Lesson 9.3: Persistent objects

Learners create a table for their extension, describe it with an eZPersistentObject definition, and store, fetch, count, change and remove rows without writing SQL for each operation. 60 minutes.

Source file:
`doc/eng-us/help/trainers/developers/module-09-the-console-and-persistent-objects/persistent-objects.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/trainers/developers/module-09-the-console-and-persistent-objects/persistent-objects.md

Extensions that keep their own data, such as a log of greetings, subscriptions or settings per user, need a table. `eZPersistentObject` maps a table to a PHP class from one description, the definition, and then stores, fetches, counts and removes rows for you, escaping every value and working on every database Exponential supports. In this lesson learners create a table and a class for it.

> **Note**
> The exercise creates a table. Run it only on your development installation.

At a glance

Course	Developers
Module	Module 9: The console and persistent objects
Duration	60 minutes (20 presenting, 40 exercise)
Level	Advanced
Audience	Developers
Prerequisites	Lesson 9.2; a development installation on SQLite (the `exp:install` default)

Learning objectives

After this lesson, learners can:

- write a `definition()` with `fields`, `keys`, `increment_key`, `class_name`, `sort` and `name`;
- create new rows with `'id' => null` and `store()`;
- fetch one row with `fetchObject()`, many with `fetchObjectList()`, and count with `count()`;
- change a row with `setAttribute()` and `store()`, and delete it with `remove()`;
- explain why persistent objects are preferred over hand-written SQL.

Trainer's notes

How to present it

Sources: `kernel/classes/ezpersistentobject.php` (the definition is documented above `definition()`), Database and SQL standards and kernel classes that use it, such as `kernel/classes/ezrole.php`.

1. **The definition** (6 minutes). Walk through `mysiteGreeting::definition()`: each field has a `name` (the PHP attribute name), a `datatype` (`integer`, `string`, `float`), a `default` and `required`; `keys` is the primary key; `increment_key` is filled by the database on insert; `class_name` is the class rows become; `name` is the table.
2. **New rows** (3 minutes). Create with `'id' => null`, so the database gives the next ID; after `store()`, `attribute( 'id' )` holds it. Kernel classes such as `eZRole` do the same.
3. **Fetching** (4 minutes). `fetchObject( $def, null, $conditions )` returns one object or `null`; `fetchObjectList( $def, null, $conditions, $sorts, $limit )` returns an array; `count( $def, $conditions )`. Conditions are a hash of field and value; values are escaped.
4. **Why** (4 minutes). Every value is escaped and cast for you; the same code runs on MySQL, PostgreSQL, SQLite and MongoDB, where hand-written SQL would not.
5. **Shipping the table** (3 minutes). A real extension ships its schema in `share/db_schema.dba` (the kernel's format, written with `ezsqldumpschema.php` and loaded with `ezsqlinsertschema.php`) or as SQL files per engine in `sql/<engine>/`. In class, a small script creates the table on SQLite.

Common questions

Can a field name differ from the column name? The keys of `fields` are the column names; `name` is the attribute name used with `attribute()`. Most classes keep them the same.

How do I add methods such as `fetchByName`? As static methods on the class that call `fetchObjectList()` with conditions, as in the exercise.

Transactions? Wrap several writes in `$db->begin()` and `$db->commit()` (lesson 9.4).

Pitfalls

- `'id' => 0` **for a new row.** A zero is a value; give `null` so the database assigns the ID.
- **A table name without prefix.** Use your extension prefix (`mysite_`) so tables never collide with the kernel's `ez*` tables.
- **The SQL of the table.** The `CREATE TABLE` of the exercise is SQLite syntax; MySQL writes `AUTO_INCREMENT`, PostgreSQL uses `SERIAL`. A `.dba` schema avoids writing it per engine.

Hands-on exercise

On the development installation on SQLite, in `extension/mysite_hello`.

1. Create `scripts/create_table.php`, which creates the table if it does not exist:

PHP

```php
<?php
// Run once: php bin/php/ezexec.php extension/mysite_hello/scripts/create_table.php (SQLite s
yntax)
$db = eZDB::instance();
$db->query( "CREATE TABLE IF NOT EXISTS mysite_greeting (
                 id INTEGER PRIMARY KEY AUTOINCREMENT,
                 name VARCHAR(100) NOT NULL DEFAULT '',
                 created INTEGER NOT NULL DEFAULT 0 )" );
eZCLI::instance()->output( 'Table mysite_greeting is ready' );
```

2. Run it: `php bin/php/ezexec.php extension/mysite_hello/scripts/create_table.php`.
3. Create `classes/mysitegreeting.php`:

PHP

```php
<?php
/**
 * One stored greeting: a row of the table mysite_greeting.
 */
class mysiteGreeting extends eZPersistentObject
{
    public static function definition()
    {
        return array( 'fields' => array( 'id' => array( 'name' => 'ID',
                                                        'datatype' => 'integer',
                                                        'default' => 0,
                                                        'required' => true ),
                                         'name' => array( 'name' => 'Name',
                                                          'datatype' => 'string',
                                                          'default' => '',
                                                          'required' => true ),
                                         'created' => array( 'name' => 'Created',
                                                             'datatype' => 'integer',
                                                             'default' => 0,
                                                             'required' => true ) ),
                      'keys' => array( 'id' ),
                      'increment_key' => 'id',
                      'class_name' => 'mysiteGreeting',
                      'sort' => array( 'created' => 'desc' ),
                      'name' => 'mysite_greeting' );
    }

    public static function create( $name )
    {
        return new self( array( 'id' => null, 'name' => $name, 'created' => time() ) );
    }

    public static function fetch( $id )
    {
        return eZPersistentObject::fetchObject( self::definition(), null, array( 'id' => (int
)$id ) );
    }

    public static function fetchByName( $name, $limit = 10 )
    {
        return eZPersistentObject::fetchObjectList( self::definition(), null, array( 'name' =
> $name ),
                                                    null, array( 'offset' => 0, 'length' => (
int)$limit ) );
    }

    public static function countAll()
```

```
    {
        return eZPersistentObject::count( self::definition() );
    }
}
```

4. Regenerate the autoloads: `php bin/php/ezpgenerateautoloads.php -e`.
5. Create `scripts/greetings.php` and run it with `ezexec.php`:

PHP

```
<?php
$cli = eZCLI::instance();
$ada = mysiteGreeting::create( 'Ada' );
$ada->store();
$grace = mysiteGreeting::create( 'Grace' );
$grace->store();
$cli->output( 'Stored IDs ' . $ada->attribute( 'id' ) . ' and ' . $grace->attribute( 'id' ) )
;
$cli->output( 'Count: ' . mysiteGreeting::countAll() );
$cli->output( 'Named Ada: ' . count( mysiteGreeting::fetchByName( 'Ada' ) ) );
$row = mysiteGreeting::fetch( $grace->attribute( 'id' ) );
$row->setAttribute( 'name', 'Grace H.' );
$row->store();
$cli->output( 'Renamed: ' . mysiteGreeting::fetch( $row->attribute( 'id' ) )->attribute( 'nam
e' ) );
$row->remove();
$cli->output( 'Count after remove: ' . mysiteGreeting::countAll() );
```

6. Run the script a second time and compare the IDs.
7. Change `create()` to `'id' => 0`, run once, read the result, and change it back.
8. Commit the class and the scripts.

Expected result

- Step 2 prints "Table mysite_greeting is ready".
- Step 5 prints two new IDs, the count, `Named Ada: 1`, `Renamed: Grace H.` and the count after the remove.
- Step 6: new, higher IDs; the count grows by one per run (Ada stays, Grace is removed).
- Step 7: the new rows get the ID 0 instead of new IDs ("Stored IDs 0 and 0"); on SQLite the second replaces the first. The database assigns IDs only when the ID is `null`.
- The trainer asks each learner to explain `increment_key`.

Quiz

1. Which key of the definition names the table?
2. Which value must the ID of a new row have so the database assigns one? A. `0` B. `null` C. `-1` D. `''`
3. Which method returns many rows, and which one row?
4. Why are persistent objects safer than SQL strings?
5. How does an extension ship the schema of its tables?

Answers

1. `name`.
2. **B.** `null`, as kernel classes such as `eZRole` do.
3. `fetchObjectList()` for many, `fetchObject()` for one.
4. Values are escaped and cast for you, and the same code runs on every supported database.
5. In `share/db_schema.dba` (dumped with `ezsqldumpschema.php`, loaded with `ezsqlinsertschema.php`) or as SQL files per engine in `sql/<engine>/`.

Further reading

- Database and SQL standards: safe SQL, dialects, schemas.
- Database support: which databases Exponential runs on.
- SQLite as the database.

Related pages

- Module 9: The console and persistent objects
- Developers: course overview
- Trainers: start here
- 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.