Modul 105: Datenbanken mit SQL bearbeiten

SQL Crashkurs IV: CREATE TABLE & Datenimport

Ausgangslage

Wir verlassen nun die uns bekannte Schülerdatenbank. Wir starten in diesem Arbeitsblatt mit einer leeren Datenbank und wollen die Daten, die sich in csv-Dateien befinden, via Konsole importieren. Danach werden wir Datensätze erfassen, ändern und löschen und diverse Abfragen durchführen.

Ziele

- CREATE TABLE-Befehl in Theorie und Praxis
- Bulk-Insert durchführen
- Praktische Übungen mit der Data Manipulation Language DML

Vorgehen

Gehen Sie auf der *bmLP1* ins Verzeichnis *Modul105* und erstellen Sie das neue Unterverzeichnis *AB100-11*. Kopieren Sie die drei Dateien *AB105-06_Kunden.csv*, *AB105-06_Artikel.csv* und *AB105-06_Kauf.csv* von Moodle in dieses Verzeichnis.

Gehen Sie dieses Arbeitsblatt Schritt für Schritt durch. Konsultieren Sie die Unterlagen, suchen Sie zusätzliche Informationen im Internet oder fragen Sie Ihre Lehrperson.

Unterlagen / Quellen

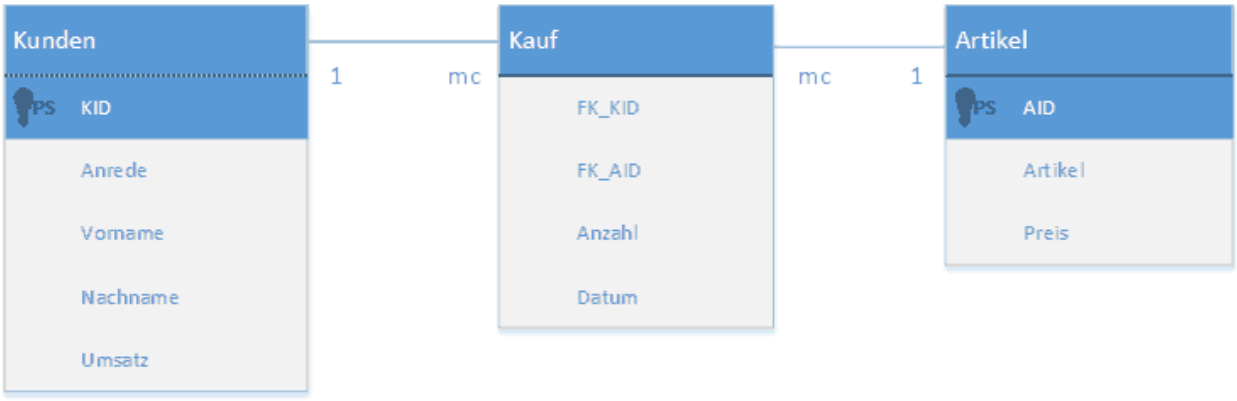
- SQL Referenz auf Moodle: SQL-Referenz.pdf
- Allgemeine SQL-Tutorials:
  - https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL
  - https://www.w3schools.com/sql
- SQLite-spezifisches Tutorial:
  - https://www.tutorialspoint.com/sqlite/

Zeit: 4 Lektionen

Kunden-Artikel-Datenbank

In der Datenbank gibt es nur zwei Entitäten mit einer m:n-Beziehung: Kunden-Artikel. Jeder Kunde kann 0, 1 oder mehrere Artikel kaufen. Jeder Artikel wird von 0, 1 oder mehreren Kunden gekauft. Wie wir gelernt haben, brauchen wir für die Abbildung einer m:n-Beziehung in einer relationalen Datenbank eine Zwischentabelle. In dieser Zwischentabelle wird der Kauf der Artikel durch die Kunden festgehalten.

Das logische Datenmodell sieht wie folgt aus:



CREATE TABLE

Der Befehl CREATE TABLE gehört zur SQL Data Definition Language DDL. Es ist der einzige DDL-Befehl, den wir in diesem Modul anschauen, um einen Import durchführen zu können. Die weiteren DDL-Befehle sind Bestandteil des Moduls 104.

```
CREATE TABLE <Tabelle> (
   <Attribut1> <Datentyp>,
   <Attribut2> <Datentyp>,
   <Attribut3> <Datentyp>,
   usw.
);
```

Attribut1

In den meisten Fällen handelt es sich beim Attribut 1 um den Primärschlüssel. In diesem Fall sieht die erweiterte Syntax wie folgt aus:

```
CREATE TABLE <Tabelle> (
   <PrimaryKey> INTEGER PRIMARY KEY [AUTOINCREMENT],
   <Attribut2> <Datentyp>,
   <Attribut3> <Datentyp>,
   usw.
);
```

<PrimaryKey>

Name des Primärschlüssels. Wir wollen uns – wie bereits an anderer Stelle erwähnt – wenn möglich an folgenden Standard halten: Anfangsbuchstabe der Tabelle plus *ID*. Beispiel: Tabelle *Orte*, der Primärschlüssel heisst *OID*.

<Attribut2>, <Attribut3> usw.

Namen der weiteren Attribute.

Im Falle von Fremdschlüsseln wollen wir folgende Regeln anwenden: *FK_* und der Name des Primärschlüssels der Primärtabelle. Beispiel: Tabelle *Kunden* mit einem Verweis auf die Tabelle *Orte*, der Fremdschlüssel heisst *FK_OID*. Der Datentyp muss identisch sein mit demjenigen des Primärschlüssels, also INTEGER.

<Datentyp>

Datentyp	Bedeutung
NULL	Der Wert ist ein NULL-Wert.
INTEGER	Der Wert ist eine vorzeichenbehaftete ganze Zahl, gespeichert in 1, 2, 3, 4, 6 oder 8 Byte in Abhängigkeit von der Grösse des Wertes.
REAL	Der Wert ist ein Fliesskommawert, gespeichert als 8-byte Gleitkommazahl IEEE.
TEXT	Der Wert ist eine Zeichenfolge, gespeichert unter Verwendung der Datenbank - Kodierung (UTF-8, UTF-16BE oder UTF-16LE).
BLOB	Nicht strukturierte Objekte (**B**inary **L**arge **Ob**jects), d.h. die Daten werden genauso gespeichert, wie sie eingegeben wurden.

Beispiel

```
CREATE TABLE Orte (
   OID INTEGER PRIMARY KEY AUTOINCREMENT,
   PLZ INTEGER,
   Ort TEXT
);
```

Datenimport (Bulk-Insert)

Ein Datenimport, bei dem eine gesamte Datei (Tabelle) auf einmal importiert wird, heisst Bulk-Insert oder BigBang-Import. Alle Daten werden mit einer einzigen Operation von der Quelle ins Ziel migriert.

Daten können in den grafischen Oberflächen von *Sqliteman* oder *DB Browser for SQLite* mit einigen Mausklicks importiert werden. Darauf verzichten wir und importieren direkt mit der Kommandozeile, da dies sehr einfach ist und später (Modul 122) in Routinen automatisiert werden kann.

Vorgehen

1. Starten Sie das Terminal auf der bmLP1.
2. Wechseln Sie ins Verzeichnis */Modul100/AB100-11*
3. Vergewissern Sie sich, dass sich die drei Dateien *AB100-11_Kunden.csv*, *AB100-11_Artikel.csv* und *AB100-11_Kauf.csv* in diesem Verzeichnis befinden. Falls dies nicht der Fall ist, ist ein Import nicht möglich bzw. muss neben dem Dateinamen der Pfad angegeben werden, was umständlich ist. Befehl dazu: *ls*.
4. Starten Sie SQLite und legen dabei die neue Datenbank *KundenArtikel.sqlite* an:
   `vmadmin@bmLP1:~/Modul100/AB100-11$ sqlite3 KundenArtikel.sqlite`
5. Erstellen Sie mit dem CREATE TABLE-Befehl die Tabelle *Kunden*:
   `sqlite> CREATE TABLE Kunden (KID INTEGER PRIMARY KEY, Anrede TEXT, Vorname TEXT, Nachname TEXT, Umsatz REAL);`
6. Erstellen Sie mit dem CREATE TABLE-Befehl die Tabelle *Artikel*:
   AID = Primärschlüssel ohne AUTOINCREMENT,
   Artikel = Textfeld für die Artikelbezeichnung,
   Preis = Gleitkommazahl

Notieren Sie den CREATE TABLE-Befehl dazu:

```
CREATE TABLE Artikel
(
   AID integer PRIMARY KEY,
   Artikel text,
   Preis decimal
);
```

7. Erstellen Sie mit dem CREATE TABLE-Befehl die Tabelle *Kauf*:
   FK_KID = Fremdschlüssel für die Kunden-ID,
   FK_AID = Fremdschlüssel für die Artikel-ID,
   Anzahl = Kaufmenge des Artikels,
   Datum = Datum des Kaufs (Textfeld)
   Notieren Sie den CREATE TABLE-Befehl dazu:

```
CREATE TAble Kauf
(
   FK_KID integer,
   FK_AID integer,
   Anzahl integer,
   Datum date,
   FOREIGN key (FK_KID) REFERENCES kunden(KID),
   FOREIGN KEY (FK_AID) REFERENCES artikel(AID)
);
```

8. Wir müssen SQLite sagen, wie das Format der Importdateien aussieht. In diesem Fall handelt es sich um csv-Dateien (= Textdateien), wo die Attributwerte mit einem Semikolon ";" getrennt sind. Wir geben dies in SQLite wie folgt an:
   `sqlite>.separator ";"` (den Punkt vor *separator* nicht vergessen!)
9. Importieren Sie die Daten in die Tabelle *Kunden*:
   `sqlite>.import AB105-06_Kunden.csv Kunden`
   (den Punkt vor *import* nicht vergessen!)
10. Importieren Sie die Daten in die Tabelle *Artikel*:
    `sqlite>.import AB105-06_Artikel.csv Artikel`
11. Importieren Sie die Daten in die Tabelle *Kauf*:
    `sqlite>.import AB105-06_Kauf.csv Kauf`
12. Wir machen einen weiteren Testimport, um zu demonstrieren was passiert, wenn eine Tabelle nicht vorhanden ist:
    `sqlite>.import AB105-06_Kunden.csv Testtabelle`

Sqliteman

Der Import ist nun abgeschlossen. Wir wollen das Ergebnis mit dem *Sqliteman* verifizieren.

Prüfen Sie bei den drei Tabellen *Kunden*, *Artikel* und *Kauf* folgendes:

- Inhalt der Tabelle (Doppelklick auf den Tabellennamen)
- Wie viele Datensätze sind in den Tabellen vorhanden:
  Kunden _____
  Artikel _____
  Kauf _____
- Attributnamen (Doppelklick auf den Tabellennamen)

- CREATE TABLE-Befehle
  (rechte Maustaste auf den Tabellennamen und *Tabelle beschreiben* auswählen)

Vergleiche Sie nun die Tabellen *Kunden* und *Testtabelle*, was fällt Ihnen auf bezüglich:

- Attributnamen

- Datentypen

- Anzahl Datensätzen
  Tabelle Kunden:
  Tabelle Testtabelle:

Schlussfolgerung: Wenn eine Tabelle beim Import nicht vorhanden ist, wird sie automatisch angelegt. Als Attributnamen wird der 1. Datensatz verwendet. Es gibt keinen Primärschlüssel, alle Attribute erhalten den Datentyp TEXT.
Eine solche Tabelle ist unbrauchbar für uns!

Unsere Regeln für den Import

Aus dem Ergebnis leiten wir ein paar Regeln ab, damit die Imports in Zukunft fehlerfrei verlaufen.

1. Vergewissern Sie sich, dass sich die zu importierenden Tabellen im aktuellen Verzeichnis befinden, **bevor** Sie SQLite starten.
2. Erstellen Sie alle Tabellen mit CREATE TABLE.
3. Überprüfen Sie, ob alle Tabellen vorhanden sind:
   `sqlite>.tables`
4. Überprüfen Sie die einzelnen Tabellendefinitionen:
   `sqlite>.schema <Tabellenname>` oder
   `sqlite>.fullschema`
5. Führen Sie den Import durch. Falls Sie die Fehlermeldung *Error: cannot open "AB105-05_Tabelle.csv"* erscheint:
   a) Sie haben entweder den Dateinamen falsch geschrieben oder die Datei befindet sich nicht im aktuellen Verzeichnis.
   b) Verlassen Sie in diesem Fall die SQLite-Konsole mit `.quit`
   c) Prüfen Sie den Dateinamen bzw. ob sich die Datei im aktuellen Verzeichnis befindet.
   d) Starten Sie den Importvorgang wieder mit der Regel 1.

   Falls andere Fehlermeldungen erscheinen, dann stimmt vermutlich die Tabellendefinition nicht mit den zu importierenden Daten überein. Weil z.B. zu viele oder zu wenig Attribute vorhanden sind oder weil der Datentyp nicht passt. Gehen Sie in diesem Fall wie folgt vor:
   a) Löschen Sie die betroffene Tabelle mit DROP TABLE <Tabellenname>.
   b) Prüfen Sie in der zu importierenden csv-Datei die Anzahl Attribute und den Datentyp.
   c) Erstellen Sie die Tabelle mit dem CREATE TABLE-Befehl.

   d) Wiederholen Sie den Import.

6. Falls der Import erfolgreich (d.h. ohne Fehlermeldung) durchgeführt werden konnte, überprüfen Sie nochmals die vorhandenen Tabellen:
   `sqlite>.tables`
   Falls es mehr Tabellen gibt als bei der ersten Überprüfung:
   a) Sie haben einen Tabellennamen falsch geschrieben oder die Tabelle war nicht vorhanden.
   b) Finden Sie heraus, um welche Tabelle(n) es sich handelt:
      `sqlite>.schema <Tabellenname>` oder
      `sqlite>.fullschema`
   c) Löschen Sie die betroffene(n) Tabelle(n) mit DROP TABLE <Tabellenname>.
   d) Gehen Sie entweder zurück zu Punkt 2 (CREATE TABLE) oder zu Punkt 5 (Import) und schreiben Sie den Tabellennamen korrekt.
7. Überprüfen Sie die Inhalte aller Tabellen mit dem SELECT-Befehl:
   `sqlite>SELECT * FROM <Tabellenname>`
8. Falls alle Prüfungen erfolgreich durchgeführt worden sind: Schliessen Sie die Konsole und überprüfen Sie das Ergebnis nochmals mit *Sqliteman*.

Aufgaben

Lösen Sie alle Aufgaben mit *Sqliteman*. Speichern Sie sämtliche SQL-Befehle in die Datei *AB105-06.sql* ab.

Abfragen mit SELECT (1 Tabelle)

Geben Sie die Datensätze gemäss folgenden Vorgaben aus:

Aufgabe 1

Alle Datensätze und alle Attribute der Tabelle Kunden.

```
select * from kunden
```

Aufgabe 2

Alle Kunden mit den Attributen Nachname und Vorname, alphabetisch aufsteigend sortiert nach Nachname und Vorname.

```
select Nachname, Vorname from kunden order by Nachname, Vorname asc;
```

Aufgabe 3

Wie Aufgabe 2, aber ohne allfälligen doppelten Datensätze und ohne Sortierung.

```
select distinct Nachname, Vorname from kunden order by Nachname, Vorname asc;
```

Aufgabe 4

Alle Attribute der Kunden mit Nachname *Meier* und Vorname *Beat*. Der Nachname soll unter der Überschrift *Gesuchte Person* aufgeführt werden.

```
select Nachname as "Gesuchte Person"
from kunden
WHERE Nachname="Meier" and Vorname="Beat"
```

Aufgabe 5

Alle Kunden, die nicht *Meier* heissen.

```
SELECT * FROM `kunden` WHERE Nachname <> "Meier";
```

Aufgabe 6

Alle Kunden, deren Nachname mit *M* beginnt.

```
SELECT * FROM `kunden` WHERE Nachname LIKE "M%";
```

Aufgabe 7

Die Anzahl Kunden mit der Überschrift *Anzahl Kunden*.

```
SELECT count(*) as "Anzahl Kunden" FROM `kunden` ;
```

Aufgabe 8

Die ersten 8 Kunden.

```
SELECT * FROM `kunden` LIMIT 8;
```

Aufgabe 9

Diejenigen 5 Kunden, die am meisten Umsatz produzieren (*Umsatz* ist ein Attribut in der Tabelle *Kunden*).

```
SELECT * FROM `kunden` ORDER by Umsatz DESC limit 5;
```

Aufgabe 10

Der durchschnittliche Umsatz aller Kunden auf 2 Stellen gerundet, unter dem Titel *Durchschnittlicher Umsatz*.
Hinweis: Zum Runden können Sie die Funktion *ROUND()* verwenden.

```
SELECT ROUND(AVG(Umsatz),2) AS 'Durchschnittlicher Umsatz' FROM kunden;
```

Aufgabe 11

Die Namen aller Kunden und eine auszuzahlende Provision von 10% des jeweiligen Umsatzes mit der Überschrift *Provision*. Sortierung: Höhe der Provision absteigend.

```
SELECT Nachname, Vorname, Umsatz/100*10 AS Provision FROM kunden ORDER BY Provision DESC;
```

Aufgabe 12

Alle Kunden, bei denen der Umsatz nicht eingetragen wurde oder 0 ist.

```
SELECT * FROM kunden WHERE Umsatz IS NULL OR Umsatz = 0;
```

Einfügen, Ändern und Löschen von Datensätzen

Aufgabe 13

Fügen Sie folgenden Datensatz an: KID = *502*, Anrede = *Frau*, Vorname = *Doris*, Nachname = *Muster*.

```
INSERT INTO kunden (KID, Anrede, Vorname, Nachname) VALUES (502, 'Frau', 'Doris', 'Muster');
```

Aufgabe 14

Der Primärschlüssel sollte immer einen Wert haben. In der letzten Aufgabe wurde der Wert *502* eingefügt (der höchste bisherige Wert plus 1). Normalerweise kennen wir den höchsten Wert nicht, können diesen aber mit einem Sub-SELECT ermitteln.

Fügen Sie folgenden Datensatz mit **einem** Befehl ein (höchster Primärschlüsselwert unbekannt): KID=(zu ermitteln), Anrede = *Frau*, Vorname = *Anna*, Nachname = *Abegglen*.

```
INSERT INTO kunden (Anrede, Vorname, Nachname) VALUES ('Frau', 'Anna', 'Abegglen');
```

Aufgabe 15

Ändern Sie den Namen von *Ralph van der Mauer* zu *Ralph Maurer* und fügen Sie gleichzeitig bei ihm einen Umsatz von *2222* ein.

```
UPDATE kunden SET Nachname = 'Maurer' AND Umsatz = 2222 WHERE Nachname = 'von der Mauer' AND Vorname = 'Ralph';
```

Aufgabe 16

Verdoppeln Sie den Umsatz von allen Personen, die einen Umsatz von weniger als 1000 haben.

```
UPDATE kunden SET Umsatz = Umsatz * 2 WHERE Umsatz < 1000;
```

Aufgabe 17

Löschen Sie den Kunden bzw. die Kunden mit Nachnamen *Gallaun*.

```
DELETE FROM kunden WHERE Nachname = 'Gallaun';
```

Aufgabe 18

Löschen Sie alle Kunden, deren Nachname mit *Kl* beginnt. Geben Sie sicherheitshalber zuerst alle zu löschenden Kunden mit einem SELECT aus!

```
SELECT * FROM kunden WHERE Nachname LIKE 'Kl%';
DELETE FROM kunden WHERE Nachname LIKE 'Kl%';
```

Zusatzaufgaben

Aufgabe 19

Geben Sie alle Kunden aus, deren Anrede einen Zusatz nach *Herr* bzw. *Frau* aufweist (z.B. *Frau Dr.*).
Hinweis: Sie können dazu die Funktion *LENGTH()* verwenden.

```
SELECT * from kunden where length(Anrede) > 4;
```

Entfernen Sie bei genau diesen Kunden den Zusatz, so dass die Anrede in jedem Fall nur noch *Frau* oder *Herr* lautet.
Hinweis: Sie können dazu die Funktion *SUBSTR()* verwenden.

```
SELECT Vorname, Nachname, Substr(Anrede, 1, 4) FROM `kunden`;
```

Aufgabe 20

Welche Artikel wurden noch nie verkauft (Artikelname anzeigen)?

```
SELECT Artikel
FROM artikel
WHERE artikel.AID NOT IN (
   SELECT kauf.FK_KID
   FROM kauf
);
```

Aufgabe 21

Welche Kunden (KID, Nachname, Vorname) haben noch nie etwas gekauft?
Die Aufgabe darf nicht über den Umsatz gelöst werden!

```
SELECT *
FROM kunden
WHERE kunden.KID NOT IN (
   SELECT kauf.FK_KID
   FROM kauf
);
```

Aufgabe 22

Zeigen Sie eine Liste mit allen Artikeln, die verkauft worden sind.
Attribute: Artikelname und wie viele Stück verkauft worden sind (Spaltentitel = *Anzahl*)
Sortierung: Nach Anzahl absteigend

```
SELECT Artikel.Artikel , COUNT(kauf.FK_AID) AS Anzahl
FROM kauf
LEFT JOIN Artikel on kauf.FK_AID = Artikel.AID
GROUP BY kauf.FK_AID
ORDER BY count(kauf.FK_AID) DESC;
```

Aufgabe 23

Welche Kunden kaufen mehr als 20% über dem durchschnittlichen Umsatz?
Attribute: Nachname, Vorname, Umsatz
Sortierung: Nach Umsatz absteigend

```
zu schwer
```

Aufgabe 24

Zeigen Sie die Kunden mit den Artikeln, die sie gekauft haben.
Attribute: Vorname, Nachname, Artikel und Totalpreis (= Anzahl * Einzelpreis, Titel = *Preis Total*)
Sortierung: Vorname und Nachname aufsteigend

Aufgabe 25

Der Umsatz für jeden Kunden ist in der Tabelle *Kunden* festgehalten. Warum ist das keine gute Idee?

Wie kann der Umsatz für jeden Kunden berechnet werden?

Geben Sie alle Kunden aus, die etwas gekauft haben (jeder Kunde soll nur 1x aufgeführt werden).
Attribute: Vorname, Nachname, Umsatz der Tabelle *Kunden* (Titel = *Umsatz falsch*) und aus den Käufen berechneter Umsatz (Titel = *Umsatz korrekt*)
Sortierung: Vorname und Nachname aufsteigend