SQLite for hosts

SQLite 3, the default database of Exponential, from a host's point of view: what the driver adds, the four files and their permissions, the shared settings, backups and restores without copying a live file, monitoring...

Source file: `doc/eng-us/help/hosting/basics/sqlite-for-hosts.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/hosting/basics/sqlite-for-hosts.md

This page is for system administrators and support staff at a hosting company whose customers run Exponential on SQLite 3, the default database. It answers: what SQLite needs from the server, how to set its files up so every process can write them, how to back it up and restore it safely, what to watch, and when a site should move to a database server. Everything here comes from chapter 9.2 of the installation book, which has the full detail.

Contents

1. Why SQLite matters to a host
2. Requirements
3. What the driver adds
4. Files and permissions
5. Settings
6. Backup and restore
7. Monitoring
8. When a site outgrows SQLite
9. Checklist
10. Related pages

Why SQLite matters to a host

SQLite is first in the official order and the installers' default (`settings/setup.ini`, `[DatabaseSettings] DefaultType=sqlite3`; `exp:install` without `--db`). For a host it means:

- **No database server and no database account** to provision, secure or bill for.
- **The database is part of the site's files**, in `var/storage/sqlite3/`, so it moves, is quota-counted and is isolated like the rest of the site.
- **It is a full production database** for a site served from one machine: the project describes it so, and the driver was load-tested for it (Database support).

Requirements

- The PHP extension `sqlite3` (`php -m | grep -i sqlite3`).
- A SQLite library of version **3.33 or later** in PHP (`php --ri sqlite3` prints `SQLite Library => ...`). The wizard accepts 3.0.1, but the driver uses `UPDATE ... FROM`, which SQLite has from 3.33.

- A writable directory for the database, by default `var/storage/sqlite3/`.
- Optional but useful: the `sqlite3` command line program, for backups and checks.

What the driver adds

From Database support:

- **Write-ahead logging (WAL)**, so page views keep reading while an editor publishes.
- **Connection settings for a web server**: `synchronous=NORMAL`, a 64 MB page cache, a 256 MB memory map, a 5-second busy timeout.
- **A write queue** (since 6.0.15): every transaction takes an exclusive lock on `<database>.writer-lock` and runs `BEGIN IMMEDIATE`, so concurrent publishes wait their turn, up to `SQLiteTransactionWait` seconds (25 shipped), instead of failing with "database is locked".

Most requests never reach the database: Velocity's response cache, the content view cache and the template-block cache answer first (chapter 9.2).

Files and permissions

A SQLite database is four files in one directory:

File	Who needs to write it
`exponential.db`	every process that writes
`exponential.db-wal`	every process that writes
`exponential.db-shm`	every process, readers too
`exponential.db.writer-lock`	any process (read-only is enough for `flock()`)

SQLite creates and removes the `-wal` and `-shm` files, so the **directory** must be writable by every process: the web server's PHP, Velocity's workers, cronjobs and console commands. If one of them creates `-shm` with mode `0644`, the others fail with "attempt to write a readonly database". Set it up once:

Shell

```
chgrp -R <web group> var/storage/sqlite3
chmod 2775 var/storage/sqlite3
chmod 664 var/storage/sqlite3/*
```

and give every process a umask of `0002` (`UMask = 0002` in a systemd unit, `umask 002` at the start of a cron line). Better still, run every process as the site's own user (Writable directories and permissions).

> **Warning**
>
> Do not use `bin/modfix.sh` to fix SQLite permissions: it makes `var/storage` and other directories world-writable, and the script itself warns that this is not secure.

Never serve the database over HTTP. The shipped rewrite rules send `var/storage` to `index.php`, and Velocity's static file list leaves it out. Check each host configuration:

Shell

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

Anything but `200` is right.

Settings

INI

```
[DatabaseSettings]
DatabaseImplementation=sqlite3
# a plain name lives in var/storage/sqlite3/; a name starting with / is an absolute path
Database=exponential.db
# seconds a transaction waits for the writers ahead of it; keep it below the web server's request
 timeout
SQLiteTransactionWait=25
# PRAGMAs over the driver's defaults, one name=value per line (empty keeps the defaults)
SQLitePragmas[]
```

`SQLiteTransactionWait` must stay below the request timeout: Velocity's `requestTimeout` is 30 seconds unless changed; under PHP-FPM it is the pool's `request_terminate_timeout`. Raise both together, the wait a few seconds below.

Memory. Count up to 64 MB of page cache per process that has opened the database; `SQLitePragmas[]=cache_size=-16384` lowers it to 16 MB, which helps on plans with many small sites.

Backup and restore

> **Warning**
>
> Never copy a database file that is in use. A copy taken during a write, or without its `-wal` file, can be inconsistent. Use SQLite's online backup.

Shell

```
sqlite3 var/storage/sqlite3/exponential.db ".backup '/backup/exponential-$(date +%F).db'"
sqlite3 /backup/exponential-2026-10-05.db "PRAGMA integrity_check"     # prints ok
```

`VACUUM INTO '/backup/<file>.db'` is an alternative that writes a compacted copy. **Restore** through SQLite's own locking, in maintenance mode:

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
```

If `.restore` reports an error, the live database is unchanged; then stop every process that may have the file open, move the database and its `-wal` and `-shm` files aside together, put the backup in place, fix owner and mode, and start again (chapter 9.2). A complete backup also contains `var/storage`, `settings/override` and `settings/siteaccess`.

Monitoring

What	How	Healthy
Lock timeouts	`grep -c "database is busy" var/log/error.log`	`0`, or rare and explained
Integrity	`sqlite3 var/storage/sqlite3/exponential.db "PRAGMA quick_check"`	`ok`
Journal mode	`sqlite3 var/storage/sqlite3/exponential.db "PRAGMA journal_mode"`	`wal`
WAL size	`ls -l var/storage/sqlite3/`	the `-wal` file stops growing after busy periods
Disk space	`df -h var/storage/sqlite3`	room for the database twice over
Ownership	`ls -l var/storage/sqlite3/`	all files group-writable, one group

A large `-wal` file left after heavy writing shrinks with `SQLitePragmas[]=journal_size_limit=67108864` or a manual `PRAGMA wal_checkpoint(TRUNCATE)`. `VACUUM` needs about twice the database's size in free disk and holds the write lock, so run it in maintenance mode (chapter 9.2).

When a site outgrows SQLite

Move the site to a database server (PostgreSQL first) when:

- **more than one machine** must write the database: WAL needs shared memory on one host, so a network file system is not a substitute, and several containers cannot share one database file;
- **many editors publish large changes at the same moment** all day, and the queue is regularly longer than `SQLiteTransactionWait`;
- the customer needs **replication**, a hot standby or point-in-time recovery from the database itself;
- the customer needs **database users and grants**.

Database size is rarely the reason. See Changing the database engine.

Checklist

☐ `sqlite3` extension, library 3.33 or later, in web and command line PHP.

☐ `var/storage/sqlite3/` group-writable and setgid, every process in one group with umask `0002`.

☐ The database file not reachable over HTTP.

☐ `SQLiteTransactionWait` below the request timeout.

☐ Backups with `.backup` or `VACUUM INTO`, checked with `PRAGMA integrity_check`.

☐ Monitoring for "database is busy" in `var/log/error.log`.

Public demos on SQLite. A demo rebuilt every few hours fits SQLite well: each build installs a fresh database file with `exp:install --db=sqlite` in demo mode (`EXP_DEMO_MODE=1`), and the

old build is removed with its file. Recipe: run a public demo on LiteSpeed and Plesk.

Related pages

- Databases in the official order
- Database support: SQLite 3
- Chapter 9.2: SQLite 3
- SQLite database support and SQLite transactions
- Database backups per engine
- 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.