Specification: the SQLite3 database driver

Source file: doc/eng-us/specifications/6.0/sqlite3-database-driver.md Last changed 2026-11-05
https://doc.exponential.earth/learninminutes/exphelp/viewer/specifications/6.0/sqlite3-database-driver.md

This page is the reference for the legacy kernel's SQLite driver, `eZSQLite3DB`, and its schema handler, `eZSQLiteSchema`: how they are registered, how a connection is opened and tuned, how transactions wait for the single writer, and which SQL the driver translates. Read it if you run an installation on SQLite, tune it, or write SQL that must run on every engine. The task-oriented introduction is SQLite database support.

Settings

File	Block	Key	Default	Scope
settings/site.ini	DatabaseSettings	ImplementationAlias[sqlite3]	eZSQLite3DB	installation
settings/site.ini	DatabaseSettings	SQLitePragmas[]	none; entries name=value override the PRAGMA defaults	installation
settings/site.ini	DatabaseSettings	SQLiteTransactionWait	25 (seconds)	installation
settings/site.ini	DatabaseSettings	SQLiteCLITransactionWait	300 (seconds)	installation, command line only
settings/site.ini	DatabaseSettings	SQLiteBusyTimeout	5000 (ms)	installation, web requests
settings/site.ini	DatabaseSettings	SQLiteCLIBusyTimeout	30000 (ms)	installation, command line only
settings/dbschema.ini	SchemaSettings	SchemaPaths[sqlite], SchemaPaths[sqlite3]	lib/ezdbschema/classes/ezsqliteschema.php	installation
settings/dbschema.ini	SchemaSettings	SchemaHandlerClasses[sqlite], SchemaHandlerClasses[sqlite3]	eZSQLiteSchema	installation
settings/setup.ini	DatabaseSettings	DefaultType	sqlite3	setup wizard

Read a value with:

Shell

```
./console exp:ini get site.ini/DatabaseSettings/SQLiteTransactionWait --allow-root-user
```

Expected output: `25`, unless an override changes it. A value that is not a number falls back to 25, and anything below 1 counts as 1 (`eZSQLite3DB::transactionWaitMs()` in

`lib/ezdb/classes/ezsqlite3db.php`).

Classes and files

Class	File	Role
`eZSQLite3DB` (extends `eZDBInterface`)	`lib/ezdb/classes/ezsqlite3db.php`	Connection, queries, transactions, escaping, database listing.
`eZSQLiteSchema`	`lib/ezdbschema/classes/ezsqliteschema.php`	Reads a live schema (`fetchTableFields`, `fetchTableIndexes`, uses the SQLite PRAGMA API), generates SQL (`generateDropIndexSql`, create and alter statements).
SQL files	`kernel/sql/sqlite/schema.sql`, `kernel/sql/sqlite/cleandata.sql`	Clean installation schema and data.

Both classes are registered in the autoload arrays; the `eZSQLite3DB` entry was added to `autoload/ezp_kernel.php` in April 2026 (`cefe8f13a2`). Without that entry the alias `sqlite3=eZSQLite3DB` ended in a fatal autoload failure and a null database connection when the kernel was loaded in a bridged setup.

Setup wizard

The setup wizard's database table (`kernel/setup/ezsetupcommon.php`) lists the type `sqlite3` with driver `sqlite3`, name "SQLite", required version `3.0.1`, demo data available, Unicode supported.

Connection

- The constructor needs the PHP `sqlite3` extension. Without it the handler reports `ERROR_MISSING_EXTENSION` through the warning list and stays disconnected.
- `connect( $fileName )` resolves the file name like this:
  - `:memory:` stays an in-memory database;
  - a name beginning with `/` is an absolute path and is used as given;
  - any other name is placed in `var/storage/sqlite3/` (constant `STORAGE_DIRECTORY`); a missing directory is created recursively.
- An unreachable file throws `eZDBNoConnectionException`, as the MySQL driver does, because the setup wizard and `index.php` depend on that exception.
- At every connection the driver applies its PRAGMAs (`applyPragmas()`), then switches the journal to WAL mode (`useWAL()`).

PRAGMA defaults and overrides

PRAGMA	Default	Effect
`synchronous`	`NORMAL`	With WAL still crash-safe; `FULL` syncs on every commit.
`cache_size`	`-65536`	64 MB page cache (SQLite default is 2 MB).

PRAGMA	Default	Effect
mmap_size	268435456	Read through 256 MB of mapped memory.
temp_store	MEMORY	Sorts and temporary tables in memory.
busy_timeout	5000 in a web request, 30000 on the command line	How long a statement waits for a lock: SQLiteBusyTimeout or SQLiteCLIBusyTimeout. A busy_timeout=<ms> entry in SQLitePragmas[] still wins over both.

Override or add entries with `[DatabaseSettings] SQLitePragmas[]=name=value` in `settings/site.ini`.

Transactions and writers

SQLite has one writer. Since 6.0.15 (1-2 October 2026) a transaction starts with `BEGIN IMMEDIATE`, so the write lock is taken at the start, where the busy timeout applies. A deferred `BEGIN` made a transaction fail at its first write, with "database is locked", once another connection had committed since it read.

Aspect	Behaviour
Waiting	begin() queues behind a writer gate (a lock file named <database>.writer-lock beside the database file, released by the operating system when a process ends) for at most [DatabaseSettings] SQLiteTransactionWait seconds (default 25); tries thin out as the wait grows and the writers that waited longest go first. A waiting writer holds a shared lock on <database>.writer-queue; a writer that finds the gate free while others wait leaves it to them for eZSQLite3DB::HANDOFF_MS (30 ms), so a loop of back-to-back transactions (a cronjob part, a content job) cannot keep the gate from the writers queued behind it. Waiting here is safe because nothing has been written yet. Keep the value below the request timeout: Velocity ends a request after Q.webserver.requestTimeout, 30 s unless changed, hence the default of 25.
Command line	Cronjobs, imports and scripts wait [DatabaseSettings] SQLiteCLITransactionWait seconds (default 300) instead: nobody waits for a page there. A web request is any process that is not PHP's command line, or a Velocity worker (eZSQLite3DB::isWebProcess()). Before 5 November 2026 a cronjob gave up after 25 s behind an import's long write and stopped with "A database transaction in Exponential failed".
Single statements	A write outside a transaction (an INSERT of a link, an UPDATE of a cronjob) queues on the same writer gate, then runs (eZSQLite3DB::execWaitingForLock()). It waits at most the statement wait: SQLiteBusyTimeout ms in a web request (default 5000), SQLiteCLIBusyTimeout ms on the command line (default 30000). A SQLITE_BUSY or SQLITE_LOCKED that SQLite returns without calling its busy handler (WAL recovery, a COMMIT waiting for readers in rollback-journal mode) is tried again with a pause of 2 to 32 ms within that wait; a statement that failed for a lock has changed nothing, so the retry is safe. Reads (SELECT, a PRAGMA that only asks) never queue and are retried the same way. Before, a single write waited only in SQLite's busy handler, which sleeps up to 100 ms between tries, while the writers on the gate took the lock back within milliseconds of each commit: behind an import's loop of short transactions it failed with "database is locked", and links saved in the online editor lost their URL id.
Failure	A start that still gets nothing is reported at once as a failed transaction with the reason, and nothing is written. The message says the database was busy, how long it waited, the limit and the setting: "database is busy: the transaction could not start within 25 s (waited 25.0 s, [DatabaseSettings] SQLiteTransactionWait), another write held the lock all that time; nothing was written", for a statement "database is busy: the statement waited 5.0 s for another write to end (limit 5 s, [DatabaseSettings] SQLiteBusyTimeout) and was not run; nothing was written". Before, the generic begin() counted it as started and later statements ran one by one, each committed on its own.

Aspect	Behaviour
Rollback	ROLLBACK is sent only when SQLite has a transaction open (no more "cannot rollback - no transaction is active").
Measured	In the change's test run, 64 concurrent publishers: before 6 of 192 failed, after 192 of 192 completed whole (not repeated for this page).

The wait is set by `[DatabaseSettings] SQLiteTransactionWait` (see Settings). Upgrade notes: SQLite transactions.

Other fixes of the same days:

- `subString( s, n )` without a length returns the rest of the string, as on the other engines. A node move used to drop the last character of `path_string` and `path_identification_string` of the moved subtree.
- A node's `path_identification_string` is rewritten from the value stored for it, not from a stale copy held by the node object.
- The asynchronous publisher closes its database connection before it forks for each publish. A connection must never cross a fork.
- The setup wizard's database field offers only real SQLite files (`eZSQLite3DB::availableDatabasesIn()`).

SQL compatibility layer

Method	Behaviour
`subString`, `concatString`	Map to `substr( s, n )` (the rest of the string, counted from 1 like the other engines) and `||`.
`md5`, `md5UDF`	An `md5()` function registered with SQLite.
`bitAnd`, `bitOr`	SQL fragments for bitwise AND and OR. The aggregates `BIT_OR` and `BIT_AND` are registered as SQLite user functions at connection time, because the content engine folds language masks with `BIT_OR` and SQLite ships none.
`rewriteJoinedUpdate`	Rewrites MySQL's `UPDATE ... JOIN` into a form SQLite accepts.
`lastSerialID`	Uses the connection's last insert row id and, for tables whose primary key has more than one column, checks that the auto-increment column matches the row id.
`escapeString`	SQLite escaping; `null` becomes an empty string, as in the MySQL driver.
`createDatabase`, `removeDatabase`, `availableDatabases`	A database is a file that `connect()` creates on demand, so `createDatabase` only makes sure the directory exists; `removeDatabase` empties it so installers can reload a schema; `availableDatabases` lists the files of the storage directory.
`eZTableList`, `relationList`, `removeRelation`	Read `sqlite_master`.

`eZPersistentObject` code and the kernel's SQL were adjusted in January 2024 so the same statements run on both engines; the test runs also found missing `is_countable()` guards, fixed in the same period (see PHP 8 support).

Known limits

- One writer at a time (see above).
- No user and password; access control is the file system's.
- No replication or clustering of its own; a shared file system is only suitable for read-mostly sites.

Related pages

- SQLite database support, SQLite for Exponential Platform
- Platform SQLite installer, Database drivers and installers
- SQLite transactions, MongoDB kernel support (the other non-MySQL engine), SQL query cache (query results on every engine)
- Changelogs: 6.0.1, 6.0.13, 6.0.14, 6.0.15
- Chronicles: January 2024, first half, January 2024, second half, April 2026, June 2026, second half

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.