

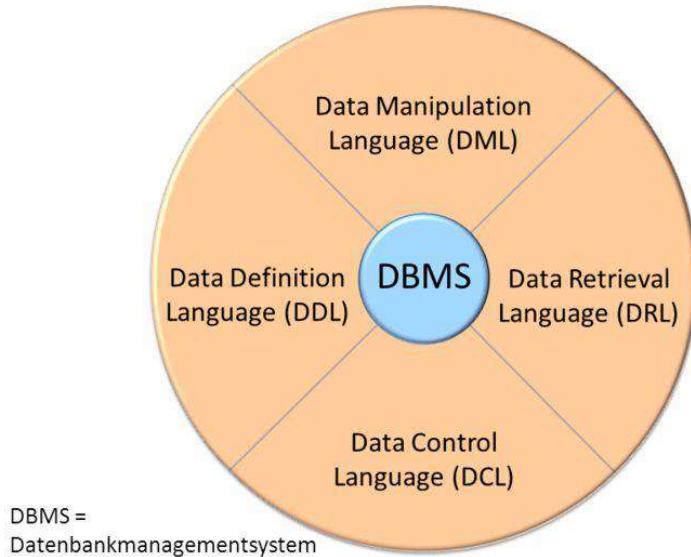
5 SQL -DDL

SQL ist eine Sprache zur Bearbeitung und Auswertung von relationalen Datenbanken. Sie umfasst drei Bereiche:

SQL Standard.

Teilsprachen.

Structured Query Language (SQL)



- Data Definition Language (DDL): Befehlssatz zum Anlegen, Ändern und Löschen von Datenbanken, Tabellen usw. und ihren Strukturen.
- Data Manipulation Language (DML): Befehlssatz zum Einfügen, Ändern, Löschen und Auslesen von Daten aus den Tabellen.
- Data Control Language (DCL): Befehlssatz zur Administration von Datenbanken

Anders als bei imperativen Programmiersprachen wie C#, C++, Java oder Pascal wird durch die Befehle nicht die Art und Weise bestimmt, wie man ein Ergebnis erhält; es wird kein Algorithmus implementiert. Vielmehr sagt man, was man haben möchte, und der Datenbanksystem ermittelt das Ergebnis. Solche Arten von Programmiersprachen nennt man *deklarativ*.

Obwohl es viele SQL-Dialekte gibt, ist der offizielle SQL-Standard in vielen Systemen implementiert und garantiert eine Wiederverwendbarkeit oder Übertragbarkeit der Befehle.

1986 wurde der erste SQL-Standard vom ANSI verabschiedet, der 1987 von der ISO ratifiziert wurde. 1992 wurde der Standard überarbeitet und als SQL-92 (oder auch SQL2) veröffentlicht. Alle aktuellen Datenbanksysteme halten sich im Wesentlichen an diese Standardversion. Die Version SQL:1999 (ISO/IEC 9075:1999, auch SQL3 genannt) ist noch nicht in allen Datenbanksystemen implementiert. Das gilt auch für die nächste Version SQL:2003. Der aktuelle Standard wurde 2008 unter SQL:2008 verabschiedet.

5.1 Anlegen/Löschen einer Datenbank

Wir gehen davon aus, dass die Datenbank komplett neu erstellt werden soll. Der Befehl zum Anlegen einer Datenbank ist CREATE SCHEMA. Damit werden allerdings noch keine Tabellen angelegt, sondern nur die diese umfassende Datenbank.

Bibliothek

```
--SQL 92:  
CREATE SCHEMA datenbankname  
    [ DEFAULT CHARACTER SET zeichensatz]  
  
--Mysql:  
CREATE { DATABASE | SCHEMA } [ IF NOT EXISTS ] datenbankname  
    [ [ DEFAULT ] CHARACTER SET zeichensatz]  
    [ COLLATE sortierung]  
;
```

5.1.1 Zuweisen eines CharacterSets

Was ist ein Zeichensatz? Auf dem Computer werden Buchstaben, Ziffern, Satz- und Sonderzeichen durch Zahlen kodiert. So ist beispielsweise der Buchstabe A im ASCII-Zeichensatz die Zahl 0x41 und a die Zahl 0x61. Da für die Kodierung des ASCII nur ein Byte (= 8 Bit) zur Verfügung steht, können nur 256 verschiedene Zahlen zur Kodierung verwendet werden. Die ersten 128 sind im Wesentlichen die Steuerzeichen (wie z.B. der Zeilenumbruch), das Leerzeichen, die lateinischen Buchstaben, die Ziffern 0 – 9, Satz- und einfache Sonderzeichen. Die restlichen 128 wurden mehr oder weniger willkürlich dazu verwendet, Umlaute oder andere sprachspezifische Sonderzeichen abzubilden. Und hier fing das Unglück an. Fast jeder Computer- oder Betriebssystemhersteller hat da sein eigenes Süppchen gekocht. So ist beispielsweise das Zeichen Ü im Zeichensatz ISO/IEC 8859-1 mit der Zahl 0xDC kodiert und in Codepage 850 mit 0x9A. Wird nun ein Text unter Windows erfasst, wird das Ü als 0xDC in die Datei geschrieben. Öffnet man nun diese Datei mit einem COMMAND-Editor wie EDIT, so erscheint aber ein anderes Zeichen und umgekehrt. Dieses Problem und die Beschränkung auf 256 Zeichen, was die Darstellung z.B. ostasiatischer Schriften unmöglich macht, haben dazu geführt, dass man eine neue, leicht erweiterbare Kodierung von Schriftzeichen baute. Unicode ward geboren! Unicode selbst liegt in verschiedenen Formatierungen vor. Derzeit gerne verwendet werden utf8, utf16 und utf32. Welche Zeichensätze von Ihrem Server unterstützt werden, können Sie leicht mit SHOW CHARACTER SET herausfinden.

5.1.2 Sortierreihenfolge

Für jede Sprache gibt selbst bei gleichen Zeichensätzen oft mehrere Arten, die Texte wie z.B. für eine Namensliste zu sortieren. In MySQL wird die Sortierreihenfolge über die Option **COLLATE** im CREATE SCHEMA festgelegt.

Die verfügbaren Sortierungen lassen sich leicht mit SHOW COLLATION anzeigen, wobei schnell deutlich wird, dass es mehrere Sortierreihenfolgen für einen Zeichensatz geben kann. Betrachten wir die Sortierreihenfolgen für den Zeichensatz cp850:

```
mysql> SHOW COLLATION LIKE 'cp850%';
+-----+-----+-----+-----+-----+
+          | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+
+          | cp850_general_ci | cp850 | 4 | Yes      | Yes        | 1       |
| cp850_bin           | cp850 | 80 |          | Yes      | 1         |
+-----+-----+-----+-----+-----+
+                                         2 rows in set (0.00 sec)
```

Bei cp850_general_ci wird nicht und bei cp850_bin wird zwischen Groß- und Kleinschreibung unterschieden.

5.1.3 Löschen einer Datenbank

Das Löschen einer Datenbank erfolgt analog zum Erstellen mit dem Befehl **DROP SCHEMA**

```
--SQL92
DROP SCHEMA datenbankname
[ CASCADE | RESTRICT ]
;
--MySQL
DROP { DATABASE | SCHEMA } [ IF EXISTS ] datenbankname;
```

Mit Hilfe von CASCADE bzw. RESTRICT werden in der Datenbank existierende Tabellen entweder gleich mitgelöscht bzw. das Löschen der Datenbank wird verhindert, solange diese noch Tabellen aufweist.

5.2 Anlegen einer Tabelle

```
CREATE TABLE tabellenname (
    spaltenspezifikation
    [, spaltenspezifikation]*
    [, PRIMARY KEY(spaltenliste) ]
) [tabellenoptionen];
```

Der erste Teil des Befehls ist selbsterklärend. Danach kommt der Tabellenname, den wir der Namenskonvention entsprechend klein schreiben. Was ist aber eine *spaltenspezifikation*? Eine Spaltenspezifikation besteht aus drei Teilen:

1. Spaltenname: Er wird klein geschrieben und ergibt sich aus dem ER-Modell.
2. Datentyp: Dieser legt fest, was für eine Art von Information in der Spalte verwaltet und wie diese kodiert wird. Mögliche Datentypen finden Sie in Abschnitt 25.1 auf Seite 371.
3. Zusätze: Mit diesen kann man eine Spalte ausführlicher bestimmen. Eine Liste möglicher Zusätze finden Sie weiter unten.

Das aus der Notation für reguläre Ausdrücke entnommene Sternchen * hinter der optionalen zweiten Spaltenspezifikation bedeutet: eine beliebige Anzahl viele, also auch 0.

```
use database artikel;
CREATE TABLE adresse (
    adresse_id INT UNSIGNED AUTO_INCREMENT,
    strasse VARCHAR(255),
    hnr VARCHAR(255),
    lkz CHAR(2),
    plz CHAR(5),
    ort VARCHAR(255),
    deleted TINYINT UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY(adresse_id)
);
```

Beispiele

```

drop database if exists employee; (1)
create database employee;

use employee; (2)

create table department (3)
(
    departmentID int not null auto_increment primary key,
    name varchar(20)
) type=InnoDB;

create table employee (4)
(
    employeeID int not null auto_increment primary key,
    name varchar(80),
    job varchar(15),
    departmentID int not null
        references department(departmentID)
) type=InnoDB;

create table employeeSkills (5)
(
    employeeID int not null references employee(employeeID),
    skill varchar(15) not null,
    primary key (employeeID, skill)
) type=InnoDB;

create table client
(
    clientID int not null auto_increment primary key,
    name varchar(40),
    address varchar(100),
    contactPerson varchar(80),
    contactNumber char(12)
) type=InnoDB;

create table assignment
(
    clientID int not null references client(clientID),
    employeeID int not null references employee(employeeID),
    workdate date not null,
    hours float,
    primary key (clientID, employeeID, workdate)
) type=InnoDB;

```

1. Diese Anweisung stellt fest, ob die DB schon existiert und löscht sie gegebenfalls. Die Anweisung ist u.U. mit Vorsicht zu genießen.
2. Mit Hilfe von CREATE wird die Datenbank erstellt und durch USE zur aktuellen DB erklärt.
3. Die Tabelle verfügt über 2 Spalten, departmentID als Primärschlüssel und name als Abteilungsnamen. Als Tabellentyp ist InnoDB angegeben.
 - departmentID: Datentyp ist int (Integer)
 - not null; die Spalte muss in jeder Zeile einen Wert haben
 - auto-increment; der Wert wird von MySQL automatisch hochgezählt
 - primary key: Die Spalte ist der Primärschlüssel der Tabelle
 - Name: Die Spalte kann maximal 20 alphanumerische Zeichen aufnehmen (varchar(20))
 - Die Spalten werden in Form einer kommaseparierten Liste geführt. Üblicherweise wird immer eine Objektbeschreibung je Zeile geschrieben. Der SQL-Interpreter führt die Zeile erst aus, nachdem er ein Semikolon (;) gefunden hat.
4. Mit Hilfe des Schlüsselwortes REFERENCES weist man die Spalte departmentID als Fremdschlüssel aus. Sie verweist auf den Primärschlüssel der Tabelle department..
5. Ein zusammengesetzter Primärschlüssel wird gebildet, indem in der Klammer auf mehrere Felder verwiesen wird. Die einzelnen Spalten werden durch Komma getrennt

5.2.1 Datentypen

Datentypen.

5.2.2 Constraints

Constraints definieren ganz allgemein gesprochen *Einschränkungen*, denen ein relationales Datenmodell entsprechen muss. Diese Einschränkungen können sein:

5.2.2.1 NOT NULL constraints

Die Einschränkung bedeutet, dass der Wert einer Spalte kein NULL-Wert sein darf.

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

5.2.2.2 CHECK constraint

Damit wird ausgedrückt, dass Werte in Spalten gewisse Bedingungen einzuhalten haben, z.B. bestimmte Größenbereiche

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CONSTRAINT positive_price CHECK (price > 0)
); --column check constraint (benannt positive_price)
```

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
); -- letzte zwei Zeilen sind table check constraints
```

Der Constraint wird erfüllt, wenn die Bedingung true ist oder einen NULL-Wert annimmt.

Der check-Constraint wird von MySQL zwar auf Syntaxfehler hin gepräst, jedoch nicht weiter umgesetzt :-).

5.2.2.3 UNIQUE constraint

Dieser Constraint fordert, dass für eine Spalte(n) innerhalb einer Tabelle eine Wert nur einmal vorkommt. NULL-Werte sind dennoch über mehrere Zeilen hinweg erlaubt. Er wird durch folgende Befehle abgebildet.

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
); -- column constraint
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no)
); -- table constraint

CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    UNIQUE (a, c) -- eindeutig über mehrere Spalten
);
```

5.2.2.4 PRIMARY/FOREIGN KEY Constraint

Ein PRIMARY KEY ist technisch gesehen ein UNIQUE constraint, der einen INDEX führt. Im Gegensatz zum UNIQUE Constraint dürfen die jeweiligen Spalten aber keinen NULL-Wert besitzen.

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

Ein FOREIGN KEY ist eine Einschränkung, dass sich der Wert einer Spalte an einem UNIQUE Wert einer anderen Tabelle orientieren muss. Ein FOREIGN KEY kann NULL-Werte besitzen, die gegenüberliegende Seite muss ein PRIMARY KEY oder ein UNIQUE constraint sein.

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
-- Zusammengesetzter Fremdschlüssel mit Namen fk_myFKey
-- Gut zum späteren Löschen des FK.
CREATE TABLE t1 (
    a integer PRIMARY KEY,
    b integer,
    c integer,
    CONSTRAINT fk_myFKey FOREIGN KEY (b, c)
        REFERENCES other_table (c1, c2)
);
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    ...
);

-- Lösch/Änderungsweitergabe verhindern oder erlauben
```

```
CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id integer REFERENCES orders

    ON DELETE CASCADE ON UPDATE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

In MySQL muss als Tabellentyp InnoDB angegeben sein. Weiterhin muss vorher ein Index auf die Fremdschlüsselspalte vergeben worden sein.

```
CREATE TABLE product (
    category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)
) TYPE=INNODB;

CREATE TABLE customer (
    id INT NOT NULL,
    PRIMARY KEY (id)
) TYPE=INNODB;

CREATE TABLE product_order (
    no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,
    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    FOREIGN KEY (product_category, product_id)
    REFERENCES product(category, id)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    INDEX (customer_id),
    FOREIGN KEY (customer_id) REFERENCES customer(id)
) TYPE=INNODB;
```

5.2.3 Ändern/Löschen von Datenstrukturen

Mit Hilfe des Befehl **ALTER** kann die Struktur einer bestehenden Tabelle verändert werden. Dazu wird dem Statement je nach Bedarf eine drop, add, change, modify - Klausel hinzugefügt.

Der grundlegende Aufbau sieht wie folgt aus.

```
ALTER [IGNORE] TABLE tbl_name alter_specification [, alter_specification ...]

alter_specification:
    ADD [COLUMN] create_definition [FIRST | AFTER column_name]
    | ADD [COLUMN] (create_definition, create_definition,...)
    | ADD INDEX [index_name] (index_col_name,...)
    | ADD PRIMARY KEY (index_col_name,...)
    | ADD UNIQUE [index_name] (index_col_name,...)
    | ADD FULLTEXT [index_name] (index_col_name,...)
    | ADD [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name, ...)
        [reference_definition]
    | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
    | CHANGE [COLUMN] old_col_name create_definition
        [FIRST | AFTER column_name]
    | MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]
    | DROP [COLUMN] col_name
    | DROP PRIMARY KEY
    | DROP INDEX index_name
    | DISABLE KEYS
    | ENABLE KEYS
    | RENAME [TO] new_tbl_name
    | ORDER BY col
    | table_options
```

Das folgende Beispiel zeigt den Umgang mit dem ALTER TABLE-Statement.

```
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.1-alpha-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database testAlter;
mysql> use testAlter;
      Wir beginnen mit dem Erzeugen einer Demodatenbank

mysql> -- Wir erzeugen eine Tabelle t1
      CREATE TABLE t1 (a INTEGER,b CHAR(10));

      -- Wir ändern den Tabellennamen von t1 in t
mysql> ALTER TABLE t1 RENAME t2;

      --Wir ändern den Spaltentyp von a in TINYINT NOT NULL und
ändern
      --den Spaltentyp von b nach CHAR(20) und geben der Spalte b
den
      -- Namen c .
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c
CHAR(20);

      --Wir fügen eine neue Spalte d mit Datentyp TIMESTAMP hinzu.
mysql> ALTER TABLE t2 ADD d TIMESTAMP;

      --Wir fügen einen Index auf die Spalte d hinzu und machen aus
      --der Spalte a einen Primärschlüssel.
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);

      --Wir löschen die Spalte c
mysql> ALTER TABLE t2 DROP COLUMN c;

      --Wir fügen eine neue Spalte c mit dem Datentyp INTEGER hinzu.
      -- Der Wert soll sich automatisch hochzählen.
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
      ADD INDEX (c);
```

5.2.4 Einfügen/Ändern/Löschen von Daten

5.2.4.1 INSERT

Mit Hilfe des Befehls INSERT wird ein neuer Datensatz in eine Tabelle eingefügt. Der INSERT-Befehl ist in verschiedenen Formen verfügbar.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
INTO tbl_name [(Feld1, Feld2, ...)]
VALUES ({expr | DEFAULT}, ...), (...), ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Nach dem Nennen der Tabelle erfolgt die Definition der Feldnamen; mit Hilfe des Schlüsselwortes **VALUES** werden dann die jeweiligen Werte übergeben.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
INTO tbl_name
SET col_name={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Das Einfügen der Werte erfolgt explizit durch eine **Feld=Wert-Zuweisung**

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
INTO tbl_name [(col_name, ...)]
SELECT ...
```

Die zum Füllen der Felder erforderlichen Werte kommen aus einem SELECT Statement

5.2.4.2 REPLACE

Mit Hilfe des Befehls REPLACE wird ebenfalls ein neuer Datensatz in eine Tabelle eingefügt. Wird jedoch ein Wert für einen Primärschlüssel oder einen UNIQUE KEY mit übergeben, so wird der alte Datensatz vor dem Einfügen des neuen Datensatzes gelöscht. Verschiedene Formen sind möglich.

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name, ...)]
VALUES ({expr | DEFAULT}, ...), (...), ...

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name, ...)]
SELECT ...
```

5.2.4.3 DELETE, UPDATE

Mit Hilfe des **DELETE**-Befehls werden Datensätze in einer Tabelle gelöscht. Es ist dabei zu beachten, dass **ohne einen WHERE-Teil alle Datensätze in einer Tabelle gelöscht** werden.

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name  
[WHERE where_definition]  
[ORDER BY ...]  
[LIMIT row_count]
```

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]  
tbl_name[.*] [, tbl_name[.*] ...]  
FROM table_references  
[WHERE where_definition]
```

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]  
FROM tbl_name[.*] [, tbl_name[.*] ...]  
USING table_references  
[WHERE where_definition]
```

5.2.4.4 Aufgaben zu UPDATE, INSERT, DELETE

Lösung Link.

1. Folgende Tabelle ist gegeben

```
mysql> DESCRIBE twounique; SELECT * FROM twounique;
```

| Field | Type | Null | Key | Default | Extra |
|-------|---------------------|------|-----|---------|-------|
| id1 | tinyint(3) unsigned | | PRI | 0 | |
| id2 | tinyint(3) unsigned | | UNI | 0 | |

| id1 | id2 |
|-----|-----|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

Sie führen folgende beiden Befehle aus. Was ist anschließend der Inhalt der Tabelle?

```
mysql> REPLACE INTO twounique VALUES (2,2);
mysql> REPLACE INTO twounique VALUES (2,6);
```

2. Wie fängt man mehrere Datensätze mit Hilfe eines einzigen INSERT-Statements hinzu
3. INSERT unterstützt den sog. IGNORE Modifier, REPLACE jedoch nicht. Warum ist das so ?
4. Mit welchem Statement kann man eine Tabelle komplett leeren ?
5. Mit welchem Statement kann man eine Tabelle teilweise leeren ?
6. Welcher Unterschied besteht in der Art und Weise, wie MySQL Fehler behandelt für ein single-row oder multiple-row-statement, wenn NULL-Werte in ein NOT NULL-Feld eingefügt werden?
7. Was für Gründe kann es geben, wenn ein UPDATE-Statement keine Auswirkungen hat, d.h. keinen einzigen Wert ändert.
8. Warum ist die Zahl der betroffenen Zeilen im folgenden UPDATE-Statement 0, obwohl die WHERE-Klausel auf 5 Zeilen zutrifft ? Warum ist die Anzahl der zutreffenden Zeilen 5 und nicht 10?

```

mysql> SELECT pid, grade FROM personnel;
+----+-----+
| pid | grade |
+----+-----+
| 1   |     1 |
| 2   |     2 |
| 3   |   NULL |
| 4   |   NULL |
| 5   |   NULL |
| 6   |     1 |
| 7   |     1 |
| 8   |     1 |
| 9   |   NULL |
| 10  |   NULL |
| 11  |   NULL |
| 12  |     1 |
| 13  |   NULL |
+----+-----+
13 rows in set

mysql> UPDATE personnel SET grade = 1 WHERE grade != 2;
Query OK, 0 rows affected (0.00 sec)
rows matched: 5  Changed: 0  Warnings: 0

```

9. Ist das folgende Statement WAHR oder FALSCH?

Um zu verhindern, dass UPDATE-Statements alle Zeilen einer Tabelle ändern würden, kann man mysql mit der **-safe-updates-Option** starten

10. Folgende Tabelle ist gegeben.

```

mysql> SELECT * FROM personnel;
+----+-----+-----+
| pid | unit | grade |
+----+-----+-----+
| 1   |    42 |     1 |
| 2   |    42 |     2 |
| 3   |    42 |   NULL |
| 4   |    42 |   NULL |
| 5   |    42 |   NULL |
| 6   |    23 |     1 |
| 7   |    23 |     1 |
| 8   |    23 |     1 |
| 9   |    23 |   NULL |
| 10  |    42 |   NULL |
| 11  |    23 |   NULL |
| 12  |    23 |     1 |
| 13  |    42 |   NULL |
+----+-----+-----+

```

Mit welchem einzigen UPDATE-Statement würde man alle Zeilen, die im Feld grade keinen Wert besitzen, auf 3 ändern.

11. Beziehen Sie sich auf die vorhergehende Tabelle. Welches REPLACE-Statement würden Sie benutzen, um das grade-Feld auf 4 und das unit-Feld auf 45 für alle Zeilen zu ändern, wo das pid-Feld den Wert 10 hat.

12. Die Tabelle personell hat den folgenden Aufbau:

```
mysql> DESCRIBE personnel; SELECT * FROM personnel;

+-----+-----+-----+-----+-----+
+
| Field | Type          | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+
+
| pid   | smallint(5) unsigned |      | PRI | NULL    | auto_incremnt |
|
| unit  | tinyint(3) unsigned | YES  |      | NULL    |             |
| grade | tinyint(3) unsigned | YES  |      | NULL    |             |
+-----+-----+-----+-----+-----+
+
+-----+-----+-----+
| pid | unit | grade |
+-----+-----+-----+
| 1   | 42   | 1     |
| 2   | 42   | 2     |
| 3   | 42   | 3     |
| 4   | 42   | 3     |
| 5   | 42   | 3     |
| 6   | 23   | 1     |
| 7   | 23   | 1     |
| 8   | 23   | 1     |
| 9   | 23   | 3     |
| 10  | 42   | 3     |
| 11  | 23   | 3     |
| 12  | 23   | 1     |
| 13  | 42   | 3     |
+-----+-----+-----+
```

Welches UPDATE-Statement benutzen Sie, um die Werte des Feldes grade mit 1000 zu multiplizieren. Welche Werte würde das Statement erzeugen.

13. In the table personnel, the unit numbers were interchanged for some reason. Unit 23 is supposed to be 42, and 42 is supposed to be 23. What statement would you use to resolve this problem? Currently the table looks like this:

```
mysql> SELECT * FROM personnel;

+----+----+----+
| pid | unit | grade |
+----+----+----+
| 1   | 42   | 255  |
| 2   | 42   | 255  |
| 3   | 42   | 255  |
| 4   | 42   | 255  |
| 5   | 42   | 255  |
| 6   | 23   | 255  |
| 7   | 23   | 255  |
| 8   | 23   | 255  |
| 9   | 23   | 255  |
| 10  | 42   | 255  |
| 11  | 23   | 255  |
| 12  | 23   | 255  |
| 13  | 42   | 255  |
```

14. The table petnames contains the following data:

```
mysql> SELECT * FROM petnames;

+----+
| name |
+----+
| Lucy |
| Macie|
| Myra |
| Cheep |
| Lucy |
| Myra |
| Cheep |
| Macie |
| Pablo |
| Stefan|
+----+
```

Assume that you issue the following statement:

```
UPDATE petnames SET name = CONCAT(name, 1) ORDER BY name LIMIT 1;
```

What will the table's contents be after the UPDATE?

15. Will the following statement delete all rows from the table mytable ?

```
TRUNCATE TABLE mytable;
```

16. Will the following statement delete all rows from the table mytable ?

```
DELETE FROM mytable;
```

