SQLite: Datenbank-Engine im Kleinformat

Datenspeicher für Sparsame

SQLite bringt zwar die Funktionalität einer relationalen SQL-Datenbank mit, kommt aber ohne aufwändig zu administrierende Server aus. Dieser Artikel vergleicht die Bibliothek mit ihren Konkurrenten und zeigt, wie man sie in eigene Programme einbaut. Christoph Dalitz

www.photocase.de

Beim Stichwort Datenbank denken die meisten wohl an Schwergewichte wie Oracle oder DB 2. Solche auf der Client-Server-Architektur beruhenden Produkte sind für kritische Anwendungen mit großen Datenmengen und viele gleichzeitige Zugriffe konzipiert. Es gibt aber auch Bereiche, in denen dieses System des Guten zu viel ist: So kann der Entwickler eines Routenplaners oder eines Telefon- und Adressbuchs seinen Kunden kaum zumuten, ein komplettes Datenbanksystem zu installieren, einzurichten und zu verwalten.

Desktop-Datenbanken wie SQLite [1] lassen sich dagegen leicht in eigene Programme integrieren, ob direkt über die C-Schnittstelle oder über eines der zahlreichen Skriptsprachen-Interfaces, zum Beispiel in Perl, Python, Ruby, PHP und vielen anderen, siehe [2]. Den konzeptionellen Unterschied zwischen beiden Ansätzen zeigt Abbildung 1. Bei einem Datenbanksystem greifen Anwendungsprogramme nicht direkt auf die Daten zu, sondern kommunizieren mit einem separaten Prozess (dem Datenbank-Managementsystem), der seinerseits die Daten verwaltet.

Bei einer Desktop-Datenbank liest und schreibt jedes Anwendungsprogramm direkt. Das hat den Vorteil, dass kein Datenbankserver gestartet und administriert werden muss, stößt aber insbesondere im Mehrbenutzerbetrieb schnell an Grenzen. Salopp formuliert ist eine Desktop-Datenbank lediglich ein SQL-Frontend für das Dateisystem (siehe Kasten „Datenbankbegriffe" und [3]).

SQLite-Komponenten

Sourcecode und vorkompilierte Binaries für diverse Plattformen finden sich als freie Software auf der SQLite-Homepage. Unter Linux lässt sich der Sourcecode problemlos gemäß der »README«-Anleitung übersetzen. Die Dokumentation fehlt dann noch, dafür sorgt ein weiteres »make doc«. Nach der Installation mit »make install« findet der Anwender folgende Komponenten auf seinem System:

- Den SQL-Interpreter »sqlite« und
- die C-Bibliothek Libsqlite und die Headerdatei »sqlite.h«.

Das Programm »sqlite« dient zur interaktiven Arbeit mit einer SQLite-Datenbankdatei. Als Desktop-Datenbank hat SQLite keine Benutzerverwaltung und kein Berechtigungskonzept, sodass sich Datenbankdateien ohne Login mit dem Befehl »sqlite *DBdatei*« öffnen und bearbeiten lassen.

Neben den SQL-Kommandos kennt »sqlite« auch Metakommandos, die mit einem Punkt beginnen und einige Operationen ermöglichen, die über die reine Datenmanipulation hinausgehen. Nützlich sind zum Beispiel ».read *SQLdatei*« zum Ausführen der SQL-Befehle in »*SQLDatei*« sowie ».tables« zum Anzeigen der vorhandenen Tabellen oder ».dump *Tabelle*« zum Export einer Tabelle in Form eines SQL-Skripts.

Datenbank mit Bibliothek

Für den Entwickler von größerem Interesse ist die C-Bibliothek Libsqlite, weil sie die Möglichkeit bietet, eigene Programme mit geringem Aufwand um eine Datenspeicherkomponente mit SQL-Zugriff zu erweitern. Wenn keine Select-Kommandos auszuführen sind, reichen drei Funktionen:

- »sqlite_open()« zum Öffnen einer Datenbankdatei
- »sqlite_close()« zum Schließen einer Datenbankdatei
- »sqlite_exec()« zum Ausführen von SQL-Kommandos

Listing 1 zeigt, wie die SQLite-Bibliothek einzusetzen ist. Die Funktion »sqlite_

open()« in Zeile 13 öffnet die Datenbank. In Zeile 22 nimmt die Variable »sql« das SQL-Insert-Statement auf, das »sql_exec()« dann ausführt.

Die Routine »sqlite_open()« legt das Datenbank-File an, wenn es noch nicht existiert. Wer das nicht möchte, muss vorher prüfen, ob es die Datei schon gibt, zum Beispiel mit der Posix-C-Funktion »access()«. Beim SQL-Statement »SELECT« wird es komplizierter, weil dessen Ergebnis im Allgemeinen eine komplette Tabelle ist. Die übliche Lösung für dieses Problem in SQL-Programmierschnittstellen sind so genannte Statement-Cursors, die in der SQLite-Dokumentation Virtual Machines heißen. Damit durchläuft der Anwender zeilenweise die als Ergebnis gelieferte Tabelle. Die SQLite-Bibliothek stellt dafür drei Funktionen bereit:

- »sqlite_compile()« zum Anlegen eines Statement-Cursors
- »sqlite_step()« zum Navigieren in der Ergebnistabelle
- »sqlite_finalize()« zum Zerstören des Statement Cursors

Auch dafür findet sich ein Beispiel in Listing 1, Zeilen 31 bis 50. Den Statement-Cursor legt »sql_compile()« an. Wenn kein Fehler auftritt, dann arbeitet »sqlite_step()« das Ergebnis ab, bis keine Zeilen mehr übrig sind (siehe Zeile 43).

Neben der C-Bibliothek enthält die SQLite-Distribution einen ODBC-Treiber, der unter anderem den Zugriff aus Office-Paketen auf SQLite-Daten ermöglicht [4]. Das ist aber mit Vorsicht zu genießen, weil SQLite keine Foreign-Key-Constraints unterstützt (siehe Kasten „Datenbankbegriffe“), sodass ein Schreibzugriff des Endanwenders zum Beispiel mit Microsoft Access schnell zum Datenchaos führen kann.

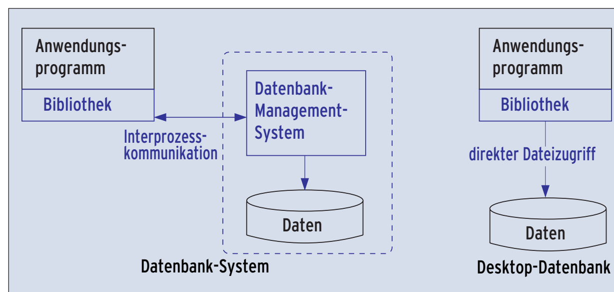

Abbildung 1: Bei einem Datenbanksystem verwaltet ein Server die Zugriffe, die Desktop-Datenbank macht alles alleine.

SQLite und der Standard

Wichtig für den Benutzer ist, wie gut seine Datenbank den SQL-Standard [5] unterstützt. Üblicherweise erfüllen Datenbanksysteme weitgehend den Standard aus dem Jahre 1992 (auch SQL 2 genannt) mit einigen ausgewählten Features der Weiterentwicklung aus dem Jahre 1999 (SQL 3). Auch SQLite orientiert sich am SQL-2-Standard, wobei es einige Basisfunktionen relationaler Datenbanken wie das Ändern von Tabellenstrukturen mit »ALTER TABLE« oder Foreign-Key-Constraints nicht unterstützt. Andererseits bringt es einige fortgeschrittene Features wie Trigger oder Views mit, die für Desktop-Datenbanken ungewöhnlich sind.

Tabelle 1 gibt einen Überblick über Abweichungen vom SQL-2-Standard. Auffallend ist, dass SQLite keine richtigen Datentypen besitzt, ähnlich wie viele Skriptsprachen. Die wichtigsten Unterschiede zum Standard sind:

- »CREATE« verarbeitet beliebige Datentypen, die SQLite aber ignoriert. Intern speichert es alle Werte als »\x00«-terminierte Strings. ▶

Listing 1: Libsqlite-Beispiel

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <sqlite.h>
04
05 sqlite *db;           /*Datenbank-Objekt*/
06 sqlite_vm *dbcursor;  /*zum Abarbeiten Select-Ergebnis*/
07 char *dberr = 0;      /*Fehlermeldungen*/
08 char *sql, *sqltail;  /*SQL-Kommandos*/
09
10 /*
11  * Datenbankdatei öffnen
12  */
13 if (! (db = sqlite_open("test.db", 0, &dberr)) ) {
14    printf("Hoppla: %s\n", dberr);
15    free(dberr);
16    return 1;
17 }
18
19 /*
20  * Beispiel für Nicht-Select Statement
21  */
22 sql = "INSERT INTO person (nr,name) VALUES (1,'Melitta Beutel')";
23 if (SQLITE_OK != sqlite_exec(db, sql, NULL, NULL, &dberr)) {
24    printf("Huch: %s\n", dberr);
25    free(dberr);
26 }
27
28 /*
29  * Beispiel für Select-Statement
30  */
31 sql = "SELECT nr, name FROM person WHERE name LIKE '%Beutel'";
32 /* Lege Statement-Cursor an (Konstruktor) */
33 if (SQLITE_OK != sqlite_compile(db, sql, &sqltail,
34                                 &dbcursor, &dberr)) {
35    printf("Oha: %s\n", dberr);
36    free(dberr);
37 } else {
38    int ncols;        /*Anzahl zurückgegebener Spalten*/
39    char **colnames;  /*Spaltennamen*/
40    char **values;    /*Spalteninhalte*/
41
42        /* arbeite Ergebnis mit Cursor ab */
43    while (SQLITE_ROW == sqlite_step(dbcursor, &ncols,
44                                     &values, &colnames)) {
45        printf("nr='%s', name='%s'\n", value[i]);
46    }
47
48        /* Speicher von Cursor freigeben (Destruktor) */
49    sqlite_finalize(dbcursor, NULL);
50 }
51
52 /*
53  * Datenbank schließen
54  */
55 sqlite_close(db);
```

- Feldinhalte fasst SQLite je nach Kontext als »TEXT«- oder »NUMBER«-Werte auf, zum Beispiel bei »SELECT *Feld1* + 1 FROM ...« als »NUMBER« oder bei »SELECT *Feld1* || 'bla' FROM ...« als »TEXT« (»||« ist der Stringverkettungsoperator in SQL).

Auch wenn SQLite-Autor Richard Hipp diese Typfreiheit in der Dokumentation als Vorteil darstellt, bringt sie einige potenzielle Probleme mit sich. So sind dann häufig Werte als Zahlen identisch, aber als String verschieden, zum Beispiel »1.0« und »1.00«. Das verletzt aber das Grundprinzip relationaler Datenbanken, dass der Anwender das interne Speicherformat nicht kennen muss.

Besonders fehlerträchtig ist das bei Datum-Strings, die je nach verwendetem Format (TT.MM.JJ, JJ.TT.MM und so weiter) verschiedene Bedeutungen haben. Im schlimmsten Fall ist nicht mal gewährleistet, dass in einem Datumsfeld überhaupt ein Datum steht, denn SQLite akzeptiert aufgrund der Typlosigkeit auch »bla« als Datum.

Tabelle 1: Abweichungen vom SQL2-Standard

SQL-Feature	Bemerkung
Datentypen	Typlos, mit zwei impliziten Datentypen: »TEXT«, »NUMBER«
Check Constraints	Nicht unterstützt
Foreign Key Constraints	Nicht unterstützt
Subqueries	Keine Correlated Subqueries (zum Beispiel mit »EXISTS«)
Alter Table	Nicht unterstützt
Benutzer- und Rechteverwaltung	Nicht unterstützt
SQL-Funktionen	Einige Funktionen fehlen, insbesondere Datumsfunktionen

Konsistenz durch ACID

Über das Fehlen einer Benutzer- und Rechteverwaltung kann man bei einer Desktop-Datenbank wohl getrost hinwegsehen, weil sie typischerweise im Einbenutzerbetrieb läuft. Schwerwiegender ist die mangelnde Unterstützung für Integrity-Constraints, besonders für Fremdschlüssel (Foreign Keys).

Wenn der Entwickler weiß, dass nur seine Anwendung auf die Daten zugreift, kann er unzulässige Einträge noch selber abfangen. Wenn aber auch das Programm »sqlite« oder ein ODBC-Frontend Tabellen verändert, ist die Datenkonsistenz nicht mehr gewährleistet. Und ohne kaskadierende Updates oder Deletes, also Änderungen, die sich über mehrere Tabellen auswirken und dabei deren Konsistenz garantieren, wird die Datenpflege mühsam und fehleranfällig.

Nach so viel Genörgel folgt ein Pluspunkt von SQLite: Es unterstützt ACID-kompatible Transaktionen (siehe Kasten „Datenbankbegriffe“). Weniger wichtig bei nur einem Benutzer ist die Isolation gleichzeitig ablaufender Transaktionen, zumal SQLite sie über eine Sperre der gesamten Datenbankdatei realisiert, also im Mehrbenutzerbetrieb überhaupt keine Parallelität bietet.

Sehr nützlich ist es dagegen, Fehler mit »ROLLBACK« zu behandeln, denn damit kehrt der Anwender zu einem definierten, fehlerfreien Zustand der Datenbank zurück. Die Durability ist eine nette Zugabe, die bewirkt, dass zum Beispiel »kill -9« laufende SQLite-Transaktionen ohne Datenkorruption abwürgt.

Performance

Auf der Homepage gibt der SQLite-Autor an, dass seine Datenbank bei den „üblichen Operationen“ doppelt so schnell wie PostgreSQL oder MySQL sei, und liefert auch die Daten seines Benchmark-Tests. Solche Benchmarks sind mit Vorsicht zu genießen, weil sich je nach Auswahl der Testfälle und des darauf zugeschnittenen Tunings fast jedes gewünschte Ergebnis produzieren lässt. Um seine Behauptung zu überprüfen, waren einige Messungen mit SQLite, PostgreSQL („The world's most advanced Open-Source Database“) und MySQL („The world's most popular Open-Source Database“) nötig.

Bei allen Tests griff nur das Testprogramm auf die Datenbank zu, bei PostgreSQL und MySQL über Unix-Domain-Sockets, um eventuellen Netzwerk-Overhead zu vermeiden. Bei allen Datenbanken blieb die Default-Konfiguration erhalten, wobei bei MySQL der Tabellentyp »InnoDB« zum Einsatz kam, da sonst wichtige Datenbankfunktionen nicht verfügbar sind. Der C++-Code der Testprogramme liegt auf dem Server des Linux-Magazins [6].

Die in Tabelle 2 dargestellten Ergebnisse zeigen ein differenzierteres Bild als die SQLite-Homepage:

- Bei Abfragen, die nur eine einzige Tabelle betreffen, ist SQLite tatsächlich schneller.
- Bei komplexeren Abfragen über mehrere Tabellen kann SQLite jedoch auch deutlich langsamer sein. Bei Abfrage 3 war SQLite sogar unbenutzbar langsam, was aber durch Indizes auf allen Fremdschlüsselfeldern behebbar war. Doch auch mit Indizes war SQLite noch langsamer als PostgreSQL (Abfrage 4).

Tabelle 2: Laufzeiten im Vergleich

Nr.	Beschreibung	Laufzeit SQLite 2.8.13	Laufzeit PostgreSQL 7.3.4	Laufzeit MySQL 4.0.20
1	110100 Inserts mit 12 Commits	54,6547 s (siehe Text)	55,1268 s	30,5562 s
2	»SELECT MAX()« auf 100000 Tupel	0,1259 s	0,2763 s	1,3934 s
3	3-Tabellen-Join ohne Indizes	1104,7357 s	0,1357 s	0,4055 s
4	3-Tabellen-Join mit Indizes	0,5690 s	0,1905 s	1,8336 s
5	Gruppieren einer Tabelle ohne Index	0,3644 s	1,6467 s	0,6772 s
6	Gruppieren einer Tabelle mit Index	0,3627 s	0,7406 s	1,2660 s
7	Gruppieren zweier Tabellen ohne Index	1,2646 s	3,5776 s	0,9119 s
8	Gruppieren zweier Tabellen mit Index	1,2659 s	2,4646 s	2,0457 s
9	Anlegen zweier Indizes für Join-Felder	4,8363 s	0,1905 s (inkl. »ANALYZE«)	7,3118 s

Im Vergleich von PostgreSQL und MySQL ergeben sich keine extremen Ausreißer wie bei SQLite, aber es gibt doch zwei interessante Beobachtungen:

- Bei MySQL führen die Indizes kurioserweise zur Verlangsamung der Abfragen. Das kann daran liegen, dass bei diesen Abfragen so viele Daten gelesen werden müssen, dass sowieso ein Tablescan nötig wird und die Indexzugriffe nur zusätzlicher Overhead sind.
- Die Inserts bei PostgreSQL dauern länger, was wohl an MVCC (Multi Version Concurrency Control) liegt. Das ist ein modernes Verfahren zur Transaktionsisolation, das aufwändiger zu implementieren ist als klassische Sperrverfahren, aber im Mehrbenutzerbetrieb höhere Parallelität ermöglicht.

Die Insert-Geschwindigkeit von SQLite verhält sich so, wie in Abbildung 2 dargestellt: Bei einer neu angelegten Datenbankdatei sind die Inserts deutlich schneller als bei einer bereits verwendeten, aber leeren Datei (nach jedem Test wurden alle Tabellen wieder gelöscht). Nach einer Einschwingphase stellt sich ein stabiler Wert ein, der die Grundlage für Tabelle 2 ist.

Fazit

Die Testergebnisse sollte niemand überbewerten. Sie bestätigen vor allem die Erkenntnis, dass durch geschickte Auswahl der Testabfragen jede Datenbank als die schnellste erscheinen kann.

Datenbankbegriffe

SQL steht für Structured Query Language, die standardisierte Abfragesprache für relationale Datenbanken, siehe [3].

Integrity-Constraints sind Bedingungen, die die logische Korrektheit der Daten gewährleisten.

Foreign-Key-Constraints (Fremdschlüssel) sind die wichtigste Variante der Integrity-Constraints, die korrekte Beziehungen zwischen Tabellen gewährleisten. Um zu testen, ob ein Wert für eine Tabelle gültig ist, bezieht die Datenbank eine andere Tabelle mit ein, daher das „Foreign".

ACID steht für Atomicity Consistency Isolation Durability und bezeichnet die Anforderungen, die Transaktionen erfüllen müssen, damit Daten konsistent bleiben.

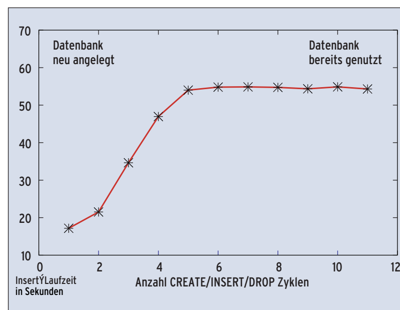

Abbildung 2: Die Insert-Geschwindigkeit von SQLite hängt davon ab, wie oft die Tabellen bereits gelöscht und wieder angelegt wurden.

SQLite ist eine attraktive Möglichkeit, Anwendungen mit einer schlanken Datenspeicherkomponente zu versehen. Die SQL-Schnittstelle bietet eine Flexibilität, die mit selbst gestrickten Dateiformaten nur schwer erreichbar ist.

Anders als bei den bekannteren kommerziellen Desktop-Datenbanken Paradox oder Access ist keine weitere Konfiguration auf dem Zielsystem erforderlich. Eine mit der SQLite-Library gelinkte Anwendung ist ohne weiteres Zutun lauffähig. Das Einsatzgebiet reicht von Embedded-Systemen über Desktop-Anwendungen bis hin zu Tests für komplexere Client-Server-Anwendungen. Die Vielfalt an Skriptsprachen-Interfaces macht SQLite auch für jene Programmierer interessant, die mit C auf dem Kriegsfuß stehen. *(ofr)* ■

Infos

[1] SQLite-Homepage: [http://www.sqlite.org/]

[2] Programmiersprachen für SQLite: [http://www.sqlite.org/cvstrac/wiki?p=SqliteWrappers]

[3] Ulrich Wolf, „SQL kurz gefasst": [http://www.linux-magazin.de/Artikel/ausgabe/2001/04/sql/sql.html]

[4] SQLite-ODBC-Treiber: [http://www.ch-werner.de/sqliteodbc/]

[5] Date, Darwen, „ SQL – Der Standard": Addison-Wesley 1998

[6] Testprogramme zum Artikel: [http://www.linux-magazin.de/Service/Listing s/2004/09/SQLite]

Der Autor

Christoph Dalitz lehrt an der Fachhochschule Niederrhein in Krefeld unter anderem das Fach Datenbanksysteme. Dabei kommt im Praktikum nicht SQLite zum Einsatz, sondern PostgreSQL.

[linux nachrichten]

Kostenlose Linux-News?

Newsletter abonnieren!

Bestellen unter:
www.linux-nachrichten.de