Database code for PHP 8 and every engine

Bring an old extension's database code up to Exponential 6 - from mysql_* calls and MySQL-only SQL to eZDB, eZPersistentObject and .dba schema files that work on SQLite, PostgreSQL, MariaDB, MySQL, Oracle and MongoDB.

Source file: `doc/eng-us/help/extension-development/part-4-upgrading/database-code-for-php-8.md`
Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/extension-development/part-4-upgrading/database-code-for-php-8.md

Part 4, chapter 10. Old extensions were written for one database: MySQL. Exponential 6 supports several, and the default for a new installation is SQLite 3. This chapter shows what to change in an old extension's database code so it works on PHP 8 and on every engine the site may run on.

The engines an extension may meet

Database support lists the supported databases, in the project's order of preference: SQLite 3 (the default), PostgreSQL, MongoDB, MariaDB, Oracle and MySQL. Your extension does not choose; the site does. Code that works only on MySQL works on one of six.

Engine	Kernel class	What changes for raw SQL
MySQL, MariaDB	`eZMySQLiDB`	nothing, that is what old code was written for
PostgreSQL	`eZPostgreSQLDB`	no backticks, no `LIMIT a,b`, stricter types, `||` for concatenation
SQLite 3	`eZSQLite3DB`	few column types, limited `ALTER TABLE`
Oracle	provided by the ezoracle extension	30-character names in older versions, no `LIMIT`
MongoDB	`expMongoDB`	no SQL at all: `arrayQuery()` returns no rows and logs a warning

The MongoDB row is the strongest argument for the rule of this chapter: **use `eZPersistentObject` for your own tables**. The MongoDB adapter turns `eZPersistentObject` 's calls (`fetchObjectList()`, `storeObject()`, `removeObject()`) into MongoDB queries; it cannot turn your hand-written SQL into anything (MongoDB kernel support).

Step 1: no mysql_* and no mysqli_* of your own

`mysql_*()` functions were removed in PHP 7. Replacing them with `mysqli_*()` and a new connection is the wrong fix: it opens a second connection, outside the kernel's transactions, to a database that may not be MySQL. Use the kernel's connection:

PHP

```php
<?php
$db = eZDB::instance();
$rows = $db->arrayQuery( 'SELECT id, name FROM myoldext_item ORDER BY name' );
foreach ( $rows as $row )
{
    echo $row['id'], ' ', $row['name'], "\n";
}
```

Old	Exponential 6
`mysql_connect()`, `mysql_select_db()`	nothing: `eZDB::instance()` is already connected
`mysql_query( "SELECT ..." )` with a fetch loop	`$db->arrayQuery( "SELECT ..." )` returns all rows as arrays
`mysql_query( "UPDATE ..." )`	`$db->query( "UPDATE ..." )`
`mysql_real_escape_string( $s )`	`$db->escapeString( $s )`
`mysql_insert_id()`	`$db->lastSerialID( 'myoldext_item', 'id' )`
`mysql_error()`	`$db->errorMessage()`
`LIMIT 10, 20` in the SQL string	`$db->arrayQuery( $sql, array( 'offset' => 10, 'limit' => 20 ) )`

The last row matters: `arrayQuery()`'s second argument takes `offset` and `limit`, and each driver writes the right SQL for its engine.

Step 2: values in SQL

Every value you put into SQL must be made safe by type:

PHP

```php
<?php
$db = eZDB::instance();
$http = eZHTTPTool::instance();

$id = (int)$http->postVariable( 'ItemID' );
$name = $db->escapeString( (string)$http->postVariable( 'Name' ) );
$ids = array_map( 'intval', (array)$http->postVariable( 'SelectedIDs', array() ) );

$db->query( "UPDATE myoldext_item SET name = '$name' WHERE id = $id" );
if ( $ids )
{
    $rows = $db->arrayQuery( 'SELECT id, name FROM myoldext_item WHERE ' . $db->generateSQLINStat
ement( $ids, 'id', false, true, 'int' ) );
}
```

- Integers: `(int)`.
- Strings: `escapeString()`, inside single quotes.
- Lists: `generateSQLINStatement()`, which also splits very long lists the way the engine needs.
- Never a value from the request without one of these.

Note the `(string)` before `escapeString()`: `postVariable()` returns `null` for a missing variable, and passing `null` to string functions is deprecated in PHP 8.1 (PHP 8.1 to 8.5

deprecations).

Step 3: portable SQL

If you keep raw SQL, keep it to what every SQL engine understands:

Avoid	Use
Backticks around names	plain lower-case names without quotes
`LIMIT a, b`	the `offset` and `limit` parameters of `arrayQuery()`
`CONCAT( a, b )`	`$db->concatString( array( 'a', 'b' ) )`
`SUBSTRING()` variants	`$db->subString( $string, $from, $len )`
`a & b` written by hand	`$db->bitAnd( 'a', 'b' )`
`INSERT ... ON DUPLICATE KEY UPDATE`, `REPLACE INTO`	fetch, then insert or update, inside a transaction
`NOW()`, `UNIX_TIMESTAMP()`	a PHP `time()` value as an integer
`AUTO_INCREMENT` in your own `CREATE TABLE`	a `.dba` schema file (step 5)

Step 4: transactions

Group writes that belong together, so a failure leaves nothing half-done:

PHP

```php
<?php
$db = eZDB::instance();
$db->begin();
$db->query( "DELETE FROM myoldext_item_tag WHERE item_id = $id" );
foreach ( $tagIDs as $tagID )
{
    $db->query( 'INSERT INTO myoldext_item_tag ( item_id, tag_id ) VALUES ( ' . (int)$id . ', ' .
 (int)$tagID . ' )' );
}
$db->commit();
```

The kernel counts nested `begin()` calls, so your transaction may run inside one the kernel opened (during a publish, for example) and commits only at the outermost level. se7enxweb/expchangeclass added transactional rollback when it was forked from eZChangeclass, "so a failed conversion leaves content untouched" (its initial import, commit 7c3f164).

Step 5: tables, schemas and .dba files

Old extensions ship `sql/mysql/schema.sql` and expect the administrator to run it. Exponential 6 extensions describe their tables once, in `share/db_schema.dba`, and the kernel's schema tools write the SQL for each engine. The whole procedure is in Database schema and .dba files.

If you keep `sql/<engine>/` files too, give every engine its own: `sql/mysql/`, `sql/postgresql/`, `sql/sqlite/`.

Step 6: eZPersistentObject

For an extension's own tables, a class that extends `eZPersistentObject` replaces most raw SQL, and works on MongoDB as well:

PHP

```php
<?php
class myOldExtItem 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' => '', 'requ
ired' => true ),
            ),
            'keys' => array( 'id' ),
            'increment_key' => 'id',
            'class_name' => 'myOldExtItem',
            'sort' => array( 'name' => 'asc' ),
            'name' => 'myoldext_item',
        );
    }

    public static function fetchList( $offset = 0, $limit = 50 )
    {
        return eZPersistentObject::fetchObjectList( self::definition(), null, null, null,
                                                    array( 'offset' => $offset, 'limit' => $limit
 ) );
    }
}
```

`store()`, `remove()` and `fetchObject()` come from the base class. The full chapter is Persistent objects and your own tables.

PHP 8 details in database code

Symptom	Cause	Fix
`count(): Argument #1 must be of type Countable` after a query	`arrayQuery()` returned `false` on an error	`$rows = $db->arrayQuery( ... ); if ( !is_array( $rows ) ) ...`
Deprecated: passing `null` to `strlen()` / `trim()`	a NULL column value	`(string)$row['name']`, or `?? ''`
`Unsupported operand types: string + int`	arithmetic on a column that came back as a string `''`	cast first: `(int)$row['count'] + 1`
A query that worked on MySQL fails on PostgreSQL	MySQL accepted `'5'` for an integer column or a missing `GROUP BY` column	write the SQL strictly; test on SQLite and PostgreSQL

Testing on more than one engine

A development installation on SQLite takes minutes to set up and is the default, so test there first: what works on SQLite and MySQL usually works everywhere else except MongoDB. If your extension's users may run MongoDB, test there as well, or say clearly in your README that the extension needs an SQL database.

Related pages

- Previous: Dynamic properties
- Next: Modernizing ezinfo.php and extension.xml
- Database support
- Persistent objects and your own tables
- Database schema and .dba files
- 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.