SQLite ein Datei-basiertes Datenbank-Managementsystem

Seminar: DT-DB42-M Datenbanksysteme 2023

Nicholas Jaguczak
nicholas.jaguczak@stud.uni-bamberg.de
Universität Bamberg
Deutschland

ABSTRACT

SQLite ist ein weit verbreitetes Datenbank-Managementsystem. Es besitzt viele nützliche Eigenschaften, die der Grund für seine heutige Popularität sind. Obwohl SQLite als OLTP Datenbank konzipiert ist, wird es auch außerhalb gewöhnlicher OLTP Anwendungen eingesetzt. In der vorliegenden Arbeit wird die Architektur von SQLite behandelt und die Ergebnisse eines OLTP und OLAP Benchmarks von SQLite und DuckDB verglichen. Durch Modifikationen an SQLites Join Algorithmus kann die Leistungsfähigkeit in einer OLAP Anwendung deutlich verbessert werden. Für OLTP Anwendungen ist SQLite ein performantes Datenbank-Managementsystem, jedoch bei OLAP Anwendungen ist es empfehlenswerter, ein dafür vorgesehenes OLAP Datenbank-Managementsystem, wie DuckDB, zu verwenden.

1 EINLEITUNG

SQLite ist das weltweit meist eingesetzte Datenbank-Managementsystem (DBMS) [1]. Ein DBMS ist eine Systemsoftware, die sich um das Erstellen und das Verwalten von Datenbanken kümmert. Ziel eines Datenbanksystems ist es, Daten für einen bestimmten Zeitraum persistent abzuspeichern. SQLite sondert sich von anderen DBMS wie MySQL ab, da es ein Datei-basiertes DBMS ist. Es gibt keine Client-Server Architektur, stattdessen wird die komplette Datenbank als einzelne Datei auf dem Dateisystem gespeichert. Diese Arbeit basiert auf dem Paper "SQLite: Past, Present, and Future "[1] und daher werden Zitate nicht nach jedem Paragraph explizit angegeben.

Die Popularität von SQLite beruht auf mehreren Charakteristiken des DBMS [1].

Cross-platform Die Datenbankdatei kann ohne Probleme zwischen 32-Bit und 64-Bit Rechnern übertragen werden. Dabei ist es nicht relevant, ob der betreffende Rechner Little-Endian oder Big-Endian verwendet. Damit SQLite auf einer Plattform läuft, wird vorausgesetzt, dass diese über einen C Compiler verfügt. Außerdem soll ein 8-Bit Byte und das Zweierkomplement bei 32-Bit und 64-Bit Integer verwendet werden.

Kompaktheit Die komplette SQLite Library ist als einzelne C-Datei verfügbar und benötigt kompiliert weniger als 1 MB Speicher. Sie benutzt nur eine geringe Menge an Methoden aus der C Standard-Bibliothek und besitzt keine externen Abhängigkeiten.

Zuverlässigkeit SQLite wird als zuverlässig angesehen. Dies liegt an der Testabdeckung von 100% Branch Coverage [1]. Ebenso werden eine Vielzahl an unterschiedlichen Testtechniken verwendet. Dazu gehören unter anderem Grenzwerttests und Regressionstests.

Geschwindigkeit Auf einer Leistungsstarken Plattform kann SQLite über zehntausend Transaktionen pro Sekunde durchführen. Ebenso kann es unter Umständen BLOB Daten schneller lesen und schreiben und benötigt dabei weniger Speicher als das Dateisystem selbst.

Datei basierte DBMS, speziell SQLite, sind vor allem für Einzelanwendungen geeignet. Es erleichtert das Entwickeln, da kein extra Prozess für das Datenbanksystem erstellt werden muss. Stattdessen ist SQLite in den Prozess der Hauptanwendung integriert. Um die Datenbank zu verwenden, entfällt dementsprechend die Kommunikation mit einem Datenbankserver. Stattdessen werden die SQLite Library Methoden verwendet. So entfällt das Debuggen eines Datenbankservers auf einem externen Prozess. Beinahe auf jeden iOS und Android Geräte wird SQLite benutzt. So können beispielsweise die Kontakte auf einem Smartphone leicht lokal in einer Datei gespeichert werden. Dadurch sind die Daten schnell abrufbar und können leicht auf anderen Geräte übertragen werden, sofern diese auch SQLite nutzen.

In den folgenden Absätzen wird die Architektur von SQLite zusammen mit dem Aufbau der Datenbank näher betrachtet. Daraufhin werden *Online Analytical Processing* (OLAP) und *Online Transaction Processing* (OLTP) erläutert. Schließlich soll SQLite als OLTP-DBMS mit DuckDB als OLAP-DBMS bezüglich der Leistungsfähigkeit in OLAP und OLTP Transaktionen verglichen werden.

2 ARCHITEKTUR

In diesem Abschnitt wird die grobe Architektur von SQLite besprochen. Als erstes wird auf die Modularität von SQLite eingegangen und daraufhin die Struktur der Datenbankdatei erläutert.

2.1 Module

SQLite besteht aus vier Modulen, siehe Abbildung 1. Es gibt drei Module, die miteinander interagieren: der *Core*, der *SQL Compiler* und das *Backend*. Der *Core* ist verantwortlich für das Laden und Ausführen von SQL Statements. Der *SQL Compiler* wandelt SQL Statements in lauffähigen Bytecode um, der vom *Core* ausgeführt werden kann. Das *Backend* greift auf die Datenbank-Pages zu und stellt sicher, dass die Daten persistent gespeichert werden. Das vierte Modul, die *Accessoires*, beinhaltet unter anderem eine große Menge an Testsuiten und weitere nützliche Werkzeuge wie zum Beispiel für die Speicherzuweisung. Auf dieses Modul wird jedoch nicht näher eingegangen, da es nicht essenziell für die Lauffähigkeit von SQLite ist.

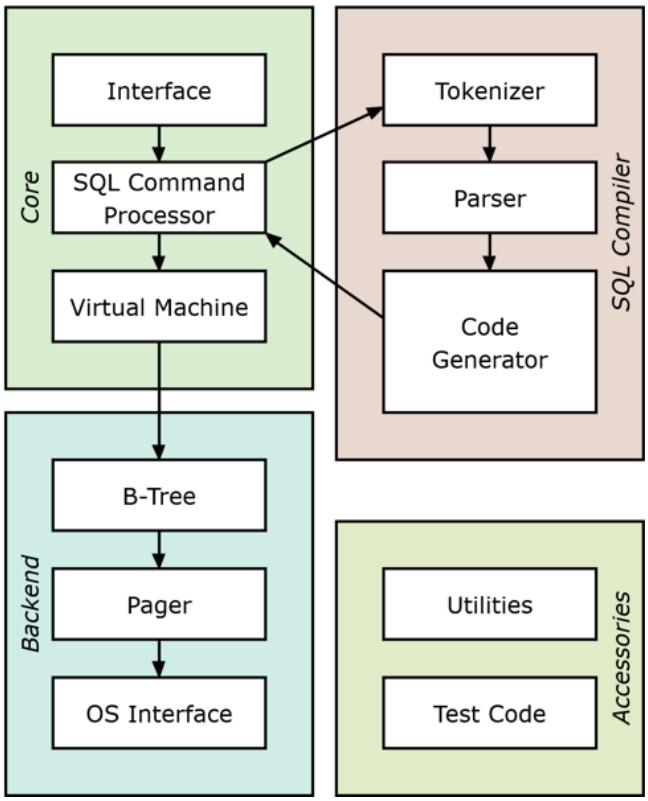

Abbildung 1: Architektur, Quelle:[2]

2.1.1 Core Modul. Der *Core* ist das Herz von SQLite [1]. Wie zuvor erwähnt, ist er dafür zuständig, den generierten Bytecode auszuführen. Die Execution Engine ist als Virtuelle Maschine aufgebaut, auch bekannt als "virtual database engine"(VDBE) [1]. Die Engine führt den Bytecode aus, beginnend mit dem Befehl an Adresse 0. Es wird jeder Befehl im Bytecode ausgeführt solange es sich nicht um ein *Halt* Befehl handelt oder ein Fehler auftritt. Im Falle eines Fehlers wird ein Rollback auf die ausstehenden Änderungen ausgeführt, damit die Datenkonsistenz erhalten bleibt.

2.1.2 SQL Compiler Modul. Jedes SQL Statement besteht aus einer Menge von Anweisungen, welche Aktionen auf der Datenbank ausführen. Damit die VDBE SQL Statements ausführen kann, müssen diese in ein ausführbares Programm übersetzt werden. Dafür sind die Komponenten innerhalb des *SQL Compilers* zuständig [2]. Der Tokenizer bekommt das SQL Statement als String und bricht diesen in Token für den Parser auf. Der Parser weist den Token basierend auf dem Kontext eine Bedeutung zu. Der Code Generator generiert aus den Ergebnissen des Parsers ein lauffähiges Bytecode Programm. Dieser Bytecode besteht aus virtuellen Befehlen, die einzeln betrachtet sehr simpel sind, jedoch zu sehr komplexen Programmen kombiniert werden können. Diese virtuellen Befehle sind das Grundgerüst der Datenverarbeitung von SQLite [1]. Jeder virtuelle Befehl verfügt über eine Adresse, einen einzigartigen Opcode und mehrere Operanden. Je nach Opcode werden die Parameter verwendet und entscheiden, welche Aktionen ausgeführt werden.

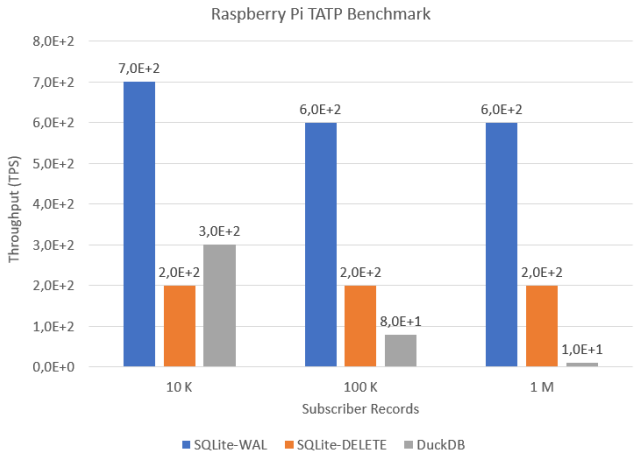

Abbildung 2: TATP Benchmark, Datenquelle:[1]

2.1.3 Backend Modul. Dieses Modul besteht aus drei Komponenten: der Datenbankdatei (B-Bäumen), dem Pager und dem Operating System (OS) Interface. In dem Absatz 2.2 wird genauer auf den Aufbau von der Datenbankdatei, B-Bäumen und Pages eingegangen. Der Pager ist verantwortlich, die Pages zu laden, zu schreiben und zu cachen. Dabei wird sichergestellt, dass die Anfragen sicher, schnell und effizient verarbeitet werden. Das OS Interface ist die Schnittstelle zum Betriebssystem und dem darin enthaltenen Dateisystem. Damit SQLite unabhängig vom Betriebsystem verwendbar ist, wird ein virtual file system (VFS) genutzt [1]. Es gibt bereits mehrere VFS für die Betriebssysteme Unix und Windows. Damit ein neues Betriebssystem unterstützt wird, muss unter Umständen ein neues VFS erstellt werden.

2.2 Struktur der Datenbankdatei

Die Datenbankdatei besteht aus einer Collection von B-Bäumen [1]. Ein B-Baum ist eine Datenstruktur, die einen Wurzelknoten besitzt. Jeder Knoten verweist auf zwei oder mehr Kindknoten. SQLite verwendet B-Bäume, welche die Daten in den Blattknoten speichert.

Es gibt zwei Arten von B-Bäumen. Die Tabellen-B-Bäume und die Index B-Bäume. Tabellen B-Bäume bilden die Datenbankschemen ab und enthalten 64-Bit signed Integer Schlüssel mit dem dazugehörigen Datensatz. Jeder Datensatz repräsentiert eine Zeile in der Datenbank. Diese besitzen jeweils einen Primärschlüssel, die sogenannte *ROWID*. Wird allerdings ein *INTEGER PRIMARY KEY* definiert ist, wird die *ROWID* damit ersetzt. Eine Tabelle ohne *ROWID* oder *INTEGER PRIMARY KEY* speichert die Daten in einem Index B-Baum anstelle eines Tabellen B-Baums. Der Schlüssel setzt sich aus den Spalten zusammen, die als *PRIMARY KEY* festgelegt sind, gefolgt von den restlichen Spalten. Der Index B-Baum speichert keine Daten, sondern nur Schlüssel ab und die Referenzen auf Tupel.

Jeder B-Baum besteht aus Interior-Pages und Leaf-Pages [3]. Bei einem Tabellen B-Baum beinhalten die Leaf-Pages Schlüssel mit dem dazugehörigen Daten. Interior-Pages enthalten Schlüssel mit

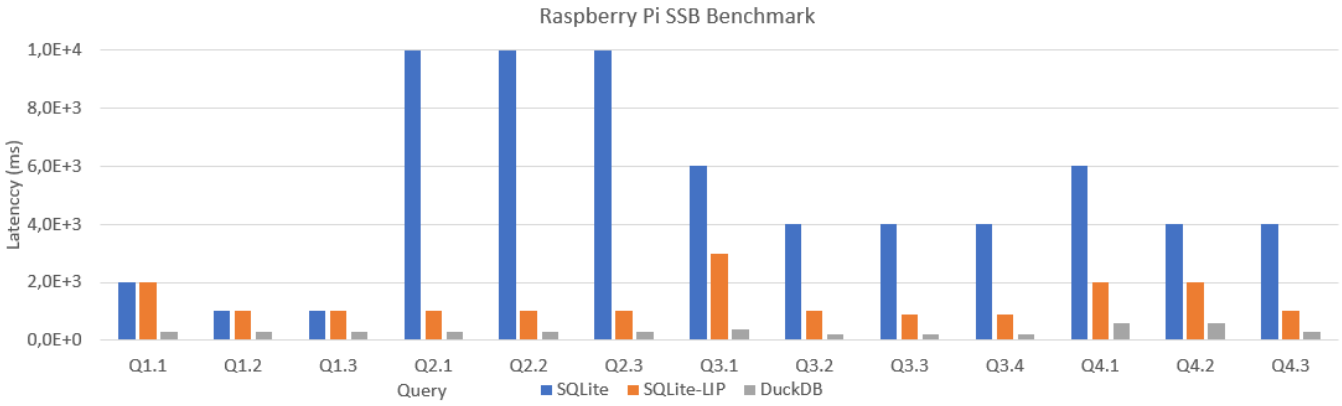

Abbildung 3: SSB Benchmark, Datenquelle:[1]

Referenzen auf Kind-Pages. Durch diese Eltern-Kind-Beziehung kann auf den kompletten B-Baum zugegriffen werden, wenn die Wurzel-Seite bekannt ist.

3 OLTP UND OLAP TRANSAKTIONEN

SQLite wurde hauptsächlich als OLTP Datenbank entworfen. OLTP steht für *Online Transaction Processing*, wobei es sich um Echtzeitdatenverarbeitung handelt. Gewöhnlich müssen diese Prozesse schnell sein, da mit diesen Anwendungen in Echtzeit interagiert wird und eine lange Wartezeit das Nutzererlebnis einschränken würde. OLAP hingegen steht für *Online Analytical Processing*, hier ist der Hauptzweck die Datenanalyse. Bei solchen Anwendungen werden große Datenmengen auf einmal gelesen. Die Querys sind dementsprechend meist deutlich komplexer aufgebaut z.B. mit Joins zwischen den Tabellen. DuckDB ist eine OLAP Datenbank, die eine sehr gute Performanz bei OLAP Transaktionen bietet. Sie wird ebenfalls als "das SQLite für Analyse" bezeichnet [1].

Obwohl SQLite als OLTP Datenbank konzipiert ist, wird sie dennoch in Szenarien außerhalb von OLTP verwendet. Deshalb wird in den folgenden Absätzen SQLite mit DuckDB bei einem OLTP Benchmark und einem OLAP Benchmark verglichen. Dabei ist deutlich zu erkennen, wie sich die Leistung eines OLTP DBMS, wie SQLite, im Gegensatz zu einem OLAP DBMS, wie DuckDB, bei OLAP und OLTP Transaktionen unterscheidet. Diese Benchmarks wurden sowohl auf einem Raspberry Pi als auch auf einem Cloud Server ausgeführt [1]. In dieser Arbeit wird nur auf die Raspberry Pi Ergebnisse eingegangen. Der Raspberry Pi verfügt über einen ARM Cortex-A72 4-Kern Prozessor mit 1.5 GHz und 8 GB LPDDR4-3200 Arbeitsspeicher. Als Speicher wird eine Micro SD Karte genutzt. Bei SQLite wurden zwei Modi für den Benchmark verwendet. Auf die unterschiedlichen Modi wird hier nicht näher eingegangen, bei bestehendem Interesse findet man dazu weitere Informationen in dem Paper "SQLite: Past, Present, and Future" [1].

3.1 OLTP Benchmark

Als OLTP Benchmark wird der *Telecommunication Application Transaction Processing* (TATP) Benchmark verwendet [1]. Dabei wird die Leistung des Datenbanksystems in einer gewöhnlichen Telekommunikationsanwendung gemessen. Die Transaktionen werden zufällig mit festgelegten Wahrscheinlichkeiten für den Transaktionstyp generiert. Jeder Durchlauf besteht aus 10 Sekunden Aufwärmphase, gefolgt von einer 60-sekündigen Bewertungsphase. Bei den Durchläufen wird die Größe der Haupttabelle angepasst. Die Haupttabelle ist die Subscriber-Tabelle und die restlichen Tabellen werden proportional zu ihr skaliert [1]. Die Ergebnisse des Benchmarks sind in Abbildung 2 zu sehen. Als Maßeinheit wird der Durchsatz in Transaktionen pro Sekunde (TPS) verwendet. Je höher der Wert, desto besser die Performanz. SQLite-WAL und SQLite-DELETE sind die verschiedenen Modi, die verwendet wurden. Diese unterscheiden sich in der Art wie sie die Daten auf die Datenbank schreiben und dabei die ACID Eigenschaften erfüllen.
Bei SQLite-WAL werden die originalen Seiten in der Datenbankdatei erhalten und die modifizierten Seiten werden in eine *write-ahead Log* (WAL) Datei geschrieben. Wenn SQLite eine Page sucht wird zuerst die WAL überprüft, falls diese dort nicht vorhanden ist, wird die eigentliche Datenbankdatei durchsucht. Wenn die WAL-Datei eine definierte Größe überschreitet werden die modifizierten Seiten auf die Datenbankdatei geschrieben und die WAL geleert.
Bei SQLite-DELETE werden die originalen Seiten in ein *Rollback Journal* geschrieben [1] und die modifizierten Seiten, werden in die Datenbankdatei geschrieben.

In jedem Durchlauf hat SQLite-WAL den höchsten TPS-Wert. Bei einer größeren Datenbank gibt es eine kleine Minderung am TPS-Wert bei SQLite-WAL. Bei der größten Datenbasis ist der Unterschied zwischen SQLite-WAL und DuckDB mit einem Faktor von 60X am größten. DuckDB schlägt sich im Gegensatz zu SQLite-DELETE bei kleineren Datenbanken besser. Je größer jedoch die Datenbank wird, desto schlechter wird der TPS-Wert in DuckDB. Wogegen bei SQLite-DELETE der TPS-Wert konstant bleibt.

3.2 OLAP Benchmark

Als OLAP Benchmark wird der *Star Schema Benchmark* (SSB) verwendet [1]. SSB misst die Leistung eines Datenbanksystems in einer typischen Anwendung eines Datenlagers. Es werden eine große Faktentabelle und vier kleinere Dimensionstabellen verwendet. Die

Anfragen beinhalten natürliche Verbünde zwischen den beiden Tabellenarten. Die Ergebnisse des SSB Benchmarks sind in Abbildung 3 zu sehen. Im Gegensatz zu davor werden hier einzelne Anfragen durchgeführt und die Leistung wird anhand der Latenz gemessen. Die Latenz ist die benötigte Zeit, vom absenden der Anfrage, bis das Ergebnis von dem DBMS zurückgegeben wird. Dementsprechend bedeutet ein niedriger Wert eine bessere Performanz.

Um die Leistung bei SQLite bei OLAP Transaktionen zu verbessern, wurde der Profiler der VDBE aktiviert. Das liefert die Anzahl von CPU Zyklen, die die VDBE je Anweisung im Bytecode benötigt. Zum Beispiel benötigt die Anweisung *SeekRowid* mit am meisten Rechenzeit. Um unnötige B-Baum Suchen im Join zu vermeiden, wurde der Bloom-Filter in SQLites Join Algorithmus eingebaut. In Abbildung 3 ist SQLite-LIP die optimierte Variante von SQLite.

Es ist klar zu erkennen, dass DuckDB bei jeder Query eine niedrigere Latenz hatte als SQLite und SQLite-LIP. Die größte Differenz ist bei den 2. Query-Anfragen, hier ist DuckDB zwischen 30-50X schneller als SQLite. Ebenso ist hier die beste Performanz-Steigerung zwischen SQLite und SQLite-LIP zu erkennen. SQLite-LIP ist dabei um den Faktor 10X schneller als SQLite. Es ist klar zu erkennen, dass die Optimierungen am Join-Algorithmus einen signifikanten Unterschied bei SQLite ausmachen.

4 FAZIT

SQLite schneidet bei den OLTP Benchmarks deutlich besser ab als DuckDB. Für diese Art von Anfragen ist DuckDB nicht entworfen. Für DuckDB ist es gewöhnlich, Daten blockweise zu laden und speichern. Bei OLTP müssen Daten oft schnell zeilenbasiert abgerufen oder gespeichert werden. Je größer die Datenbank wird, desto empfehlenswerter ist die Verwendung von SQLite anstelle von DuckDB bei einer OLTP-Anwendung.Denn bei DuckDB ist die Leistungsfähigkeit mit steigender Datenbankgröße gesunken.
Die Ergebnisse aus dem OLAP Benchmark zeigen eindeutig, dass DuckDB eine deutlich bessere Leistung als SQLite und SQLite-LIP in einer OLAP-Anwendung hat. Deshalb ist ein OLAP DBMS wie DuckDB empfehlenswerter bei einer solchen Anwendung anstelle von SQLite. Jedoch beweist der Test ebenso, dass durch Optimierungen SQLite für OLAP-Anwendung seine Leistung verbessern kann.
In der Zukunft wird es weiterhin Anwendungszwecke geben, bei denen ein dateibasiertes DBMS essenziell ist. Durch den besonderen Aufbau von SQLite ist es mit vielen Systemen kompatibel und wird weiterhin ein wichtiges DBMS sein.

LITERATUR

[1] Kevin P. Gaffney, Martin Prammer, Larry Brasfield, D. Richard Hipp, Dan Kennedy, and Jignesh M. Patel. 2022. SQLite: Past, Present, and Future. *Proc. VLDB Endow.* 15, 12 (aug 2022), 3535–3547. https://doi.org/10.14778/3554821.3554842
[2] D. Richard Hipp. [n. d.]. Architecture of SQLite. https://www.sqlite.org/arch.html [Online; accessed 18.05.2023].
[3] D. Richard Hipp. [n. d.]. Database File Format. https://www.sqlite.org/fileformat.html [Online; accessed 17.05.2023].