

SQLite

- <https://www.sqlite.org/autoinc.html>
 - <https://www.sqlite.org/quirks.html>
 - <https://docs.sqlitelcloud.io/docs/sqlite>
 - <https://github.com/asg017/sqlite-vss>
 - <https://litecli.com/>
 - <https://www.sqlite.org/cksumvfs.html>
 - <https://sqlite-utils.datasette.io/en/stable/>
 - <https://sqlite-utils.datasette.io/en/stable/cli.html#transforming-tables>
 - Leverage **unixepoch()** (or handle conversions at the application level)
 - https://www.sqlite.org/lang_datefunc.html

Potentially useful extensions

- <https://github.com/phiresky/sqlite-zstd>
 - <https://github.com/backtrace-labs/verneuil>
 - <https://gist.github.com/pkhuong/555be66f5e573a2d56186d658fd80881>

Backups

- Backup to storage box (or Cloudflare R2 or Hetzner Object Storage)
 - <https://github.com/tailscale/cpc>
 - **pg_dump dbname | lz4 | rclone cat storagebox:dst/path**
 - <https://rclone.org/crypt/>
 - <https://sqlite.org/rsync.html>
 - <https://oldmoe.blog/2024/04/30/backup-strategies-for-sqlite-in-production/>
 - <https://litestream.io/>
 - Can backup to storagebox via SFTP or FTPS (faster)
 - <https://litestream.io/guides/sftp/>
 - <https://restic.net/>
 - https://github.com/lobsters/lobsters/blob/d5d54ec92b32331f8d458d3b03c4949ff9aa4c5c/app/jobs/restic_job.rb#L9-L12

Optimizing SQLite for servers

You need to know to run it in WAL mode and be careful to wrap your write operations in short transactions - do that and it'll handle pretty much anything.

- Set these pragmas:

```
pragma auto_vacuum = incremental;  
pragma journal_mode = WAL;  
pragma wal_autocheckpoint = 1000;  
pragma busy_timeout = 5000;  
pragma synchronous = NORMAL;  
pragma cache_size = 1000000000;  
pragma foreign_keys = true;  
pragma recursive_triggers = true;  
pragma temp_store = memory;
```

- Use begin immediate

- Unless you are 100% certain that you will not write to the database within the transaction

- Use **begin concurrent** when it is available in the main branch (or manually patch it before compiling)

- https://www.sqlite.org/cgi/src/doc/begin-concurrent/doc/begin_concurrent.md

- See also <https://sqlite.org/htctree/doc/htctree/doc/htctree/index.html>

- Use strict tables

- Use check constraints for implementing booleans

```
create table if not exists markets (  
    -- This only takes up 1 byte, since integer size  
    is_flexible in SQLite  
    "is_open" int not null check ("is_open" in (0,  
1))  
) strict;
```

- Ensure text columns always have a max length

- Leverage pragma optimize to run analyze (maybe before transaction ends)

- https://docs.sqlitedatabasecloud.io/docs/sqlite/lang_analyze

- https://www.sqlite.org/lang_analyze.html#periodically_run_pragma_optimize_

- Leverage vacuum to keep the database compact

- https://docs.sqlitedoc.org/doc/sqlite/lang_vacuum
- Ensure SQLite was compiled with **SQLITE_ENABLE_UPDATE_DELETE_LIMIT**
 - Needs to be the full source, not the amalgamation
 - This is already enabled in Ecto, see: <https://github.com/elixir-sqlite/exqlite/blob/main/Makefile>
- <https://kerkour.com/sqlite-for-servers> / <https://archive.is/Xfjh6>
- <https://use.expensify.com/blog/scaling-sqlite-to-4m-qps-on-a-single-server>
- <https://blog.wesleyac.com/posts/consider-sqlite>
- <https://avi.im/blag/2021/fast-sqlite-inserts/>
- Have a single connection responsible for writing, and separate connections for reading
 - Writer connection pool size = 1
 - Reader connection pool size = N (throttle as per your application requirements and system constraints)
 - <https://github.com/bxcodec/dbresolver>
 - <https://github.com/matttn/go-sqlite3/issues/1022#issuecomment-1067353980>

Consider using without rowid if the primary key is not an integer

<https://www.sqlite.org/withoutrowid.html>

Choosing whether to use without rowid in SQLite tables depends on the specific requirements of your application. without rowid tables are a feature of SQLite designed to optimize storage and performance under certain conditions, but they are not universally beneficial. Here are key points to consider:

When to Use without rowid

- **Primary Key is NOT INTEGER:** If your primary key is not an integer type, using without rowid can save space and potentially improve performance, as SQLite does not have to maintain an internal integer primary key rowid for each row.
 - **Narrow Tables:** For tables with few columns, without rowid can reduce the storage overhead by eliminating the implicit rowid.

- **Clustered Index:** without rowid tables use the primary key as the clustered index. This means data is stored on disk in the order of the primary key, which can speed up range queries on the primary key.

When NOT to Use without rowid

- **Requires INTEGER PRIMARY KEY:** If your table benefits from an autoincrement integer primary key, without rowid is not suitable because it requires explicitly defining the primary key columns, and the autoincrement feature works with the implicit rowid.
 - **Complex Queries and Joins:** Tables with without rowid can have slower performance for complex queries and joins, especially if these operations do not align well with the primary key's order.
 - **Compatibility and Flexibility:** Not all SQLite tools and libraries may fully support without rowid tables. Additionally, opting for without rowid can limit some future schema changes, such as adding certain types of indexes.

Short-Circuiting correlated subqueries

<https://emschwartz.me/short-circuiting-correlated-subqueries-in-sqlite/>

Detecting Full Table Scans

https://sqlite.org/c3ref/stmt_status.html

<https://tenderlovemaking.com/2026/07/15/detecting-full-table-scans-with-sqlite/>

Contentless delete tables

https://www.sqlite.org/fts5.html#contentless_delete_tables