ORM and active record: eZPersistentObject

The Exponential answer to Eloquent, Doctrine, the Django ORM and ActiveRecord for your own tables - eZPersistentObject with a definition() array, fetchObject, fetchObjectList, count, store and remove, function...

Source file: `doc/eng-us/help/web-framework-developers/orm-and-ezpersistentobject.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/web-framework-developers/orm-and-ezpersistentobject.md

This page is for developers who model data with Eloquent, Doctrine, the Django ORM or Rails' ActiveRecord. Exponential has two answers. For data editors create and manage, it has the **content model** (classes, objects, versions), which you configure rather than code (The content model for ORM users). For your application's own tables it has `eZPersistentObject` (`kernel/classes/ezpersistentobject.php`), a small active record base class that the kernel itself uses for its tables (`eZSection`, `eZRole` and many more).

Contents

1. Content class or own table?
2. A model
3. Reading and writing
4. Relations
5. Transactions
6. Compared with full ORMs

Content class or own table?

From Persistent objects and custom tables:

Choose a content class when	Choose a table when
editors create and edit it in the administration	only code writes it
it needs versions, translations, permissions per item	none of that is needed
it appears as pages	it is internal data
volumes are moderate	there are millions of rows or many writes per second

A model

PHP

```php
<?php
class expDemoItem extends eZPersistentObject
{
    public static function definition()
```

```
    {
        return array(
            'fields' => array(
                'id'      => array( 'name' => 'ID', 'datatype' => 'integer', 'default' => 0, 'req
uired' => true ),
                'user_id' => array( 'name' => 'UserID', 'datatype' => 'integer', 'default' => 0,
'required' => true ),
                'title'   => array( 'name' => 'Title', 'datatype' => 'string', 'default' => '', '
required' => true ),
                'created' => array( 'name' => 'Created', 'datatype' => 'integer', 'default' => 0,
 'required' => true ) ),
            'keys' => array( 'id' ),
            'increment_key' => 'id',
            'function_attributes' => array( 'user' => 'user' ),
            'class_name' => 'expDemoItem',
            'sort' => array( 'created' => 'desc' ),
            'name' => 'expdemo_item' );
    }

    public function user()
    {
        return eZUser::fetch( $this->attribute( 'user_id' ) );
    }
}
```

The `definition()` array is the mapping (what Doctrine puts in attributes or XML, and Eloquent infers from the table): fields with types and defaults, the primary key, the auto-increment key, the table name, and **function attributes**, computed properties that templates can read as `$item.user.login`.

Reading and writing

PHP

```
$item  = eZPersistentObject::fetchObject( expDemoItem::definition(), null, array( 'id' => (int)$i
d ) );
$items = eZPersistentObject::fetchObjectList( expDemoItem::definition(), null,
                                              array( 'user_id' => (int)$userID ),
                                              array( 'created' => 'desc' ),
                                              array( 'offset' => 0, 'length' => 20 ) );
$count = eZPersistentObject::count( expDemoItem::definition(), array( 'user_id' => (int)$userID )
 );

$item = new expDemoItem( array( 'user_id' => $userID, 'title' => $title, 'created' => time() ) );
$item->store();                    // insert; the id is set from increment_key
$item->setAttribute( 'title', $newTitle );
$item->store();                    // update
$item->remove();                   // delete
```

Conditions are arrays, never SQL strings, and values are escaped for you. Most projects add static helpers such as `expDemoItem::fetch( $id )` and `expDemoItem::fetchByUser( $userID )`, the way the kernel's classes do.

Relations

There are no declared relations (`hasMany`, `ManyToOne`, `ForeignKey`) and no lazy-loading proxies. Write the method yourself, as `user()` above does, and expose it through `function_attributes` when templates need it. For many-to-many data, model the link table as

its own persistent object. Content has its own relations (object relations and the tree), read through the content API.

Transactions

PHP

```php
$db = eZDB::instance();
$db->begin();
// several store() and remove() calls
$db->commit();
```

Nested `begin()` calls are counted; only the outermost `commit()` commits (Database access and SQL). Never hold a transaction open around a slow remote call.

Compared with full ORMs

Feature	Eloquent / Doctrine / Django ORM / ActiveRecord	eZPersistentObject
Mapping	conventions, attributes, metadata	the `definition()` array
Query building	fluent query builders, DQL, querysets	condition arrays, sorts and limits; SQL through eZDB for the rest (Queries)
Relations, eager loading	declared	methods you write
Unit of work, identity map	Doctrine	no: each `store()` writes at once
Migrations from the model	Doctrine, Django, Rails	no: `.dba` schema files (Migrations)
Database portability	yes	yes: SQLite, PostgreSQL, MySQL/MariaDB, Oracle, and MongoDB through the kernel's driver
Template access	via view models	built in: fields and function attributes are readable in templates

You can still use a Composer library in your extension (Composer's autoloader is loaded by `autoload.php`), but a second ORM means a second connection, and content must still be written through the content API so that versions, caches, permissions and search stay consistent.

Related pages

- Persistent objects and custom tables
- Active record pattern
- Databases and SQL standard
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

Become an Exponential Partner

Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.

Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.