1 Data Definition Language DDL

Der Teilbereich von SQL für die Definition von Tabellen (und weiteren Datenbankelementen) ist die Data Definition Language. Es zählen folgende Aktionen hierzu – Erzeugung von:

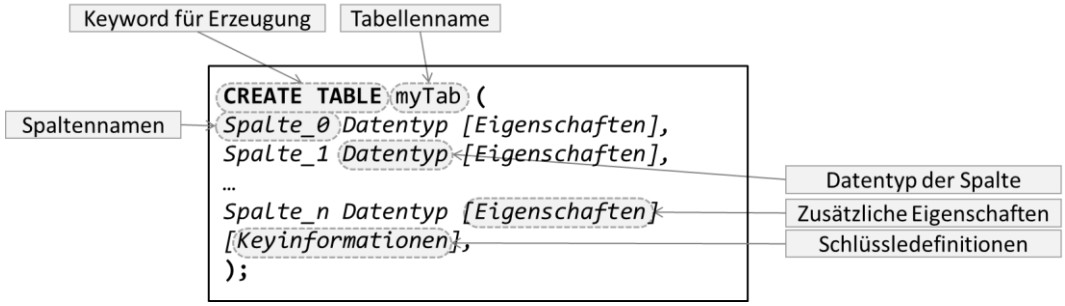
- Datenbanken
- Tabellen
- Views
- Indizes
- Trigger
- Stored Procedures und stored Functions

Oder auch die nachträgliche Änderung der entsprechenden Elemente – hier vor allem die Tabellen.

In diesem Dokument konzentrieren wir uns auf die Erzeugung von Tabellen.

2 Das CREATE TABLE Statement

Die Struktur des Statements lautet:



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.

Für normale Textinhalte wird CHAR bei fixen Längen 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 „Fließ-kommazahl“. Die Werte können Hardwarebedingt abweichen.
DOUBLE	8	-1.7976931348623157E+308	1.7976931348623157E+308	Es handelt sich hier um eine „Fließ-kommazahl“. Die Werte können Hardwarebedingt abweichen.
DECIMAL [(M[,D])]	-	-	-	Es handelt sich hier um eine „Fest-kommazahl“. M gibt die Anzahl der Ziffern und D die Anzahl der Nach-kommastellen an.

Beträge (also Geldwerte) sollten aufgrund der Rundungsfehler von Gleitkommazahlen besser in Festkomma-zahlen 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.

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

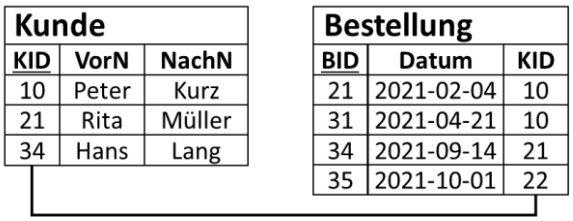
2.2 Zusätzliche Eigenschaften

Das Verhalten von Spalten bei Einfügen von Datensätzen lässt sich noch weiter festlegen. Hierbei sind die folgenden Eigenschaften die wichtigsten:

Eigenschaft:	Bedeutung:
DEFAULT	Defaultwert, wenn Spalte im Insert nicht berücksichtigt wird. Bspw. würde DEFAULT("Leer") den String „Leer“ immer dann einsetzen, wenn die Spalte beim Insert nicht vorkommt.
UNIQUE	Jeder Wert darf in dieser Spalte nur einmal vorkommen. Das Verhalten ist wie bei einem Primary Key – nur dass es kein Primary Key ist.
NOT NULL	NULL wird als Zustand der Spalte nicht akzeptiert.
PRIMARY KEY	Feld wird als Primärschlüssel festgelegt. Diese Form der Primärschlüsseldefinition geht nur, wenn ein einziges Feld ein Primärschlüssel sein soll. Bei kombinierten Schlüsseln muss ein anderer Syntax erfolgen (siehe Unten)
AUTO_INCREMENT	Bei Nichtbelegung des Feldes im Insert, wird der bis dato höchste, in dieser Spalte vorgekommene Wert + 1 eingetragen. Dies geht nur bei

2.3 Schlüsseldefinitionen

Primär- und Fremdschlüssel können als eigene Zeile im CREATE TABLE Statement festgelegt werden. In folgender Grafik ist die KID der Primärschlüssel der Tabelle Kunde und ein Fremdschlüssel in der Tabelle Bestellung.



Kunde

KID	VorN	NachN
10	Peter	Kurz
21	Rita	Müller
34	Hans	Lang

Bestellung

BID	Datum	KID
21	2021-02-04	10
31	2021-04-21	10
34	2021-09-14	21
35	2021-10-01	22

Der Primärschlüssel kann (neben dem in Kapitel 2.2 dargestellten Möglichkeit) wie folgt im CREATE TABLE Statement der Kundentabelle erstellt werden:

```
PRIMARY KEY(KID)
```

Hierbei können in den Klammern auch mehrere Spalten der Tabelle eingetragen werden, wodurch wir einen kombinierten Schlüssel erhalten. Primärschlüsselfelder sind automatisch NOT NULL, so dass dies nicht explizit angegeben werden muss.

Bei Fremdschlüsseln müssen wir neben dem Feld in der eigentlichen Tabelle auch festlegen, auf welches Feld in welcher Tabelle er verweist. Für unser Beispiel wäre dies im CREATE TABLE Statement der Bestellungstabelle:

```
FOREIGN KEY(KID) REFERENCES Kunde(KID)
```

Wenn wir Fremdschlüssel festgelegt haben, ist folgendes zu beachten:

- Die referenzierte Tabelle (also Kunde) muss vor der referenzierenden (also Bestellung) erstellt werden
- Ein Datensatz der Bestellungstabelle muss einen KID Eintrag haben, der auch tatsächlich in Kunde existiert.
- Beim Löschen von Datensätzen muss zuerst der Bestelldatensatz, dann erst der Kundendatensatz gelöscht werden.
- Beim Löschen von Tabellen muss zuerst muss zuerst die referenzierende Tabelle, dann die referenzierte Tabelle gelöscht werden.

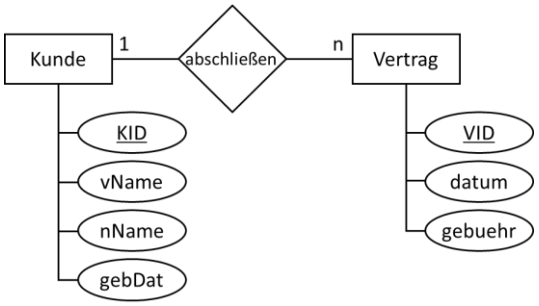
Um beim Löschen und Ändern die Referenzielle Integrität zu bewahren (also die Garantie, dass der referenzierte Wert auch tatsächlich existiert), können auch „Constraints“ hinterlegt werden, so dass das Löschen eines

Kunden automatisch das Löschen der Bestellung mit sich zieht, bzw. die Änderung einer KID in der Kundentabelle die Änderungen gleich bei der Bestellungstabelle nachzieht:

```
FOREIGN KEY(KID) REFERENCES Kunde(KID)
ON DELETE CASCADE
ON UPDATE CASCADE
```

3 Aufgabenstellung

Erstellen Sie die CREATE TABLE Statements für folgendes ER-Diagramm:



Geben Sie jedem Feld einen sinnvollen Datentypen. Weiterhin darf kein Feld den Zustand NULL annehmen. Das Feld Gebühr soll als Defaultwert 100,00 aufweisen. Sorgen Sie dafür, dass beim Löschen des Kunden automatisch alle zugehörigen Verträge gelöscht werden. Die Primärschlüsselfelder sollen sich automatisch befüllen.