EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Dr. Eberle Zentrum für digitale Kompetenzen

DataBASHing - Konsolenbasiertes Arbeiten mit SQLite

Michael Derntl

Dieses Tutorial führt sie Schritt für Schritt in die Konsolen-basierte Verwendung und Abfrage einer SQLite Datenbank ein. Grundlegende Kenntnisse über relationale Datenbanken und SQL Abfragen werden vorausgesetzt.

Beispieldaten

Zum Nachvollziehen der folgenden Schritte benötigen sie einige Dateien:

- Komplettdatenbank: schokolade.db
- Rohdaten im CSV-Format (basierend auf Chocolate Bar Ratings Datensatz)
  - company.csv : Alle Firmen, die Schokoladen herstellen mit den Spalten:
    - `id` : eindeutige Identifikationsnummer der Firma
    - `name` : Name der Firma
    - `country` : Land der Firma
  - review.csv : Alle Schokoladen-Reviews mit folgenden Spalten:
    - `id` : eindeutige Identifikationsnummer des Reviews
    - `company_id` : Identifikationsnummer der Firma (Verweis auf die id in der Tabelle company), die diese Schokolade hergestellt hat
    - `bar_name` : Name des Schokoriegels
    - `year` : Jahr des Reviews
    - `coca_pct` : Kakaoanteil zwischen 0 und 1
    - `rating` : Bewertung zwischen 1 und 5
    - `bean_type` : Kakaobohnen-Sorte (fehlt bei vielen Einträgen)
    - `bean_origin` : Kakaobohnen-Herkunft (fehlt bei vielen Einträgen)
- Datenbankdefinition: create_tables.sql

SQLite Installation

```
sudo apt install sqlite3
```

SQLite unter Windows

https://sqlite.org/download.html

Für Windows am besten `sqlite-tools-win32-x86-xxx.zip`. Entpacken in ein beliebiges Verzeichnis. Dieses Verzeichnis dann zur Windows-Umgebungsvariable „`Path`“ hinzufügen (Wie geht das?)

Ausprobieren

```
sqlite3 --version
```

Sollte mindestens `3.30` sein.

SQLite Konsole

Nun starten wir SQLite Konsole mit einer Datenbank. Dafür einfach als Argument den Dateinamen (z.B. schokolade.db) der Datenbank übergeben:

```
sqlite3 schokolade.db
```

Wenn die Datenbankdatei bereits existiert, wird sie geöffnet. Ansonsten wird eine neue Datenbank in dieser Datei angelegt.

Nun sind wir in der SQLite Konsole, die Ausgabe sollte wie folgt aussehen:

```
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite>
```

Bei diesem Prompt kann man nun zwei Arten von Kommandos absetzen:

- SQL-Befehle zum Erzeugen der Tabellen, Erstellen von Abfragen, etc.
- „Dot-Kommandos“, das sind spezielle Steuerbefehle, die mit einem Punkt (dot) beginnen und mit denen man Einstellungen ändern kann oder Daten importieren kann

---

Dot-Kommandos

Eine komplette Liste der Dot-Kommandos erhält man mit dem Kommando “ `.help` ” (hier nur eine Auswahl dargestellt):

```
sqlite> .help
.backup ?DB? FILE       Backup DB (default "main") to FILE
.cd DIRECTORY           Change the working directory to DIRECTORY
.changes on|off         Show number of rows changed by SQL
.clone NEWDB            Clone data into NEWDB from the existing database
.dump ?OBJECTS?         Render database content as SQL
.echo on|off            Turn command echo on or off
.excel                  Display the output of next command in spreadsheet
.exit ?CODE?            Exit this program with return-code CODE
.headers on|off         Turn display of headers on or off
.help ?-all? ?PATTERN?  Show help text for PATTERN
.import FILE TABLE      Import data from FILE into TABLE
.limit ?LIMIT? ?VAL?    Display or change the value of an SQLITE_LIMIT
.mode MODE ?TABLE?      Set output mode
.nullvalue STRING       Use STRING in place of NULL values
```

```
.once ?OPTIONS? ?FILE?  Output for the next SQL command only to FILE
.open ?OPTIONS? ?FILE?  Close existing database and reopen FILE
.quit                   Exit this program
.read FILE              Read input from FILE
.schema ?PATTERN?       Show the CREATE statements matching PATTERN
.separator COL ?ROW?    Change the column and row separators
.shell CMD ARGS...      Run CMD ARGS... in a system shell
.show                   Show the current values for various settings
.tables ?TABLE?         List names of tables matching LIKE pattern TABLE
.width NUM1 NUM2 ...    Set minimum column widths for columnar output
```

Detaillierte Erklärungen aller Kommandos gibt es unter https://sqlite.org/cli.html

---

Datenbanktabellen erzeugen

Das Kommando “ `.tables` ” zeigt uns, dass die Datenbank noch leer ist. Daher erzeugen wir mal die Tabellen “ `company` ” und “ `reviews` ”.

Dafür gibt es zwei Möglichkeiten: entweder die Kommandos in die Konsole schreiben, oder die Kommandos in eine Datei (z.B. create_tables.sql) speichern und diese mit “ `.read` ” ausführen:

```
sqlite> .read create_tables.sql
sqlite> .tables
company review
```

Daten aus CSV importieren

Die Tabellen sind nun angelegt aber leer. Man kann nun die Tabellen zeilenweise füllen mit dem SQL Kommando “ `insert into` ”, wir wollen aber lieber die Tabellen aus den CSV-Dateien füttern. Dafür gibt es das Kommando “ `.import` ”. Lassen wir uns mal die Hilfe zum Import anzeigen:

```
sqlite> .help import
.import FILE TABLE       Import data from FILE into TABLE
  Options:
    --ascii              Use \037 and \036 as column and row separators
    --csv                Use , and \n as column and row separators
    --skip N             Skip the first N rows of input
    -v                   "Verbose" - increase auxiliary output
  Notes:
    * If TABLE does not exist, it is created. The first row of input
      determines the column names.
    * If neither --csv or --ascii are used, the input mode is derived
      from the ".mode" output mode
    * If FILE begins with "|" then it is a command that generates the
      input text.
```

Wir werden also die Option “ `--csv` ” verwenden, und die erste Zeile (Spaltennamen) mittels “ `--skip 1` ” übergehen. Wenn keine Fehlermeldung kommt, kann man davon ausgehen, dass alles funktioniert hat. Wir gucken trotzdem wie viele Zeilen in company eingetragen wurden:

```
sqlite> .import --csv --skip 1 company.csv company
sqlite> select count(*) from company;
418
```

Das gleiche noch mit den Reviews, und voila die Datenbank ist gefüllt und bereit für unsere Abfragen:

```
sqlite> .import --csv --skip 1 review.csv review
sqlite> select count(*) from review;
1795
```

Quiz Warum haben wir erst “ `company` ” und dann erst “ `review` ” gefüllt?

Antwort Da die Einträge in Spalte “ `company_id` ” in Tabelle “ `review` ” auf die `id` Daten in “ `company` ” verweisen.

Abfragen

Man kann nun Abfragen in SQL formulieren, die Ergebnisse werden entsprechend ausgegeben, z.B. Jahreszahl und Bewertung aller Reviews für Schokoladen aus Vietnam:

```
sqlite> select year, rating from review where bean_origin = "Vietnam";
2014|2.75
2017|3.5
2016|3.5
2016|3.25
2010|3.25
[... einige ausgeblendete Zeilen...]
2016|3.0
2016|4.0
2015|3.25
2015|3.25
2012|3.5
```

Wir sehen die Ausgabe ist ohne Spaltenüberschrift und das verwendete Trennzeichen ist die Pipe. Das kann man mit den Kommandos .headers und .separator ändern, z.B. mit Tabulator als Trennzeichen:

```
sqlite> .headers on
sqlite> .separator "\t"
sqlite> select year, rating from review where bean_origin = 'Vietnam';
year rating
2014 2.75
2017 3.5
2016 3.5
2016 3.25
2010 3.25
[... einige ausgeblendete Zeilen...]
2016 3.0
2016 4.0
```

```
2015 3.25
2015 3.25
2012 3.5
```

Modus

Das Format der Ausgabe von Abfrageergebnissen kann man unter anderem über das Kommando “ `.mode` ” voreinstellen. Der Standardmodus ist “ `list` ”, man kann aber z.B. auch CSV oder als Tabelle ausgeben lassen:

```
sqlite> .help mode
.mode MODE ?TABLE? Set output mode
  MODE is one of:
    ascii       Columns/rows delimited by 0x1F and 0x1E
    box         Tables using unicode box-drawing characters
    csv         Comma-separated values
    column      Output in columns. (See .width)
    html        HTML <table> code
    insert      SQL insert statements for TABLE
    json        Results in a JSON array
    line        One value per line
    list        Values delimited by "|"
    markdown    Markdown table format
    quote       Escape answers as for SQL
    table       ASCII-art table
    tabs        Tab-separated values
    tcl         TCL list elements
```

Wollen wir beispielsweise ein Abfrageergebnis als CSV-Datei speichern, so stellen wir den Ausgabemodus auf “ `csv` ” und verwenden das Kommando “ `.once` ” um die Ausgabe in eine Datei zu leiten:

```
sqlite> .once ergebnis.csv
sqlite> select year, rating from review where bean_origin = 'Vietnam';
```

Über Kommandozeile steuern

Um SQLite in einen Verarbeitungspipeline zu verwenden, kann man SQLite auch ohne interaktive Konsole verwenden:

```
> sqlite3 -help
Usage: sqlite3 [OPTIONS] FILENAME [SQL]
FILENAME is the name of an SQLite database. A new database is created
if the file does not previously exist.
OPTIONS include:
  -A ARGS...           run ".archive ARGS" and exit
  -append              append the database to the end of the file
  -ascii               set output mode to 'ascii'
  -bail                stop after hitting an error
  -batch               force batch I/O
  -box                 set output mode to 'box'
  -column              set output mode to 'column'
  -cmd COMMAND         run "COMMAND" before reading stdin
```

```
  -csv                 set output mode to 'csv'
  -deserialize         open the database using sqlite3_deserialize()
  -echo                print commands before execution
  -init FILENAME       read/process named file
  -[no]header          turn headers on or off
  -help                show this message
  -html                set output mode to HTML
  -interactive         force interactive I/O
  -json                set output mode to 'json'
  -line                set output mode to 'line'
  -list                set output mode to 'list'
  -lookaside SIZE N    use N entries of SZ bytes for lookaside memory
  -markdown            set output mode to 'markdown'
  -maxsize N           maximum size for a --deserialize database
  -memtrace            trace all memory allocations and deallocations
  -mmap N              default mmap size set to N
  -newline SEP         set output row separator. Default: '\n'
  -nofollow            refuse to open symbolic links to database files
  -nonce STRING        set the safe-mode escape nonce
  -nullvalue TEXT      set text string for NULL values. Default ''
  -pagecache SIZE N    use N slots of SZ bytes each for page cache memory
  -quote               set output mode to 'quote'
  -readonly            open the database read-only
  -safe                enable safe-mode
  -separator SEP       set output column separator. Default: '|'
  -stats print         memory stats before each finalize
  -table               set output mode to 'table'
  -tabs                set output mode to 'tabs'
  -version             show SQLite version
  -vfs NAME            use NAME as the default VFS
  -zip                 open the file as a ZIP Archive
```

(Je nach verwendeter SQLite-Version können bei Ihnen weniger oder mehr als die hier gelisteten Optionen möglich sein.)

Nun wollen wir das Ergebnis unserer Vietnam-Abfrage direkt in der BASH ausgeben lassen (ohne interaktive SQLite-Konsole):

```
$ sqlite3 -csv -readonly schokolade.db \
 "select year, rating from review where bean_origin = 'Vietnam'"
```

Beachten: das “ `\` ” am Zeilenende führt zum Ignorieren des Zeilenumbruchs, sodass der Befehl in der nächsten Zeile fortgeführt werden kann. (Aber nur, wenn nach dem “ `\` ” kein weiteres Leerzeichen etc. sondern direkt der Zeilenumbruch folgt!)

CSV | Datenbank | CSV

Nun wollen wir ein CSV-File in eine temporäre SQLite-Datenbank füttern und anschließend eine Abfrage machen, deren Ergebnis als CSV ausgegeben wird.

Gucken wir zunächst in die CSV-Datei review.csv, die ersten paar Zeilen sehen wie folgt aus:

```
$ head -n 10 review.csv
```

```
id,company_id,bar_name,year,cocoa_pct,rating,bean_type,bean_origin
436,1,"Agua Grande",2016,0.63,3.75,,"Sao Tome"
437,1,Kpime,2015,0.7,2.75,,Togo
438,1,Atsane,2015,0.7,3.0,,Togo
439,1,Akata,2015,0.7,3.5,,Togo
440,1,Quilla,2015,0.7,3.5,,Peru
441,1,Carenero,2014,0.7,2.75,Criollo,Venezuela
442,1,Cuba,2014,0.7,3.5,,Cuba
443,1,"Sur del Lago",2014,0.7,3.5,Criollo,Venezuela
444,1,"Puerto Cabello",2014,0.7,3.75,Criollo,Venezuela
```

Jetzt pipen wir die komplette CSV-Datei in eine temporäre SQLite-Datenbank, führen die SQL Abfrage aus und gucken uns die ersten 10 Zeilen des Ergebnisses an:

```
$ cat review.csv | sqlite3 :memory: ".mode csv" ".import /dev/stdin review" \
"select year, rating from review where bean_origin = 'Vietnam'" | head -n 10
```

```
2014,2.75
2017,3.5
2016,3.5
2016,3.25
2010,3.25
2015,3.5
2015,2.75
2016,2.75
2016,3.0
2016,3.5
```

Um zu erklären, was bei obigen Aufruf von sqlite3 passiert, wird dieser im Folgenden nochmal in seine Teile zerlegt und diskutiert:

```
sqlite3 \
:memory: \
".mode csv" \
".import /dev/stdin review" \
"select year, rating from review where bean_origin = 'Vietnam'"
```

- `:memory:` ist ein spezieller "Dateiname" in SQLite, der andeutet, dass wir die Datenbank nicht in eine Datei auf den Datenträger speichern wollen, sondern mit einer temporären In-Memory-Datenbank arbeiten wollen, die so lange im Arbeitsspeicher existiert, wie das sqlite3 Kommando läuft.
- `".mode csv"` müssen wir vor dem Import-Kommando verwenden, um CSV importieren zu können, und auch um CSV ausgeben zu lassen. Diese Notwendigkeit ergibt sich aus der Dokumentation: *„Note that it is important to set the "mode" to "csv" before running the ".import" command. This is necessary to prevent the command-line shell from trying to interpret the input file text as some other format."*
- `".import /dev/stdin review"` veranlasst SQLite, von /dev/stdin zu lesen; das ist die Standard-eingabe, also das was via cat review.csv an sqlite3 geliefert wird. Das CSV soll in die Tabelle review importiert werden.
- `"select ..."` ist die SQL-Abfrage, die wir ausführen wollen.

Um beispielsweise die Anzahl der Zeilen im Abfrageergebnis zu ermitteln, können wir den Output von sqlite3 zum Zeilenzählen an `wc` pipen, und sehen es gibt 38 Reviews von vietnamesischen Schokoladen:

```
$ cat review.csv | sqlite3 :memory: ".mode csv" ".import /dev/stdin review" \
"select * from review where bean_origin = 'Vietnam'" | wc -l
```

```
38
```

May 16, 2023