9. Databases

Source file: `doc/eng-us/install/09-databases.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/install/09-databases.md

Exponential stores its content repository in one of six database systems on five drivers: SQLite 3, PostgreSQL, MongoDB, MariaDB, Oracle through the `ezoracle` extension, and MySQL (MariaDB and MySQL share one driver). This chapter explains each of them as an operator meets it: what the driver does, how to create the database and its user, the exact settings block, what goes wrong and how to recognise it. It spends most of its pages on SQLite, because SQLite is the installer's default and, with the driver Exponential ships, a full production database that carries heavy read load and queues concurrent writes instead of failing them. The chapter ends with a comparison table and the notes that matter when code or content moves from one engine to another.

> **Warning**
>
> `.htaccess` **slows down every request.** When `.htaccess` files are enabled (`AllowOverride` anything but `None`), let alone used, Apache looks for them and reads and evaluates them on every request, in every directory of the path, and page loading times go up drastically (measured). The only real fix is to switch the feature off for the virtual host: `AllowOverride None` in the HTTPS virtual host, and the rules in the virtual host's own configuration as mod_rewrite rules. That is far faster in production speed tests and well worth the time it takes to optimize a production configuration. Switch over right away: copy the rules of the example HTTPS virtual host into your virtual host (with the static cache, also `staticcache_offload.conf`), set `AllowOverride None`, run `apachectl configtest`, reload Apache and test the site. The same rules for other servers: nginx, LiteSpeed, Caddy, IIS, Velocity (built in); where virtual host rules go in a control panel: Plesk, cPanel, DirectAdmin, ISPConfig. **On LiteSpeed or OpenLiteSpeed,** the server's own page cache can also answer anonymous visitors and crawlers without PHP: Page caching with LiteSpeed and OpenLiteSpeed.

> **Supported databases (Exponential 6.0.x), in order of preference:** 1. SQLite 3 (the default) · 2. PostgreSQL · 3. MongoDB · 4. MariaDB · 5. Oracle · 6. MySQL. Microsoft SQL Server is known to be possible with additional work at this time. Details: Database support.

The sections below keep the order of the drivers (MariaDB and MySQL share section 9.3); the support page above gives the versions checked against the code, the test coverage of each database and what Microsoft SQL Server would need.

Contents · Previous: 8. Serving the site · Next: 10. After installing

Contents

- 9.1 How Exponential chooses a database driver
- 9.2 SQLite 3: a production database in one file
  - What the driver does at every connection
  - WAL: readers and the writer side by side

    - Queued writes: BEGIN IMMEDIATE and the writer lock
    - The settings block
    - What carries the read load: the caching layers
    - Scaling with Velocity's persistent workers
    - How it behaves under load
    - Files, ownership and mixed users
    - Backup and restore
    - ANALYZE, VACUUM and the WAL file
    - Monitoring
    - Limits, and when to choose another engine
- 9.3 MySQL and MariaDB
- 9.4 PostgreSQL
- 9.5 MongoDB
- 9.6 Oracle 19c and later (ezoracle)
- 9.7 Comparison
- 9.8 Cross-engine notes
- 9.9 Switching an existing site to another engine
- References

9.1 How Exponential chooses a database driver

Every database setting lives in `settings/site.ini`, block `[DatabaseSettings]`. You never edit that file; you override it in `settings/override/site.ini.append.php` (for the whole installation) or in `settings/siteaccess/<name>/site.ini.append.php` (for one siteaccess). The installers write the override for you.

Why the override and not `settings/site.ini` itself: an upgrade replaces the shipped file with its new version, and every change made in it is lost without a warning. The override files are yours; nothing that ships with Exponential writes to them except the installers.

The key `DatabaseImplementation` names the driver by an alias. `eZDB::instance()` (`lib/ezdb/classes/ezdb.php`) reads the alias, looks it up in `ImplementationAlias[]` and instantiates the class it names. An alias that maps to nothing does not stop the request at once: `eZDB::instance()` returns a null database (`eZNullDB`) that carries the message "No database handler was found for '...'", and writes "Database implementation not supported: ..." to `var/log/error.log`. Every page that needs content then fails. A typing error in `DatabaseImplementation` therefore looks like a database that is down; read `error.log` first.

Alias in `DatabaseImplementation`	**Class**	**File**
`sqlite3`	`eZSQLite3DB`	`lib/ezdb/classes/ezsqlite3db.php`
`ezmysqli`, `mysqli`, `ezmysql`, `mysql`	`eZMySQLiDB`	`lib/ezdb/classes/ezmysqlidb.php`

Alias in DatabaseImplementation	Class	File
ezpostgresql, postgresql, pgsql	eZPostgreSQLDB	lib/ezdb/classes/ezpostgresqldb.php
mongodb	expMongoDB	lib/ezdb/classes/expmongodb.php
ezoracle, oracle	eZOracleDB	in the ezoracle extension, registered by its own site.ini.append.php

The aliases are the ones in `settings/site.ini` (`ImplementationAlias[...]`); the Oracle aliases are added by `extension/ezoracle/settings/site.ini.append.php` when the extension is active.

The other keys of the block apply to the server engines; SQLite ignores the server, the user and the password:

Key	Default	Used by	Meaning
Server	localhost	MySQL, PostgreSQL, MongoDB	Host name or address of the database server.
Port	empty	MySQL, PostgreSQL, MongoDB	Empty means the driver's default port.
Socket	disabled	MySQL	A Unix socket path instead of host and port.
User, Password	root, empty	all but SQLite	The login. Never leave the shipped root in production.
Database	nextgen	all	The database name; for SQLite the file; for Oracle the connect string.
Charset	utf-8	MySQL, Oracle	The connection character set (see the engine sections).
ConnectRetries	0	MySQL, PostgreSQL, SQLite	How often to try again when the first connection fails. MySQL and PostgreSQL wait between the attempts; SQLite tries again at once. Oracle has its own RetryCount (section 9.6) and falls back to this value.
UsePersistentConnection	disabled	MySQL, PostgreSQL, Oracle	Reuse a connection across requests of the same PHP process.
UseSlaveServer, SlaveServerArray[] ...	disabled	MySQL	Send reads to replicas.
Transactions	enabled	all	Run grouped writes in transactions. Leave it on.
SQLOutput, SlowQueriesOutput	disabled, 0	all	Show statements in the debug output; with a value above 0 only the slower ones (milliseconds).
QueryAnalysisOutput	disabled	MySQL	With SQLOutput=enabled, MySQL's EXPLAIN of each statement in the debug output.
DebugTransactions	disabled	all	Record a stack trace for every begin and commit, to find an unbalanced transaction.

Key	Default	Used by	Meaning
`SQLitePragmas[]`, `SQLiteTransactionWait`	empty, 25	SQLite	The connection's PRAGMAs and the wait for the write lock; see the SQLite settings block.

Read the effective value of any key, with every override applied:

Shell

```
php bin/php/console exp:ini get site.ini/DatabaseSettings/DatabaseImplementation --allow-root-use
r
```

The console first prints a line naming the script it runs (`running exp:ini → bin/php/ini.php`, on the error stream), then the value alone, for example `sqlite3`. A value you expected to have changed but that still shows the old one means the override went into a file or a block that is not read; `exp:ini where` (see exp:ini) shows which file each value comes from.

After you change a database setting, clear the INI cache so every process reads it:

Shell

```
php bin/php/ezcache.php --clear-tag=ini --allow-root-user
```

Under Velocity the workers read the settings when they start; restart them as well (`php bin/php/console exp:velocity restart --allow-root-user`, see chapter 10).

9.2 SQLite 3: a production database in one file

SQLite is not a toy database that Exponential happens to start on. The SQLite project itself describes it as a database for websites with up to a few hundred thousand requests a day and more (see Appropriate uses for SQLite), and the Exponential driver adds what a web application needs on top of the library: write-ahead logging, connection settings chosen for a web server, and a write queue that makes concurrent publishing wait its turn instead of failing. Most requests of a running site do not touch the database at all, because three cache layers answer them first. This section explains each part, so you can judge for yourself where the limits lie.

The installer's default (`php bin/php/console exp:install --apply`, the setup wizard's preselected choice, `settings/setup.ini [DatabaseSettings] DefaultType=sqlite3`) is SQLite. Requirements:

- The PHP extension `sqlite3` (`php -m | grep -i sqlite3` prints `sqlite3`).
- A SQLite library of version **3.33 or later** in PHP (`php --ri sqlite3` prints `SQLite Library => ...`). The setup wizard accepts 3.0.1 (`kernel/setup/ezsetupcommon.php`), but the driver turns MySQL's `UPDATE ... JOIN`, which the content engine uses to rebuild language masks, into `UPDATE ... FROM`, which SQLite supports from 3.33 on (`eZSQLite3DB::rewriteJoinedUpdate()`).
- A writable directory for the database: by default `var/storage/sqlite3/`.
- The `sqlite3` command-line program is optional but makes backups, `ANALYZE` and checks simple (`sqlite3 --version`).

What the driver does at every connection

The driver class is `eZSQLite3DB` in `lib/ezdb/classes/ezsqlite3db.php`. When PHP lacks the `sqlite3` extension it stops before anything else and logs "SQLite3 extension was not found, the

DB handler will not be initialized."; under PHP-FPM, remember that the web server's PHP and the command line's PHP are configured separately, so check both (`php -m` and a `phpinfo()` page, or Setup > System information). Otherwise, opening a connection does this, in order:

1. **Resolve the file.** The `Database` value is a file name. A name without a slash is placed in `var/storage/sqlite3/` (the constant `eZSQLite3DB::STORAGE_DIRECTORY`); a name that starts with `/` is an absolute path and is used as it is; `:memory:` is a throw-away in-memory database. A missing directory is created with mode `0775`; a missing file is created by SQLite itself, empty. A file that cannot be opened (after `ConnectRetries` further attempts) raises `eZDBNoConnectionException` with the full path in its message, which the setup wizard and `index.php` report as a database that cannot be reached. Because a missing file is simply created, a wrong `Database` value does not fail: it opens a new, empty database, and the site then reports missing tables. Compare the path in the error with `ls var/storage/sqlite3/` when that happens.
2. **Register functions** SQLite lacks: `md5()`, and the aggregates `BIT_OR` and `BIT_AND`, which the content engine uses to fold language masks.
3. **Apply the PRAGMAs** (`applyPragmas()`), the busy timeout first, so that every following statement already waits for a lock instead of failing at once.
4. **Switch to WAL** (`useWAL()`), but only when the file is not yet in WAL mode: the mode is stored in the file, so it is set once, not by every connection.

The PRAGMA defaults, and why each was chosen:

PRAGMA	Driver default	SQLite's own default	Why
`synchronous`	`NORMAL`	`FULL`	With WAL, `NORMAL` is still safe against corruption on a crash; only the last commits before a power loss can be lost. `FULL` syncs on every commit.
`cache_size`	`-65536` (64 MB)	about 2 MB	The page cache of each connection. Negative numbers are kibibytes.
`mmap_size`	`268435456` (256 MB)	`0`	Reads go through memory-mapped I/O instead of a `read()` call and a copy per page. The mapped pages are the operating system's file cache, shared by all processes.
`temp_store`	`MEMORY`	file	Sorts and temporary tables stay in memory.
`busy_timeout`	`5000` ms	`0`	A statement that meets a lock waits up to 5 seconds. With 0 a request that met another's write failed at once.

You can change or add PRAGMAs with `SQLitePragmas[]`, one `name=value` per line (next section). Two rules the driver enforces: the name may contain only letters and underscores, and the value only letters, digits, `_` and `-`; anything else is skipped. And WAL always wins: a `journal_mode` entry is applied first and then switched back to WAL by `useWAL()`.

Remember that `cache_size` is per connection. Each PHP-FPM worker and each Velocity worker that has opened the database can grow a page cache of up to 64 MB. Size memory for that, or lower it (`SQLitePragmas[]=cache_size=-16384` is 16 MB).

WAL: readers and the writer side by side

In WAL mode (Write-Ahead Logging) a write does not change the database file directly. It appends the changed pages to `<database>-wal`; readers see a consistent snapshot made of

the database file and the part of the WAL that was committed when their read began. The consequences for a web site:

- **Readers never block the writer, and the writer never blocks readers.** Page views continue while an editor publishes.
- **There is still only one writer at a time.** That is a property of SQLite, not of the driver. The driver's job is to make the second writer wait well (next part).
- **A checkpoint** copies the WAL back into the database file. SQLite runs it automatically when the WAL reaches about 1000 pages (`wal_autocheckpoint`); a checkpoint cannot finish past a page that a running reader still needs, so under constant reading the WAL file can grow for a while (see ANALYZE, VACUUM and the WAL file).
- **Two extra files** sit next to the database: `<database>-wal` and the shared-memory index `<database>-shm`. Both belong to the database; never delete them while any process has it open.
- **All processes must be on one machine.** WAL uses shared memory, so it does not work across a network file system. One server (any number of processes) is fine; several web servers sharing one database file over NFS are not.

Queued writes: BEGIN IMMEDIATE and the writer lock

This is the part that makes SQLite suitable for an editorial site. It was introduced in 6.0.15 (1 and 2 October 2026); read SQLite: transactions queue for the write lock for the upgrade notes.

The problem it solves. A plain `BEGIN` in SQLite is *deferred*: the transaction reads first and asks for the write lock only at its first write. If another connection committed in between, the snapshot the transaction read is stale, and SQLite answers `SQLITE_BUSY` immediately, without waiting, because waiting cannot make the snapshot current. A publish failed with "database is locked" however long the busy timeout was.

What the driver does instead. `eZSQLite3DB::beginQuery()` starts every transaction like this:

1. It opens (once per connection, and again in every forked process) a lock file next to the database: `<database file>.writer-lock`, for example `var/storage/sqlite3/exponential.db.writer-lock`.
2. It takes an exclusive `flock()` on that file. If another writer holds it, it tries again every 1 to 15 ms, thinning out as the wait grows; a writer that has waited more than a second tries more often than those that came later, which serves the writers roughly in the order they arrived.
3. With the lock in hand it runs `BEGIN IMMEDIATE`, which takes SQLite's write lock at the start of the transaction, where the busy timeout applies. Writers that do not use the queue (a single statement outside a transaction, another program) are waited for by SQLite's busy handler within the time that is left.
4. At `COMMIT` or `ROLLBACK` the lock file is released. If the process dies, the operating system releases it, so a crash can never block the queue.

Why waiting at the start is safe. Before `BEGIN IMMEDIATE` has succeeded, the transaction has written nothing. Once it has succeeded, nothing inside it can lose a lock race, and WAL makes the commit all or nothing. The start is therefore the only place a transaction can fail for a lock, and the one place where waiting costs nothing but time.

How long it waits. Up to `[DatabaseSettings] SQLiteTransactionWait` seconds (default `25`; values below 1 are raised to 1). Keep it below your web server's request timeout, so a waiting request fails with a clear message rather than being killed. The default is chosen for Velocity: Velocity replaces a worker whose request runs longer than its engine setting `requestTimeout` (`Q.webserver.requestTimeout`), which is **30 seconds** unless you change it (the client gets a 504; see the worker pool specification). 25 seconds leaves the driver time to give up and report before that. Releases before this one shipped `60`, which let the server cut off a queued publish first; if your override still sets `60`, lower it to `25`. To wait longer, raise `requestTimeout` and `SQLiteTransactionWait` together, the wait always a few seconds below the timeout. Under PHP-FPM the limit is the pool's `request_terminate_timeout` (off unless set); PHP's `max_execution_time` does not count time spent waiting on Linux, because it measures CPU time.

What a timeout looks like. When a transaction could not start within the wait, the driver reports, in `var/log/error.log` and on the error page:

```
database is busy: the transaction could not start within 25 s, another write held the lock all th
at time; nothing was written
```

(25 is your `SQLiteTransactionWait`.) Any other failure at the start reads `the transaction could not start: <SQLite's message>; nothing was written`. In both cases nothing was written; retry the action. A rollback is sent only when SQLite really has a transaction open, so you no longer see "cannot rollback - no transaction is active" hiding the real error.

When timeouts repeat. One write held the lock for the whole wait. Look for what was running at the time in `var/log/error.log` and in the cron log: a `VACUUM`, a large import, a script that opened a transaction and then waited for something else, or a `sqlite3` shell left open inside `BEGIN`. Raising `SQLiteTransactionWait` only helps when the holder was legitimately long; it must stay below the web server's request timeout (PHP-FPM's `request_terminate_timeout`, Velocity's `requestTimeout`), or the request is killed before the driver can report.

Forked processes. `flock()` belongs to an open file, so a forked process must not reuse its parent's handle. The driver remembers the process id that opened the lock file and opens a new one in each forked process (`writerGate()`); Velocity workers and the asynchronous publisher's children therefore queue correctly against each other. The asynchronous publisher also closes its database connection before it forks for each publish.

Lock file owned by another user. The driver opens `.writer-lock` for writing and, failing that, read-only: `flock()` works on a read-only handle, so a lock file created by another user still queues correctly.

The SQLite settings block

All SQLite settings, as they belong in `settings/override/site.ini.append.php`:

INI

```ini
[DatabaseSettings]
# Use the SQLite driver.
DatabaseImplementation=sqlite3

# The database file: a plain name lives in var/storage/sqlite3/,
# a name starting with / is an absolute path.
Database=exponential.db

# Seconds a transaction waits at its start for the writers ahead of it
```

```
# (default 25). Keep it below the web server's request timeout: Velocity's
# requestTimeout is 30 s unless changed; raise both together.
SQLiteTransactionWait=25

# PRAGMAs over the driver's defaults, one per line as name=value.
# An empty list keeps the defaults (synchronous=NORMAL, cache_size=-65536,
# mmap_size=268435456, temp_store=MEMORY, busy_timeout=5000).
SQLitePragmas[]
#SQLitePragmas[]=cache_size=-131072
#SQLitePragmas[]=busy_timeout=10000
#SQLitePragmas[]=journal_size_limit=67108864

# How often to try again (at once, without a pause) when the file cannot
# be opened at first.
ConnectRetries=0
```

The setup wizard accepts a plain file name ending in `.db`, `.db3`, `.sqlite` or `.sqlite3` in `var/storage/sqlite3/`, made of letters, digits, dots, dashes and underscores and starting with a letter or digit (anything else gets "The database file name is not valid. ..."); an absolute path is possible from `kickstart.ini` and from `exp:install`. The wizard proposes `sqlite.db`; `exp:install` uses `exponential.db`. Check what your site uses:

Shell

```
php bin/php/console exp:ini get site.ini/DatabaseSettings/Database --allow-root-user
php bin/php/console exp:ini get site.ini/DatabaseSettings/SQLiteTransactionWait --allow-root-user
ls -l var/storage/sqlite3/
```

Clear the INI cache after every change (`php bin/php/ezcache.php --clear-tag=ini --allow-root-user`) and, under Velocity, restart the workers.

What carries the read load: the caching layers

A database that answers every page view would be the bottleneck of any engine. Exponential keeps most requests away from it with three layers, from the outermost:

Layer	Setting	What a hit costs	Cleared by
Velocity response cache	`settings/velocity.ini` `[CacheSettings]` `Enabled=enabled`, `DefaultTtl=30`	Answered by the Velocity parent process: no PHP worker, no database. Anonymous visitors only; signed-in visitors are never served from it.	`php bin/php/console exp:velocity cache clear --apply --allow-root-user`
Content view cache	`site.ini` `[ContentSettings]` `ViewCaching=enabled`	The rendered main area of a content page is read from a file; the page layout around it still runs.	`php bin/php/ezcache.php --clear-id=content --allow-root-user`, and automatically on publish
Template-block cache	`site.ini` `[TemplateSettings]` `TemplateCache=enabled`	{cache-block} sections of the page layout (menus, footers) are read from files.	`php bin/php/ezcache.php --clear-id=template-block --allow-root-user`

Below them, two more caches reduce what reaches SQLite:

- The **SQL query cache** (`settings/querycache.ini`, `[QueryCacheSettings] Mode=off` by

default; `request` or `shared`) answers a repeated `SELECT` until a write to one of its tables makes it stale. The SQLite driver tells it about every write (`eZDBQueryCache::noteWrite()`). See SQL query cache.

- The **operating system's file cache** and SQLite's memory map: a database of a few hundred megabytes is read from memory after the first pass.

The static cache (`[ContentSettings] StaticCache`) is **disabled** in the shipped installer and should stay off: with it on, every publish regenerated every affected page over HTTP inside the editor's request.

So the database sees the uncached page views (first views after a clear, signed-in editors, search) and the writes. That is the load the next two parts measure.

Scaling with Velocity's persistent workers

Under PHP-FPM every worker process opens its own SQLite connection when a request needs one. Under Velocity with persistent workers (`settings/velocity.ini [ServerSettings] ForkPerRequest=disabled`, the shipped value), each worker keeps the application loaded between requests; the database connection, its page cache and the compiled templates stay warm in the worker. What this means for SQLite:

- **Reads scale with the number of workers.** Each worker reads its own snapshot; WAL lets them all read while one writes. The size of the pool is `[ServerSettings] Workers` (shipped `4`), optionally dynamic with `SpareWorkers` and `IdleWorkerTimeout`.
- **Writes queue on the writer lock**, across workers, PHP-FPM, cronjobs and command-line scripts alike, because they all take the same `.writer-lock`.
- **Memory**: count up to `cache_size` (64 MB) per worker that has used the database, plus the shared memory map.
- **After a settings change** restart the workers: they read `site.ini` when they start.

Read Velocity persistent-worker server and Velocity worker pool for the pool itself.

How it behaves under load

The driver was load-tested on the development server with concurrent processes reading and writing the same database. In general terms:

- **Concurrent readers scaled** with the number of processes: many times the single-process rate at 16 to 32 readers, after which the machine's cores, not the database, set the ceiling. No reader failed.
- **Short write transactions** stayed free of errors at 16 concurrent writers, at about the rate a single writer reaches: writers wait in line, they do not fail.
- **Mixed load** (many readers and several writers at once) kept reads fast while writes queued.
- **Long transactions** (each holding the write lock for 50 to 300 ms, as a large publish does) failed with the earlier driver once the 5-second busy timeout ran out; with the writer queue all of them completed, only waiting longer. In the change's own test, 192 publishes from 64 processes at once all completed whole, where the earlier start lost 6 of them; 48 transactions of 300 ms each, started at once, were done in 14.7 seconds, about the 14.4 seconds they take one after another. The queue costs almost nothing beyond the waiting itself.

- **The WAL file grew** to well over a hundred megabytes during sustained heavy writing with readers running, and was checkpointed back afterwards (see the next parts for `journal_size_limit`).

The practical reading: SQLite handles a busy public site and a normal editorial team comfortably. Where many editors publish large changes at the same moment, each waits for the others; that is when a server database is worth its extra operation (see Limits).

Files, ownership and mixed users

A SQLite database is four files in one directory:

File	What it is	Who needs to write it
`exponential.db`	the database	every process that writes
`exponential.db-wal`	the write-ahead log	every process that writes
`exponential.db-shm`	the shared-memory index of the WAL	**every process, readers too**
`exponential.db.writer-lock`	the driver's write queue	any process (read-only is enough for `flock()`)

SQLite also creates and removes the `-wal` and `-shm` files, so the **directory** must be writable by every process. The driver creates `var/storage/sqlite3/` with mode `0775`, but it does not set the mode or owner of the files; they get the owner and umask of whichever process created them first.

On a typical server three kinds of process use the database: the web server's PHP (PHP-FPM or mod_php, as the site user), Velocity workers, and cronjobs and console commands (often another user, sometimes `root`). If one of them creates `-shm` with mode `0644`, the others fail with "attempt to write a readonly database". Set it up once:

Shell

```
# one group that every process runs with (the web server's group is usual)
chgrp -R <web group> var/storage/sqlite3
# the setgid bit: files created in the directory inherit its group
chmod 2775 var/storage/sqlite3
chmod 664 var/storage/sqlite3/*
```

and give every process a umask of `0002`, so new files are group-writable (`UMask = 0002` in a systemd unit, or `umask 002` at the top of a cron line). Check the result after a day of normal running:

Shell

```
ls -l var/storage/sqlite3/
```

Every file should be group-writable and carry the common group. Expected output looks like this (owner, group and sizes are yours):

```
-rw-rw-r-- 1 example www-data 187678720 Oct  5 10:14 exponential.db
-rw-rw-r-- 1 example www-data     32768 Oct  5 10:20 exponential.db-shm
-rw-rw-r-- 1 example www-data   4124152 Oct  5 10:20 exponential.db-wal
-rw-rw-r-- 1 example www-data         0 Oct  5 09:58 exponential.db.writer-lock
```

Do **not** reach for `bin/modfix.sh` (`bin:modfix`) here: it makes `var/storage` and other

directories world-writable (mode `777`, files `666`), and the script itself warns that this is not secure. Fix the group and mode as above instead; chapter 8 covers the users the servers run as.

Never serve the database over HTTP. It lives below `var/storage`, which the shipped `.htaccess` sends to `index.php` and Velocity's list of static files leaves out. Check on your own host; anything but `200` is right:

Shell

```
curl -sI https://www.example.com/var/storage/sqlite3/exponential.db | head -1
```

SQLite backup and restore

Never copy a database file that is in use. A copy taken while a write is in progress, or without its `-wal` file, can be inconsistent. Use SQLite's online backup API through the `sqlite3` program; it takes a consistent copy while the site keeps running and writing:

Shell

```
sqlite3 var/storage/sqlite3/exponential.db ".backup '/backup/exponential-$(date +%F).db'"
```

Alternatively, `VACUUM INTO` (SQLite 3.27 and later) writes a compacted copy in one statement (VACUUM):

Shell

```
sqlite3 var/storage/sqlite3/exponential.db "VACUUM INTO '/backup/exponential-$(date +%F).db'"
```

Check a backup before you rely on it:

Shell

```
sqlite3 /backup/exponential-2026-10-05.db "PRAGMA integrity_check"
```

It prints `ok`. A complete backup also contains the files (`var/storage`, `settings/override`, `settings/siteaccess`); chapter 10 gives the full procedure.

Restore. Processes keep the database open: PHP-FPM workers while they run a request, Velocity workers for their whole life. Restore through SQLite's own locking rather than by replacing the file under them:

Shell

```
php bin/php/maintenance.php on --message="Restoring" --allow-root-user
sqlite3 var/storage/sqlite3/exponential.db ".restore '/backup/exponential-2026-10-05.db'"
php bin/php/maintenance.php off --allow-root-user
php bin/php/ezcache.php --clear-all --allow-root-user
php bin/php/console exp:velocity cache clear --apply --allow-root-user
```

`.restore` copies the backup into the live database page by page under a write lock. It prints nothing when it succeeds. If it reports an error instead (for example because a writer held the lock, or because the backup was made with another page size, which a database in WAL mode cannot take over), the live database is unchanged; use the file replacement below. Clear the caches afterwards in any case: they still hold pages rendered from the content before the restore. If you must replace the file itself instead, stop every process that may have it open first (Velocity with `exp:velocity stop`, PHP-FPM, cron), move the old database and its `-wal` and `-shm` files aside together, put the backup in place, fix its owner and mode, and start the processes again.

ANALYZE, VACUUM and the WAL file

ANALYZE. SQLite's query planner chooses indexes better with statistics. Run `ANALYZE` once after an install or a large import, and again after the content has grown a lot:

Shell

```
sqlite3 var/storage/sqlite3/exponential.db "ANALYZE"
```

`PRAGMA optimize` runs only the analysis that is needed and is cheap enough for a weekly cron entry (ANALYZE, PRAGMA optimize).

VACUUM. Deleted content leaves free pages in the file; SQLite reuses them but does not give them back to the file system. `VACUUM` rebuilds the file compactly. It needs about twice the database's size in free disk space and holds the write lock for its whole run, so writers queue behind it and may time out after `SQLiteTransactionWait`. Run it in maintenance mode, and only when the file has become much larger than its content (after removing a large part of the site):

Shell

```
php bin/php/maintenance.php on --allow-root-user
sqlite3 var/storage/sqlite3/exponential.db "VACUUM"
php bin/php/maintenance.php off --allow-root-user
```

The WAL file. After a checkpoint SQLite reuses the WAL file from its start but does not shrink it. A burst of heavy writing can leave a large `-wal` file behind. Two remedies:

- `SQLitePragmas[]=journal_size_limit=67108864` truncates the WAL to 64 MB after each checkpoint that empties it.
- A manual checkpoint that truncates the file: `sqlite3 var/storage/sqlite3/exponential.db "PRAGMA wal_checkpoint(TRUNCATE)"`. It prints three numbers; a first number of `0` means the checkpoint was not blocked.

Monitoring a SQLite site

What	How	Healthy
Lock timeouts	`grep -c "database is busy" var/log/error.log`	`0`, or rare and explained by a known long job
Other database errors	`grep -i "eZSQLite3DB" var/log/error.log | tail`	nothing new
Integrity	`sqlite3 var/storage/sqlite3/exponential.db "PRAGMA quick_check"` (fast) or `"PRAGMA integrity_check"` (thorough)	ok
WAL size	`ls -l var/storage/sqlite3/`	the `-wal` file stops growing after busy periods; it shrinks only with `journal_size_limit` or a `TRUNCATE` checkpoint (above)
Journal mode	`sqlite3 var/storage/sqlite3/exponential.db "PRAGMA journal_mode"`	`wal`

What	How	Healthy
Disk space	`df -h var/storage/sqlite3`	room for the database twice over (for `VACUUM` and backups)
Ownership	`ls -l var/storage/sqlite3/`	all files group-writable, one common group

`exp:benchmark kernel` measures the database round trip among its probes, and `exp:benchmark` with `--save` and `--baseline` catches a slowdown between two runs (Benchmark).

Limits, and when to choose another engine

SQLite fits when the site runs on **one machine** and its write load is that of an editorial team, however many visitors read it. Choose MySQL, MariaDB or PostgreSQL when:

- **More than one machine** must write the same database. WAL needs shared memory on one host; a network file system is not a substitute.
- **Many editors publish large changes at the same moment**, all day. Each publish waits for the ones before it; if the queue is regularly longer than `SQLiteTransactionWait`, a server database with row-level locking will serve them better.
- You need **replication**, a hot standby or point-in-time recovery from the database itself. SQLite has none of its own.
- You need **database users and grants**. SQLite's access control is the file system's.

Database size is rarely the reason: SQLite handles databases far larger than an Exponential site's content (Implementation limits). Moving later is possible (see 9.9).

9.3 MySQL and MariaDB

The driver is `eZMySQLiDB` (`lib/ezdb/classes/ezmysqlidb.php`), on the PHP extension `mysqli`. MySQL and MariaDB are the classic production choice and the engine most extensions are written against.

Character set and storage engine

- **Create the database in `utf8mb4`.** The setup wizard checks the database's default character set and accepts `utf8mb4` as UTF-8 (`kernel/setup/steps/ezstep_installer.php`); the installer creates its tables without a character set of their own, so they take the database's default.
- **The connection charset.** `Charset=utf-8` (the default) is mapped to MySQL's `utf8` for the connection (`lib/ezdb/classes/ezmysqlcharset.php`); MySQL treats that name as `utf8mb3` , three bytes per character, so characters outside the Basic Multilingual Plane (emoji, some CJK extensions) cannot pass through the connection. Keep the database in `utf8mb4` anyway: it costs nothing and is ready for a connection change.
- **When the connection charset cannot be set.** The driver asks the connection which charset it really has, tries once more, and if it still differs it records the warning "Connection warning: could not set the connection charset to 'utf8'; it is '...'. Text will be transliterated to that character set on its way out of the database." The site keeps running, and text comes back with `?` in place of the characters the other charset lacks. The warning is a warning, not an error: it appears in the debug output and in `var/log/warning.log`, but only when

warnings are logged (`site.ini [DebugSettings] AlwaysLog[]=warning`; the shipped value logs errors only). Enable that while you set up a MySQL site.

- **InnoDB.** The installer creates the tables as InnoDB when the server offers it (`kernel/setup/steps/ezstep_create_sites.php`). Transactions need it. `php bin/php/ezconvertmysqltabletype.php` (`exp:ezconvertmysqltabletype`) lists the table types of an older installation with `--list` and converts them with `--newtype=InnoDB`. The MySQL update files of an upgrade start with `SET default_storage_engine=InnoDB;`, which every current MySQL and MariaDB accepts (chapter 11).

Create the database and the user

SQL

```sql
CREATE DATABASE exponential CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'exponential'@'localhost' IDENTIFIED BY '<a strong password>';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER,
      LOCK TABLES, CREATE TEMPORARY TABLES
   ON exponential.* TO 'exponential'@'localhost';
```

The user needs no global privilege, no `SUPER` and no right to list databases: the wizard works with ordinary per-database privileges. `LOCK TABLES` and `CREATE TEMPORARY TABLES` are listed because the driver's `lock()` uses `LOCK TABLES` and the kernel creates temporary tables for some operations.

Settings

INI

```ini
[DatabaseSettings]
DatabaseImplementation=ezmysqli
Server=localhost
# empty: 3306
Port=
User=exponential
Password=<the password>
Database=exponential
# a Unix socket instead of host and port, e.g. /var/lib/mysql/mysql.sock
Socket=disabled
Charset=utf-8
# enabled: mysqli persistent connections (the driver prefixes the host with p:)
UsePersistentConnection=disabled
ConnectRetries=0
```

Install with the console (the password from the environment, so it stays out of the shell history):

Shell

```
EXP_INSTALL_DB_PASSWORD='<the password>' php bin/php/console exp:install --apply --db=mysql \
    --db-host=127.0.0.1 --db-name=exponential --db-user=exponential --allow-root-user
```

`--db=mariadb` is the same. The kickstarter and the wizard take the same values.

Replicas

`UseSlaveServer=enabled` with `SlaveServerArray[]`, `SlaveServerPort[]`, `SlaverServerUser[]`, `SlaverServerPassword[]` and `SlaverServerDatabase[]` (the keys really are spelled `Slaver...`) sends reads to a replica chosen at random per request; writes go

to `Server`. Replication lag means a page may show content a few seconds old.

Gotchas

Symptom	Cause
The "database cannot be reached" page	Wrong host, user, password or database. Since PHP 8.1 `mysqli` throws; the driver turns that into `eZDBNoConnectionException`.
Text with ? where accented letters were	The connection fell back to the server's default charset. Set `AlwaysLog[]=warning`, load a page, and look for "could not set the connection charset" in `var/log/warning.log`.
The wizard refuses the database's charset	The database was created in `latin1` or another single-byte charset. Recreate it in `utf8mb4`.
Slow queries	`[DatabaseSettings] QueryAnalysisOutput=enabled` with `SQLOutput=enabled` shows MySQL's analysis in the debug output (MySQL only).

Backups: `mysqldump --single-transaction -u exponential -p exponential > backup.sql`, consistent without locking InnoDB tables (mysqldump).

9.4 PostgreSQL

The driver is `eZPostgreSQLDB` (`lib/ezdb/classes/ezpostgresqldb.php`), on the PHP extension `pgsql`. The wizard checks for PostgreSQL 8.0, but use a supported release; `pgcrypto` is a trusted extension from PostgreSQL 13 on, which the installer relies on (below).

Create the database and the user

Shell

```
sudo -u postgres createuser --pwprompt exponential
sudo -u postgres createdb --owner=exponential --encoding=UTF8 --template=template0 exponential
```

Making the user the **owner** of the database matters: from PostgreSQL 15 on, ordinary users may no longer create tables in the `public` schema of a database they do not own.

Extensions: pgcrypto

The driver computes MD5 with `encode(digest(..., 'md5'), 'hex')`, and `digest()` comes from pgcrypto. The installer runs `CREATE EXTENSION IF NOT EXISTS pgcrypto` when the server ships it; as a trusted extension the database owner may create it. If the server lacks it, install your distribution's PostgreSQL contrib package, then, as the owner or a superuser:

Shell

```
sudo -u postgres psql -d exponential -c 'CREATE EXTENSION IF NOT EXISTS pgcrypto;'
```

Sequences

Auto-increment columns are integers with a default of `nextval('<table>_<column>_seq')`; the schema handler creates each sequence before its table (`CREATE SEQUENCE IF NOT EXISTS ...`), and `lastSerialID()` reads `currval('<table>_<column>_seq')`. When data is loaded with explicit ids (an import, a restore of selected tables), the sequences stay behind and the next insert fails with a duplicate key. The schema installer corrects them after loading data (`correctSequenceValues()`); for one table by hand:

SQL

```sql
SELECT setval('ezcontentobject_id_seq', (SELECT max(id) FROM ezcontentobject));
```

Settings

INI

```ini
[DatabaseSettings]
DatabaseImplementation=ezpostgresql
Server=localhost
# 5432 is the server's default
Port=5432
User=exponential
Password=<the password>
Database=exponential
# enabled: pg_pconnect instead of pg_connect
UsePersistentConnection=disabled
```

Shell

```sh
EXP_INSTALL_DB_PASSWORD='<the password>' php bin/php/console exp:install --apply --db=pgsql \
    --db-host=127.0.0.1 --db-name=exponential --db-user=exponential --allow-root-user
```

The wizard prefills port 5432 and user `postgres` (`settings/setup.ini` `DefaultPort_pgsql`, `DefaultUser_pgsql`) and installs into the database you name; it opens `template1` only to offer a list when no name is given.

Gotchas

Symptom	Cause
"The 'digest' function is not available in your database, and Exponential cannot run without it. ..." (the setup wizard)	`pgcrypto` could not be created. Install the contrib package and create the extension as shown above, then click Next again.
Connection refused	`listen_addresses` in `postgresql.conf` or a missing line in `pg_hba.conf` (client authentication).
Duplicate key on insert after an import	Sequences behind their tables; correct them as above.
An extension's SQL fails on PostgreSQL only	MySQL syntax such as backtick quoting; see 9.8.

Backups: `pg_dump -U exponential -Fc exponential > backup.dump`, restored with `pg_restore` (pg_dump).

9.5 MongoDB

The adapter is `expMongoDB` (`lib/ezdb/classes/expmongodb.php`), on the PHP extension `mongodb` and the `MongoDB\Client` of the MongoDB PHP library. It turns the kernel's SQL-shaped calls into `find`, `aggregate`, `insert`, upsert and delete operations. MongoDB arrived with 6.0.14; read the limits before you choose it for production.

Requirements

	Minimum	Tested
PHP	8.2	8.5

	Minimum	Tested
MongoDB server	6.0 (the wizard checks 4.0)	8.3
PHP `mongodb` extension	1.18	current

Create the database user

The adapter connects with the URI `mongodb://<user>:<password>@<server>:<port>/<database>`. MongoDB uses the database in the URI's path as the authentication database when no `authSource` is given (connection strings), so create the user **in the site's database**:

Shell

```
mongosh --eval '
  db.getSiblingDB("exponential").createUser({
    user: "exponential",
    pwd: passwordPrompt(),
    roles: [ { role: "readWrite", db: "exponential" }, { role: "dbAdmin", db: "exponential" } ]
  })'
```

The database itself is created by the first write; there is no `CREATE DATABASE`.

The user is required, also on a test server that does not enforce access control: the adapter always writes `<user>:<password>@` into the URI, and with an empty `User` the MongoDB driver refuses the result (`mongodb://:@...`) with "Failed to parse MongoDB URI: ... 'default' authentication mechanism requires a username".

Settings

INI

```
[DatabaseSettings]
DatabaseImplementation=mongodb
Server=localhost
# empty: 27017
Port=27017
User=exponential
Password=<the password>
Database=exponential
```

User name and password are URL-encoded by the adapter. One client per connection URI is kept for the life of the process, so a Velocity worker reuses its connections.

Install

1. Give the setup wizard time: on MongoDB the configuration step can take one to three minutes. Raise PHP-FPM's `request_terminate_timeout` before you start, or install from the console:

   Shell

   ```
   EXP_INSTALL_DB_PASSWORD='<the password>' php bin/php/console exp:install --apply --db=mongodb
    \
       --db-host=127.0.0.1 --db-name=exponential --db-user=exponential --allow-root-user
   ```

2. Create the indexes the kernel's queries rely on; without them subtree queries and URL alias lookups scan whole collections:

Shell

```
mongosh "mongodb://exponential@localhost:27017/exponential" --file bin/mongodb/create_indexes
.js
```

3. Run the cronjobs once to build the search index (`php runcronjobs.php --allow-root-user`).

Limits

From MongoDB as the database and section 26 of the porting reference:

- The adapter never runs real SQL. Hand-written SQL sent through `arrayQuery()` that it cannot translate returns an empty result and logs a warning; search the logs for `MONGO TODO`. Extensions with their own SQL may need a MongoDB path.
- Deep URL alias regeneration after moving subtrees is incomplete; role and policy listing pages can be empty although permissions are enforced; some cronjobs are not ported; `eztags` is not usable; the shop checkout and `ezflow` scheduling are untested end to end.
- The SQL query cache does not apply to MongoDB.

Moving an existing MySQL site to MongoDB: the scripts in `bin/mongodb/` (`export_mysql.sh`, `mysql2ndjson.py`, `import_all.sh`, `validate_ndjson.sh`) and section 18 of the porting reference.

Backups: `mongodump --uri="mongodb://exponential@localhost:27017/exponential" --out=/backup/$(date +%F)` (mongodump).

9.6 Oracle 19c and later (ezoracle)

Oracle support comes from the `ezoracle` extension (version 2.3.7, the version `composer.json` requires with `~2.3.7`): the driver `eZOracleDB`, the schema handler `eZOracleSchema`, console commands and cronjobs. It is a dependency in `composer.json`, so it is in `extension/ezoracle` after `composer install`. Its own `README.md` and `INSTALL` are the complete reference; this section is the operator's path through them.

Requirements

- Exponential 6.0.15 or later (earlier versions do not offer Oracle in the installers).
- **Oracle Database 19c or later** (the wizard checks 19.0). The driver uses features of 12.1 (`STANDARD_HASH`, `OFFSET/FETCH`); 11g and older are not supported. Tested with Oracle AI Database 26ai Free (23.26).
- Database character set **AL32UTF8**. The driver connects in AL32UTF8; `NLS_LANG` is not needed.
- PHP 8.0 to 8.5 with **oci8** (PECL oci8 3.4 for PHP 8.4 and 8.5, 3.2 for 8.1 to 8.3), built against an Oracle client; the Instant Client is enough. `ORACLE_HOME` is not needed with the Instant Client; set `LD_LIBRARY_PATH` (and `TNS_ADMIN` for TNS aliases) only when the client is outside the system library path, and then in the PHP-FPM pool (`env[...]`) and in Velocity's environment as well.
- An Oracle user that may create sessions, tables, views, triggers, sequences and procedures.

Create the user

The extension prints the SQL for you to run as a DBA:

Shell

```
ORACLE_NEW_PASSWORD='<a strong password>' php extension/ezoracle/bin/php/ora-grant.php exponentia
l - USERS
```

(`-` takes the password from `ORACLE_NEW_PASSWORD`, so it does not appear in the process list; `USERS` is the tablespace.) The same is available as `php bin/php/console ext:ezoracle:ora-grant`. For the health and report commands, also grant `SELECT_CATALOG_ROLE`.

Install

Shell

```
EXP_INSTALL_DB_PASSWORD='<the password>' php bin/php/console exp:install --apply --db=oracle \
    --db-host=127.0.0.1 --db-port=1521 --db-name=FREEPDB1 --db-user=exponential --allow-root-user
```

`--db-name` is the **service name**; `host:port/service` is built from it. A full connect string or `@<tns alias>` is used as given. The installers activate the extension for the new site. For the kickstarter set `[database_choice] Type=oci8` (`oracle` is accepted as well) and the connect string as `Database` in `[database_init]` and `[site_details]`.

Settings

The connection is in `site.ini`:

INI

```
[DatabaseSettings]
DatabaseImplementation=ezoracle
User=exponential
Password=<the password>
# The connect string: Easy Connect host:port/service_name,
# Easy Connect Plus tcp://host:port/service?connect_timeout=5,
# a TNS alias, or a full (DESCRIPTION=...) descriptor.
# Server and Port are not used by this driver.
Database=127.0.0.1:1521/FREEPDB1
Charset=utf-8
# Oracle stores '' as NULL; enabled returns '' for NULL in text columns
OracleEmptyStringForNull=disabled
# enabled: ALTER SESSION SET NLS_COMP=LINGUISTIC NLS_SORT=<sort> (case-insensitive =, LIKE, ORDER
 BY)
OracleCaseInsensitive=disabled
OracleCaseInsensitiveSort=BINARY_CI

[ExtensionSettings]
ActiveExtensions[]=ezoracle
```

Everything beyond the plain connection is in `ezoracle.ini` (`extension/ezoracle/settings/ezoracle.ini`), overridden in `settings/override/ezoracle.ini.append.php`. Every feature is off by default.

Persistent connections and DRCP

A new dedicated Oracle session costs tens of milliseconds per request. Two settings remove that cost:

INI

```ini
# settings/override/ezoracle.ini.append.php
[ConnectionSettings]
# oci_pconnect: inherit (site.ini UsePersistentConnection, the shipped value),
# enabled or disabled
Persistent=enabled
# Database Resident Connection Pooling (shipped: disabled): appends :POOLED to an Easy Connect st
ring,
# SERVER=POOLED to a descriptor built from Hosts[]; for a TNS alias put
# (SERVER=POOLED) into tnsnames.ora yourself
DRCP=enabled
# oci8.connection_class: sessions of one class are shared
ConnectionClass=EXPONENTIAL
# for long-running processes (Velocity workers, cronjobs): after this many idle
# seconds a round trip checks the connection first and replaces a dead one
# (shipped: 0, never)
KeepAliveInterval=60
```

- **Persistent connections** live as long as the PHP process: a PHP-FPM worker, a Velocity worker, a CLI script. `oci8.persistent_timeout`, `oci8.max_persistent` and `oci8.ping_interval` in `php.ini` govern them; they cannot be set at run time.
- **DRCP** (Database Resident Connection Pooling) shares a pool of server processes between all those PHP processes, so hundreds of workers do not need hundreds of dedicated server processes. The pool must be started in the database, once, as a DBA: `EXECUTE DBMS_CONNECTION_POOL.START_POOL;`. PHP's side is described in the oci8 connection handling chapter.

Extension order matters for these settings. When two active extensions both provide `ezoracle.ini` settings, the one listed **earlier** in `ActiveExtensions[]` wins: `eZExtension::activateExtensions()` prepends each extension's settings directory in turn, so a later extension ends up with lower priority. `ezoracle` ships a complete `ezoracle.ini`; if you keep your connection settings in a site extension of your own instead of `settings/override/`, list that extension **before** `ezoracle`:

INI

```ini
[ExtensionSettings]
ActiveExtensions[]=mysite_settings
ActiveExtensions[]=ezoracle
```

`settings/override/` always wins over every extension. Check the effective values after a change with `php bin/php/console exp:ini get ezoracle.ini/ConnectionSettings/DRCP --allow-root-user`. The loading order can also be changed in the admin under **Setup > Extensions** (Extension loading order).

More settings

Block	Key	What it does

Block	Key	What it does
[ConnectionSettings]	ConnectString, TnsAdmin	A connect string that replaces site.ini Database (empty: use Database); the directory of tnsnames.ora for TNS aliases (the PHP-FPM pool's env[TNS_ADMIN] or Velocity's environment is the surer place, because the client reads it once per process).
[ConnectionSettings]	Hosts[], ServiceName, Failover, LoadBalance, ConnectTimeout	Build a descriptor over several listeners (when ConnectString is empty).
[ConnectionSettings]	RetryCount, RetryDelay, RetryBackoff	Connection retries with back-off (default: site.ini ConnectRetries).
[ConnectionSettings]	ReconnectErrors[], RetryReads	On a lost connection outside a transaction, reconnect and run a read again; a write is reported.
[ConnectionSettings]	CallTimeout	Milliseconds a round trip may take (client 18c or later); 0 = no limit.
[PerformanceSettings]	Prefetch, LobPrefetch, CursorSharing	Rows per round trip; LOB bytes with the row (shipped 2000, and capped there by the driver); CURSOR_SHARING (shipped FORCE).
[TraceSettings]	ClientIdentifier, ModuleName, Action, ClientInfo	What a DBA sees in V$SESSION, with patterns such as %siteaccess%, %module%/%view%.
[LogSettings]	SlowQueryThreshold, SlowQueryLog, MaskLiterals	Slow statements to var/log/oracle-slow.log, string literals masked.

Maintenance

Console commands (`php bin/php/console ext:ezoracle:<name>`): `health`, `gather-stats`, `recompile`, `purge-recyclebin`, `sequence-sync`, `schema-diff`, `indexes`, `report`, `datapump`, `ci-indexes`. Each prints one PASS/WARN/FAIL/INFO line per check. Cronjob parts `ezoracle` (nightly) and `ezoraclehealth` (hourly) run them; each job does nothing until it is enabled in `ezoracle.ini [CronjobSettings]`:

```
15 3 * * *  cd /path/to/installation && php runcronjobs.php -q -s <admin siteaccess> ezoracle
0 * * * *   cd /path/to/installation && php runcronjobs.php -q -s <admin siteaccess> ezoracleheal
th
```

Gotchas

Symptom	Cause
ORA-00933 in an extension's SQL	A table alias written with AS; Oracle allows AS only for column aliases.
An empty string comes back as NULL	Oracle stores '' as NULL. OracleEmptyStringForNull=enabled returns '' for text columns.
A login works only in one case	Oracle compares strings binary. OracleCaseInsensitive=enabled, then create linguistic indexes with ext:ezoracle:ci-indexes --create.

Symptom	Cause
CLOB values cut short	An older `ezoracle`, or a client version that cuts long CLOBs when `LobPrefetch` is above 2000. The current driver caps the prefetch at 2000 itself and writes a notice when more is set (oci8 3.4.1 with the Oracle 23.26 client cut every longer CLOB without an error). Update the extension.
Settings in your site extension are ignored	It is listed after `ezoracle` in `ActiveExtensions[]`; move it before.

9.7 Comparison

	SQLite 3	MySQL / MariaDB	PostgreSQL	MongoDB	Oracle
`DatabaseImplementation`	`sqlite3`	`ezmysqli`	`ezpostgresql`	`mongodb`	`ezoracle`
PHP extension	`sqlite3`	`mysqli`	`pgsql`	`mongodb`	`oci8`
Server to run	none	yes	yes	yes	yes
Installer default	yes				
Concurrent writers	one at a time, queued (`SQLiteTransactionWait`)	row locks (InnoDB)	row locks (MVCC)	document-level	row locks
Several application servers	no (one host)	yes	yes	yes	yes
Replicas for reads	no	`UseSlaveServer`	external	replica sets	external (Data Guard)
Persistent connections	not needed (a file)	`UsePersistentConnection`	`UsePersistentConnection`	client kept per process	`Persistent`, DRCP
SQL query cache	yes	yes	yes	no	yes
Users and grants	file system	yes	yes	yes	yes
Backup	`.backup`, `VACUUM INTO`	`mysqldump --single-transaction`	`pg_dump`	`mongodump`	`ext:ezoracle:datapump`, `expdp`
Known gaps	one host; queued writes	4-byte characters through the `utf8` connection		see limits	`''` is NULL; case-sensitive
Best for	one server, any number of readers, an editorial team	the classic production setup, most extensions	production, strict SQL	organisations standardised on MongoDB	organisations standardised on Oracle

9.8 Cross-engine notes

Exponential's own SQL runs on every engine; extension code and hand-written SQL often do not. What to know when you write or review such code (there is no separate document on cross-engine query semantics; these points are taken from the drivers):

- **Ask the driver which engine it is**: `eZDB::instance()->databaseName()` returns `mysql`, `postgresql`, `sqlite`, `mongo` or `oracle`.
- **No backticks.** MySQL and SQLite accept `` `column` ``; PostgreSQL and Oracle do not, and the MongoDB adapter's parser does not expect them. Name tables and columns plainly.
- **No `AS` before a table alias** (Oracle rejects it); `AS` before a column alias is fine everywhere.
- **Use the driver's helpers** for what differs: `subString()`, `concatString()`, `md5()`, `bitAnd()`, `bitOr()`, `escapeString()`, `lastSerialID()`. SQLite's driver also registers `BIT_OR`/`BIT_AND` aggregates and rewrites `UPDATE ... JOIN`; PostgreSQL computes MD5 through `pgcrypto`; Oracle uses `STANDARD_HASH` and `BITAND`.
- **MySQL session statements**: the SQLite driver accepts `SET FOREIGN_KEY_CHECKS`, `SET NAMES`, `SET CHARACTER SET`, `SET AUTOCOMMIT`, `SET SQL_MODE` and `SET UNIQUE_CHECKS` (also with `SESSION` or `GLOBAL`) and does nothing, because SQLite has no such switches. PostgreSQL understands `SET NAMES` (as `client_encoding`) but rejects the others; Oracle has no `SET` statement of this kind. Leave them out of code meant for every engine.
- **Empty strings**: Oracle returns NULL for `''` unless `OracleEmptyStringForNull=enabled`.
- **Case**: MySQL's default collations compare without case. PostgreSQL and Oracle compare with case. SQLite compares with case for `=` but ignores case for ASCII letters in `LIKE`. A login or identifier lookup written for MySQL may therefore find nothing elsewhere; compare `lower(...)` on both sides when case must not matter.
- **Do not infer from a result what the result cannot tell you.** A failed statement and an empty result are not reported the same way on every engine; the MongoDB adapter returns an empty result for SQL it cannot translate. To ask whether a table exists, ask the catalogue (`eZTableList()` of the driver), not a `SELECT` that may fail.
- **Index names** are database-wide on SQLite and PostgreSQL; the schema handlers create a clashing name as `<table>__<name>`.

9.9 Switching an existing site to another engine

Why switch at all: the reasons are those of Limits, and when to choose another engine. The content stays the same; only where it is stored changes. Plan the switch as a maintenance window: no editor may publish between the dump and the moment the site runs on the new database, or that work is lost.

The schema tools move a database between the SQL engines through Exponential's engine-neutral `.dba` format. `--type` takes a schema handler from `settings/dbschema.ini`: `mysql`, `postgresql`, `sqlite3` and, with the `ezoracle` extension, `oracle` (the scripts' own `--help` lists only some of them). For SQLite the database argument is the file name, as in `Database`:

Shell

```
# dump schema and data of the current database (MySQL in this example):
# ezsqldumpschema.php [OPTION]... [DATABASE] [FILENAME]
php bin/php/ezsqldumpschema.php --type=mysql --host=localhost --user=exponential --password='<pas
sword>' \
    --format=generic --output-array --output-types=all exponential var/backup/site.dba --allow-ro
ot-user
# load them into the new one:
# ezsqlinsertschema.php [OPTION]... [FILENAME] [DATABASE]
php bin/php/ezsqlinsertschema.php --type=postgresql --host=localhost --user=exponential --passwor
d='<password>' \
    --insert-types=all --schema-file=var/backup/site.dba var/backup/site.dba exponential --allow-
root-user
```

`--password=` takes its value on the command line, where other users of the machine can see it in the process list; run the transfer on a machine you control, or with a temporary password you change afterwards. Run both scripts with `--help` first: the types they accept and the options per engine (host, port, socket, table type, table charset) are listed there. Then set `DatabaseImplementation` and the connection keys in `settings/override/site.ini.append.php` to the new engine, clear all caches and restart the workers. A forgotten `DatabaseImplementation` is the usual reason no page loads after a switch. For Oracle the extension has its own transfer commands (`ext:ezoracle:mysql2oracle-schema`, `ext:ezoracle:mysql2oracle-data`); for MongoDB see 9.5.

Check the result before you open the site again. Count the rows of a few large tables on both sides (`ezcontentobject`, `ezcontentobject_tree`, `ezcontentobject_attribute`); the numbers must match. On PostgreSQL, correct the sequences if `ezsqlinsertschema.php` reported a problem with them (see 9.4). Then sign in to the administration, open a few pages and publish a test object; a publish exercises the writes that a page view does not.

Schema updates are per engine. When you later upgrade Exponential, the database update files are chosen by the engine the site runs on now, not by the one it was installed on: `update/database/mysql/`, `update/database/postgresql/` and `update/database/sqlite/`, and for Oracle the files in the `ezoracle` extension. MongoDB has no SQL update files. Chapter 11 lists them.

9.10 Microsoft SQL Server

Exponential 6.0.x has no driver for Microsoft SQL Server; it is known to be possible with additional work at this time. The database layer is the interface `eZDBInterface` with pluggable driver aliases and schema handlers, which is how the Oracle and MongoDB support were added. The work list (a driver on `sqlsrv`, the `BIT_OR` aggregate SQL Server lacks, a schema handler, SQL and update files, installer entries, tests) is in Database support.

SQLite for public demos. SQLite is the usual database of a public demo that is rebuilt from scratch every few hours: `EXP_DEMO_MODE=1 ./console exp:install --apply --db=sqlite --password=<the published password>` installs one in about a minute, with nothing to drop between builds. Demo mode itself does not depend on the database engine (Demo mode).

References

In this repository:

- Database support: the supported databases in their official order, the versions checked against the code, installer support per path and CI coverage.
- SQLite database support: the task-oriented introduction.
- Specification: the SQLite3 database driver: connection, PRAGMAs, transactions and writers, the SQL compatibility layer.
- SQLite: transactions queue for the write lock: the 6.0.15 change and upgrade notes.
- Database drivers and installers, 16 to 30 September 2026: SQLite, PostgreSQL, MySQL and Oracle changes, the setup wizard.
- SQLite for Exponential Platform and Platform SQLite installer.
- SQL query cache.
- MongoDB as the database and the MongoDB kernel support porting reference (sections 18, 24, 26).
- Installing in one command, Kickstarter.
- Velocity persistent-worker server, Velocity response cache, Velocity worker pool.
- Extension loading order.
- Operating a site, section 7 (database driver) and section 6 (backups).
- Chapter 11, the update files, for schema updates per engine.
- The code: `ezsqlite3db.php`, `ezmysqlidb.php`, `ezpostgresqldb.php`, `expmongodb.php`, `settings/site.ini` (`[DatabaseSettings]`), `settings/setup.ini`, `settings/dbschema.ini`, `bin/modfix.sh`, `bin/mongodb/create_indexes.js`.
- The Oracle extension: `extension/ezoracle/README.md` and `extension/ezoracle/INSTALL` in an installation, and se7enxweb/ezoracle.

External:

- SQLite: Write-Ahead Logging, File locking and concurrency, BEGIN TRANSACTION (DEFERRED, IMMEDIATE), PRAGMA statements, Online backup API, VACUUM, ANALYZE, Appropriate uses for SQLite, Implementation limits, the LIKE operator.
- MySQL: Unicode support (utf8mb4), GRANT, InnoDB, mysqldump; MariaDB: character sets.
- PostgreSQL: CREATE DATABASE, pgcrypto, sequences, client authentication, SET (including SET NAMES), pg_dump.
- MongoDB: PHP library, connection strings, db.createUser(), mongodump.
- Oracle: Database Resident Connection Pooling, DBMS_CONNECTION_POOL, PHP oci8 connection handling and oci8 configuration.

Contents · Previous: 8. Serving the site · Next: 10. After installing

> **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.