



SQLite-Tutorial

Joerg Schwarze-Schütte
CJD Christophorusschule Elze

Version 1.3

Stand: 27.04.09

Inhaltsverzeichnis

Inhaltsverzeichnis	2
Grundlagen	3
Eingabeaufforderung	3
Starten	3
SQL-Befehle und Steuerbefehle	4
SQLite-Hilfe.....	4
Verknüpfte Datenbank	4
Tabelle erstellen	5
Datensatz eingeben.....	5
Datensätze anzeigen.....	5
Datensätze anzeigen nach Sortierung.....	5
Tabellen anzeigen.....	6
Aufbau einer Tabelle	6
Ausgabe formatieren.....	6
Anzeige im Spaltenformat.....	6
Spaltenüberschriften einblenden	6
Spaltenbreiten festlegen	7
Datensatz ändern/aktualisieren	7
Datensatz löschen.....	7
Tabellenfeld/-spalte einfügen.....	8
Tabelleninhalte exportieren	8
Das CSV-Format.....	8
SQLite Vorbereitungen	9
Der Export.....	9
Die Kontrolle	10
Daten importieren.....	10

Grundlagen

Bevor man mit SQLite arbeiten kann, muss man ein paar Voraussetzungen schaffen. Man benötigt einen USB-Stick und einen Internetanschluß. Im Browser www.sqlite.org eingeben und über die Registerkarte Download in den Downloadbereich wechseln. Unter „Precompiled Binaries For Windows“ findet man die momentan aktuelle Version von SQLite als ZIP-File (hier: `sqlite-3_6_13.zip`). Für weitere Betriebssysteme (Linux, MacOS) befinden sich ebenfalls hier die Dateien.

Entsprechende Datei anklicken, herunterladen und per 2-Klick entpacken nach USB-Stick (in diesem Tutorial als Laufwerk E: festgelegt). Wer SQLite auf dem USB-Stick in einem eigenen Ordner haben möchte sollte ihn vor dem Entpacken erstellen.

In dem Pfad, in dem man die Dateien entpackt hat, befindet sich nun eine Datei `sqlite3.exe`. Diese Datei benötigen wir später zum Bearbeiten und Erstellen von Datenbanken. In diesem Tutorial unter `E:\sqlite3.exe`

Eingabeaufforderung

Da es sich bei SQLite um ein Kommandozeilenprogramm handelt müssen wir es über die Eingabeaufforderung (`cmd.exe`) ausführen. Hierzu starten wir über START-Programme-Zubehör die Eingabeaufforderung und bekommen einen blinkenden Cursor. Wir navigieren mit folgenden DOS-Befehlen zu unseren benötigten Dateien.

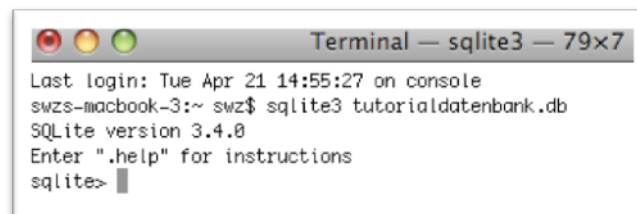
- `cd ORDNERNAME` - Ordnerwechsel (Ebene tiefer)
- `cd..` - Ordnerwechsel (Ebene höher)
- `dir` - Inhaltsverzeichnis anzeigen
- `e:` - Laufwerkswechsel (hier LW E:)

Alle angegebenen Befehle werden nach der Eingabe durch Drücken der ENTER-Taste (im weiteren Verlauf als ↵ gekennzeichnet) ausgeführt.

Starten

Nachdem wir mit den DOS-Befehlen auf unser Laufwerk gewechselt und den richtigen Ordner ausgewählt haben, befinden wir uns auf `E:\` und der Cursor blinkt.

Wir geben nun folgendes Beispiel ein: `E:\sqlite3 tutorialdatenbank.db` und drücken die ENTER-Taste. Gibt es keine Fehlermeldung, so befinden wir uns im Programm SQLite und die `tutorialdatenbank.db` wurde erstellt oder geöffnet.



```
Terminal — sqlite3 — 79x7
Last login: Tue Apr 21 14:55:27 on console
swzs-macbook-3:~ swz$ sqlite3 tutorialdatenbank.db
SQLite version 3.4.0
Enter ".help" for instructions
sqlite>
```

SQL-Befehle und Steuerbefehle

SQLite unterscheidet zwischen den reinen SQL-Befehlen und den SQLite-Steuerbefehlen. Die Steuerbefehle (z.B. `.databases`) beginnen immer mit einem Punkt. Die SQL-Befehle besitzen keinen Punkt, sondern werden mit einem Semikolon (;) beendet (z.B. `select * from personen;`)

SQLite-Hilfe

Über den Steuerbefehl `.help` kann man sich jederzeit die Hilfe aufrufen.

```
Terminal — sqlite3 — 79x33

sqlite> .help
.bail ON|OFF          Stop after hitting an error. Default OFF
.databases             List names and files of attached databases
.dump ?TABLE? ...     Dump the database in an SQL text format
.echo ON|OFF          Turn command echo on or off
.exit                Exit this program
.explain ON|OFF        Turn output mode suitable for EXPLAIN on or off.
.header(s) ON|OFF     Turn display of headers on or off
.help                Show this message
.import FILE TABLE   Import data from FILE into TABLE
.indices TABLE       Show names of all indices on TABLE
.mode MODE ?TABLE?   Set output mode where MODE is one of:
                    csv      Comma-separated values
                    column   Left-aligned columns. (See .width)
                    html     HTML <table> code
                    insert   SQL insert statements for TABLE
                    line     One value per line
                    list     Values delimited by .separator string
                    tabs     Tab-separated values
                    tcl      TCL list elements
.nullvalue STRING     Print STRING in place of NULL values
.output FILENAME      Send output to FILENAME
.output stdout        Send output to the screen
.prompt MAIN CONTINUE Replace the standard prompts
.quit                Exit this program
.read FILENAME        Execute SQL in FILENAME
.schema ?TABLE?       Show the CREATE statements
.separator STRING     Change separator used by output mode and .import
.show                Show the current values for various settings
.tables ?PATTERN?     List names of tables matching a LIKE pattern
.timeout MS           Try opening locked tables for MS milliseconds
.width NUM NUM ...    Set column widths for "column" mode
sqlite>
```

Verknüpfte Datenbank

Nachdem man SQLite gestartet hat sollte man überprüfen, ob auch die gewollte Datenbank richtig verknüpft wurde und ich darauf zugreifen kann. Mit `.databases` kann ich diese Information aufrufen.

```
Terminal — sqlite3 — 80x5

sqlite> .databases
seq  name          file
---  -
0    main           /Users/swz/tutorialdatenbank.db
sqlite>
```

Sollte in der Spalte „file“ ein falscher Eintrag stehen oder komplett fehlen, so ist SQLite ein weiteres Mal zu starten, nachdem man das Programm mit `.quit` verlassen hat.

Tabelle erstellen

Um Daten in der Datenbank speichern zu können, benötigen wir eine Tabelle. Jede Tabelle besteht aus Feldern, die einen Feldnamen und einen Felddatentyp besitzen. Wir wollen eine Schülertabelle „schueler“ erstellen mit den Feldern: vorname, nachname, schuelernr, klassestufe, klassenbezeichnung, klassenlehrer.

Mit dem SQL-Befehl CREATE lässt sich dieses Vorhaben umsetzen. Wir wählen für vorname, nachname, klassenbezeichnung, klassenlehrer den Felddatentyp VARCHAR, da wir Buchstaben und Zahlen eingeben wollen und für schuelernr, klassestufe den Felddatentyp INT, da es sich bei diesen Einträgen um Ganzzahlen handeln wird.

Befehl:

```
CREATE TABLE schueler (vorname VARCHAR(10), nachname VARCHAR(10), schuelernr INT, klassestufe INT, klassenbezeichnung VARCHAR(1), klassenlehrer VARCHAR(15));
```

Die Zahlenangabe hinter VARCHAR gibt an welche Länge als Minimum für dieses Feld im Speicher reserviert wird. Dadurch die Variabilität des Datentyps, wird die Speichergröße bei Bedarf erweitert.

Datensatz eingeben

Mit INSERT können Datensätze in die vorhandene Tabelle eingegeben werden.

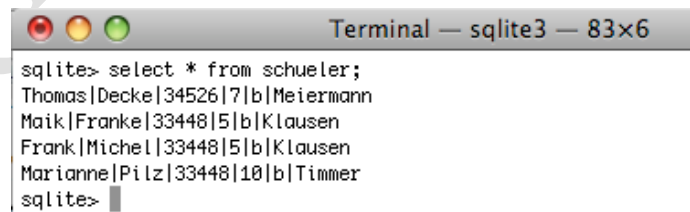
Befehl:

```
INSERT INTO schueler VALUES („Thomas“, „Decke“, 34526, 7, „b“, „Herr Meiermann“);
```

Mit diesem Befehl können nun weitere Datensätze hinzugefügt werden.

Datensätze anzeigen

Mit SELECT * from TABELLENNAME; kann man sich alle eingetragenen Datensätze anzeigen lassen. Da der SELECT-Befehl noch weitere Attribute besitzt wird er in den nachfolgenden Kapiteln noch häufiger auftauchen.



```
Terminal — sqlite3 — 83x6
sqlite> select * from schueler;
Thomas|Decke|34526|7|b|Meiermann
Maik|Frankel|33448|5|b|Klausen
Frank|Michel|33448|5|b|Klausen
Marianne|Pilz|33448|10|b|Timmer
sqlite>
```

Datensätze anzeigen nach Sortierung

Mit einer genaueren Bestimmung des SELECT-Befehls können die Datensätze in einer vom Benutzer bestimmten Reihenfolge ausgegeben werden.

SQL-Befehl:

```
SELECT * FROM tabellenname ORDER BY feldname;
```

(Sortiert alphabetisch aufsteigend nach den angegebenen Feldnamen)

SELECT * FROM tabellenname ORDER BY feldname DESC;

(Sortiert alphabetisch absteigend nach dem angegebenen Feldnamen)

Tabellen anzeigen

Mit dem Steuerbefehl `.tables` werden alle in der Datenbank angelegten Tabellen angezeigt.

Hier eine Tabelle „schueler“.

```
Terminal — sqlite3 — 66x5
SQLite version 3.4.0
Enter ".help" for instructions
sqlite> .tables
schueler
sqlite> |
```

Aufbau einer Tabelle

Mit dem Steuerbefehl `.schema TABELLENNAME` wird der Aufbau einer Tabelle angezeigt inkl. der Feldnamen und Felddatentypen in Form eines CREATE-Befehls.

```
Terminal — sqlite3 — 77x5
sqlite> .schema schueler
CREATE TABLE schueler (vorname VARCHAR(10), nachname VARCHAR(10), schuelernr
INT, klassestufe INT, klassenbezeichnung VARCHAR(1), klassenlehrer VARCHAR(3)
);
sqlite> |
```

Ausgabe formatieren

Anzeige im Spaltenformat

Mit dem Steuerbefehl `.mode column TABELLENNAME` wird der Inhalt einer Tabelle im Spaltenformat angezeigt.

```
Terminal — sqlite3 — 82x7
sqlite> .mode column schueler
sqlite> select * from schueler;
Thomas   Decke   34526   7       b       Meiermann
Maik     Franke  33448   5       b       Klausen
Frank    Michel  33448   5       b       Klausen
Marianne Pilz    33448   10      b       Timmer
sqlite> |
```

Spaltenüberschriften einblenden

Mit dem Steuerbefehl `.headers ON` werden die Spaltenüberschriften bei der Anzeige im Spaltenformat mit angezeigt.

```
Terminal — sqlite3 — 82x9
sqlite> .headers on
sqlite> select * from schueler;
vorname    nachname    schuelernr  klassestufe  klassenbezeichnung  klassenlehrer
-----
Thomas     Decke       34526       7            b                    Meiermann
Maik       Franke      33448       5            b                    Klausen
Frank      Michel      33448       5            b                    Klausen
Marianne   Pilz        33448       10           b                    Timmer
sqlite>
```

Spaltenbreiten festlegen

Mit dem Steuerbefehl `.width SPALTENBREITE 1 SPALTENBREITE 2 ...` wird die Breite jeder Tabellenspalte festgelegt, wenn man nicht mit der automatisch vergebenen Spaltenbreite zufrieden ist.

```
Terminal — sqlite3 — 82x9
sqlite> .width 10 10 5 2 1 15
sqlite> select * from schueler;
vorname    nachname    schue kl k klassenlehrer
-----
Thomas     Decke       34526 7 b Meiermann
Maik       Franke      33448 5 b Klausen
Frank      Michel      33448 5 b Klausen
Marianne   Pilz        33448 10 b Timmer
sqlite>
```

Datensatz ändern/aktualisieren

SQL-Befehl:

`UPDATE tabellenname SET feld=neuer_wert where feld=alter_wert; oder`

`UPDATE tabellenname SET feld=neuer_wert where feld=bezugswert;`

In diesem Beispiel wurde der Schülernachname falsch eingetragen anstatt „Deckel“ wurde „Decke“ eingetragen, was nun geändert werden soll.

```
Terminal — sqlite3 — 82x9
sqlite> update schueler set nachname="Deckel" where nachname="Decke";
sqlite> select * from schueler;
vorname    nachname    schue kl k klassenlehrer
-----
Thomas     Deckel      34526 7 b Meiermann
Maik       Franke      33448 5 b Klausen
Frank      Michel      33448 5 b Klausen
Marianne   Pilz        33448 10 b Timmer
sqlite>
```

Datensatz löschen

SQL-Befehl:

`DELETE from tabellenname;`

(löscht alle Datensätze der angegebenen Tabelle)

`DELETE from tabellenname where feld=wert;`

(löscht nur bestimmte Datensätze, die dem Löschkriterium entsprechen)

In diesem Beispiel wird der Schüler mit der Schülernummer 33448 gelöscht. Da es mehrere Schüler mit dieser Nummer gibt, wurden drei Datensätze gelöscht.

```
Terminal — sqlite3 — 59x6
sqlite> delete from schueler where schuelernr=33448;
sqlite> select * from schueler;
vorname    nachname   schue kl  k  klassenlehrer
-----
Thomas     Deckel     34526 7   b  Meiermann
sqlite>
```

Tabellenfeld/-spalte einfügen

SQL-Befehl:

ALTER TABLE tabellenname ADD feldname felddatentyp;

In diesem Beispiel soll in der Schülertabelle noch ein Feld „groesse“ ergänzt werden, welches die Größe in cm aufnimmt und den Datentyp INT (Integer) bekommen soll.

```
Terminal — sqlite3 — 66x6
sqlite> alter table schueler add groesse int;
sqlite> select * from schueler;
vorname    nachname   schue kl  k  klassenlehrer  groesse
-----
Thomas     Deckel     34526 7   b  Meiermann
sqlite>
```

Tabelleninhalte exportieren

Für eine Datensicherung oder eine Übertragung von Daten in ein anderes Programm oder auf einen weiteren PC ist es notwendig, dass man die Datensätze per Export aus seiner Datenbank bekommt, um sie danach weiterverarbeiten zu können.

Die Exportdaten können nur dann sinnvoll weiterverarbeitet werden, wenn die importierende Stelle die gleiche „Sprache“ spricht, d.h. sie muss das Exportformat der Daten ebenfalls lesen bzw. verarbeiten können.

Das CSV-Format

Fast jedes Datenbankprogramm beherrscht das CSV-Format. Das Dateiformat CSV beschreibt den Aufbau einer Textdatei zur Speicherung oder zum Austausch einfach strukturierter Daten. Die Dateiendung CSV ist eine Abkürzung für Comma-Separated Values (seltener Character Separated Values oder Colon Separated Values). Ein allgemeiner Standard für das Dateiformat CSV existiert nicht.

Innerhalb der Textdatei haben einige Zeichen eine Sonderfunktion zur Strukturierung der Daten.

- Ein Zeichen wird zur Trennung von Datensätzen benutzt. Dies ist in der Regel der Zeilenumbruch des datei-erzeugenden Betriebssystems – damit sind es in der Praxis oft tatsächlich zwei Zeichen.

- Ein Zeichen wird zur Trennung von Datenfeldern(Spalten) innerhalb der Datensätze benutzt. Allgemein wird dafür das Komma eingesetzt. Abhängig von beteiligter Software und Benutzereinstellungen sind auch Semikolon, Doppelpunkt, Tabulator, Leerzeichen und andere Zeichen üblich.
- Um Sonderzeichen innerhalb der Daten nutzen zu können (z. B. Komma in Dezimalzahlwerten), wird ein Feldbegrenzerzeichen benutzt. Normalerweise ist dieser Feldbegrenzer das Doppelhochkomma ". Wenn der Feldbegrenzer selbst in den Daten enthalten ist, wird dieser im Datenfeld verdoppelt.

Der erste Datensatz kann ein Kopfdatensatz sein, der die Spaltennamen definiert.

SQLite Vorbereitungen

Für den Datenexport sieht das Programm SQLite keine eigenständige Exportfunktion vor, die darauf schließen lässt, dass man mit ihr Daten sichern könnte. Man behilft sich hierbei mit der Umleitung des Ausgabekanals von der Standardausgabe in eine externe Datei. Hierzu ein Blick in die Hilfe:

```

Terminal — sqlite3 — 79x33

sqlite> .help
.bail ON|OFF          Stop after hitting an error. Default OFF
.databases             List names and files of attached databases
.dump ?TABLE? ...     Dump the database in an SQL text format
.echo ON|OFF           Turn command echo on or off
.exit                 Exit this program
.explain ON|OFF        Turn output mode suitable for EXPLAIN on or off.
.header(s) ON|OFF     Turn display of headers on or off
.help                 Show this message
.import FILE TABLE   Import data from FILE into TABLE
.indices TABLE        Show names of all indices on TABLE
.mode MODE ?TABLE?    Set output mode where MODE is one of:
                      csv      Comma-separated values
                      column   Left-aligned columns. (See .width)
                      html     HTML <table> code
                      insert    SQL insert statements for TABLE
                      line      One value per line
                      list      Values delimited by .separator string
                      tabs      Tab-separated values
                      tcl       TCL list elements
.nullvalue STRING     Print STRING in place of NULL values
.output FILENAME       Send output to FILENAME
.output stdout         Send output to the screen
.prompt MAIN CONTINUE Replace the standard prompts
.quit                 Exit this program
.read FILENAME         Execute SQL in FILENAME
.schema ?TABLE?        Show the CREATE statements
.separator STRING      Change separator used by output mode and .import
.show                 Show the current values for various settings
.tables ?PATTERN?      List names of tables matching a LIKE pattern
.timeout MS            Try opening locked tables for MS milliseconds
.width NUM NUM ...     Set column widths for "column" mode
sqlite>
  
```

Man findet hier zwei Einträge die interessant werden können.

⇒ .output FILENAME

⇒ .output STDOUT

Grundsätzlich steht .output immer auf STDOUT, da es sich bei dem Eintrag STDOUT um die Ausgabe auf dem Monitor handelt. Diesen Eintrag werden wir im weiteren Verlauf abändern.

Weitere wichtige Einträge sind natürlich

⇒ .mode

⇒ .separator

Da wir in das CSV-Format exportieren wollen, müssen wir

das Format auf jeden Fall umstellen und mit .separator legen wir das Datentrennzeichen fest, welches die Feldinhalte voneinander abgrenzen soll.

Als letzten Schritt müssen wir uns einen Dateinamen mit der Endung .csv überlegen, unter dem die Exportdaten abgelegt werden sollen.

Der Export

1. Eingabe: .mode csv TABELLENNAME ↵ (hier: TABELLENNAME=schueler)
2. Eingabe: .separator ; ↵

3. Eingabe: `.output DATEINAME.csv` ↵ (hier: `export_schueler.csv`)
4. Eingabe: `SELECT * FROM schueler;` ↵

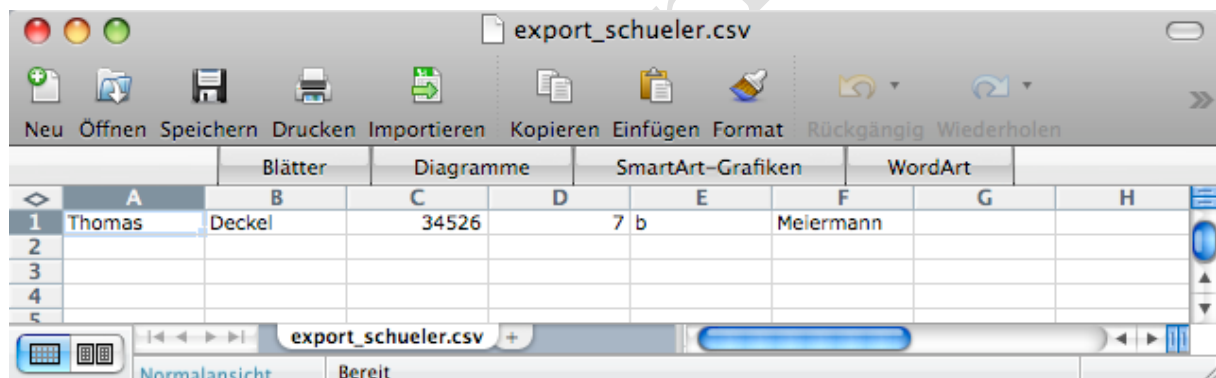
Nachdem man die Befehle der Reihenfolge nach ausgeführt hat, erhält man nach Befehl Nr.4 keine Ausgabe mehr auf dem Bildschirm. Es blinkt nur noch der SQL-Prompt (`sqlite>`)

```
Terminal — sqlite3 — 80x5
sqlite> .mode csv schueler
sqlite> .separator ;
sqlite> .output export_schueler.csv
sqlite> select * from schueler;
sqlite>
```

Die Kontrolle

Da man auf dem Bildschirm keine Kontrollinformation angezeigt bekommt, ob der Export geklappt hat, ist es notwendig sich die Datei anzusehen, die wir für den Export bestimmt hatten (`export_schueler.csv`)

Die angegebene Datei befindet sich in dem Pfad, in dem sich auch unsere `sqlite3.exe` befindet, die wir zum Beginn dieses Tutorials aufgerufen haben. Mit dem Explorer oder Dateimanager können wir den Pfad anzeigen und mit einem Tabellenkalkulationsprogramm (MS Excel, OO Calc) können wir die Datei öffnen.



Befinden sich jedes Datum in einer eigenen Zelle hat der Export geklappt und die Daten wurden richtig getrennt und übermittelt.

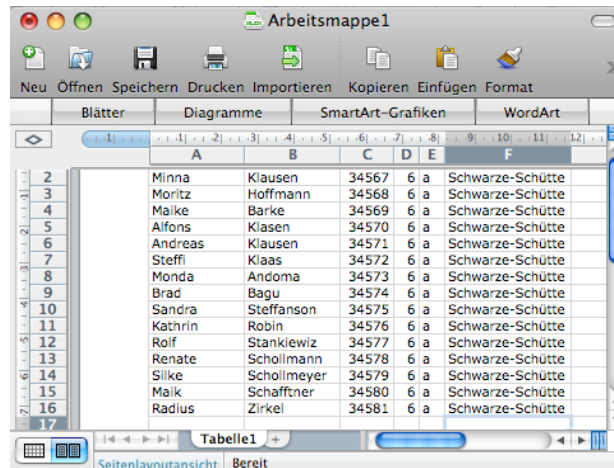
Nach dem Exportvorgang sollte man `.output` wieder auf `STDOUT` umstellen, da sonst auch die nächsten `SELECT`-Anweisungen und alle weiteren Befehlsausgaben in die Datei umgeleitet werden.

Daten importieren

Um Daten auf einem neuen Rechnersystem nutzen zu können ist es notwendig die Daten über eine Importfunktion zu übertragen, um nicht alle Informationen neu eingeben zu müssen. Wir bereiten uns in diesem Beispiel eine Excel-Tabelle vor uns speichern sie im CSV-Format. Dabei müssen wir darauf achten, dass die Reihenfolge der Spalten mit der Reihenfolge der Importtabelle übereinstimmt. In unserem Fall „Vorname, Nachname, Schuelernummer, Klassenstufe, Klassenbuchstabe, Klassenlehrer“.

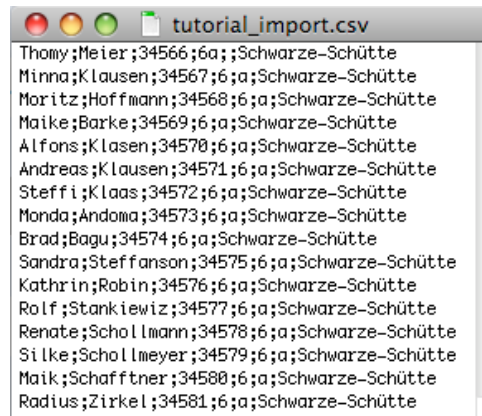
Siehe Bildfolge:

Abbildung 1: Daten in Excel vor dem Speichern



	A	B	C	D	E	F
2	Minna	Klausen	34567	6	a	Schwarze-Schütte
3	Moritz	Hoffmann	34568	6	a	Schwarze-Schütte
4	Maika	Barke	34569	6	a	Schwarze-Schütte
5	Alfons	Klasen	34570	6	a	Schwarze-Schütte
6	Andreas	Klausen	34571	6	a	Schwarze-Schütte
7	Steffi	Klaas	34572	6	a	Schwarze-Schütte
8	Monda	Andoma	34573	6	a	Schwarze-Schütte
9	Brad	Bagu	34574	6	a	Schwarze-Schütte
10	Sandra	Steffanson	34575	6	a	Schwarze-Schütte
11	Kathrin	Robin	34576	6	a	Schwarze-Schütte
12	Rolf	Stankiewicz	34577	6	a	Schwarze-Schütte
13	Renate	Schollmann	34578	6	a	Schwarze-Schütte
14	Silke	Schollmeyer	34579	6	a	Schwarze-Schütte
15	Maik	Schaffner	34580	6	a	Schwarze-Schütte
16	Radius	Zirkel	34581	6	a	Schwarze-Schütte

Abbildung 2: Daten im Editor nach dem Speichern



```
tutorial_import.csv
Thomy;Meier;34566;6;a;;Schwarze-Schütte
Minna;Klausen;34567;6;a;Schwarze-Schütte
Moritz;Hoffmann;34568;6;a;Schwarze-Schütte
Maika;Barke;34569;6;a;Schwarze-Schütte
Alfons;Klasen;34570;6;a;Schwarze-Schütte
Andreas;Klausen;34571;6;a;Schwarze-Schütte
Steffi;Klaas;34572;6;a;Schwarze-Schütte
Monda;Andoma;34573;6;a;Schwarze-Schütte
Brad;Bagu;34574;6;a;Schwarze-Schütte
Sandra;Steffanson;34575;6;a;Schwarze-Schütte
Kathrin;Robin;34576;6;a;Schwarze-Schütte
Rolf;Stankiewicz;34577;6;a;Schwarze-Schütte
Renate;Schollmann;34578;6;a;Schwarze-Schütte
Silke;Schollmeyer;34579;6;a;Schwarze-Schütte
Maik;Schaffner;34580;6;a;Schwarze-Schütte
Radius;Zirkel;34581;6;a;Schwarze-Schütte
```

Um nun die Werte importieren zu können benötigt man lediglich zwei Befehle:

1. `.separator ;`
2. `.import DATEINAME TABELLENNAME` (hier `.import tutorial_import.csv schueler`)

Nach der Eingabe blinkt wieder der SQLITE-Prompt.

Mit `SELECT * FROM schueler;` können wir überprüfen, ob der Import geklappt hat.