SQLite: die Schreibwarteschlange und Sperren

Wie der SQLite-Treiber von Exponential 6.0.15 gleichzeitige Schreiber mit einer Sperrdatei und BEGIN IMMEDIATE einreiht, wie lange eine Transaktion wartet (SQLiteTransactionWait) und wie sich das zu den Anfrage-Timeouts…

Source file: `doc/ger-de/hilfe/datenbankadministratoren/sqlite-schreibwarteschlange-und-sperren.md`
Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/database-administrators/sqlite-write-queue-and-locking.md

Diese Seite richtet sich an Datenbankadministratoren, die Exponential auf SQLite betreiben und verstehen müssen, was passiert, wenn zwei Redakteure im selben Moment veröffentlichen. Sie beantwortet: warum einfache SQLite-Transaktionen mit "database is locked" scheitern, was der Treiber seit 6.0.15 stattdessen tut, wie lange ein Schreibzugriff wartet, wie ein Timeout aussieht und wie Sie seine Ursache finden. Quellen: Schreibzugriffe in der Warteschlange und SQLite: Transaktionen reihen sich für die Schreibsperre ein.

Inhalt

1. Das Problem
2. Was der Treiber tut
3. Wie lange eine Transaktion wartet
4. Wie ein Timeout aussieht
5. Wenn sich Timeouts wiederholen
6. Geforkte Prozesse und andere Benutzer
7. Gemessenes Verhalten
8. Checkliste

Das Problem

Ein einfaches `BEGIN` in SQLite ist verzögert: Die Transaktion liest zuerst und verlangt die Schreibsperre erst bei ihrem ersten Schreibzugriff. Hat eine andere Verbindung dazwischen committet, ist der gelesene Schnappschuss veraltet, und SQLite antwortet sofort mit `SQLITE_BUSY`, ohne zu warten, weil Warten den Schnappschuss nicht aktuell machen kann. Vor 6.0.15 konnte eine Veröffentlichung mit "database is locked" scheitern, egal wie lang das Busy-Timeout war.

Was der Treiber tut

`eZSQLite3DB::beginQuery()` startet jede Transaktion so:

1. Es öffnet, einmal je Verbindung und erneut in jedem geforkten Prozess, eine Sperrdatei neben der Datenbank: `<database file>.writer-lock`, zum Beispiel `var/storage/sqlite3/exponential.db.writer-lock`.

2. Es nimmt ein exklusives `flock()` auf diese Datei und versucht es alle 1 bis 15 ms erneut; Schreiber, die schon länger als eine Sekunde warten, versuchen es öfter, was sie ungefähr in der Reihenfolge ihrer Ankunft bedient. Solange es wartet, hält es ein geteiltes `flock()` auf eine zweite Datei, `<database file>.writer-queue`, die den anderen sagt, dass ein Schreiber wartet. Ein Schreiber, der die Sperre frei vorfindet, während andere warten, lässt sie ihnen 30 ms lang, bevor er es selbst versucht. Ohne das nahm eine Schleife, die festschreibt und sofort wieder beginnt (der Cronjob-Teil `indexcontent`, die Stapel eines Inhaltsauftrags), die Sperre Bruchteile einer Millisekunde nach jedem Commit wieder und hielt sie minutenlang; das Speichern einer Seite dahinter scheiterte nach seiner ganzen Wartezeit mit "database is busy".
3. Mit gehaltener Sperre führt es `BEGIN IMMEDIATE` aus, das die Schreibsperre von SQLite am Anfang nimmt, wo das Busy-Timeout gilt. Auf Schreiber außerhalb der Warteschlange (eine einzelne Anweisung, ein anderes Programm) wartet der Busy-Handler von SQLite innerhalb der verbleibenden Zeit.
4. Bei `COMMIT` oder `ROLLBACK` gibt es die Sperrdatei frei. Stirbt der Prozess, gibt das Betriebssystem sie frei; ein Absturz blockiert die Warteschlange also nie.

Am Anfang zu warten ist sicher: Bevor `BEGIN IMMEDIATE` gelingt, hat die Transaktion nichts geschrieben, und danach kann nichts in ihr ein Wettrennen um eine Sperre verlieren.

Jeder Prozess, der in die Datenbank schreibt, verwendet dasselbe `.writer-lock`: PHP-FPM-Worker, Velocity-Worker, Cronjobs und Konsolenskripte reihen sich gegeneinander ein.

Wie lange eine Transaktion wartet

Bis zu `[DatabaseSettings] SQLiteTransactionWait` Sekunden (Standard `25`; Werte unter 1 werden auf 1 angehoben). Halten Sie den Wert unter dem Anfrage-Timeout des Webservers, damit eine wartende Anfrage mit einer klaren Meldung scheitert, statt abgebrochen zu werden:

Server	Anfrage-Timeout	Hinweis
Velocity	`requestTimeout` (`Q.webserver.requestTimeout`), 30 s, sofern nicht geändert	der Client erhält ein 504, wenn es überschritten wird
PHP-FPM	`request_terminate_timeout` des Pools (aus, sofern nicht gesetzt)	`max_execution_time` zählt Wartezeit unter Linux nicht

Releases vor 6.0.15 lieferten `60` aus; setzt Ihr Override noch `60`, senken Sie den Wert auf `25`. Um länger zu warten, erhöhen Sie das Anfrage-Timeout und `SQLiteTransactionWait` zusammen, die Wartezeit immer ein paar Sekunden unter dem Timeout.

Shell

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

Wie ein Timeout aussieht

In `var/log/error.log` und auf der Fehlerseite:

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

Jeder andere Fehler beim Start lautet `the transaction could not start: <SQLite's message>; nothing was written`. In beiden Fällen wurde nichts geschrieben; wiederholen Sie die Aktion.

Wenn sich Timeouts wiederholen

Ein Schreibzugriff hat die Sperre während der ganzen Wartezeit gehalten. Sehen Sie in `var/log/error.log` und im Cron-Log nach, was zu dieser Zeit lief:

- ein `VACUUM` (es hält die Schreibsperre während seines ganzen Laufs);
- ein großer Import oder ein Skript, das eine Transaktion geöffnet und dann auf etwas anderes gewartet hat;
- eine `sqlite3`-Shell, die innerhalb von `BEGIN` offen gelassen wurde.

Zählen Sie die Timeouts mit `grep -c "database is busy" var/log/error.log`. `SQLiteTransactionWait` zu erhöhen hilft nur, wenn der Halter berechtigt lange lief. Ist die Warteschlange beim normalen Redigieren regelmäßig länger als die Wartezeit, bedient ein Datenbankserver mit Sperren auf Zeilenebene die Redakteure besser (Eine Engine wählen).

Geforkte Prozesse und andere Benutzer

`flock()` gehört zu einer geöffneten Datei, ein geforkter Prozess darf also das Handle seines Elternprozesses nicht wiederverwenden. Der Treiber merkt sich die Prozess-ID, die die Sperrdatei geöffnet hat, und öffnet in jedem Fork eine neue; Velocity-Worker und die Kindprozesse des asynchronen Veröffentlichers reihen sich korrekt ein. Der Treiber öffnet `.writer-lock` zum Schreiben und, falls das scheitert, nur lesend: `flock()` funktioniert auf einem nur lesenden Handle, eine von einem anderen Benutzer angelegte Sperrdatei reiht also weiterhin korrekt ein.

Gemessenes Verhalten

Aus den eigenen Tests der Änderung auf dem Entwicklungsserver: 192 Veröffentlichungen aus 64 gleichzeitigen Prozessen wurden alle abgeschlossen, wo der frühere Start 6 davon verlor; 48 Transaktionen von je 300 ms, gemeinsam gestartet, waren nach 14,7 Sekunden fertig, etwa die 14,4 Sekunden, die sie nacheinander brauchen. Die Warteschlange kostet über das Warten hinaus fast nichts.

Checkliste

- ☐ `SQLiteTransactionWait` liegt unter `requestTimeout` von Velocity und `request_terminate_timeout` von PHP-FPM.
- ☐ Kein Override setzt noch den alten Wert `60`.
- ☐ `VACUUM` und große Importe laufen im Wartungsmodus.
- ☐ `grep -c "database is busy" var/log/error.log` ist 0 oder erklärt.

Verwandte Seiten

- SQLite 3 für Datenbankadministratoren
- SQLite-Wartung
- Datenbankfehler beheben
- Velocity-Worker und Pools
- Freie Dokumentation: Diese Seite darf unter der GNU FDL 1.3 frei kopiert, übersetzt und verbessert werden.

Werden Sie Exponential-Partner

Partner teilen ihre Informationen, stellen Mittel bereit, geben Beiträge zurück und entwickeln die Produkte mit uns, und so wächst Exponential. Eine Partner-Website und ein Dashboard sind für Mitte 2027 geplant: Werden Sie noch heute Partner, damit wir schneller wachsen und neue Funktionen früher liefern.

Partner: hier beginnen | Wie Sie sich bewerben

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.