MUSTERLÖSUNG ÜBUNGSBLATT 5

Musterlösung der zweiten Laborübung Datenbanken (DBA), Studiengang IB,
Sven Klaus, s.klaus@hs-mannheim.de, http://www.informatik.hs-mannheim.de/~klaus/

AUFGABE 1

a) CREATE DATABASE dba*xx*; (*xx* entspricht Ihrer Gruppennummer 01 – 20)

b) USE dba*xx*;

c)
```
CREATE TABLE t_artikel
      (id      INTEGER          NOT NULL          PRIMARY KEY,
      name     VARCHAR(150),
      preis    FLOAT);
```

d)
```
DROP TABLE t_artikel;
DROP DATABASE testdb;
```

AUFGABE 2

a)
```
CREATE DATABASE dba xx;          (xx entspricht Ihrer Gruppennummer 01 – 20)
USE dba xx;
```

b)
```
CREATE TABLE t_person
      (id      INTEGER          NOT NULL          PRIMARY KEY,
      vname    VARCHAR(150)     NOT NULL,
      name     VARCHAR(150)     NOT NULL);
```

c) ALTER TABLE t_person ADD lebenslauf BLOB;

d) ALTER TABLE t_person ADD beschaeftigt_seit DATE;

e) ALTER TABLE t_person DROP beschaeftigt_seit;

f) DROP TABLE t_person;

AUFGABE 3

a)
```
CREATE TABLE t_lager
      (stueck  INTEGER          DEFAULT 1,
      preis    FLOAT            NOT NULL);
```

Es gibt keinen geeigneten Schlüssel. Wenn wir ehrlich sind, so ist die obige Tabelle für die Realität eher ungeeignet: *WAS* wird denn in jeder Zeile eingelagert?

Führen Sie dennoch das obige SQL Kommando aus. Es funktioniert. MySQL erlaubt sogar die Anlage einer Tabelle komplett ohne einen Schlüssel. Wird jetzt die Gesamtheit aller Attribute als Schlüssel verwendet (so wie von Codd gefordert) oder wird ein unsichtbarer Primärschlüsselfeld generiert (so wie MS Access es umsetzt)? Finden Sie es heraus!

b) ALTER TABLE t_lager ADD name VARCHAR(100);

c) INSERT INTO t_lager (name,stueck,preis) VALUES("Kugelschreiber",88,2.99);
INSERT INTO t_lager (name,stueck,preis) VALUES("Ordner",67,2.50);
INSERT INTO t_lager (name,stueck,preis) VALUES("Heftklammern",423,0.99);
INSERT INTO t_lager (name,stueck,preis) VALUES("Bleistift",88,0.99);
INSERT INTO t_lager (name,stueck,preis) VALUES("Umschläge B6",67,0.80);

Spätestens an dieser Stelle nervt SQL ein wenig.
Zum Glück arbeiten wir ja aber mit MySQL: Schauen Sie auf dev.mysql.com in das MySQL-Handbuch. Es gibt eine Möglichkeit, sich dieses ewige INSERT INTO ... zu ersparen. Ebenfalls sollten Sie einen Blick auf das SOURCE Kommando werfen.

Tipp: Wenn Sie mit (My)SQL arbeiten, so sollten Sie immer einen Texteditor geöffnet haben.

d) INSERT INTO t_lager (name,stueck,preis) VALUES("Schreibblock A4",null,1.99);

e) SELECT * FROM t_lager;

f) UPDATE t_lager SET stueck=270 WHERE name="Bleistift";

g) UPDATE t_lager SET preis=2.80 WHERE name="Ordner";

h) DELETE FROM t_lager WHERE name="Schreibblock A4";

AUFGABE 4

a) CREATE TABLE t_ma_dt
(pid INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
vname VARCHAR(100),
name VARCHAR(100),
str VARCHAR(100),
plz CHAR(5),
ort VARCHAR(100),
alt INTEGER);

Der Datentyp INTEGER ist für Postleitzahlen ungeeignet, da führende Nullen unterdrückt würden. Der Feldname *alter* musste sinnvoll abgewandelt werden, da ALTER ein SQL-Schlüsselwort darstellt.

Die Aufgabenstellung schweigt sich über einen Schlüssel aus. Wir müssen selber einen wählen, nur ist kein guter Schlüsselkandidat erkennbar. Die Lösung besteht wieder einmal in der Einführung eines künstlichen Primärschlüssels.

b) INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
VALUES('Haas','Elsbeth','Berliner Str. 223','63067','Offenbach',42);
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
VALUES('Richter','Hans','Frankfurter Str. 61','63067','Offenbach',32);

```
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
        VALUES('Friedrich','Irmgard','Goethestr. 61','63067','Offenbach',40);
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
        VALUES('Hartmann','Jochen','Berliner Str. 223','60528','Frankfurt',29);
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
        VALUES('Goldbach','Martin','Frankfurter Str. 61','60529','Frankfurt',35);
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
        VALUES('Naumann','Norbert','Goethestr. 61','60594','Frankfurt',38);
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
        VALUES('Haas','Tanja','Berliner Str. 223','30323','Hannover',36);
INSERT INTO t_ma_dt (name,vname,str,plz,ort,alt)
        VALUES('Neppe','Martin','Goethestr. 61','30324','Hannover',43);
```

Nochmal ;-) es geht auch weniger tipp-intensiv.

c)
```
CREATE TABLE t_ma_frankfurt
        (pid    INTEGER   NOT NULL   PRIMARY KEY,
        vname   VARCHAR(100),
        name    VARCHAR(100),
        str     VARCHAR(100),
        plz     CHAR(5),
        ort     VARCHAR(100);
        alt     INTEGER);

INSERT INTO t_ma_frankfurt (pid, name,vname,str,plz,ort,alt)
        SELECT pid, name,vname,str,plz,ort,alt
        FROM t_ma_dt WHERE ort='Frankfurt' or ort='Offenbach';
```

Hier lauert erneut eine versteckte Gefahr. Klar, bei dieser "Ableger-"Tabelle wurde ebenfalls ein künstlicher Primärschlüssel eingefügt. Aber warum muss dieser Wert diesmal explizit angegeben werden und ist nicht über AUTO_INCREMENT gelöst?

Nur so kann sicher gestellt werden, dass die gleiche Entität, also die gleiche Person in diesen beiden Tabellen auch die gleiche Personal-ID hat – zumindest beim Anlegen der Daten.

d)
```
SELECT * FROM t_ma_frankfurt;
```

e)
```
UPDATE t_ma_frankfurt SET ort='Frankfurt/Main' WHERE ort='Frankfurt';
```