameable relations: created: class: User
disabled: true foreign: id name: CreatedBy updated: class: User disabled: true foreign: id name: UpdatedBy
```

EventLoggable

Loggable behavior tracks your record changes and is able to manage versions.

YML

```
actAs: EventLoggable: logger: path: ..\logs\ type: string
```

Geographical

The Geographical behavior can be used with any data record for determining the number of miles or kilometers between 2 records.

YML

```
actAs: Geographical: latitude: name: latitude size: 255 type: float longitude: name: longitude size: 255 type: float
```

For more details [check the documentation](#).

GoogleI18n

YML

```
actAs: GoogleI18n: fields: [name] languages: [EN]
```

I18n

I18n translatable behavior offers a very handy solution for translating specific record fields in different languages.

YML

```
actAs: I18n: className: "%CLASS%Translation" generateFiles: false length: 2 pluginTable: false table: false
type: string
```

For more details [check the documentation](#).

Nested-set

Nested-set behavior will implement the tree behavior on your Entity.

YML

```
actAs: NestedSet: hasManyRoots: true levelColumnName: name rootColumnName: id
```

For more details [check the documentation](#).

Searchable

The Searchable behavior is a fulltext indexing and searching tool.



YML

```
actAs: Searchable: analyzer: Doctrine_Search_Analyzer_Standard batchUpdates: false className:
"%CLASS%Index" generateFiles: false generatePath: false pluginTable: false type: 1
```

For more details [check the documentation](#).

Sluggable

Sluggable behavior will build the slug of predefined fields on a given field which should store the slug.

YML

```
actAs: Sluggable: alias: slug builder: Doctrine_Inflector, urlize canUpdate: false indexName: sluggable
length: 255 name: slug type: string unique: true uniqueIndex: true
```

For more details [check the documentation](#).

Sortable

Sortable behavior will maintain a position field for ordering entities.

YML

```
actAs: Sortable: manyListsColumn: [name, publisherId]
```

Softdelete

SoftDeleteable behavior allows to “soft delete” objects, filtering them at SELECT time by marking them as with a timestamp, but not explicitly removing them from the database.

YML

```
actAs: SoftDelete: length: 255 name: deleted_at options: notnull: false type: timestamp
```

For more details [check the documentation](#).

Timestampable

Timestampable behavior will automate the update of date fields on your Entities or Documents.

YML

```
actAs: Timestampable: created: alias: createdAt disabled: false expression: false format: Y-m-d H:i:s name:
created_at type: timestamp updated: alias: updatedAt disabled: false expression: false name: updated_at
format: Y-m-d H:i:s onInsert: true type: timestamp
```

For more details [check the documentation](#).

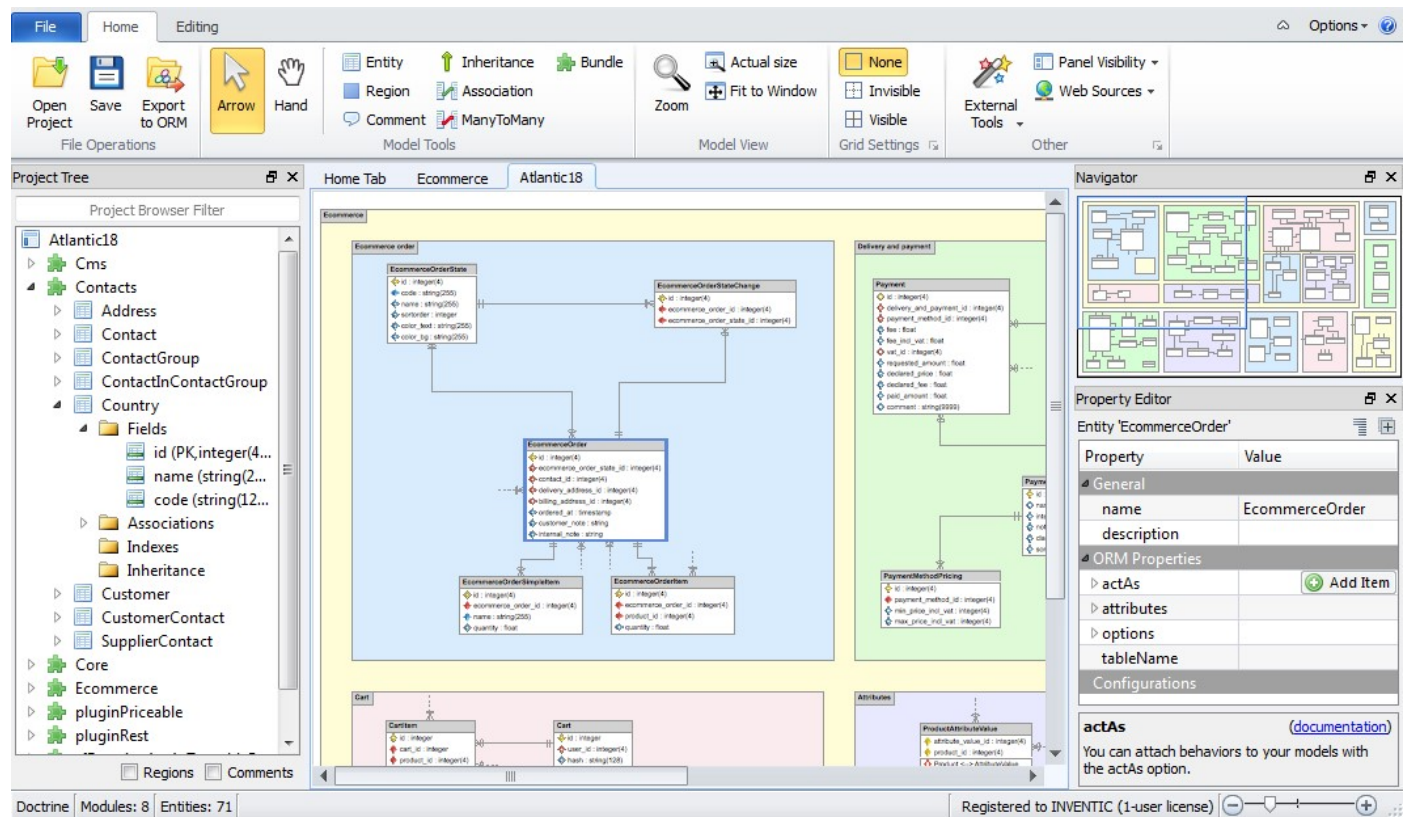
Taggable

YML

```
actAs: Taggable: className: "%CLASS%TaggableTag" generateFiles: false pluginTable: false table: false
tagAlias: Tags tagClass: Taggabletag tagField: name
```

Skipper

[Visit website](#)



Definitions on this page were modelled and generated by Skipper, visual schema editor for ORM frameworks.

Skipper greatly simplifies work with Doctrine and saves huge amount of time. Every example entity and relation used in this Cheatsheet can be achieved with just a few clicks with Skipper. Generated code is clean and elegant and complies with all coding standards.

To learn how Skipper works visit the [product tour](#).

Export to Doctrine definitions

Entity name Description

Entity location

Field Properties | Indexes | Associations | Inheritance | ☐ Show property editor

Field Name	Type	Size	PK	FK	AI	NN	UQ	Default	Description	Enum
id	integer	255	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1		1, 2, 3, 4, 7
publisherId	integer	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
eanId	integer		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
name	string	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
item	string		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			

Entity editor

Skipper allows to model and export the definition for every Doctrine element and its properties. Further advantages of automated export are:

- Editing and generating of definitions is fully repeatable.
- Standardized definitions are immediately ready-to-use.
- All typos and syntax errors are 100% eliminated.

Useful links:

[Project export](#) - more info about Skipper definitions export

[Export to Doctrine](#) - how to export your changes to definition schema files

Import of a project

Status of Exported Files

File Name	Status
entities\Doctrine.Tests.Models.ECommerce.ECommerceCart.dcm.xml	New File
entities\Doctrine.Tests.Models.ECommerce.ECommerceCategory.dcm.xml	New File
entities\Doctrine.Tests.Models.ECommerce.ECommerceFeature.dcm.xml	Updated
entities\Doctrine.Tests.Models.ECommerce.ECommerceProduct.dcm.xml	Updated
entities\Doctrine.Tests.Models.ECommerce.AffiliateCustomer.dcm.xml	No Changes
entities\Doctrine.Tests.Models.ECommerce.ECommerceCustomer.dcm.xml	No Changes
entities\Doctrine.Tests.Models.ECommerce.ECommerceShipping.dcm.xml	No Changes
entities\Doctrine.Tests.Models.ECommerce.SystemCustomer.dcm.xml	No Changes

Project Root

Export dialog

Any existing Doctrine project can be simply and quickly imported to Skipper. This enables:

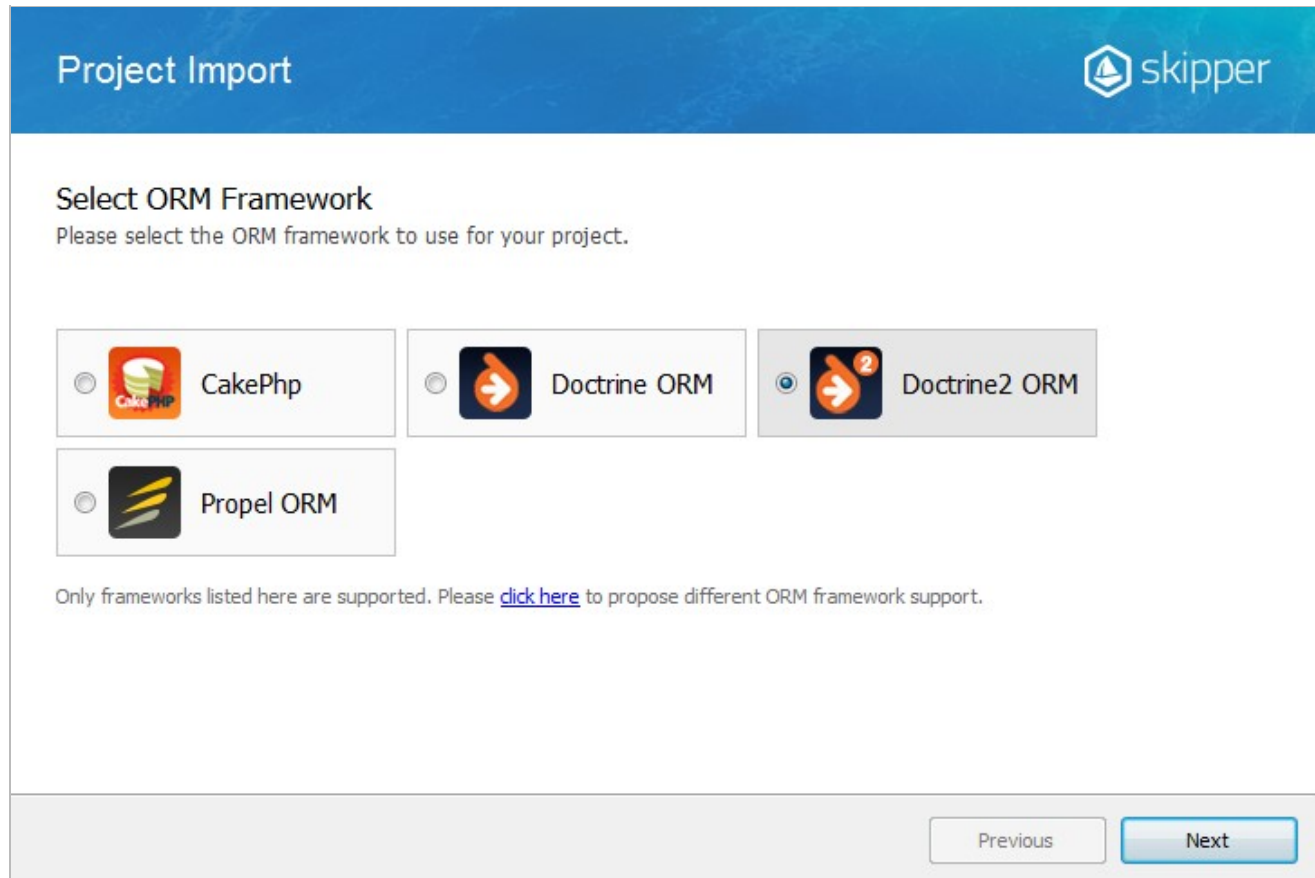
- To visualize logic of any project.
- To start to use application in any phase of the project.

Useful links:

[Project import](#) - general information about import feature

[Doctrine project import](#) - how to import existing project to Skipper

Summary of Skipper benefits



Project Import

Select ORM Framework
Please select the ORM framework to use for your project.

☐ CakePHP

☐ Doctrine ORM

☒ Doctrine2 ORM

☐ Propel ORM

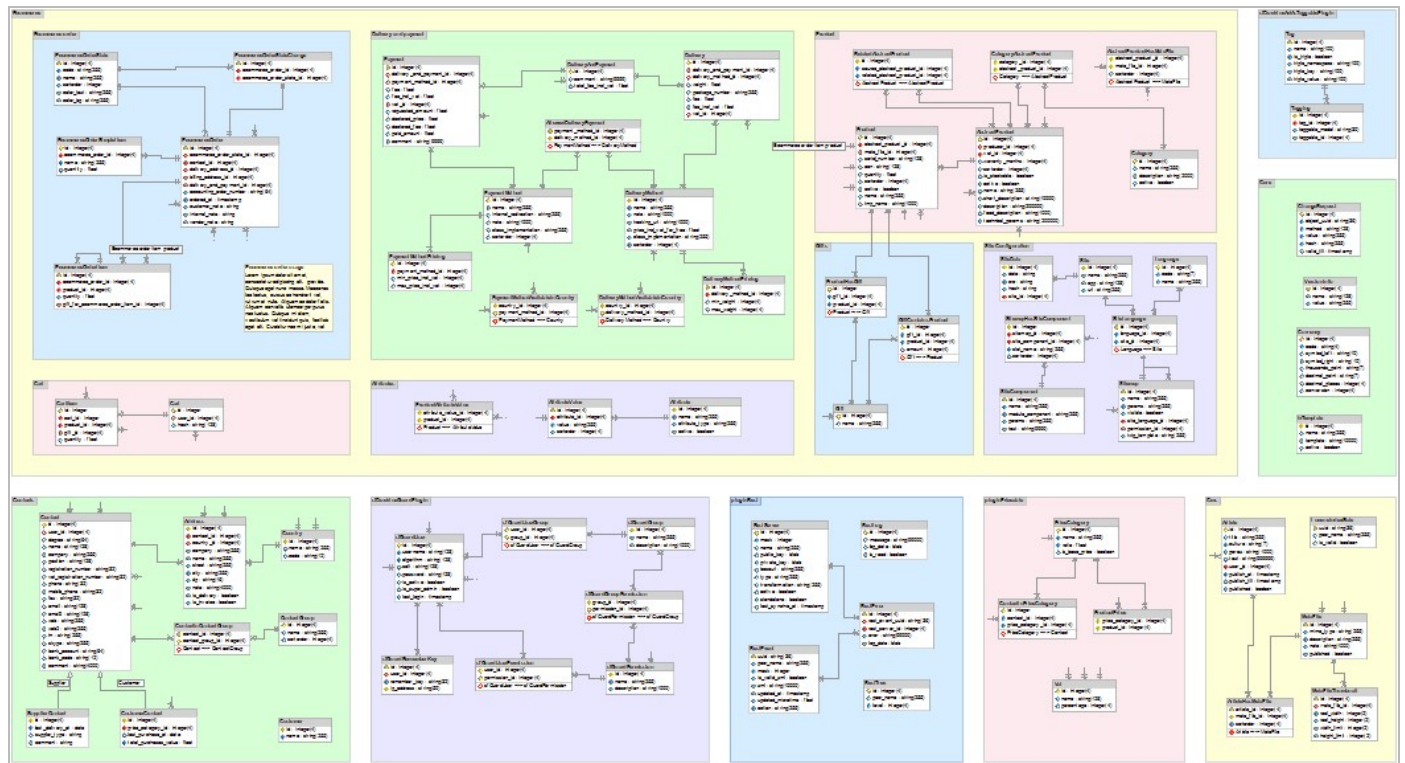
Only frameworks listed here are supported. Please [click here](#) to propose different ORM framework support.

Previous Next

Import dialog

- Allows to create and maintain the project four times faster.
- Replaces manual definitions writing and reduces errors.
- Displays the model schema in a form of interactive enhanced ER diagram.
- Emphasizes the creative part of a project and eliminates the stereotype.
- Increases work comfort.
- Provides quality project documentation.
- Reduces requirements on knowledge and experience of programmers.
- Simplifies the cooperation between team members.

Skipper download



Atlantic model

You can try Skipper during its 14-day evaluation period. Trial version offers full application functionality without any limitation and no credit card is needed.

Download trial version from the tool websites at www.skipper18.com/download.



See also

- [Symfony website](#)
- [Zend framework website](#)
- [Doctrine project](#)
- [Doctrine documentation](#)
- [Symfony Cheatsheet](#)

Do you know any other helpful or interesting sites that should be linked here?

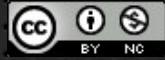
Let us know: developers@ormcheatsheet.com

Found a typo? Something is wrong or missing in the Cheatsheet? Just [fork and edit it!](#)

ORM Cheat Sheet

Created by [Inventic](#) developed by community

If you want to leave feedback, contact us developers@ormcheatsheet.com



This work is licensed under a
[Creative Commons Attribution-NonCommercial 3.0 Unported License](https://creativecommons.org/licenses/by-nc/3.0/)