Installation Pakete

SuSE, Debian: `sqlite3`

CentOS: `sqlite`

Unix Benutzer in SQLite Datenbank überführen

Ein einfaches Beispiel als Schnell-Einstieg in SQL

SQLite erstmals aufrufen

```
sqlite3 accounts.db
.help
.quit
```

Bei der ersten Schreiboperation wird die Datenbank-Datei erzeugt, z.B:

CREATE und INSERT

Tabelle für Benutzer anlegen

Aufbau gemäß `/etc/passwd` :

```
sqlite3 accounts.db <<SQL

CREATE TABLE users (
 uid text,
 password text,
 uidNumber INTEGER,
 gidNumber INTEGER,
 gecos text,
 home text,
 shell text
);

SQL
```

Benutzer importieren

Schrittweise Aufbau eines komplexen Einzeilers

Einlesen einer Zeile mit `read`

```
read line < /etc/passwd
echo $line
```

Auftrennen der Zeile in Einzelwerte mit Variable `IFS`

```
IFS=':' read uid password uidNumber gidNumber gecos home shell < /etc/passwd
echo $uid $uidNumber $shell
```

SQL INSERT Statement zum Einfügen der Daten in die Tabelle `users`:

```
cat <<SQL

INSERT INTO users ( uid, password, uidNumber, gidNumber, gecos, home, shell
)
  VALUES ( '$uid', '$password', '$uidNumber', '$gidNumber', '$gecos',
'$home', '$shell' );

SQL
```

Mit ''while''-Schleife alle Benutzer importieren

Der Einzeiler ist der Übersicht halber auf mehrere Zeilen verteilt:

```
while IFS=':' read uid password uidNumber gidNumber gecos home shell;
do
  echo "
    INSERT INTO users ( uid, password, uidNumber, gidNumber, gecos, home,
shell )
    VALUES ( '$uid', '$password', '$uidNumber', '$gidNumber', '$gecos',
'$home', '$shell' );
  "
done < /etc/passwd | sqlite3 accounts.db
```

Tabellen für Gruppen anlegen

Für die `/etc/group` werden zwei Tabellen benötigt:

```
sqlite3 accounts.db <<SQL

CREATE TABLE groups ( gid text, password text, gidNumber INTEGER );
```

```
CREATE TABLE members ( gidNumber INTEGER, uid text );

SQL
```

Gruppen importieren

Genauso wie Benutzer importieren:

```
while IFS=':' read gid password gidNumber members;
do
  echo "
    INSERT INTO groups ( gid, password, gidNumber )
      VALUES ( '$gid', '$password', '$gidNumber' );
  "
done < /etc/group | sqlite3 accounts.db
```

Gruppenmitglieder importieren

Gruppenmitglieder sind etwas komplizierter. In einer Zeile der `/etc/group` können mehrere Mitglieder auftauchen. In der Tabelle `members` hat aber jedes Mitglied seine eigene Zeile. Eine Mitgliederliste in der `/etc/group` hat z.B. folgende Form: `hans,klaus,franz`. Mit

```
while IFS=':' read gid password gidNumber members;
do
  echo $members
done < /etc/group
```

können wir diese Liste in die Variable $members einlesen. Mit

```
while IFS=':' read gid password gidNumber members;
do
  echo $members | tr ',' ' '
done < /etc/group
```

werden die Kommata in der Liste durch Leerzeichen ersetzt. Mit zwei verschachtelten Schleifen ...

```
while IFS=':' read gid password gidNumber members;
do
  for uid in $(echo $members | tr ',' ' ');
  do
    echo "INSERT INTO members ( gidNumber, uid ) VALUES ( '$gidNumber',
'$uid' );"
  done
done < /etc/group | sqlite3 accounts.db
```

... können die Mitglieder dann einer nach dem anderen in die Variable `$member` eingelesen und zusammen mit der `$gidNumber` in die Tabelle `members` geschrieben werden.

Das Skript in einer Datei

unix_users2sql.sh

```
#!/bin/bash

# Tabelle für Benutzer anlegen
sqlite3 accounts.db <<SQL
  CREATE TABLE users (
    uid text,
    password text,
    uidNumber integer,
    gidNumber integer,
    gecos text,
    home text,
    shell text
  );
SQL

# Benutzer importieren
while IFS=':' read uid password uidNumber gidNumber gecos home shell;
do
  cat <<SQL
    INSERT INTO users ( uid, password, uidNumber, gidNumber, gecos,
home, shell )
    VALUES ( '$uid', '$password', '$uidNumber', '$gidNumber', '$gecos',
'$home', '$shell' );
SQL
done < /etc/passwd | sqlite3 accounts.db

## Tabellen für Gruppen anlegen
sqlite3 accounts.db <<SQL
  CREATE TABLE groups ( gid text, password text, gidNumber integer );
  CREATE TABLE members ( gidNumber integer, uid text );
SQL

## Gruppen importieren
while IFS=':' read gid password gidNumber members;
do
  cat <<SQL
    INSERT INTO groups ( gid, password, gidNumber )
      VALUES ( '$gid', '$password', '$gidNumber' );
SQL
done < /etc/group | sqlite3 accounts.db

## Gruppenmitglieder importieren
while IFS=':' read gid password gidNumber members;
do
  for uid in $(echo $members | tr ',' ' ');
  do
```

```
    echo "INSERT INTO members ( gidNumber, uid ) VALUES ( '$gidNumber',
'$uid' );"
  done
done < /etc/group | sqlite3 accounts.db
```

Der fertige SQL-Dump

Beispiel für eine `/etc/passwd` :

```
root:x:0:0:root:/root:/bin/bash
system1:x:1:1:System Benutzer 1:/bin:/sbin/nologin
system2:x:2:2:System Benutzer 2:/bin:/sbin/nologin
nutzer1:x:500:100:nutzer1:/home/nutzer1:/bin/bash
nutzer2:x:501:100:nutzer2:/home/nutzer2:/bin/sh
nutzer3:x:502:100:nutzer3:/home/nutzer3:/bin/bash
```

Beispiel für eine `/etc/group` :

```
root:x:0:root
system-group1:x:1:
system-group2:x:2:
nogroup:x:4:
users:x:100:
projekt-x:x:101:nutzer1,nutzer3
nutzer1:x:500:
nutzer2:x:501:
nutzer3:x:502:
```

Mit den oben gezeigten Skripten wird daraus folgendes SQL-Datei:

```
CREATE TABLE users (
 uid text,
 password text,
 uidNumber INTEGER,
 gidNumber INTEGER,
 gecos text,
 home text,
 shell text
);
INSERT INTO "users" VALUES('root', 'x', 0, 0, 'root', '/root', '/bin/bash');
INSERT INTO "users" VALUES('system1', 'x', 1, 1, 'bin', '/bin',
'/sbin/nologin');
INSERT INTO "users" VALUES('system2', 'x', 2, 2, 'bin', '/bin',
'/sbin/nologin');
INSERT INTO "users" VALUES('nutzer1', 'x', 500, 100, 'nutzer1',
'/home/nutzer1', '/bin/bash');
INSERT INTO "users" VALUES('nutzer2', 'x', 501, 100, 'nutzer2',
'/home/nutzer2', '/bin/sh');
```

```
INSERT INTO "users" VALUES('nutzer3', 'x', 502, 100, 'nutzer3',
'/home/nutzer3', '/bin/bash');
CREATE TABLE groups ( gid text, password text, gidNumber INTEGER );
INSERT INTO "groups" VALUES('root', 'x', 0);
INSERT INTO "groups" VALUES('system-group1', 'x', 1);
INSERT INTO "groups" VALUES('system-group2', 'x', 2);
INSERT INTO "groups" VALUES('nogroup', 'x', 4);
INSERT INTO "groups" VALUES('users', 'x', 100);
INSERT INTO "groups" VALUES('projekt-x', 'x', 101);
INSERT INTO "groups" VALUES('nutzer1', 'x', 500);
INSERT INTO "groups" VALUES('nutzer2', 'x', 501);
INSERT INTO "groups" VALUES('nutzer3', 'x', 502);
CREATE TABLE members ( gidNumber INTEGER, uid text );
INSERT INTO "members" VALUES(0, 'root');
INSERT INTO "members" VALUES(101, 'nutzer1');
INSERT INTO "members" VALUES(101, 'nutzer3');
```

Benutzerdatenbank abfragen

```
sqlite3 accounts.db
```

Alle weiteren SELECT-Befehle beziehen sich auf die `sqlite3`-Shell

einfache SELECT Abfragen

Alle Benutzer in ''/etc/passwd'' ausgeben

```
SELECT * FROM users;
```

analog zu

```
cat /etc/passwd
```

Benutzer root ausgeben

```
SELECT * FROM users WHERE uid = 'root';
```

analog zu

```
grep '^root' /etc/passwd
```

Shell des Benutzers root ausgeben

```
SELECT shell FROM users WHERE uid = 'root';
```

analog zu

```
grep '^root' /etc/passwd | cut -d: -f 7
```

Benutzer nach Name sortiert ausgeben

```
SELECT * FROM users ORDER BY uidNumber;
```

analog zu

```
sort -n -t: -k3 /etc/passwd
```

Anzahl der Benutzer ausgeben

```
SELECT COUNT(*) FROM users;
```

analog zu

```
wc -l /etc/passwd
```

Wie viele Benutzer verwenden welche Shell ?

```
SELECT shell, COUNT(shell) FROM users GROUP BY shell;
```

analog zu

```
cut -d : -f 7 /etc/passwd|sort|uniq -c
```

JOIN

Alle Benutzer, ihre Passwörter und ihre primäre Gruppe namentlich ausgeben

traditioneller JOIN

```
SELECT uid,users.password,uidNumber,gid,gecos
  FROM users, groups
  WHERE users.gidNumber = groups.gidNumber;
```

JOIN

Und noch mal mit JOIN:

```
SELECT uid, users.password, gid
  FROM users
  JOIN groups
  WHERE users.gidNumber = groups.gidNumber;
```

JOIN ON

Das selbe mit JOIN ON:

```
SELECT uid, users.password, gid
  FROM users
  JOIN groups
  ON users.gidNumber = groups.gidNumber;
```

JOIN USING

Das selbe mit JOIN USING:

```
SELECT uid, users.password, gid
  FROM users
  JOIN groups
  USING ( gidNumber );
```

NATURAL JOIN

Was ähnliches mit NATURAL JOIN:

```
SELECT *
  FROM users
  NATURAL JOIN groups;
```

Subselect

Den Durchschnitt der Gruppennummern aus der Tabelle `users` berechnen:

```
  SELECT avg(gidNumber) FROM users;
```

Alle Gruppen, deren Nummer größer als der Durchschnitt ist:

```
  SELECT gid FROM groups WHERE gidNumber > ( SELECT avg(gidNumber) FROM
```

```
users );
```

Join mit Hilfe von Subselect:

```
SELECT users.uid
  FROM users
  WHERE gidNumber IN (
    SELECT gidNumber
      FROM groups
      WHERE gid = 'root'
  );
```

Bei allen Benutzern mit Benutzernummern zwischen 500 und 10000 die Benutzernummer um 500 erhöhen:

```
UPDATE users
  SET uidNumber = uidNumber + 500
  WHERE uidNumber >= 500 AND uidNumber <10000;
```

Datenbank verändern

Datensätze verändern

Zeile hinzufügen

Gruppe mit neuer uidNumber hinzufügen

```
INSERT INTO groups (gid, password, gidNumber) VALUES ( 'projekt-y', 'x', (
SELECT MAX(gidNumber) FROM groups WHERE gidNumber < 65000)+1);
```

Zeile kopieren

```
INSERT INTO groups SELECT * FROM groups WHERE gid = 'projekt-y';
```

Zeile verändern

```
UPDATE groups SET gid = 'projekt-z' WHERE gid = 'projekt-y';
```

Zeile löschen

```
DELETE FROM groups WHERE gid = 'projekt-y';
```

Spalte hinzufügen

Übung

Schreiben sie Skripte mit SQL Anweisungen, die als Ersatz für die Linuxbefehle

```
id
useradd
userdel
groupadd
groupdel
```

dienen können. Vielleicht erst mal ohne die vielen Optionen, die diese Befehle beherrschen.

Erweitern Sie die Datenbank, so daß sie auch einen Ersatz für den Befehl

```
passwd
```

schreiben können.

Dokumentation

- http://www.sqlite.org/lang.html
- (Quelle)