1 Data Definition Language DDL

DDL ist eine Untergruppe von SQL und dient dazu, die Struktur der Datenbank festzulegen. Hierbei wird hauptsächlich das Augenmerk auf die Tabellen gelegt – sprich das „`CREATE TABLE`“ Statement. Im Weitesten Sinne ist jedoch auch das „`CREATE DATABASE`“ Statement ein Teil von DDL.

Wenn wir eine Datenbank erzeugen wollen, so wird dies mit dem folgenden Statement erledigt, wobei „*Datenbankname*“ für einen eindeutigen Namen der Datenbank innerhalb des RDBMS steht:

```
CREATE DATABASE Datenbankname;
```

Je nach RDBMS gibt es hier eine Vielzahl von Ergänzungen dieses Befehls, bspw. um den Standardzeichensatz zu festzulegen, worauf wir hier verzichten. Um nun mit der Datenbank zu arbeiten, muss sie geöffnet werden:

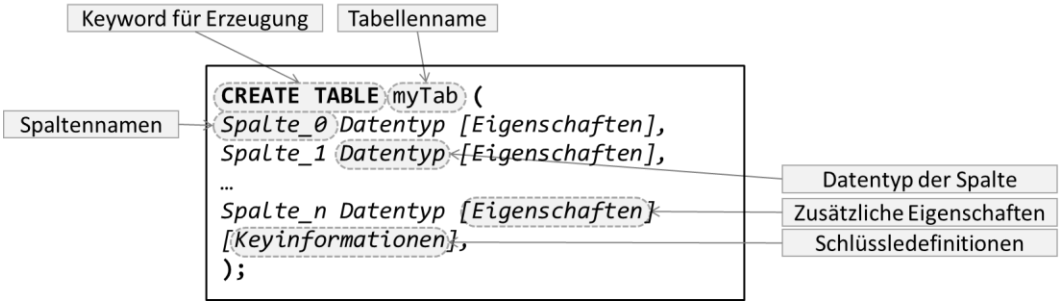
```
USE Datenbankname;
```

Ab diesem Statement wird jeder Befehl innerhalb der Datenbank mit dem Namen „*Datenbankname*“ durchgeführt. Mit „`SHOW DATABASES`“ kann man übrigens prüfen, ob die Datenbank überhaupt existiert, bevor man sie mit `USE` öffnet.

Die beiden wohl wichtigsten DDL-Aktionen auf Tabellenebene sind `CREATE TABLE` (Tabelle erzeugen) und `ALTER TABLE` (Tabelle strukturell anpassen). Wir werden uns an dieser Stelle erstmal auf das `CREATE TABLE` Statement beschränken.

2 CREATE TABLE

Es beginnt immer mit `CREATE TABLE`, gefolgt von einem Tabellennamen. Dieser muss innerhalb der Datenbank eindeutig sein, darf kein anderweitig belegtes Schlüsselwort sein und darf nicht mit einer Ziffer beginnen. Danach folgend die Spaltennamen. Diese müssen wiederum innerhalb der Tabelle eindeutig sein und dürfen ebenfalls kein Schlüsselwort sein und nicht mit Ziffern beginnen. Nach jedem Spaltennamen folgen die Eigenschaften der Spalte. Zuerst kommt der Datentyp, gefolgt von zusätzlichen Eigenschaften. Vor dem nächsten Spalteneintrag folgt immer ein Komma. Am Schluss können noch Primärschlüssel oder Fremdschlüssel festgelegt werden.



Es ist zwar noch möglich, weitere zusätzliche Informationen abzulegen, dies ist für den ersten Schritt jedoch nicht notwendig. Details finden sich in der Onlinedokumentation von MySQL (oder entsprechend anderen Datenbanken).

2.1 Datentypen

In Datenbanken spielt die möglichst exakt an die Anforderung angepasste Auswahl der Datentypen eine sehr große Rolle, da bei einer entsprechend großen Anzahl an Datensätzen eventuelle „Verschwendung“ von Speicher schnell ins Gewicht fällt. Insofern ist eine Abwägung zwischen

- Flexibilität des Datentyps vs. Performance
- Zu erwartende Größe eines Datenwertes vs. Zukunftssicherheit

wichtig. Wenn wir bspw. nur Zahlen zwischen 0 und 100 erwarten, so ist `TINYINT` ausreichend – wir würden hier nicht wie bspw. in Java einen `INT` Datentyp verwenden.

Beginnen wir mit den Datentypen für Textinformationen. Hier unterscheiden wir folgende Typen:

Typ:	**Bytes[1]:**	**Bemerkung:**
`CHAR(M)`	1 Byte pro „M“	String mit fester Länge „M“. M darf die Werte 0 bis 255 annehmen.
`VARCHAR(M)`	1 Byte pro „L“ + 1 (bzw. wenn M > 255 + 2)	String mit variabler Länge „L“ bis maximal „M“. M darf die Werte 0 bis 65.532 annehmen.
`TINYTEXT`	L + 1 Byte	String mit variabler Länge „L“ bis maximal 2^8 Zeichen.
`TEXT`	L + 2 Byte	String mit variabler Länge „L“ bis maximal 2^{16} Zeichen.
`MEDIUMTEXT`	L + 3 Byte	String mit variabler Länge „L“ bis maximal 2^{24} Zeichen.
`LONGTEXT`	L + 4 Byte	String mit variabler Länge „L“ bis maximal 2^{32} Zeichen.

Alternativ zu `CHAR` und `VARCHAR` kann man `BINARY` und `VARBINARY` verwenden, wobei hier immer case-sensitiv verglichen wird. Wir nutzen im Regalfall `CHAR` und `VARCHAR` Für normale Textinhalte wird `CHAR` bei fixen Längen (wie bspw. PLZ oder Kennzeichen wie „Geschlecht“) und `VARCHAR` bei variablen Längen verwendet. Ganze Zahlen werden in `*INT` Datentypen abgelegt:

Typ:	**Bytes:**	**Min:**	**Max:**	**Bemerkung:**
`TINYINT`	1	-128	127	Synonym wird auch BOOL verwendet.
`SMALLINT`	2	-32768	32767	
`MEDIUMINT`	3	-8388608	8388607	Ist nicht Teil des SQL Standards
`INT`	4	-2147483648	2147483647	Es geht auch INTEGER
`BIGINT`	8	-9223372036854775808	9223372036854775807	

Nachkommazahlen werden entweder in Gleitkomma- oder Festkommadatentypen abgelegt:

Typ:	**Bytes:**	**Min:**	**Max:**	**Bemerkung:**
`FLOAT`	4	-3.402823466E+38	3.402823466E+38	Es handelt sich hier um eine „Fließkommazahl“. Die Werte können Hardwarebedingt abweichen.
`DOUBLE`	8	-1.797693134862315 7E+308	1.797693134862315 7E+308	Es handelt sich hier um eine „Fließkommazahl“. Die Werte können Hardwarebedingt abweichen.
`DECIMAL [(M[,D])]`	-	-	-	Es handelt sich hier um eine „Festkommazahl“. M gibt die Anzahl der Ziffern und D die Anzahl der Nachkommastellen an.

Beträge (also Geldwerte) sollten aufgrund der Rundungsfehler von Gleitkommazahlen besser in Festkommazahlen abgelegt werden (also bspw. `DECIMAL(8,2)`).

Datum und Zeit werden in eigenen Formaten abgelegt:

Typ:	**Bytes:**	**Bemerkung:**
`DATE`	3 Bytes	Datum im Format „YYYY-MM-DD“.
`TIME`	3 Bytes	Zeit im Format „HH:MM:SS“
`DATETIME`	8 Bytes	Datum und Zeit im Format „YYYY-MM-DD HH:MM:SS“
`TIMESTAMP`	4 Bytes	Datum und Zeit im Format „YYYY-MM-DD HH:MM:SS“, wobei MySQL bei der Standardkonfiguration den Wert bei jeder Änderung des Datensatzes den Wert auf die Änderungszeit (+Datum) anpasst.

Nutzen Sie für Datums- und Zeitangaben niemals Zeichenketten, sondern immer die Datums- und Zeitdatentypen. Bei diesen können die Zeitfunktionen für Berechnungen genutzt werden.

[1] Wir gehen hier von einem Zeichensatz aus, der 1 Byte pro Zeichen benötigt.

3 Einfaches Create TABLE Statement

In der „First Table“ Übung haben wir bereits eine Tabelle „Kunde“ erzeugt. Hierbei wurden schon einige wichtige Datentypen verwendet:

```
CREATE TABLE Kunde (
  ID INT,
  Vorname VARCHAR(64),
  Nachname VARCHAR(64),
  Geschlecht CHAR(1),
  eMail VARCHAR(64),
  Telefon VARCHAR(32),
  Geburtsdatum DATE,
  KundeSeit DATE,
  LetzterLogin DATETIME
);
```

Bevor wir weitermachen, schauen wir uns die Struktur der Tabelle mit dem DESC Befehl an:

```
DESC Kunde;
```

Erzeugen Sie nun eine weitere Tabelle „Adresse“ für die Adressinformationen. Hierbei benötigen wir folgende Informationen:

Spaltenname:	**Bedeutung:**	**Datentyp:**
ID	Eindeutige Identifikation jedes Datensatzes.	
Bundesland	Bundesland, als Text ausgeschrieben	
PLZ	Postleitzahl – Achtung mit führende 0!	
Ort	Ortsname, als Text ausgeschrieben	
Strasse	Straßenname	
Hausnr	Hausnummer – Achtung, auch Buchstaben	
Typ	Kennzeichen ob Liefer-, Rechnungsadresse oder beides (L, R oder B)	

Überlegen Sie sich einen Datensatz und tragen Sie ihn mit einem `INSERT` Statement ein.

4 Nullable und Default

Ein Adressdatensatz kann nun nicht sinnvoll verarbeiten werden, wenn bspw. der Straßenname fehlt. Eine fehlender Spalteneintrag in einer Datenbank wird immer mit **`NULL`** gekennzeichnet – `NULL` ist also ein Zustand, kein Wert!

Eine Datenbank gibt uns nun auch die Möglichkeit, diesen Zustand zu verhindern. Hier können wir unsere Spaltendefinition um die Eigenschaft `NOT NULL` erweitern (sie sind damit „not nullable“).

```
CREATE TABLE Adresse (
  ID INT NOT NULL,
  Bundesland VARCHAR(32) NOT NULL,
  …
);
```

Entfernen Sie die Adresstabelle mit `DROP TABLE Adresse;` wieder und erzeugen Sie sie neu, so dass alle Felder „not nullable“ sind. Nun können wir das Verhalten analysieren.

Für nullable Felder gibt es zwei Möglichkeiten, den NULL-Zustand zu erzeugen. Entweder man setzt das Feld im INSERT Statement explizit auf NULL oder man lässt die Spalte im INSERT Statement einfach weg. Versuchen wir die erste Option:

```
INSERT INTO Adresse SET
ID = 1,
Bundesland = 'Mecklenburg-Vorpommern',
PLZ = '19205',
Ort = 'Holdorf',
Strasse = 'Clemensstr.',
Hausnr = '74a',
Typ = NULL;
```

Die Datenbank meldet folgenden Fehler:

```

```

Wir haben also den NULL-Zustand verhindert. Versuchen wir nun die zweie Möglichkeit, NULL zu provozieren:

```
INSERT INTO Adresse SET
ID = 1,
Bundesland = 'Mecklenburg-Vorpommern',
PLZ = '19205',
Ort = 'Holdorf',
Strasse = 'Clemensstr.',
Hausnr = '74a';
```

Für den Fall, dass die Datenbank im „strict mode“[2] läuft, erhalten wir als Fehlermeldung:

```
ERROR 1364 (HY000): Field 'Typ' doesn't have a default value
```

Ansonsten wird der Datensatz mit einem Warning geschrieben. Dieses können wir mit SHOW WARNINGS ausgeben lassen:

```
mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1364 | Field 'Typ' doesn't have a default value |
+---------+------+-----------------------------------------+
```

Es gibt also die Möglichkeit, einen Default-Wert anzugeben, der beim „Weglassen“ einer Spalte im INSERT Statement greift. Wir entfernen die Tabelle wieder und ergänzen beim Typ nun folgenden Default-Wert:

```
CREATE TABLE Adresse (
  ID INT NOT NULL,
  …
  Typ CHAR(1) NOT NULL DEFAULT "B"
);
```

Nun wird der Datensatz ohne Fehler oder Warning geschrieben. Sehen wir uns die Werte mit `SELECT * FROM Adresse;` an, sehen wir den Wert „B“ in der Spalte Typ.

[2] Wenn `SELECT @@sql_mode;` *STRICT_TRANS_TABLES* oder *STRICT_ALL_TABLES* ausgibt.